McCann, Michael John

AN APPLICATION OF CONVEX PROGRAMMING TO CONSTRUCTION SCHEDULING

Iowa State University PH.D. 1986

PLEASE NOTE:

In all cases this material has been filmed in the best possible way from the available copy. Problems encountered with this document have been identified here with a check mark __✓__.

1. Glossy photographs or pages _____
2. Colored illustrations, paper or print _____
3. Photographs with dark background _____
4. Illustrations are poor copy _____
5. Pages with black marks, not original copy _____
6. Print shows through as there is text on both sides of page _____
7. Indistinct, broken or small print on several pages ✓
8. Print exceeds margin requirements _____
9. Tightly bound copy with print lost in spine _____
10. Computer printout pages with indistinct print _____
11. Page(s) _____ lacking when material received, and not available from school or author.
12. Page(s) _____ seem to be missing in numbering only as text follows.
13. Two pages numbered _____. Text follows.
14. Curling and wrinkled pages _____
15. Dissertation contains pages with print at a slant, filmed as received _____
16. Other _____

_____

_____

University
Microfilms
International

An application of convex programming to construction scheduling

by

Michael John McCann

A Dissertation Submitted to the
Graduate Faculty in Partial Fulfillment of the
Requirements for the Degree of
DOCTOR OF PHILOSOPHY
Department: Industrial Engineering
Major: Engineering Valuation

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For the Major Department

Signature was redacted for privacy.

For the Graduate College

Iowa State University
Ames, Iowa

1986

TABLE OF CONTENTS

INTRODUCTION

The first commercially available digital computer was introduced in 1956. With this computer, or the UNIVAC computer, the DuPont and Remington Rand corporations developed the project scheduling method called the critical path method (CPM) and successfully applied the method to the planning and control of several of DuPont's construction projects.

DuPont's successful applications of the CPM demonstrated the value of computer aided scheduling methods as a construction management tool, and encouraged many large companies and government agencies to specify CPM in their construction related contracts.

As a result of these CPM contract specifications, the courts of the United States in 1972[1] required the use of CPM schedules for construction litigation. This decision set a precedent for the use of project scheduling and firmly established CPM as the basic tool of both government and industry for the administration of construction contracts.

Unfortunately, the benefits derived from CPM by industry and government were not shared by construction contractors. Contractors often viewed, and sometimes found, owner required CPM schedules a restriction on their freedom to execute a project. Even though the Association of General Contractors supported the use of CPM as early as 1965, by 1971 only a minority of contractors[2] had actually integrated the method into their business procedures.

This fact was again confirmed in 1983 by the President's Report on Federal Construction Management.

> "Formal detailed (CPM) schedules are often not prepared and more often not maintained. Management information systems concentrate on financial data rather than the physical status of construction and related activities. Project control systems are not clearly promulgated nor adequately utilized."[3]

This lack of acceptance of CPM, and CPM related methods, by construction contractors is due in part to the following inadequacies of the method.

(1) CPM lacked data structure to integrate with the estimating and cost control methods used by most contractors. (Although the government tried to integrate cost control and CPM with the Cost Schedule Control System criteria (CSCS), the results were controversial.)

(2) CPM lacked the flexibility needed to track the rapid and dynamic progress of construction projects. Changing activity durations and construction sequences often made complex CPM networks obsolete before a project could even start[2].

(3) CPM lacked the ability to systematically seek optimal combinations of time, cost, manpower, and materials in a manner that would reduce contractor costs. This inadequacy became critical when the court case of Blackhawk Heating and Plumbing Company[1] implied that any contractor submitted CPM schedule was considered "optimal" and any free schedule time (float) was available for the owner's (the government's) use.

Extensive research has been done on network scheduling since the initial Dupont CPM method; but in general, this research has not been applied to the construction contractor's needs. From the evident reluctance of construction contractors to adopt CPM, it appears that the current theories and technologies should be used, possibly in "hybrid" forms, to solve some of the inadequacies of the present CPM network scheduling method.

History of Construction Scheduling Methods

"Scientific management" was first applied to project scheduling by Gantt[4] in 1917 with the use of the Gantt or bar chart. Much of the noncomputerized scheduling done today is still based on this turn of the century method.

It was not until 40 years later with the introduction of the UNIVAC computer that Walker and Sayer[5] of DuPont developed the network scheduling method of CPM. With CPM, large companies and government agencies had the tool to manage large scale multi-phased construction contracts.

Parallel to the development of CPM, the consulting team of Booz, Allen, and Hamilton developed the probabilistic networking method called Project Evaluation and Review Technique[6] (PERT) for the Navy's Polaris missile program. Unfortunately, the complexity and the added information required by the method discouraged its widespread use.

These methods laid the foundation of network scheduling and represent the majority of scheduling method applications in the

construction industry today.

In an attempt to expand CPM to include resources other than time, such as costs, manpower, and materials, and to search for optimal combinations of these resources; research has continued since 1958 in three (3) major directions -- the analytical methods, the heuristic methods, and simulation[7].

The Analytical Methods

The work of Dantzig, Gomory, and Bellman provided the framework for the research in the analytical methods. Unfortunately, little of this work has had commercial application in construction scheduling although manufacturing has used extensively the linear programming techniques developed by Dantzig.

Of the analytical methods, Dantzig's simplex method naturally lent itself to optimizing CPM schedules with linear activity cost functions[8]. This led to the work of Kelley[9], Charnes and Cooper[10], and Fulkerson[11] who in the early 1960s solved CPM network schedules with linear activity cost functions by means of a primal-dual flow interpretation of the simplex method. This research provided the basis for Kapur's[12] method for optimizing CPM schedules with parabolic activity cost functions in 1973 using a modification of Fulkerson's method. Neither approach has seen widespread application in construction scheduling, and Kapur's method is not available in a computerized form.

Gomory's work with integer programming in the 1960s provided the solutions to the integer resource constrained machine schedule models of

Wagner[13] (processing order), Bowman[14] (time period), and Manne[15] (start time period), but the scale of the models were more suited for manufacturing than for construction. In practice, processing time made these models impractical for full scale construction applications.

Pritsker, Watters and Wolfe[16] (selected time periods) derived a more efficient model which Patterson and Huber[17] optimized using a branch and bound method in 1974; but again, the number of variables in the models proved a stumbling block to actual application.

Attempts at using Bellman's dynamic programming for scheduling of resources proved of little use because of its complexity and problem dependence; but Petrovic[18] demonstrated in 1968 the feasibility of the approach.

The Heuristic Methods

The heuristic methods are offered in almost every major commercial CPM computer packages for the scheduling of activity costs and resources. Since these are privately developed commercial applications, most of the detailed information about the methods are proprietary.

Two (2) basic heuristic approaches have been developed[7], the first of which is the "parallel" activity selection approach. Major vendors whose software packages use this approach are IBM's Project Management Systems IV[19] (PMS4), MacDonald Douglas's Management Scheduling and Control Systems[20] (MSCS) and Project Software and Development's Project/2[21]. The second approach is the "serial" activity selection approach which was first used by Sun Information Services for PREMIS[22].

Both approaches schedule activities on heuristic rules and do not guarantee optimal solutions. In many cases the solutions are far from optimal which has discouraged even further use of the algorithms.

The Simulation Methods

The last direction of research is the simulation methods. From the foundation of the probabilistic PERT, Pritsker developed Graphic Evaluation and Review Technique[23] (GERT) based on the signal flow graph and its simulation extensions of Graphic Evaluation and Review Technique Simulation (GERTS) and Simulation Language for Alternative Modeling[24] (SLAM). Although these methods have extensive applications in manufacturing, they have not been specifically adopted to construction scheduling.

Objective of Research

Other than the heuristic methods, little of the research of the past twenty (20) years has been applied directly to construction scheduling in the form of commercially available computerized methods. Considering that construction scheduling is closely tied to the computer technology and that CPM is specified extensively in contract documents, the objective of this research is to draw on the current analytical methods[25] and computer technology to:

(1) Increase the flexibility of CPM by eliminating the requirement of fixed or predetermined activity durations.

(2) Incorporate costs and crew sizes (an integer convex function used extensively in construction estimating) into a CPM related

networking model so that CPM can be more compatible with the current estimating and cost control procedures used by construction contractors.

(3) Adapt the optimizing algorithms to the solution of the CPM related problem as defined above.

This appears to be a reasonable objective, considering the developments in convex and integer mathematical programming and the rapid growth in both the availability and power of computer equipment.

Research Methods

The methodology of the dissertation is to derive and code into the IBM BASIC computer language selected "hybrid" convex programming algorithms and to solve on a limited scale with a Panasonic Sr. Partner[26] micro-computer (8088 processor with no 8087 co-processor with clock time of 4.77 MHz) several CPM related models applicable to construction scheduling. All computer programs will be limited to small 64k bytes BASIC routines to encourage copying and expansion by others and will be compiled using the IBM PC BASIC compiler routine[27] for greater machine transportability.

Form of Results

The dissertation will consists of a set of convex programming algorithms adapted to the solution of construction scheduling problems with computer benchmarks indicating their applicability to construction scheduling problems.

The programs will be written in a modular fashion in which many subroutines will be shared with the intention of increasing the flexibilty of the code. To encourage further research based on these programs, the BASIC code subroutines will be saved on a 5 $\frac{1}{4}$" floppy disk disk under file names listed in the text.

SECTION I. SCHEDULING PROBLEMS WITH REAL NUMBER SOLUTIONS

Most scheduling problems eventually reduce to finding a minimum balance between estimated fixed and variable costs. Beyond the fixed costs of materials, subcontracts, and direct labor which are determined by the contract specification more than the schedule, fixed overhead costs are related to project duration and can be estimated as a linear function of the duration of the tasks, or activities, which must be performed in a sequence determined by the schedule. Variable costs of direct labor supervision, can also be approximated by a linear function of the scheduled activity durations, but are more accurately estimated as a parabolic function of direct labor man counts where man counts are a hyperbolic function of activity durations.

In this section, models will be formulated for project duration as determined by a given schedule; for project cost as determined by linear function of activity duration; for activity man counts as determined by a hyperbolic function of activity durations; and for project supervision cost as determined by a parabolic function of activity man counts. These models will then be minimized by the critical path method and variations of the simplex method.

Each method will be accompanied by a BASIC program code listing with which an example problem will be solved to demonstrate the solution of the problems and the efficiency of the algorithm. For simplicity, no consideration will be given to the cases were integer solutions are required, such as for man counts and schedule dates.

MINIMUM PROJECT DURATION PROBLEM

Before costs can even be considered, a project must be broken down into a sequence of tasks which are small enough in scope so that the individual tasks can be understood in detail. These tasks or activities can then be sequenced in a logical order to form a schedule.

A project schedule is needed to determine if a sufficient amount of time has been alloted to complete the project. Although most schedules are "thought-out", this mental process does not have the rigor or the capacity to handle large projects. To find a minimum project duration, when the time alloted for the work becomes critically short, requires a mathematical model.

Theory of Critical Path Method

A method first developed by Kelley and Walker can determine the minimum amount of time required to complete a project schedule. This method is called the critical path method[28] and is primarily a modeling technique, an accounting system for time, which determines those tasks in a work schedule whose timely completion are critical to the completion of a project schedule.

CPM Time Scale

The first component of the CPM model is the time scale as shown in figure 1. The scale is a set of "points" in time, usually represented by sequential integers, separated by equal "increments" of scheduled time or worked time which are not necessarily equal increments of elapsed time.

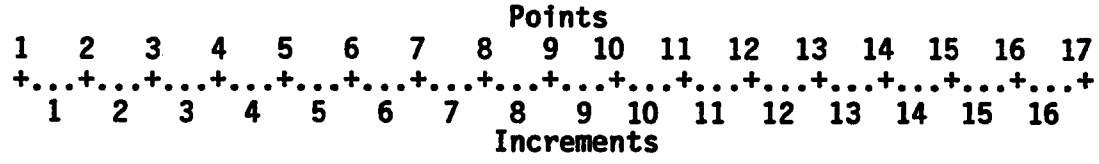


Figure 1. Critical path method project time scale

By convention, increment one (1) is started at point one (1) and finished at point two (2). Increment two (2) at point two (2), etc.

Work Activities and Durations

The second component of the CPM model is the work "activity". Most projects, too complex to analyze as a whole, are divided into smaller units of work or activities which are of a repetitive nature and easily studied in detail. These activities are usually the work done by a single crew. They have an easily identifiable "start" and "finish" point, and they have a "duration" (dur_{ij}) of a fixed number of CPM work increments required for their completion.

Events

To associate an activity with the CPM time scale, each activity is defined with two (2) "events". Each activity must start after an event and finish before an event. Events must occur only at one of the points on the CPM time scale and must be separated by at least the duration of the activity which they define.

Activities on Arrows

In the CPM model, an activity is graphically represented by two (2) nodes and an arrow shown in figure 2. The nodes represent two (2) events, often labeled i and j; and the arrow an activity. In this

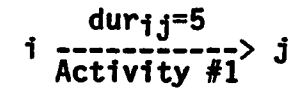


Figure 2. Critical path method arrow diagram work activity

example, activity #1 is defined in the CPM model as ij with a duration of five (5) work increments or $dur_{ij}=5$.

CPM Logic Diagram -- Arrow Diagramming Technique

Activities, drawn as arrows, can be combined into a CPM logic diagram or "arrow" diagram as shown in figure 3, example 1. The

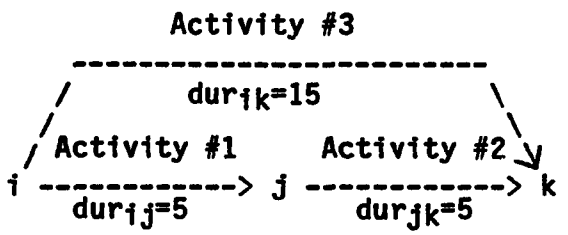


Figure 3. Critical path method example 1

sequential requirements of a schedule can be modeled by sequencing the arrows in a directed network. In a network, the nodes now represent the event of completion of all the activities entering the node.

Constructing the CPM network is an art in itself and will not be covered in detail here, but several conventions are commonly followed for the arrow diagram. Activities are linked together at their nodes. The network must have one (1) start node and one (1) finish node. For activity identification and processing purposes, the nodes are labeled with a set of integer numbers such that each activity can be identified by an ascending ordered pair of labeled nodes of which there are no

duplicate pairs. If any two (2) activities have a duplicate pair of labeled nodes, then "dummy" activities with zero (0) duration are added to the network to eliminate duplicate pairs.

(Although duplicate pairs are allowed in the following BASIC program, the use of the arrow diagramming conventions will prevent the logical contradictions of doubling back or "looping" in a CPM network.)

CPM Node Parameters

If activity ij is placed on the time scale in figure 4 at the

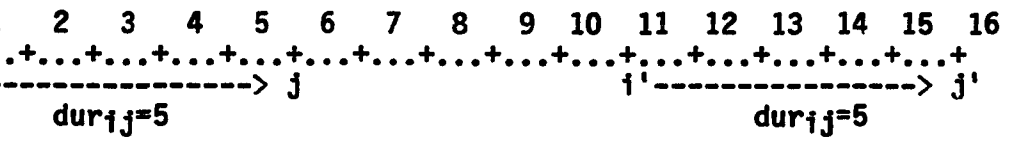


Figure 4. Critical path method time scale with activities

earliest possible position, then event i is at point one (1) and event j is at six (6). Assuming a maximum scale point at sixteen (16) and placing activity ij at its latest possible position, event i is at point twelve (12) and event j is at sixteen (16).

In algebraic notation, the above values can be expressed as:

$$i=1$$
$$j=i+dur_{ij}=1+5=6$$

$$j'=16$$
$$i'=j'-dur_{ij}=16-5=11$$

In CPM, these calculations define the node parameters defined by:

Earliest start of node i (ES_i) = 1
Latest start of node i (LS_i) = 11
Earliest start of node j (ES_j) = 6
Latest start of node j (LS_j) = 16

The range of points on the CPM time scale which the nodes i and j can assume without violating any activity's duration are defined by the CPM parameters:

$$\text{Node float node i } (NF_i) = LS_i - ES_i = 10$$
$$\text{Node float node j } (NF_j) = LS_j - ES_j = 10$$

CPM Forward Pass and Backward Pass

If the activities of example 1 are now positioned on the CPM time scale in the earliest possible position as in figure 5, then i is at one

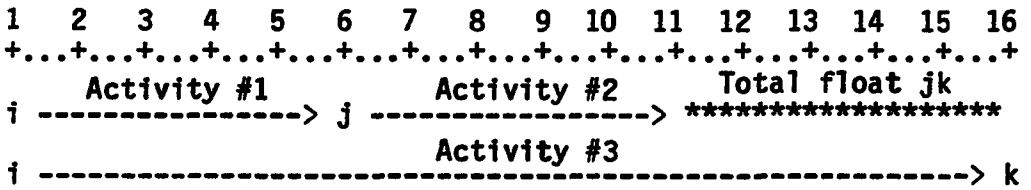


Figure 5. Critical path method example 1 early start time scaled network

(1) j is at six (6), and k at sixteen (16). If fifteen (15) work increments is the time allowed for the example network, then the sixteenth (16) point is the end of the CPM time scale. Event sixteen (16) is now used as the latest possible finish point, so the network is placed in its late finish position as in figure 6 where i is at one (1), j is at eleven (11), and k at sixteen (16).

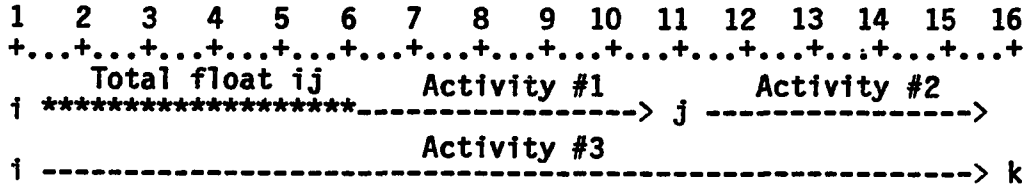


Figure 6. Critical path method example 1 late start time scaled network

In general, these calculations can be defined by a "forward pass":

ES_j=max (1, ES_i+dur_{ij}) for all activities entering node j

and a "backward pass":

LS_j=min (max point, LS_k-dur_{jk}) for all activities leaving node j

Calculating the CPM parameters for example 1 results in the following for the forward pass:

ES_i=maximum (1)=1
ES_j=maximum (1,6)=6
ES_k=maximum (1,11,16)=16

and for the backward pass:

LS_k=minimum (16)=16
LS_j=minimum (16,11)=11
LS_i=minimum (16,6,1)=1

and for the node calculations:

NF_i=LS_i-ES_i=1-1=0
NF_j=LS_j-ES_j=11-6=5
NF_k=LS_k-LS_k=16-16=0

CPM Activity Parameters

Most people who schedule are more concerned with activities than with nodes. In order to translate node parameters into activity parameters, CPM defines the following activity parameters:

Earliest start of activity ij (ES_{ij}) =ES_i
Latest start of activity ij (LS_{ij}) =LS_j-dur_{ij}
Earliest finish of activity ij (EF_{ij}) =ES_i+dur_{ij}
Latest finish of activity ij (LF_{ij}) =LS_j
Total float activity ij (TF_{ij}) =LS_j-ES_i-dur_{ij}

"Earliest start", "latest start", "earliest finish", and "latest finish" times are self-explanatory from their descriptions. "Total float", in terms of the actual execution of the schedule, gives the work

increments that activity's "actual" start can be delayed beyond its CPM early start without delaying any other activity's actual start beyond its CPM late start. Since one late start is the completion point of the project, if an activity's total float is zero (0), then the activity's early start is "critical" to the completion of the project. Together, all critical activities define a "critical path" for which all actual starts must equal the early start for timely completion of the project.

Again, using example 1 and the formula for total floats:

$TF_{ij}=11-1-5=5$
$TF_{jk}=16-6-5=5$
$TF_{ik}=16-1-15=0$ critical

CPM has further parameters of float[29] as defined by:

Free float activity ij $(FF_{ij}) = ES_j - ES_i - dur_{ij}$
Safety float activity ij $(SF_{ij}) = LS_j - LS_i - dur_{ij}$
Independent float activity ij $(IF_{ij}) = ES_j - LS_i - dur_{ij}$

Figure 7 is a graphic summary of the parameters of the above parameters.

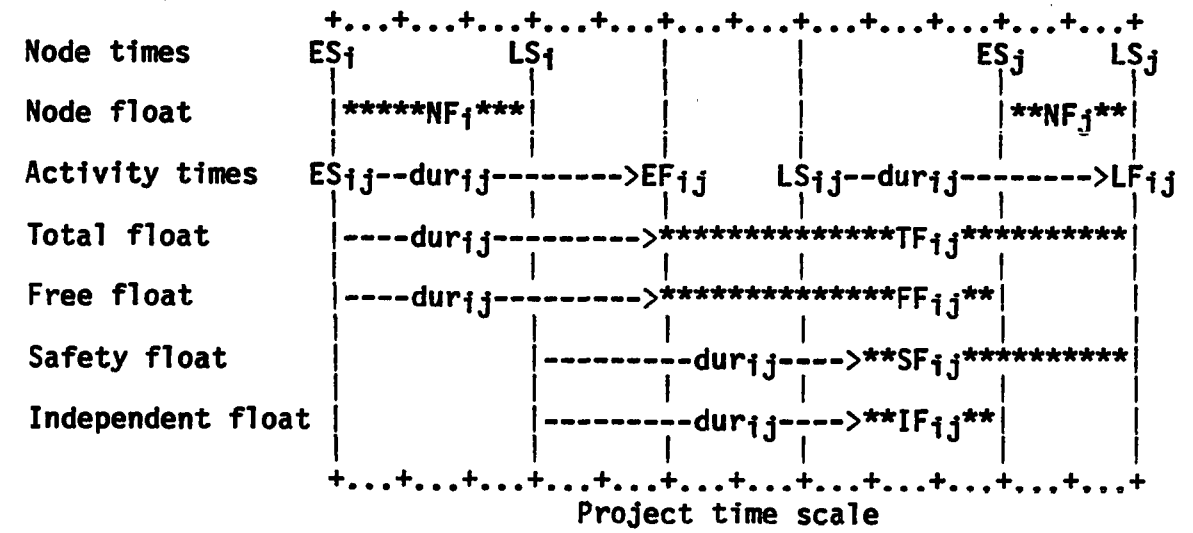


Figure 7. Critical path method parameters on time scale

"Free float" is the work increments that the actual start of an activity can be delayed beyond its CPM early start without delaying the actual start of any other activity beyond its CPM early start. "Safety float" is the time increments that the actual start of an activity can be delayed beyond its CPM late start without delaying the actual start any other activity beyond its CPM late start, and "independent float" is the time increments that the actual start of an activity can be delayed beyond its CPM late start without delaying the actual start of any other activity beyond its CPM early start.

Negative Float

In actual applications, the time scale point at which the project must be completed is a requirement of the contract. To introduce this completion time into the CPM model, the backward pass is started at the event corresponding to the completion time on the CPM time scale.

Using example 1 and a completion at point fifteen (15), the total

ES_i =maximum (1)=1
ES_j =maximum (1,6)=6
ES_k =maximum (1,11,16)=16
LS_k =minimum (15)=15
LS_j =minimum (15,10)=10
LS_i =minimum (15,5,0)=0
NF_i =LS_i-ES_i=0-1=-1
NF_j =LS_j-ES_j=10-6=5
NF_k =LS_k-LS_k=15-16=-1
TF_{ij}=LS_j-ES_i-Dur_{ij}=10-1-5=4
TF_{jk}=LS_k-ES_j-Dur_{jk}=15-6-5=4
TF_{ik}=LS_k-ES_i-Dur_{ik}=15-1-15=-1

float of activity ik is a negative (-1) value. This "negative float" indicates that either the duration of activity ij is one (1) time unit too long or that the logic of the schedule is not adequate for the

project requirements. Through a process of trial and error, the negative float can be eliminated by shortening activity durations or changing the arrow diagram logic.

Critical Path Method BASIC Code

The following code is a modification of a BASIC program first published by Gary Whitehouse of the University of Central Florida. Although it does not contain all the options currently used in commercial CPM programs, it does provide a simple algorithm from which to expanded to larger more complex programs. For other enhancements, such as the precedence diagramming modification to CPM, see Appendix A.

All programs listed in the text consist of a main calling routine and a series of input, output, and processing subroutines. These BASIC routines are listed in the text as well as on a computer disk compatible with IBM micro-computers.

Critical Path Method Main Routine -- File MAIN-CPM

The CPM main program (MAIN-CPM) dimensions the two (2) data

```
                CRITICAL PATH METHOD

NUMBER OF ACTIVITIES                10

                M-RETURN TO MENU

              A-ACTIVITIES
              U-EXECUTE ALGORITHM
              R-REPORT
              S-SAVE  F-FETCH

        OPTION ?
```

Figure 8. Critical path method main menu screen

arrays; writes the options menu as shown in figure 8; calls the utility subroutines UTIL-OPT, UTIL-ERS, UTIL-CON, UTIL-CHX; calls the input and output routines INPT-ACT and REPT-CPM; times and calls the processing algorithm ALGR-CPM; and saves and fetches the input data to and form disk file "DATA".

```
1 REM                    * CRITICAL PATH METHOD *
2 REM-------------------------MAIN-CPM---------------------------------

3 REM      AR  - NUMBER OF NETWORK ARCS OR ACTIVITIES
4 REM      BI# - MACHINE INFINITE
5 REM      DR  - PROJECT DURATION
6 REM      SM# - MACHINE ZERO
7 REM      A(AR,5) - (I,J,MIN DUR,MAX DUR,INC CST)
8 REM      B(AR,8) - WORK FILE (NO,I,J,DR,ES,LS,TF,FF)
9 REM-------------------------------------------------------------------
```

Sets BI# to a number considered infinite and SM# to a number considered zero (0). Reads from the keyboard the number of activities in the arrow diagram, and dimensions the activity array A(AR,5) and the work space array B(AR,8).

The array A(AR,8) contains the activity i node, j node, minimum duration, normal duration, and incremental cost. The array B(AR,8) is a temporary work array with activity numbers which are the row numbers from array A(AR,8), i node, j node, minimum duration, early start, late start, total float, and free float.

```
10 AR=0
11 BI#=1E+10
12 SM#=1E-10
13 CLS
14 LOCATE 1,7:PRINT "CRITICAL PATH METHOD"
15 LOCATE 3,1:PRINT "NUMBER OF ACTIVITIES":LOCATE 3,31:INPUT "",L$
16 GOSUB 1870:REM UTIL-CHX
17 IF Z#<>BI# THEN AR=Z#
18 LOCATE 3,30:PRINT AR,"    "
19 DIM A(AR,5)
20 DIM B(AR,8)
```

Prints the "option" menu to the screen, calls the option line routine UTIL-OPT, and pauses the program execution for entry of "A","U","R","S", or "F" for the option variable L$.

```
21 LOCATE 8,10:PRINT     "M-RETURN TO MENU"
22 LOCATE 10,5:PRINT   "A-ACTIVITIES"
23 LOCATE 11,5:PRINT   "U-EXECUTE ALGORITHM"
24 LOCATE 12,5:PRINT   "R-REPORT"
25 LOCATE 13, :PRINT   "S-SAVE F-FETCH"
26 GOSUB 1800. EM UTIL-OPT
27 LOCATE 21,8:INPUT "",L$
28 CLS
```

Calls either the input subroutine INPT-ACT, the processing subroutine ALGR-CPM, or the report subroutine REPT-CPM based on the option variable L$.

```
29 IF L$<>"A" THEN 32
30 GOSUB 1100:REM INPT-ACT
31 GOTO 28
32 IF L$<>"U" THEN 37
33 TM=3600*VAL(MID$(TIME$,1,2))+60*VAL(MID$(TIME$,4,2))+VAL(MID$(TIME$,7
   ,2))
34 GOSUB 3300:REM ALGR-CPM
35 TM=3600*VAL(MID$(TIME$,1,2))+60*VAL(MID$(TIME$,4,2))+VAL(MID$(TIME$,7
   ,2))-TM
36 GOTO 21
37 IF L$<>"R" THEN 40
38 GOSUB 2000:REM REPT-CPM
39 GOTO 28
```

Saves the content of A(AR,8) to disk file "DATA" as an ASCI file.

```
40 IF L$<>"S" THEN 50
41 OPEN "O",#1,"DATA"
42 PRINT #1,STR$(AR)
43 PRINT #1,""
44 FOR I=1 TO AR
45 FOR J=1 TO 5
46 PRINT #1,STR$(A(I,J))
47 NEXT J
48 NEXT I
49 CLOSE #1
```

Loads into array A(AR,8) the disk file "DATA".

```
50 IF L$<>"F" THEN 63
51 OPEN "I",#1,"DATA"
52 INPUT #1,X$
53 AR=VAL(X$)
54 INPUT #1,X$
55 FOR I=1 TO AR
```

```
56 FOR J=1 TO 5
57 INPUT #1,X$
58 A(I,J)=VAL(X$)
59 NEXT J
60 NEXT I
61 CLOSE #1
62 GOTO 21
63 IF L$="N" THEN RUN
64 GOTO 21
```

Utility Subroutines -- Files UTIL-OPT,UTIL-ERS,UTIL-CON, and UTIL-CHX

The following routines are used as utilities for printing prompt lines, causing program pauses at report page breaks, and checking data entry for data type errors. The option line subroutine (UTIL-OPT) writes the OPTION ? line at the bottom of the screen.

```
1800 REM                  * OPTION LINE SUBROUTINE *
1801 REM-------------------------UTIL-OPT-------------------------------

1802 LOCATE 21,1:PRINT "OPTION ?"
1803 RETURN
```

The erase option subroutine (UTIL-ERS) erases the OPTION ? line.

```
1850 REM               * ERASE OPTION LINE SUBROUTINE *
1851 REM-------------------------UTIL-ERS-------------------------------

1852 LOCATE 21,1:PRINT "          "
1853 RETURN
```

The continue line subroutine (UTIL-CON) stops reports at the end of a screen with the prompt line PRESS RETURN TO CONTINUE before continuing to the next screen or taking another option.

```
1860 REM                 * CONTINUE LINE SUBROUTINE *
1861 REM-------------------------UTIL-CON-------------------------------

1862 LOCATE 21,7:PRINT "(PRESS RETURN TO CONTINUE)"
1863 LOCATE 21,8:INPUT "",L$
1864 CLS
1865 G=1
1866 RETURN
```

The data check routine (UTIL-CHX) checks numeric input entered as the character variable L$ for invalid characters and either sets Z# to the number entered or, if an error is found, to infinite BI#.

```
1870 REM                    * DATA CHECK SUBROUTINE *
1871 REM--------------------------UTIL-CHX-----------------------------

1872 IF L$="" THEN 1878
1873 Z#=LEN(L$)
1874 FOR Y=1 TO Z#
1875 P$=MID$(L$,Y,1)
1876 IF P$="E" THEN 1880
1877 IF P$="0" OR P$="1" OR P$="2" OR P$="3" OR P$="4" OR P$="5" OR
     P$="6 " OR P$="7" OR P$="8" OR P$="9" OR ((P$="-" OR P$=".") AND
     Z#>1) THEN 1880
1878 Z#=BI#
1879 RETURN
1880 NEXT Y
1881 Z#=VAL(L$)
1882 RETURN
```

Activity Input Subroutine -- File INPT-ACT

The activity input subroutine (INPT-ACT) is the interactive screen input of data into the activity array A(AR,8). To enter the screen from the main menu, type "A" as the OPTION ?. The screen shown in figure 9

```
                    ACTIVITIES
NO.     I     J     MIN DUR   NORM DUR   INC COST
 1      0     3     1         10         5
 2      2     4     1         20         10
 3      0     1     1         30         15
 4      1     2     1         40         20
 5      2     3     1         50         25
 6      3     4     1         60         30
 7      1     3     1         70         35
 8      0     2     1         80         40
 9      1     4     1         90         45
 10     0     4     1         100        50
 _

 OPTION ?
```

Figure 9. Critical path method activity input screen

will appear which consists of a data line for each activity. The cursor "_" will locate under the activity number column NO. and will move across the columns as the ENTER key is pressed to enter the data typed on the screen under the column heading. When a blank is entered, the current values for the activity numbered are retained in the file.

If an error is made in entry, the current activity line must be completed and a new entry made using the same activity number as in the first column of the mistyped activity line.

If in the NO. column an activity number is not entered, but an option letter is entered, then the option is taken.

```
1100 REM               * ACTIVITY INPUT SUBROUTINE *
1101 REM-------------------------INPT-ACT-------------------------------

1102 H=0
1103 G=2
1104 LOCATE 1,15:PRINT "ACTIVITIES":LOCATE 2,1:PRINT "NO.   I
     J     MIN DUR  NORM DUR  INC COST"
1105 G=G+1
1106 H=H+1
1107 GOSUB 1800:REM UTIL-OPT
```

Enters activity number.

```
1108 LOCATE G,2:INPUT "",L$
1109 GOSUB 1870:REM UTIL-CHX
1110 IF L$<>"" AND Z#=BI# THEN RETURN
1111 IF Z#<>BI# THEN H=Z#
1112 IF H>AR OR H<=0 THEN 1107
1113 GOSUB 1850:REM UTIL-ERS
```

Prints activity number and enters new i node number.

```
1114 LOCATE G,1:PRINT H,"  ":LOCATE G,6:PRINT A(H,1):LOCATE G,7:INPUT ""
     ,L$
1115 GOSUB 1870:REM UTIL-CHX
1116 IF Z#<>BI# AND Z#>=0 THEN A(H,1)=Z#
```

Prints i node and enters new j node.

If the j node is less than or equal to the i node then the new j node number is rejected. This is a protection against entering a loop in the CPM network logic.

```
1117 LOCATE G,6:PRINT A(H,1),"    ":LOCATE G,11:PRINT A(H,2):LOCATE G,12
     :INPUT "",L$
1118 GOSUB 1870:REM UTIL-CHX
1119 IF Z#<=A(H,1) THEN 1117
1120 IF Z#<>BI# THEN A(H,2)=Z#
```

Prints j node and enters new minimum duration.

```
1121 LOCATE G,11:PRINT A(H,2),"    ":LOCATE G,16:PRINT A(H,3):LOCATE
     G,17:INPUT "",L$
1122 GOSUB 1870:REM UTIL-CHX
1123 IF Z#<>BI# THEN A(H,3)=Z#
```

Prints minimum duration and enters new normal duration.

```
1124 LOCATE G,16:PRINT A(H,3),"    ":LOCATE G,25:PRINT A(H,4):LOCATE
     G,26:INPUT "",L$
1125 GOSUB 1870:REM UTIL-CHX
1126 IF Z#<>BI# THEN A(H,4)=Z#
```

Prints normal duration and enters new incremental cost.

```
1127 LOCATE G,25:PRINT A(H,4),"    ":LOCATE G,38:PRINT A(H,5):LOCATE
     G,39:INPUT "",L$
1128 GOSUB 1870:REM UTIL-CHX
1129 IF Z#<>BI# THEN A(H,5)=Z#
1130 LOCATE G,38:PRINT A(H,5),"    "
```

Checks for end the of screen and starts a new screen if the lines used are greater than eighteen (18).

```
1131 IF G<18 THEN 1105
1132 GOSUB 1860:REM UTIL-CON
1133 GOTO 1106
```

CPM Report Subroutine -- File REPT-CPM

The CPM report (REPT-CPM) lists the processing time of the ALGR-CPM subroutine and, in early start sequence, the activities and their CPM parameters of activity number, i node, j node, early start, late start, total float, and free float. At the end of each screen shown in figure

10, the option line is printed and the listing pauses.

```
                CRITICAL PATH LISTING
     SOLUTION FOUND IN                    4 SEC
     MINIMUM NETWORK DURATION             190
 NO.     I       J       ES      LS      TF      FF
  3      0       1       1       11      10      0
  8      0       2       1       1       0       0
  1      0       3       1       121     120     120
 10      0       4       1       91      90      90
  4      1       2       31      41      10      10
  7      1       3       31      61      30      30
  9      1       4       31      101     70      70
  5      2       3       81      81      0       0
  2      2       4       81      171     90      90
  6      3       4       131     131     0       0

 OPTION ?
```

Figure 10. Critical path method critical path listing screen

```
2000 REM                  * CPM REPORT SUBROUTINE *
2001 REM-----------------------REPT-CPM-------------------------------

2002 LOCATE 1,8:PRINT " CRITICAL PATH LISTING"
2003 LOCATE 2,4:PRINT "SOLUTION FOUND IN ":LOCATE 2,30:PRINT TM;"SEC"
2004 LOCATE 3,4:PRINT "MINIMUM NETWORK DURATION ":LOCATE 3,30:PRINT DR
2005 LOCATE 4,1:PRINT " NO.   I     J     ES    LS    TF    F     F"
2006 G=5
2007 FOR I=1 TO AR
2008 LOCATE G,1:PRINT B(I,1):LOCATE G,7:PRINT B(I,2):LOCATE G,13:PRINT
     B(I,3):LOCATE G,19:PRINT B(I,5):LOCATE G,25:PRINT B(I,6):LOCATE
     BG,31:PRINT B(I,7):LOCATE G,37:PRINT B(I,8)
2009 G=G+1
2010 IF G<20 THEN 2016
2011 GOSUB 1860:REM UTIL-CON
2012 LOCATE 21,8:INPUT "",L$
2013 GOSUB 1870:REM UTIL-CHX
2014 IF L$<>"" AND Z#=BI# THEN RETURN
2015 GOSUB 1864:REM UTIL-CON
2016 NEXT I
2017 GOSUB 1800:REM UTIL-OPT
2018 LOCATE 21,8:INPUT "",L$
2019 RETURN
```

Critcal Path Algorithm Subroutine -- File ALGR-CPM

The CPM algorithm subroutine (ALGR-CPM) first copies the activity data from the activity array A(AR,5) to the working array B(AR,8). The working file is then sorted in i node order; and within the i node order, into j node order. With the activities sorted as above, the early finish EF_{ij} calculations of forward pass are executed to determine the earliest project completion time DR. Using the earliest project completion time DR, the late finish LF_{ij} calculations or backward pass are executed. These two calculations yield EF_{ij} and LF_{ij} for each activity from which the total float TF_{ij} and free float FF_{ij} are calculated.

```
3300 REM              * CRITICAL PATH ALGORITHM SUBROUTINE *
3301 REM----------------------------ALGR-CPM--------------------------
```

Copies the array A(AR,5) into the first five (5) positions of the array B(AR,8) and set all early finish times to one (1), late finish to zero (0), etc.

```
3302 FOR I=1 TO AR
3303 B(I,1)=I
3304 B(I,2)=A(I,1)
3305 B(I,3)=A(I,2)
3306 B(I,4)=A(I,4)
3307 B(I,5)=1
3308 FOR J=6 TO 8
3309 B(I,J)=0
3310 NEXT J
3311 NEXT I
```

Sorts the array B(AR,8) on positions two (2), then three (3), using a Schell sort[30].

```
3312 IF AR=1 THEN 3335
3313 Y=AR
3314 Y=INT(Y/2)
3315 FOR Z=1 TO AR-Y
3316 A=B(Z+Y,1)
```

```
3317 B=B(Z+Y,2)
3318 C=B(Z+Y,3)
3319 D=B(Z+Y,4)
3320 FOR W=Z TO 1 STEP -Y
3321 IF B(W,2)<B THEN 3329
3322 IF B(W,2)>B THEN 3324
3323 IF B(W,3)<C THEN 3329
3324 B(W+Y,1)=B(W,1)
3325 B(W+Y,2)=B(W,2)
3326 B(W+Y,3)=B(W,3)
3327 B(W+Y,4)=B(W,4)
3328 NEXT W
3329 B(W+Y,1)=A
3330 B(W+Y,2)=B
3331 B(W+Y,3)=C
3332 B(W+Y,4)=D
3333 NEXT Z
3334 IF Y>1 THEN 3314
```

Executes the CPM forward pass to find early finish times.

This is done by selecting the first activity of the i node sequence, adding to its early start its duration, searching the remaining activities of the J sequence (which because of the sort have larger node numbers) for all activities with the same start node as the current activities completion node, and assigning these activities new early start times equal to the current activity's early finish if the current activity's finish time is the latest. After all connected activities are found, select the next activity in the i node sequence to continue the process until the last activity is reached.

```
3335 FOR I=1 TO AR
3336 B(I,5)=B(I,5)+B(I,4)
3337 IF I=AR THEN 3344
3338 FOR J=I+1 TO AR
3339 IF B(I,3)<>B(J,2) THEN 3342
3340 IF B(J,5)>B(I,5) THEN 3342
3341 B(J,5)=B(I,5)
3342 NEXT J
3343 NEXT I
```

Selects the latest early finish time as the project duration.

```
3344 DR=0
3345 FOR I=1 TO AR
3346 IF DR>B(I,5) THEN 3348
3347 DR=B(I,5)
3348 NEXT I
```

Sets all activity late finish times to project duration.

```
3349 FOR I=1 TO AR
3350 B(I,6)=DR
3351 NEXT I
```

Executes the CPM backward pass.

This is the same process as the forward pass above except that the file is sequenced backwards and the activity duration is subtracted from the selected activity rather than added as in the forward pass.

```
3352 FOR I=1 TO AR
3353 K=AR-I+1
3354 FOR J=1 TO AR-I
3355 IF B(K,2)<>B(J,3) THEN 3358
3356 IF B(K,6)-B(K,4)>B(J,6) THEN 3358
3357 B(J,6)=B(K,6)-B(K,4)
3358 NEXT J
3359 NEXT I
```

Calculates the total float times TF_{ij} using the formula $LF_{ij}-EF_{ij}$ and the free float FF_{ij} as follows. For each current activity of the i node sequence finds all activities of the j node sequence which directly follow, of these finds the minimum early start time and subtract it from the late finish of the current activity.

```
3360 FOR I=1 TO AR
3361 B(I,7)=B(I,6)-B(I,5)
3362 IF B(I,3)=B(AR,3) THEN 3364
3363 GOTO 3365
3364 A=B(AR,6)
3365 FOR J=I TO AR
3366 IF B(I,3)=B(J,2) THEN 3368
3367 GOTO 3369
3368 A=B(J,5)-B(J,4)
3369 NEXT J
3370 B(I,8)=A-B(I,5)
3371 NEXT I
```

Converts early finish EF_{ij} and late finish LF_{ij} times to early start ES_{ij} times and early finish EF_{ij} times.

```
3372 DR=DR-1
3373 FOR I=1 TO AR
3374 B(I,5)=B(I,5)-B(I,4)
3375 B(I,6)=B(I,6)-B(I,4)
3376 NEXT I
```

3377 RETURN

Program Table of Contents

The following table 1 can be used to reconstruct the above

Table 1. Critical path method BASIC program table of contents

File	Program lines	Page	Routines
MAIN-CPM	0001-0064	20	Critical path method
INPT-ACT	1100-1133	24	Activity input subroutine
UTIL-OPT	1800-1803	22	Option line subroutine
UTIL-ERS	1850-1853	22	Erase option subroutine
UTIL-CON	1860-1866	22	Continue line subroutine
UTIL-CHX	1870-1882	23	Data check subroutine
REPT-CPM	2000-2019	26	CPM report subroutine
ALGR-CPM	3300-3377	27	Critical path algorithm subroutine

computer code from the computer disk files. Since BASIC code is dependent on program line numbers for subroutine branching, the line number must be maintained as listed in above

Critical Path Method:
Solutions to Example 2 Minimum Project Duration Model

A small ten (10) activity project network schedule is used as an example throughout the dissertation for the purpose of establishing benchmarks for the different methods except in cases where the resulting models or problems would exceed the capacity of a 64k CPU. This example also serves as the source of the input and output data for the figures displaying computer screens.

Figure 11 is the CPM arrow diagram used for example 2 and consists

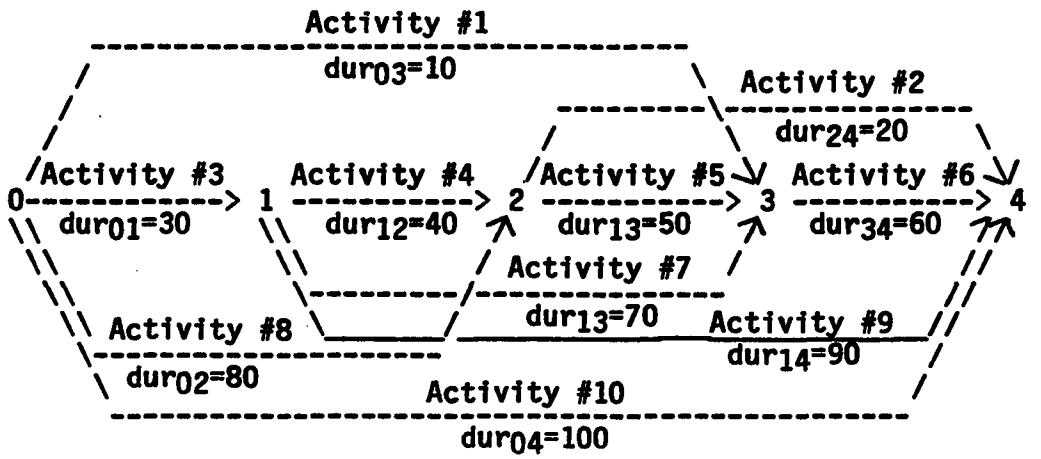


Figure 11. Minimum project duration model example 2 arrow diagram

of ten (10) activities and five (5) nodes. When this network is entered into the the CPM program, it appears as shown in figure 12. For each

ACTIVITIES

NO.	I	J	MIN DUR	NORM DUR	INC COST
1	0	3	0	10	0
2	2	4	0	20	0
3	0	1	0	30	0
4	1	2	0	40	0
5	2	3	0	50	0
6	3	4	0	60	0
7	1	3	0	70	0
8	0	2	0	80	0
9	1	4	0	90	0
10	0	4	0	100	0

OPTION ?

Figure 12. Minimum project duration model example 2 activity input screen

activity number, NO., there is an i to j node pair in ascending order and a "normal" activity duration entered under the column heading NORM

DUR. The input screen is also used in later programs in which the activity durations are variables, so the minimum durations column, MIN DUR, and incremental costs columns, INC COST, are left as zero (0) in this method.

After an execution time of four (4) seconds, the report shown in figure 13 was listed by the CPM BASIC program to the computer screen.

CRITICAL PATH LISTING
SOLUTION FOUND IN 4 SEC
MINIMUM NETWORK DURATION 190

NO.	I	J	ES	LS	TF	FF
3	0	1	1	11	10	0
8	0	2	1	1	0	0
1	0	3	1	121	120	120
10	0	4	1	91	90	90
4	1	2	31	41	10	10
7	1	3	31	61	30	30
9	1	4	31	101	70	70
5	2	3	81	81	0	0
2	2	4	81	171	90	90
6	3	4	131	131	0	0

OPTION ?

Figure 13. Minimum project duration model example 2 critical path listing screen

MINIMUM PROJECT COST PROBLEM WITH LINEAR COST FUNCTIONS

The critical path method combines work activities, each with a fixed duration and a schedule sequence, into a mathematical model described by a set of parameters. These parameters can then be evaluated relative to some objective. If the parameters fail to meet the objective, a new set of activity durations or schedule sequences can be modeled; and in an iterative manner, a better set of parameters can be found which are closer to meeting the objective.

Unfortunately, this does not quarantee an optimal set of parameter even with fixed activity durations. When variable activity durations and associated costs are introduced, finding an optimum network becomes too complex for a simple trial and error approach.

Theory of Dual Simplex Method

The application of the "simplex" methods to CPM schedules[31] was first proposed by Kelley. The simplex method had been used extensively to solve sets of linear equations, and it provided a means to find an optimal solution to the mathematical model of the CPM network.

If the network logic of example 1 is written as a set of simultaneous equations with all variables greater than or equal zero (0), then:

$$T_j - dur_{ij} - T_i >= 0$$
$$T_k - dur_{jk} - T_j >= 0$$
$$T_k - dur_{ik} - T_i >= 0$$

$$T_k <= Dur.$$

$$T_i, T_j, T_k, Dur. >= 0$$

where T_i are the node times not restricted to integer points, dur_{ij} are the activity durations, and Dur. is the maximum duration of the network.

Let R_{ij} be the reduction in the duration of activity ij so:

$$\text{Activity duration} = dur_{ij}-R_{ij} \quad \text{for all ij}$$

then:

$$T_j-(dur_{ij}-R_{ij})-T_i>=0$$
$$T_k-(dur_{jk}-R_{jk})-T_j>=0$$
$$T_k-(dur_{ik}-R_{ik})-T_i>=0$$

$$T_k<=Dur.$$

$$R_{ij}< dur_{ij}$$
$$R_{jk}< dur_{jk}$$
$$R_{ik}< dur_{ik}$$

$$T_i,T_j,T_k,R_{ij},R_{jk},R_{ik},Dur.>=0$$

Using the variable (or direct) cost versus fixed (or overhead) cost economic model[32], the total cost of the project can be written as:

$$\text{Total cost}=nc_{ij}+nc_{jk}+nc_{ik}+c_{ij}*R_{ij}+c_{jk}*R_{jk}+c_{ik}*R_{ik}+fix.*(T_k-T_i)$$

where nc_{ij} is the normal duration cost of the activity ij; c_{ij} is the incremental variable cost to reduce the normal duration of activity ij by R_{ij}; and fix. is the incremental fixed cost over the duration (T_k-T_i).

The CPM arrow diagram can now be written as a set of linear "constraints" as demonstrated above. If the completion time of the network is fixed, then the constraints can be satisfied by only a limited set of values or points called the "feasible region"; and all the point of the region can be evaluated with the total project cost equation, or "objective function", for the optimal solution point.

These constraints and objective function define a linear

programming (LP) problem called the "primal problem"[33]. (For every LP problem there is also another problem called the "dual problem" from which the method first was derived.) This problem can be solved with the "dual" simplex method[34] which minimize the objective function while generating a series of "primal infeasible" points satisfying an ever increasing number of unsatisfied constraints until a minimum optimal solution from the feasible region of the primal problem is found.

The Two Dimensional Case of the Dual Simplex Method

To visualize the method and to provide a simple geometric problem

minimize a*X+b*Y

subject to: X >=0 or lower bound of X
Y>=0 or lower bound of Y
c*X+d*Y>=e
g*X+h*Y>=f
a,b,c,d,e,f,g,h>=0

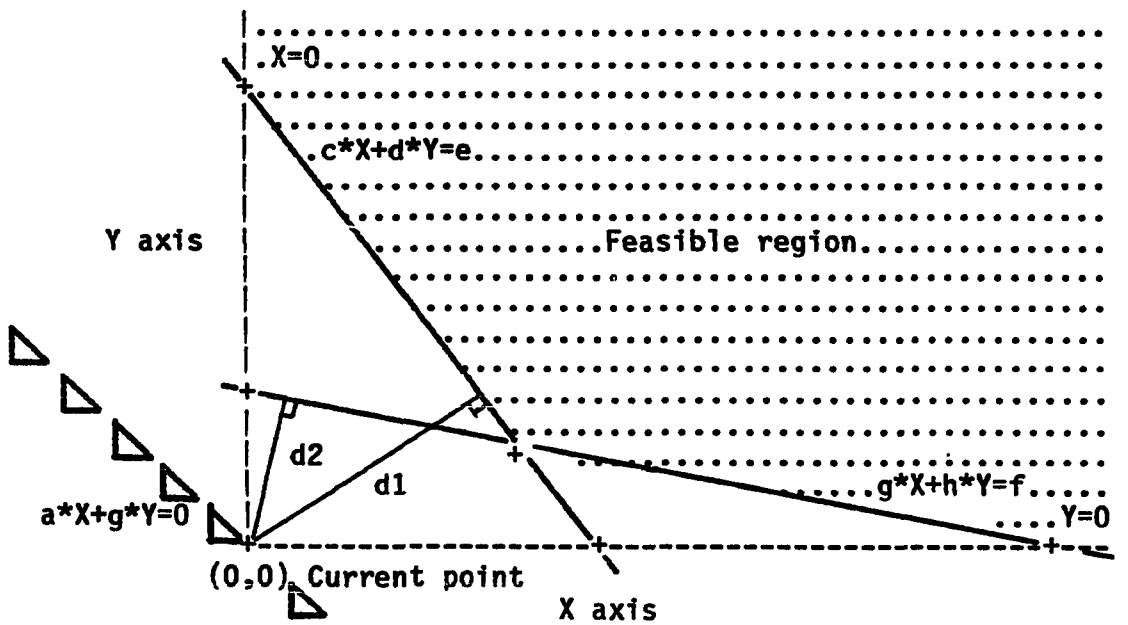


Figure 14. Dual simplex method example 3

to solve with the dual simplex algorithm, the two (2) dimensional example displayed in figure 14 will be used as a demonstration.

Example 3 is in a format for which there is a dual simplex method solution. All lower case letters represent positive coefficients. All constraints are in the greater than or equal form. The positive restrictions on the X and Y variables and the positive coefficients of the objective function (a condition called "dual feasible" since the constraints of the dual problem are satisfied) are a requirement of the dual simplex method.

Slack

If the objective function is added to the constraints and a set of "slack" values are chosen which will satisfy the constraints as equalities, then the problem can be written as:

$$\text{minimize} \quad z$$

$$\begin{aligned} \text{subject to:} \quad & a*X+b*Y-z = 0*1 \\ & X - x = 0*1 \\ & Y-y = 0*1 \\ & c*X+d*Y-s1 = e*1 \\ & g*X+h*Y-s2 = f*1 \end{aligned}$$

Starting Point

The above equations can be rewritten so that only the slacks are on the left hand side:

$$\text{minimize} \quad z$$

$$\begin{aligned} \text{subject to:} \quad & z = 0*1+a*X+b*Y && (1) \\ & x = 0*1 + X && (2) \\ & y = 0*1 + Y && (3) \\ & s1 = -e*1+c*X+d*Y && (4) \\ & s2 = -f*1+g*X+h*Y && (5) \end{aligned}$$

Since all the coefficients of the objective function are greater than or equal to zero (0), setting X and Y to the lower bound of zero (0) results in the minimum value of the objective function and the best possible starting value or "current point" from which to search for a minimum primal feasible point. (If any variable has a greater lower bound then the starting point can be at that lower bounds.)

Graphically, example 2 is as shown in figure 14 with the current point (0,0) at the origin of an X,Y coordinate system in which x=0 is the displacement of the current point on the X axis; y=0 is the displacement of the current point on the Y axis; z=0 is the displacement of the objective function from the origin; and s1=-e and s2=-f are the negative slack of the last two (2) unsatisfied or "violated" constraints.

Selection of a Violated Constraint

The current point (0,0) violates several of the constraints so it is "primal infeasible" and not a solution. Any "new point" which might be in the feasible region must first "satisfy" at least one (1) of the violated constraints without violating any of the satisfied constraints.

To find a new point, a constraint is selected to be satisfied. The simplest choice would be to choose the equation with the most negative slack. Unfortunately, this does not lead to a feasible solution in later methods, so the greatest distance along the normal line from the current point to the violated constraint is used as the selection criterion.

A line through the point (x,y) and normal to the line c*X+d*Y=e

can be written parametically in T as the line X=x+c*T and Y=y+d*T. The intersection of the parametric line and the constraint can be found by substitution in c*X+d*Y=e and solving for T.

$$c*(x+c*T)+d*(y+d*T)=e$$

$$T=\frac{e-c*x-d*y}{c\hat{}2+d\hat{}2}$$

The distance from (x,y) to c*X+d*Y=e can now be defined by:

$$d1\hat{}2=(x-(x+c*T))\hat{}2+(y-(y+d*T))\hat{}2$$

and:

$$d1\hat{}2=\frac{(e-c*x-d*y)\hat{}2}{c\hat{}2+d\hat{}2}$$

or:

$$d1\hat{}2=\frac{(-s1)\hat{}2}{c\hat{}2+d\hat{}2}$$

The distance from the current point to the constraint is then the slack "normalized" by the coefficients of the constraint.

In the example problem, equation (4) is geometrically the most distant so that:

$$d1\hat{}2=\frac{(-e)\hat{}2}{c\hat{}2+d\hat{}2}>=\frac{(-f)\hat{}2}{g\hat{}2+h\hat{}2}=d2\hat{}2$$

Selecting a New Point

Any new point must satisfy the selected constraint equation (4) so that s1>=0. Using this fact, algebraically, a new point can be found by rewriting the constraint equation in either the X variable and s1 or in the Y variable and s1 and setting s1 to zero (0).

$$s1=0 \quad X=0 \quad Y=\frac{e}{d}+\frac{s1}{d}-\frac{c}{d}*X=\frac{e}{d} \quad \text{point } (0,\frac{e}{d})$$

$$s1=0 \quad Y=0 \quad X=\frac{e}{c}-\frac{d}{c}*Y+\frac{s1}{c}=\frac{e}{c} \quad \text{point } (\frac{e}{c},0) \qquad (6)$$

Of the two possible points, $(0,\frac{e}{d}),(\frac{e}{c},0)$, the point to be selected

should yield the minimum value of the objective function. By substitution:

$$z=a*0+b*\frac{e}{d}=b*\frac{e}{d}$$

$$z=a*\frac{e}{c}+b*0=a*\frac{e}{c}$$

or if $a*\frac{e}{c}<=b*\frac{e}{d}$, or $\frac{a}{c}<=\frac{b}{d}$, choose (6) as the new point.

The geometry of the algebra can be represented by the intersection of the violated constraint with the constraints defining the current point or the "basis". As in the algebraic solution, the graphics in figure 15 show two (2) intersection points, both possibly in the feasible

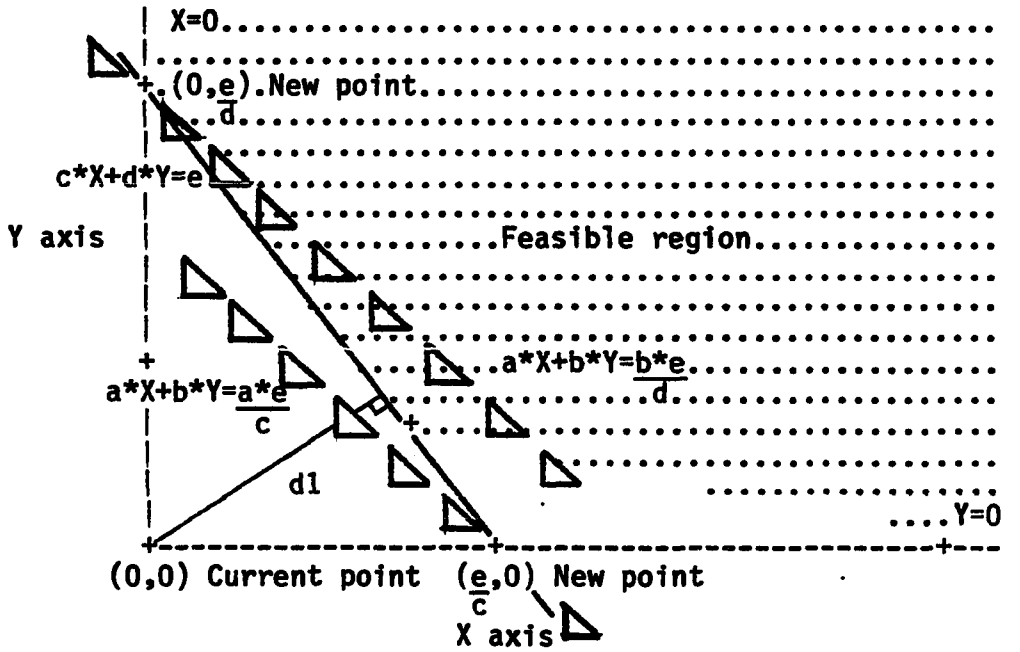


Figure 15. Dual simplex method example 3, iteration one

region, which can be evaluated with the displacement of the objective

function from the origin.

Simplex Tableau and Transformation

The simplex method tableau and transformation provides a more convenient format than the algebra derived above to search for a simplex new point.

The simplex algorithm is based on the transformation of a matrix by a Gauss-Jordan elimination. The matrix or tableau for example 3 is shown as tableau 1. In the first column is the constants column or the

	1	X	Y
z=	0	a	b
x=	0	1	0
y=	0	0	1
s1=	-e	c	d

Tableau 1. Dual simplex method tableau example 3, iteration one

slacks column. The second column is the X column or the coefficients of the X variable. The third column is the Y column or the coefficients of the Y variable. The rows of the tableau consist of the objective function row followed by the constraints defining the current point. This second set of equations is called the basis and consists of the minimum number of equations needed to define the current point. (For the point (0,0) the X,Y>=0 constraint are always used.) To the bottom of the tableau is appended the selected violated constraint called the

"pivot row".

Simplex Transformation

The algebraic solution for the new point can be duplicated in tableau 1 with a Gauss-Jordan elimination on the pivot row[35]. Again, there are two possible new points. To set s1 equal to zero (0) in the transformed tableau, the appended row can be transformed to either:

$$s1=0+0*X+1*Y \quad \text{or:} \quad s1=0+1*X+0*Y$$

To transform the original pivot row into the form above using a Gauss-Jordan elimination, either a "pivot" on the "pivot element" of the coefficient of the Y "pivot column" can be performed by dividing the pivot row by the Y coefficient d and then subtracting out multiples of the new Y element from the remaining element, or by a similar pivot on X. In either case, the same operations as performed on the pivot row must be performed on all the tableau rows, transforming the first row or objective row of the tableau to one (1) of the following equations.

$$z=\frac{b*e}{d}+(a-\frac{b*c}{d})*X+b*s1$$

$$z=\frac{a*e}{c}+a*s1+(b-\frac{a*d}{c})*Y$$

Of the two possible pivots yielding new points, the pivot which results in the minimum increase in z is selected. Since X, Y, s1 are zero (0) and a, b, d, c are greater than zero (0), selecting the pivot column with minimum ratio $(\frac{b}{d},\frac{a}{c})$ minimizes the objective function value.

Assuming $\frac{a}{c}<=\frac{b}{d}$, then the Gauss-Jordan elimination is as shown in tableaus 2 and 3. Tableau 2 is the division of the pivot row by c

	1	X	Y
z=	0	a	b
x=	0	1	0
y=	0	0	1
s1=	$\frac{-e}{c}$	1	$\frac{d}{c}$

Tableau 2. Dual simplex method tableau example 3, iteration one

and tableau 3 is the reduction of the tableau with the "pivot column".

	1	s1	Y
z=	$\frac{a*e}{c}$	a	$b-\frac{a*d}{c}$
x=	$\frac{e}{c}$	1	$\frac{-d}{c}$
y=	0	0	1
s1=	0	1	0

Tableau 3. Dual simplex method tableau example 3, iteration one

The new point can now be read directly from tableau 3 since the second and third row of tableau 3 are the same as equations (6); and geometrically, x and y are the displacement of the new point along the X and Y axes as shown in figure 16.

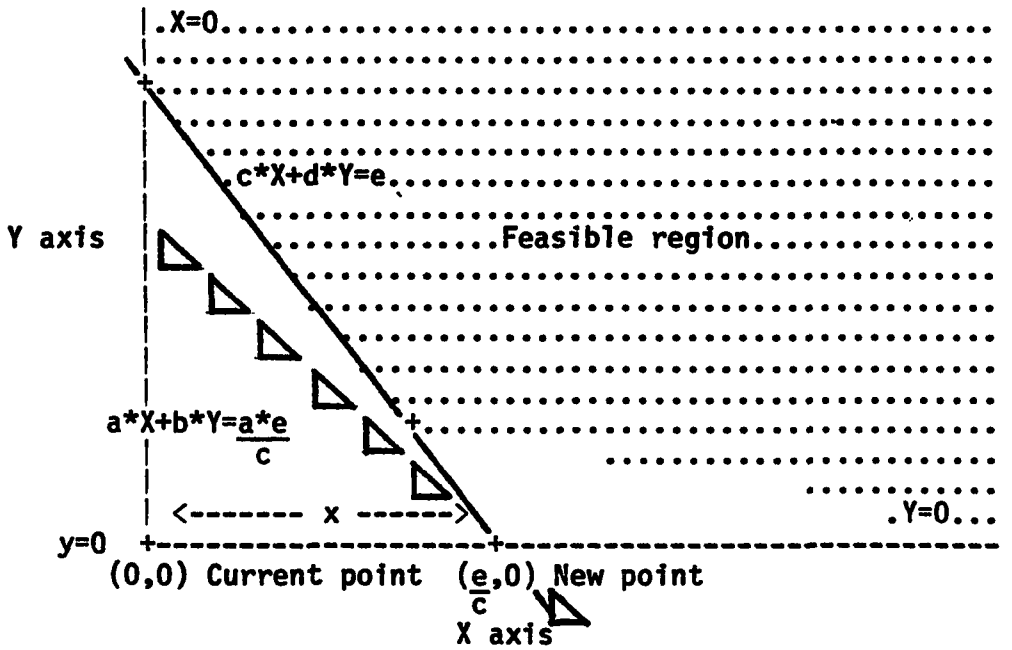


Figure 16. Dual simplex method example 3, iteration one

Another result of the transformation is that the objective function row, or first row elements of the tableau, remain greater than or equal to zero (0) which is an assumption of the method. This can be shown true by:

$$\frac{a}{c} <= \frac{b}{d} \text{ guarantees } b-\frac{a*d}{c} >= b-\frac{b*d}{d}=0$$

In the above example, the coefficients c and d were assumed greater than zero (0). In the general case, c or d may have any value. If a pivot were selected so that the pivot row element were zero (0), then the pivot would result in a division by zero (0) which by the geometry is a basis constraint that does not intersect with the pivot row. If a pivot were selected so that the pivot row element was negative, then the new point would violate one of the currently satisfied constraints and

could not possibly improve on the feasiblity of the current solution.

Perturbation Method

In the method described above, the selection of the pivot element from the pivot row was determined by the minimum $(\frac{a}{c},\frac{b}{d})$ ratio. If a tie, such as $\frac{a}{c}=\frac{b}{d}$, exists, then it is possible to return to a point previously evaluated and the method could "cycle". A modification to dual simplex method used to prevent cycling is a form of the "perturbation method"[36] which can be described geometrically as follows.

Figure 17 is the two (2) dimensional example 3 modified to demonstrate the perturbation method. As seen in the figure 17, the two

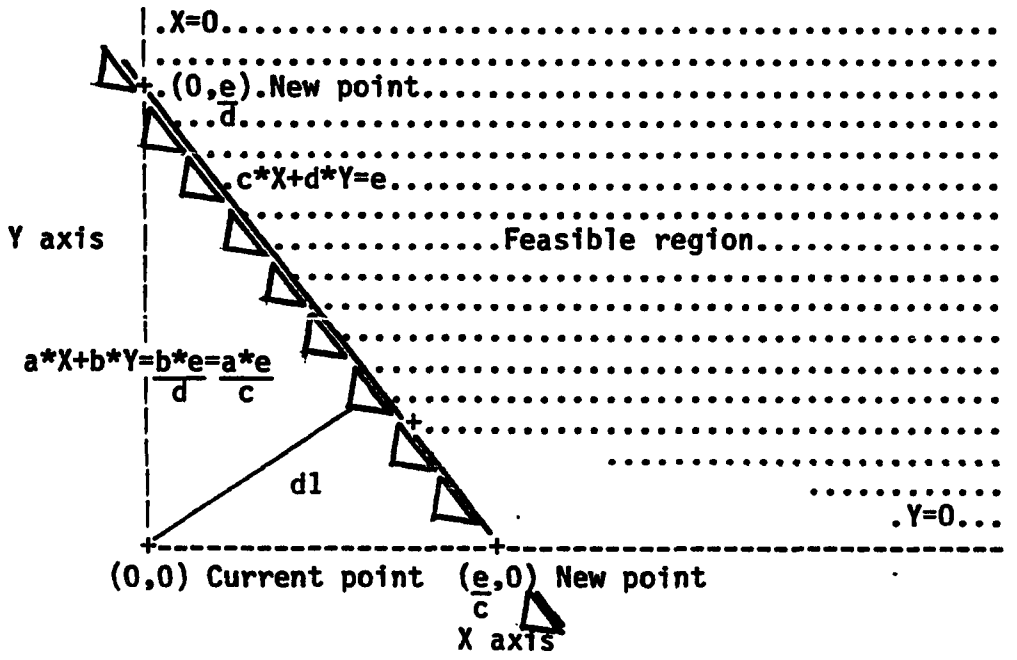


Figure 17. Dual simplex method perturbation iteration

(2) points eligible for a pivot are on a pivot row parallel to the objective function, so that either are equal candidates. To break the tie, select the point with the minimum value of X. If there is still a tie, select the point with the minimum value of Y, etc. Before all the dimensions of the problem are exceeded, a unique point must be found.

This is accomplished in the tableau by using the second row of the tableau to break the tie. Using the example, choose the:

$$\text{minimum } \left(\frac{1}{c},\frac{0}{d}\right)$$

If this is a tie then:

$$\text{minimum } \left(\frac{0}{c},\frac{1}{d}\right)$$

In the modified of example 3, using the Y column as the pivot, the new tableau is shown in tableau 4.

	1	X	s1
z=	$\frac{b*e}{c}$	$a-\frac{b*c}{d}=0$	b
x=	0	1	0
y=	$\frac{-e}{d}$	$\frac{-c}{d}$	1
s1=	0	0	1

Tableau 4. Dual simplex method tableau perturbation iteration

This tie breaking method will maintain not only a positive element in the first row, but also a positive element as the first greater than zero (0) element of the every variable coefficient column. This

tableau is defined as "lexicographical positive"[37].

Constraint Transformation

If the transformed tableau is rewritten as equations in s1 and Y, it would appear as:

$$z=\frac{(a*e)}{c}*1+a*s1+\frac{(b-a*d)}{c}*Y$$

$$x=\frac{e}{c}*1+\ s1\ \ \frac{-d}{c}*Y$$

$$y=\ 0*1+0*s1\ \ +1*Y$$

$$s1=\ 0*1+1*s1\ \ +0*Y$$

To find the violated constraints, substitute $X=\frac{e}{c}$ and Y=0 into the original constraints:

$$x=\frac{e}{c}>0 \quad y=0 \quad s1=\frac{-e}{c}+c*e+d*0=0 \quad s2=\frac{-f}{c}+g*e-h*0=\frac{-f}{c}+g*e$$

and select a violated constraint by the distance formula.

$$d1^2=\frac{(-e+c*e+d*0)^2}{c^2+d^2}<=\frac{(-f+g*e+h*0)^2}{g^2+h^2}=d2^2$$

Up to now, the search or "iteration" for the third point has been the same as the for the second. But the new tableau is now in terms of s1 and Y. Before the selected constraint can be appended to the current tableau, its equation has to be transformed to the current variables.

Algebraically, this can be done by substituting for X from the tableau's second row's displacement:

$$x=\frac{e}{c}+1*s1-\frac{d}{c}*Y$$

into the violated constraint:

$$s2=-f+g*X+h*Y$$

resulting in:

$$s2=\frac{(g*e}{c}-f)+g*s1+\frac{(h-g*d)}{c}*Y$$

This transformation can be accomplished directly in the simplex tableau by multiplying the coefficients of the constraint by the elements of the columns of the tableau and totaling in the manner demonstrated in the following expression of the transformed constraint.

$$\begin{array}{ccccc} & \text{column 1} & & \text{column 2} & \text{column3} \\ s2= & \{-f+g*(\frac{e}{c})+h*(0)\} & + & \{g*(1)+h*(0)\}*s1+ & \{g*(\frac{-d}{c})+h*(1)\}*Y \end{array}$$

Again, appending the selected constraint to the tableau as a pivot row results in tableau 5.

	1	s1	Y
z=	$\frac{a*e}{c}$	a	$b-\frac{a*d}{c}$
x=	$\frac{e}{c}$	1	$\frac{-d}{c}$
y=	0	0	1
s2=	$\frac{g*e-f}{c}$	g	$h-\frac{g*d}{c}$

Tableau 5. Dual simplex method tableau example 3, iteration two

Graphically, the current point and the selected constraint can be drawn as shown in figure 18. The intersection of the selected constraint and the current basis constraints form the two (2) possible new points from which the simplex algorithm will select a pivot to transform the tableau to represent the next simplex pivot.

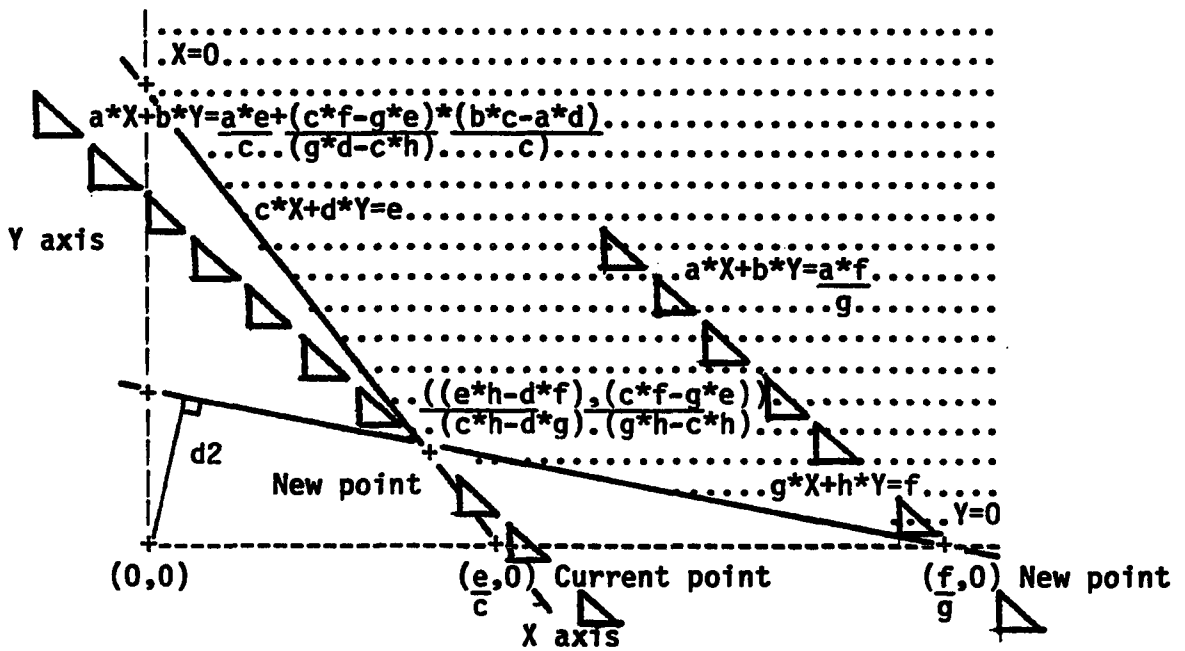


Figure 18. Dual simplex method example 3, iteration two

Assuming $g<0$ and $h-\frac{g*d}{c}<0$ and $\frac{a}{g}>=\frac{b*c-a*d}{c*h-g*d}$ and pivoting on Y results in tableaus 6 and 7.

	s1	Y	
z=	$\frac{a*e}{c}$	a	$b-\frac{a*d}{c}$
x=	$\frac{e}{c}$	1	$-\frac{d}{c}$
y=	0	0	1
s2=	$\frac{(g*e-c*f)}{(h*c-g*d)}$	$\frac{c*g}{(h*c-g*d)}$	1

Tableau 6. Dual simplex method tableau example 3, iteration two

		s1	s2
z=	$\frac{a*e-(g*e-c*f)*(b*c-a*d)}{c \quad (h*c-g*d) \quad c}$	$a-\frac{(c*g)*(b*c-a*d)}{(c*h-g*d) \quad c}$	$\frac{c*b-a*d}{c}$
x=	$\frac{e+(g*e-c*f)*d}{c \quad (h*c-g*d) \quad c}$	$1+\frac{(c*g)*d}{(h*c-g*d) \quad c}$	$\frac{-d}{c}$
y=	$\frac{-(g*e-c*f)}{(h*c-g*d)}$	$\frac{-c*g}{(h*c-g*d)}$	1
s2=	0	0	1

Tableau 7. Dual simplex tableau example 3, solution

Again checking the current point with the constraints.

$$x=e-\frac{(g*e-c*f)}{c*h-g*d}*\frac{d}{c}>0 \quad y=-\frac{(e*g-c*f)}{c*h-g*d}>0 \quad s1=c*\{\frac{e}{c}+\frac{(e*g-c*f)}{h*c-g*d}*\frac{d}{c}\}+d*\{\frac{c*f-g*e}{h*c-g*d}\}-e=0$$

$$s2=0$$

Since all of the constraints are satisfied, the current point is a feasible solution. Having selected each successive point in the search for a feasible point such that it was always the minimum value of the objective function guaranteed a solution from the feasible region which also minimized the objective function.

Geometrically, the current point is shown in figure 19. Again the point satisfies all constraints and is the point of minimum value of the objective function.

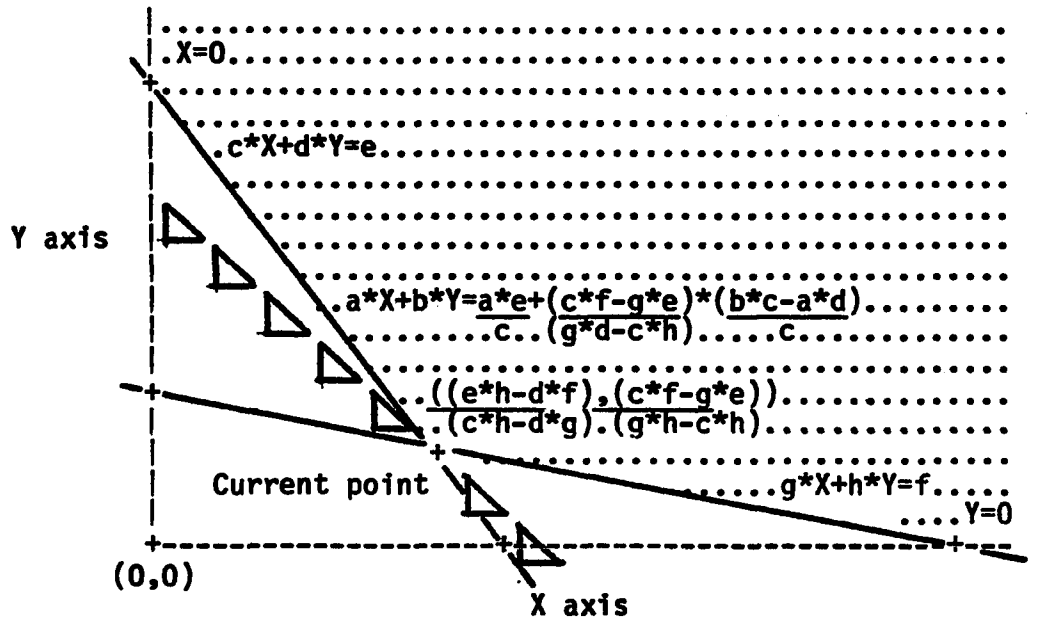


Figure 19. Dual simplex method example 3, solution

Dual Simplex Method in Matrix Notation

To provide a more rigorous description of the dual simplex method based on LP theory[38] and to extend the method into more than two (2) dimensions, the method will be presented in matrix notation.

The constraints of the problem derived from the CPM model can be

$$\text{minimize} \quad \overline{c}'\overline{x}$$

$$\text{subject to:} \quad \overline{Ax} >= \overline{a}$$
$$\overline{x} >= \overline{0}$$

written in the matrix notation above, where $\overline{c}$ is the vector of coefficients of the linear objective function; $\overline{a}$ is the vector of the constants of the constraints; and $\overline{A}$ is the constraint coefficients in a matrix with as many rows as constraints and as many columns as variables.

From LP theory, the above problem has a dual which can be written in

the following form.

$$\text{maximize} \quad \bar{a}'\bar{s}$$

$$\text{subject to:} \quad \bar{A}'\bar{s} \geq \bar{c}, \quad \bar{s} \geq \bar{0}$$

As in the two (2) dimensional example, a slack vector $\bar{x}$ can be added to change the inequalities to equalities.

$$\text{minimize} \quad z$$

$$\text{subject to:} \quad z - \bar{a}'\bar{s} - \bar{0}'\bar{x} = \bar{0}, \quad \bar{0}'z + \bar{A}'\bar{s} + \bar{I}'\bar{x} = \bar{c}, \quad \bar{z}, \bar{s}, \bar{x} \geq 0$$

If the matrices are now partitioned, the above matrices can be written as:

$$\left|\begin{array}{c|c|c} 1 & \bar{0}' & -\bar{a}' \\ \hline \bar{0} & \bar{I} & \bar{A}' \end{array}\right| * \left|\begin{array}{c} z \\ \hline \bar{x} \\ \hline \bar{s} \end{array}\right| = \left|\begin{array}{c} 0 \\ \hline \bar{c} \end{array}\right|$$

If the $\bar{A}'$ matrix is partitioned further into the linearly independent $\bar{B}'$ and dependent $\bar{D}'$ columns of the $\bar{A}'$ matrix (assuming a feasible solution exists) then:

$$\left|\begin{array}{c} -\bar{a}' \\ \hline \bar{A}' \end{array}\right| = \left|\begin{array}{c|c} -\bar{b}' & -\bar{d}' \\ \hline \bar{B}' & \bar{D}' \end{array}\right|$$

where $\bar{B}'$ is now a square matrix with as many rows and columns as variables in the problem. The partitioned dual problem can now be written in the matrix notation:

$$\begin{array}{|c|c|c|c|} \hline 1 & \bar{0}' & -\bar{b}' & \bar{d}' \\ \hline \bar{0} & \bar{I}' & \bar{B}' & \bar{D}' \\ \hline \end{array} * \begin{array}{|c|} \hline z \\ \hline \bar{x} \\ \hline \bar{s} \\ \hline \bar{s} \\ \hline \end{array} = \begin{array}{|c|} \hline 0 \\ \hline \bar{c} \\ \hline \end{array}$$

Again from LP theory the simplex transformation results in the "inversion" of the "augmented" $\bar{B}'$ matrix or:

$$\bar{B}' = \begin{array}{|c|c|} \hline 1 & -\bar{b}' \\ \hline \bar{0} & \bar{B}' \\ \hline \end{array}^{-1} = \begin{array}{|c|c|} \hline 1 & \bar{b}'\bar{B}'^{-1} \\ \hline \bar{0} & \bar{B}'^{-1} \\ \hline \end{array}$$

The tableau of the simplex method can now be constructed with these components:

Z	X	S	S		
1	$\bar{0}'$	$-\bar{b}'$	$\bar{d}'$	0	$=z$
$\bar{0}$	$\bar{I}'$	$\bar{B}'$	$\bar{D}'$	$\bar{c}$	$=\bar{s}$

From the transform of the above tableau, the simplex tableau is constructed.

	1	X	
	1	$\bar{0}'$	Z
$\bar{x}=$	$\bar{0}$	$\bar{I}$	X
$\bar{s}=$	$-\bar{b}$	$\bar{B}$	S
$\bar{s}=$	$-\bar{d}$	$\bar{D}$	S
$z=$	0	$\bar{c}'$	
	$=$	$=$	
	z	$\bar{s}'$	

To minimize computer storage, the dual simplex tableau is reduced

further. Using the equation format of the two (2) dimensional example 3:

$$\begin{aligned} &\text{minimize} && z \\ &\text{subject to:} && \underline{z+\bar{c}'\bar{x}=0} \\ & && -\bar{a}+\bar{A}\bar{x} \geq \bar{0} \end{aligned}$$

the equations can be written as the partitioned matrix and repartitioned as:

$$\begin{array}{c|c} 0 & \bar{c}' \\ \hline -\bar{a} & \bar{A} \end{array} = \begin{array}{c|c} 0 & \bar{c}' \\ \hline -\bar{b} & \bar{B} \\ \hline -\bar{d} & \bar{D} \end{array}$$

From LP theory, only the augmented $\bar{B}$ inverse is needed to reconstruct any transformed row of the dual simplex tableau. To construct the objective function:

$$| \; 0 \; , \bar{c}' | * \begin{array}{c|c} 1 & \bar{0}' \\ \hline \bar{B}^{-1}\bar{b} & \bar{B}^{-1} \end{array} = | \; \bar{c}'\bar{B}^{-1}\bar{b}, \; \bar{c}'\bar{B}^{-1} |$$

and any constraint:

$$| -a_p \; , \bar{a}_p' | * \begin{array}{c|c} 1 & \bar{0}' \\ \hline \bar{B}^{-1}\bar{b} & \bar{B}^{-1} \end{array} = | -a_p+\bar{a}_p'\bar{B}^{-1}\bar{b}, \; \bar{a}_p'\bar{B}^{-1} |$$

which is the same transformation in matrix form as used in the two (2) dimensional example 3.

The final form of the simplex tableau, as used in the computer program, can now be constructed. For each section of the partitioned tableau, there is defined a program array. The first section of the tableau is the original coefficients of the objective and constraints. The next section is the transformed objective coefficients and the augmented $\bar{B}$ inverse followed by the constraint coefficients from which

		program variables
0	$\bar{c}'$	A#(1,ND+1)
$-\bar{a}$	$\bar{A}$	A#(2 to MD+1,ND+1)
		(column one negative)
$z=\ \bar{c}'\bar{B}^{-1}\bar{b}$	$\bar{c}'\bar{B}^{-1}$	B#(1,ND+1)
$x=\ \bar{B}^{-1}\bar{b}$	$\bar{B}^{-1}$	B#(2 to ND+1,ND+1)
$-\bar{b}$	$\bar{B}$	B#(ND+2 to ND+1+ND,ND+1)
		(column one negative)
$s=\ -a_p+\bar{a}_p'\bar{B}^{-1}\bar{b}$	$\bar{a}_p'\bar{B}^{-1}$	P#(ND+1)

the current basis inverse was constructed. At the bottom of the matrix is appended the transformed violated constraint or the pivot row.

With this tableau, using the procedures of the two (2) dimensional example, a pivot column can be found and the Gauss-Jordan transformation used to invert the new basis in which the violated constraint has replaced a row of the augmented $\bar{B}$ matrix.

$$\begin{vmatrix} 1 & \bar{0}' \\ -\bar{b}_{new} & \bar{B}_{new} \end{vmatrix}^{-1} = \begin{vmatrix} 1 & \bar{0}' \\ \bar{B}^{-1}\bar{b} & \bar{B}^{-1} \end{vmatrix}$$

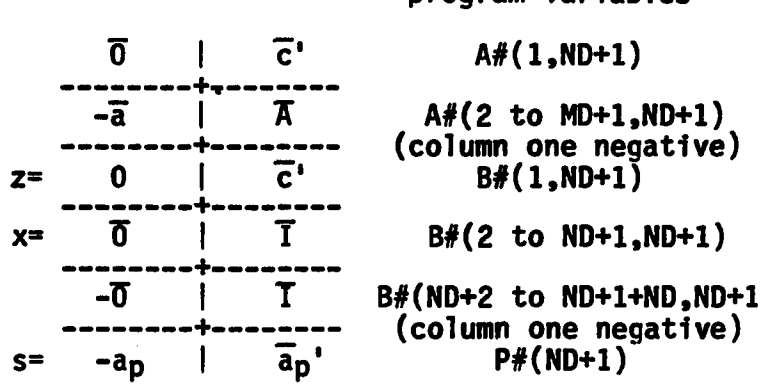
To start the iterative process, the origin point is defined by the identity matrix. To construct the objective function:

$$\left| -0\ ,\ \bar{c}' \right| * \begin{vmatrix} 1 & \bar{0}' \\ \bar{0} & \bar{I} \end{vmatrix} = \left| 0\ ,\ \bar{c}' \right|$$

and to construct the pivot row from the selected constraint:

$$\left| -a_p\ ,\ \bar{a}_p' \right| * \begin{vmatrix} 1 & \bar{0}' \\ \bar{0} & \bar{I} \end{vmatrix} = \left| -a_p\ ,\ \bar{a}_p' \right|$$

The resulting tableau and program arrays:



Dual Simplex Method BASIC Code

The following code is a modification of the dual simplex method with extra code provided for use in later programs. If only the dual simplex application is needed, then the code can easily be reduced for faster processing and more efficient data storage.

All programs listed in the text consist of a main calling routine and a series of input, output, and processing subroutines. These BASIC routines are listed in the text as well as on a computer disk which is compatible with IBM micro-computers.

Dual Simplex Main Routine -- File MAIN-SMP

The dual simplex main calling routine (MAIN-SMP) dimension nine (9) arrays; writes the options menu to the screen as show in figure 20; calls the utility subroutines UTIL-OPT, UTIL-ERS, UTIL-CON, UTIL-CHX; calls the data input and output subroutines INPT-OBJ, INPT-CON, INPT-TYP, INPT-BND, and REPT-SMP; call and times the processing algorithm ALGR-SMP; and saves and fetches the input data to disk.

```
                    DUAL SIMPLEX METHOD

          NUMBER OF CONSTRAINTS              10
          NUMBER OF VARIABLES                14
          MAXIMUM ITERATIONS               1000

                    M-RETURN TO MENU

               O-OBJECTIVE COEFFICIENTS
               A-CONSTRAINT COEFFICIENTS
               C-CONSTRAINT TYPES
               B-BOUNDED VARIABLES
               U-EXECUTE ALGORITHM
               R-REPORT LISTING
               N-NEW PROBLEM
               S-SAVE  F-FETCH

          OPTION ?
```

Figure 20. Dual simplex method main menu screen

```
1 REM                      * DUAL SIMPLEX METHOD *
2 REM--------------------------MAIN-SMP-------------------------

3 REM     BI# - MACHINE INFINITE
4 REM     CO  - PIVOT COLUMN FOR SIMPLEX TRANSFORMATION
5 REM     ER  - ERROR KEY
6 REM     IR  - MAXIMUM NUMBER OF ITERATIONS
7 REM     MD  - NUMBER OF CONSTRAINTS
8 REM     ND  - NUMBER OF VARIABLES
9 REM     OB# - VALUE OF THE OBJECTIVE FUNCTION
10 REM    RO  - PIVOT ROW FOR SIMPLEX TRANSFORMATION
11 REM    SM# - MACHINE ZERO
12 REM    A#(MD+1,ND+1) - ORIGINAL DATA AND KEYS
13 REM    B#(ND+ND+1,ND+1) - PRIMAL-DUAL MATRIX
14 REM    C(ND+1) - COLUMNS ELIGIBLE FOR PIVOT
15 REM    H#(ND+1) - SOLUTION TO LAST ITERATION
16 REM    M#(ND+1,2) - UPPER AND LOWER BOUNDS OF VARIABLES
17 REM    P#(ND+1) - CURRENT PIVOT ROW
18 REM    R(MD+1) - CONSTRAINT TYPES
19 REM    T#(ND+1,ND+1) - REINVERSION WORKSPACE
20 REM    X#(ND) - SOLUTION VECTOR
21 REM ------------------------------------------------------------
```

Sets MD to the default number of constraints in the problem to be optimized and ND to the number of variables. Sets IN to the number of iterations before reinversion of the augmented $\bar{B}$ basis matrix. Sets the

default maximum number of iterations to one thousand (1000). Sets BI# to a number considered infinite and SM# to a number considered zero (0).

```
22 MD=0
23 ND=0
24 IN=20
25 IR=1000
26 BI#=1E+10
27 SM#=1E-10
```

Prompts and reads from the keyboard the number of constraints, MD; the number of variables, ND; and the maximum number of iterations, IR.

```
28 CLS
29 LOCATE 1,10:PRINT "DUAL SIMPLEX METHOD"
30 LOCATE 3,1:PRINT "NUMBER OF CONSTRAINTS":LOCATE 3,31:INPUT "",L$
31 GOSUB 1870:REM UTIL-CHX
32 IF Z#<>BI# THEN MD=Z#
33 LOCATE 3,30:PRINT MD,"    ":LOCATE 4,1:PRINT "NUMBER OF VARIABLES"
   :LOCATE 4,31:INPUT "",L$
34 GOSUB 1870:REM UTIL-CHX
35 IF Z#<>BI# THEN ND=Z#
36 LOCATE 4,30:PRINT ND,"    ":LOCATE 5,1:PRINT "MAXIMUM ITERATIONS"
   :LOCATE 5,31:INPUT "",L$
37 GOSUB 1870:REM UTIL-CHX
38 IF Z#<>BI# THEN IR=Z#
39 LOCATE 5,30:PRINT IR,"    "
```

Dimensions the objective function and constraint coefficient array A#(MD+1,ND+8), the basis inverse and basis array B#(2*ND+1,ND+1), the eligible entering column array C(ND+1), the holding array H#(ND+1), the upper and lower bounds array M#(ND+1,2), the pivot row array P#(ND+1), the constraint type array R(MD+1), the reinversion work space array T#(ND+1,ND+1), and the solution array X#(ND).

The arrays A#(MD+1,ND+8), B#(2*ND+1,ND+1), and P#(ND+1) have been defined previously. The only differences is that array A#(MD+1,ND+8) contains seven extra columns used in later algorithms. The array C(ND+1) contains elements with value either zero (0) or one (1) corresponding to each column of the simplex tableau where one (1) is a column eligible for a pivot operation and zero (0) is column not eligible. In the case of ties there would be more than one one (1) in the array. H#(ND+1) is a holding array for the last point before the current point. In later algorithms this is used to reduce duplicate processing if the current point values remain the same as at the last iteration. The array M#(ND+1,2) contains two (2) values for each variable (offset by one (1)). The first value is the upper bound, and the second value is the lower bound of the variable. The array R(MD+1)

contains a value for each constraint (offset by one) that represents the relationship type where one (1) is for ">=", minus one (-1) for "<=", and zero (0) for "=". The array T#(ND+1,ND+1) is a work space array in which the current basis is held for a Gauss-Jordan reinversion. And the array X#(ND) is the point that is both feasible and optimal.

```
40 DIM A#(MD+1,ND+8)
41 DIM B#(2*ND+1,ND+1)
42 DIM C(ND+1)
43 DIM H#(ND+1)
44 DIM M#(ND+1,2)
45 DIM P#(ND+1)
46 DIM R(MD+1)
47 DIM T#(ND+1,ND+1)
48 DIM X#(ND)
49 FOR I=2 TO MD+1
50 R(I)=1
51 NEXT I
```

Prints the option menu to the screen; calls the option line routine, UTIL-OPT; and pauses for the entry of "M", "O", "A", "C", "B", "U", "R", "N", "S", "F" for the option variable L$.

```
52 LOCATE 8,12:PRINT       "M-RETURN TO MENU"
53 LOCATE 10,7:PRINT   "O-OBJECTIVE COEFFICIENTS"
54 LOCATE 11,7:PRINT   "A-CONSTRAINT COEFFICIENTS"
55 LOCATE 12,7:PRINT   "C-CONSTRAINT TYPES"
56 LOCATE 13,7:PRINT   "B-BOUNDED VARIABLES"
57 LOCATE 14,7:PRINT   "U-EXECUTE ALGORITHM"
58 LOCATE 15,7:PRINT   "R-REPORT LISTING"
59 LOCATE 16,7:PRINT   "N-NEW PROBLEM"
60 LOCATE 17,7:PRINT   "S-SAVE  F-FETCH"
61 GOSUB 1800:REM UTIL-OPT
62 LOCATE 21,8:INPUT "",L$
```

Calls either the objective function input subroutine INPT-OBJ, the constraint input subroutine INPT-CON, the constraint type input subroutine INPT-TYP, the upper and lower bounds input subroutine INPT-BND, the processing subroutine ALGR-SMP, or the report subroutine REPT-SMP based on the option variable L$.

```
63 CLS
64 IF L$<>"O" THEN 67
65 GOSUB 1200:REM INPT-OBJ
66 GOTO 63
67 IF L$<>"A" THEN 70
68 GOSUB 1300:REM INPT-CON
69 GOTO 63
```

```
70 IF L$<>"C" THEN 73
71 GOSUB 1400:REM INPT-TYP
72 GOTO 63
73 IF L$<>"B" THEN 76
74 GOSUB 1500:REM INPT-BND
75 GOTO 63
76 IF L$<>"U" THEN 85
77 TM=3600*VAL(MID$(TIME$,1,2))+60*VAL(MID$(TIME$,4,2))+VAL(MID$(TIME$,7
   ,2))
78 GOSUB 3300:REM ALGR-SMP
```

Sets the last current point of the simplex tableau to the optimal solution.

```
79 OB#=B#(1,1)
80 FOR I=1 TO ND
81 X#(I)=B#(I+1,1)
82 NEXT I
83 TM=3600*VAL(MID$(TIME$,1,2))+60*VAL(MID$(TIME$,4,2))+VAL(MID$(TIME$,7
   ,2))-TM
84 GOTO 52
85 IF L$<>"R" THEN 88
86 GOSUB 2200:REM REPT-SMP
87 GOTO 63
```

Saves the contents of MD, ND, M#(ND+1,2), A#(MD+1,ND+8), and R(ND+1) to disk file "DATA" as an ASCI file if option "S" is selected.

```
88 IF L$<>"S" THEN 106
89 OPEN "O",#1,"DATA"
90 PRINT #1,STR$(MD)
91 PRINT #1,STR$(ND)
92 FOR I=1 TO ND+1
93 FOR J=1 TO ND+1
94 PRINT #1,""
95 NEXT J
96 PRINT #1,STR$(M#(I,1))
97 PRINT #1,STR$(M#(I,2))
98 NEXT I
99 FOR I=1 TO MD+1
100 FOR J=1 TO ND+8
101 PRINT #1,STR$(A#(I,J))
102 NEXT J
103 PRINT #1,STR$(R(I))
104 NEXT I
105 CLOSE #1
```

Loads to MD, ND, M#(ND+1,2), A#(MD+1,ND+8), and R(ND+1) the disk

file "DATA" if option "F" is selected.

```
106 IF L$<>"F" THEN 131
107 OPEN "I",#1,"DATA"
108 INPUT #1,X$
109 MD=VAL(X$)
110 INPUT #1,X$
111 ND=VAL(X$)
112 FOR I=1 TO ND+1
113 FOR J=1 TO ND+1
114 INPUT #1,X$
115 NEXT J
116 INPUT #1,X$
117 M#(I,1)=VAL(X$)
118 INPUT #1,X$
119 M#(I,2)=VAL(X$)
120 NEXT I
121 FOR I=1 TO MD+1
122 FOR J=1 TO ND+8
123 INPUT #1,X$
124 A#(I,J)=VAL(X$)
125 NEXT J
126 INPUT #1,X$
127 R(I)=VAL(X$)
128 NEXT I
129 CLOSE #1
130 GOTO 52
```

Restarts the program for a new run if option "N" is selected.

```
131 IF L$="N" THEN RUN
132 GOTO 52
```

Utility Subroutines -- Files UTIL-OPT, UTIL-ERS, UTIL-CON, and UTIL-CHX

Same as for critical path method.

Objective Function Input Subroutine -- Files INPT-OBJ

The objective function input subroutine (INPT-OBJ) is the interactive screen input of the first row of the A#(MD+1,ND+8) array, columns two (2) through ND+1. To reach the screen from the main menu, type "O" as the OPTION ?. The screen, as shown in figure 21, consists

```
NON-0 COEFFICIENTS IN THE OBJECTIVE
     VARIABLE NO.   COEFFICIENT
     5              5
     6              10
     7              15
     8              20
     9              25
     10             30
     11             35
     12             40
     13             45
     14             50

OPTION ?
```

Figure 21. Dual simplex method objective function input screen

of a variable number and the objective function coefficient for that variable. When the screen is first entered, the cursor will be in the VARIABLE NO. column. When a variable number or a blank is typed and entered, the cursor will move to the COEFFICIENT column so that the coefficient can be typed and entered.

If an error is entered, then the variable number can be reentered and the corrections made. To select other options and leave the screen, the option letter is typed in the VARIABLE NO. column and entered.

```
1200 REM                * OBJECTIVE INPUT SUBROUTINE *
1201 REM-------------------------INPT-OBJ-----------------------------

1202 H=0
1203 G=2
1204 LOCATE 1,1:PRINT "NON-0 COEFFICIENTS IN THE OBJECTIVE":LOCATE 2,5:
     PRINT "VARIABLE NO.  COEFFICIENT"
1205 G=G+1
1206 H=H+1
1207 GOSUB 1800:REM UTIL-OPT
1208 LOCATE G,11:INPUT "",L$
1209 GOSUB 1870:REM UTIL-CHX
1210 IF L$<>"" AND Z#=BI# THEN RETURN
1211 IF Z#<>BI# THEN H=Z#
```

```
1212 IF H>ND OR H<=0 THEN 1207
1213 GOSUB 1850:REM UTIL-ERS
1214 LOCATE G,10:PRINT H,"  ":LOCATE G,21:PRINT A#(1,H+1):LOCATE G,22
     :INPUT "",L$
1215 GOSUB 1870:REM UTIL-CHX
1216 IF Z#<>BI# THEN A#(1,H+1)=Z#
1217 LOCATE G,21:PRINT A#(1,H+1),"     "
1218 IF G<18 THEN 1205
1219 GOSUB 1860:REM UTIL-CON
1220 GOTO 1206
```

Constraint Input Subroutine -- File INPT-CON

The constraint input subroutine (INPT-OBJ) is the interactive screen input of rows two (2) to MD+1 and columns two (2) to ND+1 of the A#(MD+1,ND+8) array. To reach the screen from the main menu, type "A" for the OPTION ?. The screen, as shown in figure 22, consists of a

```
NON-0 COEFFICIENTS A MATRIX
  ROW COLUMN  COEFFICIENTS
  1   3        1
  1   5        1
  2   4        1
  2   6        1
  2   2       -1
  3   1        1
  3   7        1
  4   2        1
  4   8        1
  4   1       -1
  5   3        1
  5   9        1
  5   2       -1

OPTION ?
```

Figure 22. Dual simplex method constraint coefficient input screen

constraint or ROW number, a variable or COLUMN number, and the variable COEFFICIENT column which correspond to the entries in the simplex $\bar{A}$ matrix. When the screen is first entered, the cursor will be in the ROW

column. When a row number or a blank is typed and entered, the cursor will move to the COLUMN column so that the column number can be typed and entered. The cursor will then move to the COEFFICIENT column so that the coefficient of the variable can be typed and entered.

If an error is entered, then the row and column numbers must be re-entered and the corrections made. To select other options and leave the screen, the option letter is typed in the ROW column and entered.

```
1300 REM          * CONSTRAINT COEFFICIENT INPUT SUBROUTINE *
1301 REM------------------------INPT-CON-----------------------------
1302 LOCATE 1,1:PRINT "NON-ZERO COEFFICIENTS A MATRIX":LOCATE 2,2:PRINT
     "ROW COLUMN  COEFFICIENTS"
1303 H=1
1304 G=2
1305 R=0
1306 G=G+1
1307 R=R+1
1308 GOSUB 1800:REM UTIL-OPT
1309 LOCATE G,3:INPUT "",L$
1310 GOSUB 1870:REM UTIL-CHX
1311 IF L$<>"" AND Z#=BI# THEN RETURN
1312 IF Z#<>BI# THEN H=Z#
1313 IF R<=ND THEN 1316
1314 R=1
1315 IF L$="" THEN H=H+1
1316 IF H>MD OR H<=0 THEN 1308
1317 GOSUB 1850:REM UTIL-ERS
1318 LOCATE G,2:PRINT H:LOCATE G,9:INPUT "",L$
1319 GOSUB 1870:REM UTIL-CHX
1320 IF Z#<>BI# THEN R=Z#
1321 IF R>ND OR R<=0 THEN 1318
1322 LOCATE G,8:PRINT R:LOCATE G,15:PRINT A#(H+1,R+1):LOCATE G,16:INPUT
     "",L$
1323 GOSUB 1870:REM UTIL-CHX
1324 IF Z#<>BI# THEN A#(H+1,R+1)=Z#
1325 LOCATE G,15:PRINT A#(H+1,R+1),"   "
1326 IF G<18 THEN 1306
1327 GOSUB 1860:REM UTIL-CON
1328 GOTO 1307
```

Constraint Type Input Subroutine -- File INPT-TYP

The constraint type input subroutine (INPT-OBJ) is the interactive screen input of the R(MD+1) array and of the ND+2 through ND+8 columns of the A#(MD+1,ND+8) array. All constraints are stored in the A#(MD+1,ND+8) array in the greater than or equal form. To save the relationship type, a one (1) for constraints of the ">=" type, a zero (0) for constraints of the "=" type, and a minus one (-1) for constraints of the "<=" type is entered in the array R(MD+1). The last columns of the A#(MD+1,ND+8) array are used in later algorithms for storing keys to the nonlinear constraints.

To reach the screen from the main menu, type "C" as the OPTION ?. The screen, as shown in figure 23, consists of a column of row numbers,

```
                VALUE AND TYPE OF CONSTRAINTS
ROW   LINEAR PARABOLIC HYPERBOLIC SHEET
1 >= 10      0         0          0
2 >= 20      0         0          0
3 >= 30      0         0          0
4 >= 40      0         0          0
5 >= 50      0         0          0
6 >= 60      0         0          0
7 >= 70      0         0          0
8 >= 80      0         0          0
9 >= 90      0         0          0
10>= 100     0         0          0

OPTION ?
```

Figure 23. Dual simplex method constraint type input screen

a relationship type, a constant column for linear constraints, a constant column for a hyperbolic constraints, and a constant column for a hyperbolic of two sheet constraints.

As an example of the input format using two (2) variables constraints, for the linear cases:

X+Y>=LINEAR constant or X+Y<=LINEAR constant or X+Y=LINEAR constant

for the parabolic case:

Y+a*(X-PARABOLIC constant)^2>=LINEAR constant

for the hyperbolic case:

X*Y>=HYPERBOLIC constant

and for the hyperbolic in two sheets case:

X*(Y-Z)>=SHEET constant

When the screen is first entered, the cursor will be under the ROW column. When a constraint number or a blank is typed and entered, the cursor will move to the relationship column so that ">=" or "<=" or "=" can be typed and entered. The cursor will then move to the constant columns and the constants can be typed and entered in the same manner.

If an error is entered, then the constraint number is reentered and the corrections made. To select other options and leave the screen, the option letter is typed in the ROW column and entered.

```
1400 REM              * CONSTRAINT TYPE INPUT SUBROUTINE *
1401 REM-------------------------INPT-TYP-----------------------------

1402 H=0
1403 G=2
1404 LOCATE 1,3:PRINT " VALUE AND TYPE OF CONSTRAINTS":LOCATE 2,1:PRINT
     "ROW  LINEAR PARABOLIC HYPERBOLIC PLANE"
1405 G=G+1
1406 H=H+1
1407 GOSUB 1800:REM UTIL-OPT
1408 LOCATE G,2:INPUT "",L$
1409 GOSUB 1870:REM UTIL-CHX
1410 IF L$<>"" AND Z#=BI# THEN RETURN
1411 IF Z#<>BI# THEN H=Z#
```

```
1412 IF H>MD OR H<=0 THEN 1407
1413 GOSUB 1850:REM UTIL-ERS
1414 IF R(H+1)=1 THEN L$=">="
1415 IF R(H+1)=0 THEN L$="="
1416 IF R(H+1)=-1 THEN L$="<="
1417 LOCATE G,1:PRINT H,"  ":LOCATE G,4:PRINT L$:LOCATE G,4:INPUT "",L$
1418 IF L$<>">=" AND L$<>"=" AND L$<>"<=" AND L$<>"" THEN 1417
1419 IF L$="" THEN 1423
1420 IF L$=">=" THEN R(H+1)=1
1421 IF L$="=" THEN R(H+1)=0
1422 IF L$="<=" THEN R(H+1)=-1
1423 LOCATE G,4:PRINT L$:LOCATE G,6:PRINT A#(H+1,1):LOCATE G,7:INPUT ""
     ,L$
1424 GOSUB 1870:REM UTIL-CHX
1425 IF Z#<>BI# THEN A#(H+1,1)=Z#
1426 LOCATE G,6:PRINT A#(H+1,1),"    "
1427 IF R(H+1)<1 THEN 1438
1428 LOCATE G,13:PRINT A#(H+1,ND+2):LOCATE G,14:INPUT "",L$
1429 GOSUB 1870:REM UTIL-CHX
1430 IF Z#<>BI# THEN A#(H+1,ND+2)=Z#
1431 LOCATE G,13:PRINT A#(H+1,ND+2),"    ":LOCATE G,23:PRINT A#(H+1,ND+3
     ):LOCATE G,24:INPUT "",L$
1432 GOSUB 1870:REM UTIL-CHX
1433 IF Z#<>BI# THEN A#(H+1,ND+3)=Z#
1434 LOCATE G,23:PRINT A#(H+1,ND+3),"    ":LOCATE G,34:PRINT A#(H+1,ND+4
     ):LOCATE G,35:INPUT "",L$
1435 GOSUB 1870:REM UTIL-CHX
1436 IF Z#<>BI# THEN A#(H+1,ND+4)=Z#
1437 LOCATE G,34:PRINT A#(H+1,ND+4),"     "
1438 IF G<18 THEN 1405
1439 GOSUB 1860:REM UTIL-CON
1440 GOTO 1406
```

Upper and Lower Bounds Input Subroutine -- File INPT-BND

The upper and lower bounds input subroutine (INPT-BND) is the interactive screen input of the M#(MD+1,2) array.

To reach the screen from the main menu, type "B" as the OPTION ?. The screen, as shown in figure 24, consists of a variable number, its upper bound, and its lower bound. When the screen is first entered, the cursor will be in the VARIABLE NO. column. When a variable number or a blank is typed and entered, the cursor will move to the UPPER BOUND

```
          BOUNDS ON VARIABLES
VARIABLE NO.  UPPER BOUND  LOWER BOUND
4             160          0
5             9            0
6             19           0
7             29           0
8             39           0
9             49           0
10            59           0
11            69           0
12            79           0
13            89           0
14            99           0

OPTION ?
```

Figure 24. Dual simplex method variable bounds input screen

column so that the value can be typed and entered. The cursor will then move to the LOWER BOUND column where the lower bound can be typed and entered.

If an error is entered, then the variable number can be reentered and the corrections made. To select other options and leave the screen, the option letter is typed in the VARIABLE NO. column and entered.

```
1500 REM              * VARIABLE BOUNDS INPUT SUBROUTINE *
1501 REM-------------------------INPT-BND-----------------------------

1502 H=0
1503 G=2
1504 LOCATE 1,10:PRINT "BOUNDS ON VARIABLES":LOCATE 2,1:PRINT "VARIABLE
     NO.  UPPER BOUND  LOWER BOUND"
1505 G=G+1
1506 H=H+1
1507 GOSUB 1800:REM UTIL-OPT
1508 LOCATE G,2:INPUT "",L$
1509 GOSUB 1870:REM UTIL-CHX
1510 IF L$<>"" AND Z#=BI# THEN RETURN
1511 IF Z#<>BI# THEN H=Z#
1512 IF H>ND OR H<=0 THEN 1507
1513 GOSUB 1850:REM UTIL-ERS
1514 LOCATE G,1:PRINT H,"  ":LOCATE G,14:PRINT M#(H+1,1):LOCATE G,15
```

```
    :INPUT "",L$
1515 GOSUB 1870:REM UTIL-CHX
1516 IF Z#<>BI# AND Z#>=0# THEN M#(H+1,1)=Z#
1517 LOCATE G,14:PRINT M#(H+1,1),"    ":LOCATE G,27:PRINT M#(H+1,2)
    :LOCATE G,28:INPUT "",L$
1518 GOSUB 1870:REM UTIL-CHX
1519 IF Z#<>BI# AND Z#>=0# THEN M#(H+1,2)=Z#
1520 LOCATE G,27:PRINT M#(H+1,2),"    "
1521 IF G<18 THEN 1505
1522 GOSUB 1860:REM UTIL-CON
1523 GOTO 1506
```

Dual Simplex Report Subroutine -- File REPT-SMP

The simplex report subroutine (REPT-SMP) is the interactive screen output of the X#(ND) array. To reach the screen from the main menu, type "R" as the OPTION ?. The screen, as shown in figure 25. The dual

```
SOLUTION FOUND IN   7 ITR        5 SEC
VALUE OF OBJECTIVE               750

        VARIABLE NO.    VALUES
        1               30
        2               80
        3               10
        4               160
        5               0
        6               0
        7               0
        8               0
        9               30
        10              0
        11              0
        12              0
        13              0
        14              0

    OPTION ?
```

Figure 25. Dual simplex method solution screen.

simplex report lists the number of iterations and seconds required to reach an optimal solution, the value of the objective function, and the

variable numbers and their values at the solution. At the end of each screen of output, the OPTION ? line is printed and the program pauses. To continue, enter blank. Otherwise, any other option.

If the dual simplex method fails to reach a solution then instead of a report, an error code is printed where:

ER 0 = Iteration limit exceeded
ER 1 = Successful execution
ER 2 = Problem has no feasible solutions
ER 3 = Reinversion failed because of numerical accuracy
ER 4 = Maximum number of branch and bound nodes exceeded

```
2200 REM                *SIMPLEX REPORT SUBROUTINE*
2201 REM------------------------REPT-SMP-----------------------------

2202 IF ER<>2 AND ER<>3 AND ER<>4 THEN 2205
2203 LOCATE 1,6:PRINT "NO SOLUTION FOUND.";ER;"ER"
2204 GOTO 2210
2205 IF ER<>0 THEN 2208
2206 LOCATE 1,6:PRINT "MAXIMUM ITERATIONS REACHED "
2207 GOTO 2210
2208 IF ER<>1 THEN 2222
2209 LOCATE 1,1:PRINT "SOLUTION FOUND IN ";IT;"ITR"
2210 LOCATE 1,30:PRINT TM;"SEC":LOCATE 2,1:PRINT "VALUE OF OBJECTIVE"
     :LOCATE 2,30:PRINT OB#:LOCATE 4,8:PRINT "VARIABLE NO.   VALUES"
2211 G=5
2212 FOR I=1 TO ND
2213 LOCATE G,7:PRINT I:LOCATE G,22:PRINT X#(I)
2214 G=G+1
2215 IF G<20 THEN 2221
2216 GOSUB 1860:REM UTIL-CON
2217 LOCATE 21,8:INPUT "",L$
2218 GOSUB 1870:REM UTIL-CHX
2219 IF L$<>"" AND Z#=BI# THEN RETURN
2220 GOSUB 1864:REM UTIL-CON+4
2221 NEXT I
2222 GOSUB 1800:REM UTIL-OPT
2223 LOCATE 21,8:INPUT "",L$
2224 RETURN
```

Dual Simplex Algorithm Subroutine -- File ALGR-SMP

The dual simplex algorithm first initializes the simplex tableau to

represent the point at the origin and the set of constraints that form the zero axes. Then, in an iterative fashion, it proceeds to execute the following steps.

(1) From the constraints of the LP problem, select the constraint violated by the current point which is geometrically farthest from the current point. If no constraints are violated, then the current point is a feasible optimal solution. Otherwise, go to step (2).

(2) From the points of intersection formed by the constraint selected in step (1) and the constraints defining the current point, select a new point which minimizes the value of the objective and does not violate the basis constraints. This can be done directly from the simplex tableau as demonstrated earlier. If no intersection is found, set ER=2, since the the problem is infeasible, and return to the main routine. Otherwise, go to step (3).

(3) Use a Gauss-Jordan elimination to transform the current tableau to represents the new point selected in step (2) and the constraints which define it. If the transformation fails because of numerical accuracy, set ER=3 and return to main routine. Otherwise, go to step (4).

(4) Set the current point equal to the new point and return step (1). If the cycle is repeated more than a preset number of iterations, then set ER=0 and return to main routine.

```
3300 REM          *DUAL SIMPLEX ALGORITHM SUBROUTINE*
3301 REM-----------------------ALGR-SMP-------------------------------
```

Initializes the simplex tableau to represent the origin by setting the $\bar{B}$ matrix to the identity matrix. Sets the iteration count IT to zero (0) and the error code ER to zero (0).

```
3302 FOR I=1 TO 2*ND+1
3303 FOR J=1 TO ND+1
3304 B#(I,J)=0#
3305 NEXT J
3306 NEXT I
3307 FOR I=2 TO ND+1
3308 H#(I)=BI#
3309 B#(I,I)=1#:B#(I,1)=M#(I,2)
3310 B#(I+ND,I)=1#:B#(I+ND,1)=M#(I,2)
3311 B#(1,I)=A#(1,I)
3312 NEXT I
3313 IT=0
3314 ER=0
```

Increments the iteration count by one and checks if maximum iteration has been reached.

```
3315 IT=IT+1
3316 IF IT>IR THEN RETURN
```

Checks the upper bound for the most violated constraint and saves the slack amount in MI#. In the case of the upper bound, the geometric distance is also the amount by which the constraint is violated. Sets RO to the row number of the violated constraint if the violation is currently the largest found in the current iteration.

To accommodate upper and lower bounds, the number of constraints is automatically increased in the algorithm to ND+1+ND+1+MD+1. The extra constraint on each grouping of constraints is to allow for the offset formed by the objective function in the simplex tableau. This simplifies addressing at the expense of phantom constraints.

```
3317 MI#=-SM#
3318 RO=0
3319 FOR I=2 TO ND+1
3320 IF B#(I,1)-M#(I,2)>=MI# THEN 3323
3321 MI#=B#(I,1)-M#(I,2)
3322 RO=I
3323 NEXT I
```

As with the upper bound, checks the lower limits.

```
3324 FOR J=2 TO ND+1
3325 IF M#(J,1)<=0# THEN 3331
```

```
3326 A#=M#(J,1)
3327 IF A#=BI# THEN A#=0#
3328 IF A#-B#(J,1)>=MI# THEN 3331
3329 MI#=A#-B#(J,1)
3330 RO=J+ND+1
3331 NEXT J
```

Checks for the violated constraint the greatest geometric distance form the current point. The REM statements are for later nonlinear constraints.

```
3332 FOR K=2 TO MD+1
3333 REM------------------------------------------------------------
3334 REM
3335 REM        SUPPORTING PLANE AND DEEP CUT SUBROUTINES
3336 REM  (if no nonlinear subroutines used, set Z#=1# in line 3345)
3337 REM------------------------------------------------------------
3338 B#=-A#(K,1)
3339 Z#=0#
3340 FOR J=2 TO ND+1
3341 B#=B#+A#(K,J)*B#(J,1)
3342 Z#=Z#+A#(K,J)*A#(K,J)
3343 NEXT J
3344 IF Z#=0# THE 3350
3345 SN=R(K):REM Z#=1#
3346 IF SN=0 THEN SN=-SGN(B#)
3347 IF (B#*CDBL(SN))/CDBL(SQR(Z#))>=MI# THEN 3350
3348 MI#=(B#*CDBL(SN))/CDBL(SQR(Z#))
3349 RO=K+ND+1+ND+1
3350 NEXT K
```

Sets ER=1 and returns to main routine if no constraints are violated.

```
3351 ER=1
3352 IF RO=0 THEN RETURN
```

Transforms the original constraint coefficients into the coefficients of the current simplex tableau.

```
3353 GOSUB 3600:REM TRAN-CON
```

Selects the pivot column for a Gauss-Jordan elimination using a modification of the perturbation method. In practice it is seldom used for the dual simplex method.

```
3354 REM------------------------------------------------------------
3355 REM CN=0
```

```
3356 REM FOR I=2 TO ND+1
3357 REM IF P#(I)<=SM# OR M#(I,1)=BI# THEN 600
3358 REM CN=CN+1
3359 REM C(CN)=I
3360 REM NEXT I
3361 REM CO=C(1)
3362 REM IF CN=1 THEN 810
3363 REM FOR I=1 TO ND+2
3364 REM IF I=ND+2 THEN STOP
3365 REM MI#=BI#
3366 REM FOR J=1 TO CN
3367 REM IF C(J)=0 THEN 690
3368 REM IF MI#>B#(I,C(J))/P#(C(J)) THEN MI#=B#(I,C(J))/P#(C(J))
3369 REM NEXT J
3370 REM C=0
3371 REM FOR J=1 TO CN
3372 REM IF C(J)=0 THEN 780
3373 REM IF B#(I,C(J))/P#(C(J))>MI# THEN 770
3374 REM C=C+1
3375 REM CO=C(J)
3376 REM GOTO 780
3377 REM C(J)=0
3378 REM NEXT J
3379 REM IF C=1 THEN 810
3380 REM NEXT I
3381 REM----------------------------------------------------------------
```

Selects the pivot column for a Gauss-Jordan elimination using the minimum ratio of the objective function row coefficient B#(1,ND+1) to the pivot row coefficient P#(ND+1). Ties are broken by using the largest divisor, then by the smallest column number.

```
3382 MI#=BI#
3383 MA#=0#
3384 CO=0
3385 FOR I=2 TO ND+1
3386 IF P#(I)<=SM# OR M#(I,1)=BI# THEN 3365
3387 IF MI#<B#(1,I)/P#(I) THEN 3365
3388 IF MI#=B#(1,I)/P#(I) AND MA#>P#(I) THEN 3365
3389 MI#=B#(1,I)/P#(I)
3390 MA#=P#(I)
3391 CO=I
3392 NEXT I
```

Sets ER=2 and returns to main routine if no column is found and the problem is infeasible.

```
3393 ER=2
```

```
3394 IF CO=0 THEN RETURN
```

Performs a Gauss-Jordan elimination using the pivot row RO and column CO.

```
3395 GOSUB 3700:REM TRAN-INV
```

Sets ER=3 and return to main routine if the Gauss-Jordan elimination fails because of numerical precision.

```
3396 IF ER=3 THEN RETURN
3397 REM
```

Returns to start next interation

```
3398 GOTO 3314
```

Constraint Tranformation Subroutine -- File TRAN-CON

The constraint transformation subroutine (TRAN-CON) transforms the constraint RO selected in ALGR-SMP subroutine from either the upper and lower bound M#(ND+1,2) array or the A#(MD+1,ND+8) array into an equation in the current tableau's coefficients and then stores the transformed constraint in the pivot row array P#(ND+1). This transformation is simplest to stated as the matrix operation:

$$\left| -a_p \;,\; \bar{a}_p' \right| * \left| \begin{array}{c|c} 1 & \bar{0}' \\ \hline \bar{B}^{-1}\bar{b} & \bar{B}^{-1} \end{array} \right| = \left| -a_p + \bar{a}_p' \bar{B}^{-1}\bar{b},\; \bar{a}_p' \bar{B}^{-1} \right|$$

where the $\bar{a}'$ vector is the selected row of the A#(MD+1,ND+8) array and the inverted augmented $\bar{B}$ matrix is the multiplier.

```
3600 REM            * CONSTRAINT TRANSFORMATION SUBROUTINE *
3601 REM------------------------TRAN-CON-----------------------------
```

Sets the pivot row P#(ND+1) array equal to the transformed upper bound constraint.

If the RO value is less or equal to ND+1, then the selected constraint is an upper bound. The upper bound constraint can be

directly transformed from the basis inverse as seen if the constraint is substituted into the matrix operation defining the transformation.

```
3602 IF RO>ND+1 THEN 3608
3603 P#(1)=B#(RO,1)-M#(RO,2)
3604 FOR I=2 TO ND+1
3605 P#(I)=B#(RO,I)
3606 NEXT I
3607 GOTO 3632
```

Sets the pivot row P#(ND+1) array equal to the transformed lower bound constraint.

If the RO value is greater than ND+1 and less or equal to ND+1+ND+1 then the selected constraint is a lower bound constraint.

```
3608 IF RO>ND+1+ND+1 THEN 3617
3609 J=RO-ND-1
3610 A#=M#(J,1)
3611 IF A#=BI# THEN A#=0#
3612 P#(1)=A#-B#(J,1)
3613 FOR I=2 TO ND+1
3614 P#(I)=-B#(J,I)
3615 NEXT I
3616 GOTO 3632
```

Sets the pivot row P#(ND+1) array equal to the transformed A#(MD+1,ND+8) array row.

If the RO value is greater than ND+1+ND+1 then the selected constraint is one of the original LP problem constraints stored in their original form in the A#(MD+1,ND+8) array. The transformation is as defined earlier except that the order of the matrix operations are chosen to take advantage of a sparse A#(MD+1,ND+8) matrix.

```
3617 K=RO-ND-1-ND-1
3618 P#(1)=-A#(K,1)
3619 FOR I=2 TO ND+1
3620 P#(I)=0#
3621 P#(1)=P#(1)+A#(K,I)*B#(I,1)
3622 NEXT I
3623 SN=R(K)
3624 IF SN=0 THEN SN=-SGN(P#(1))
3625 P#(1)=P#(1)*CDBL(SN)
3626 FOR J=2 TO ND+1
3627 IF A#(K,J)=0# THEN 3631
3628 FOR L=2 TO ND+1
3629 P#(L)=P#(L)+A#(K,J)*B#(J,L)*CDBL(SN)
```

```
3630 NEXT L
3631 NEXT J
3632 RETURN
```

Inversion Subroutine -- File TRAN-INV

The inversion subroutine (TRAN-INV) using the pivot row P#(ND+1) performs a Gauss-Jordan elimination on column CO of the inverted augmented $\overline{B}$ matrix or the B#(ND+1,ND+1+ND) array rows one (1) to ND+1. The pivot row constraint in the original variables is then stored in the augmented $\overline{B}$ matrix or the B#(ND+1,ND+1+ND) array rows ND+2 to ND+1+ND in a row corresponding to the column CO's transpose row position.

```
3700 REM                    *INVERSION SUBROUTINE*
3701 REM-------------------------TRAN-INV-------------------------
```

Prints to the screen the current iteration, objective function value, selected constraint, and pivot column.

```
3702 PRINT IT;B#(1,1);RO;CO
```

Searches for the sequence of operations in the transformation with the fewest mathematic operations; sets the holding array to the current point; and loads into the augmented $\overline{B}$ matrix the selected constraint as defined by RO.

```
3703 X=0
3704 Y=0
3705 FOR I=1 TO ND+1
3706 H#(I)=B#(I,1)
3707 IF RO<=ND+1+ND+1 THEN B#(CO+ND,I)=0#
3708 IF RO>ND+1+ND+1 THEN B#(CO+ND,I)=A#(RO-ND-1-ND-1,I)*CDBL(SN)
3709 IF P#(I)=0# THEN X=X+1
3710 IF B#(I,CO)=0# THEN Y=Y+1
3711 NEXT I
3712 IF RO>ND+1 THEN 3716
3713 B#(CO+ND,1)=M#(RO,2)
3714 B#(CO+ND,RO)=1#
3715 GOTO 3719
3716 IF RO>ND+1+ND+1 THEN 3719
3717 B#(ND+CO,1)=-M#(RO-ND-1,1)
3718 B#(ND+CO,RO-ND-1)=-1#
```

If the number of iterations is equal to an integer multiple of IN then the augmented $\overline{B}$ matrix is completely inversed or:

$$\begin{vmatrix} 1 & \overline{0}' \\ -\overline{b}_{new} & \overline{B}_{new} \end{vmatrix}^{-1} = \begin{vmatrix} 1 & \overline{0}' \\ \overline{B}^{-1}\overline{b} & \overline{B}^{-1} \end{vmatrix}$$

Fortunately, with only a slight lose of precision, only a single column of the inverted augmented $\overline{B}$ matrix has to be updated.

Selects to do a full inversion.

```
3719 IF IT/IN=INT(IT/IN) THEN 3742
```

Pivots on the pivot row if the pivot row has a greater number of zero (0) entries that the CO column of the inverted augmented $\overline{B}$ matrix.

```
3720 L#=P#(CO)
3721 IF X<Y THEN 3730
3722 FOR J=1 TO ND+1
3723 P#=P#(J)
3724 IF J=CO OR P#=0# THEN 3728
3725 FOR I=1 TO ND+1
3726 B#(I,J)=(B#(I,J)*L#-(P#*B#(I,CO)))/L#
3727 NEXT I
3728 NEXT J
3729 GOTO 3737
```

Pivots on the CO column rather than the pivot row if the CO column of the inverted augmented $\overline{B}$ matrix has a greater number of zero (0) entries than the pivot row.

```
3730 FOR I=1 TO ND+1
3731 P#=B#(I,CO)
3732 IF P#=0# THEN 3736
3733 FOR J=1 TO ND+1
3734 IF J<>CO THEN B#(I,J)=(B#(I,J)*L#-(P#*P#(J)))/L#
3735 NEXT J
3736 NEXT I
```

Divides the CO column by minus one (-1) if the pivot element is a negative value.

In the primal simplex algorithm for which this inversion is also used, the pivot element has a negative coefficient. In this case the transformation has a extra division by minus one (-1) of the CO column of the inverted augmented $\overline{B}$ matrix.

```
3737 IF SGN(L#)=1 THEN RETURN
3738 FOR I=1 TO ND+1
3739 B#(I,CO)=-B#(I,CO)
3740 NEXT I
3741 RETURN
3742 GOSUB 3900:REM TRAN-RIV
3743 RETURN
```

Reinversion Subroutine -- File TRAN-RIV

The reinversion subroutine[39] completely inverts the augmented $\overline{B}$ matrix or in matrix notation:

$$\begin{vmatrix} 1 & \overline{0}' \\ -\overline{b}_{new} & \overline{B}_{new} \end{vmatrix}^{-1} = \begin{vmatrix} 1 & \overline{0}' \\ \overline{B}^{-1}\overline{b} & \overline{B}^{-1} \end{vmatrix}$$

This inversion is done using a Gauss-Jordan elimination starting with:

$$\left|\begin{array}{cc|cc} 1 & \overline{0}' & 1 & \overline{0}' \\ -\overline{b}_{new} & \overline{B}_{new} & \overline{0} & \overline{I} \end{array}\right|$$

where the augmented $\overline{B}$ matrix is copied into the T#(ND+1,ND+1) array and the identity matrix is loaded into the B#(ND+1,ND+1) array. By the transformation, the new matrix is:

$$\left|\begin{array}{cc|cc} 1 & \overline{0}' & 1 & \overline{0}' \\ \overline{0} & \overline{I} & \overline{B}^{-1}\overline{b} & \overline{B}^{-1} \end{array}\right|$$

where the B#(ND+1,ND+1) array is the reinverted augmented B matrix.

```
3900 REM                    * REINVERSION SUBROUTINE *
3901 REM-------------------------TRAN-RIV-------------------------------
```

Initializes the T#(ND+1,ND+1) array as the augmented $\overline{B}$ matrix and the B#(ND+1,ND+1+ND) array rows one (1) through ND+1 as the identity matrix.

```
3902 FOR I=2 TO ND+1
3903 FOR J=2 TO ND+1
3904 T#(I,J)=B#(I+ND,J)
3905 B#(I,J)=0
3906 NEXT J
3907 B#(I,I)=1
3908 NEXT I
```

Inverts the $\overline{B}$ matrix.

To allow for eliminating parts of the computer code in later algorithms, the actual inversion is done on the $\overline{B}$ matrix and then the first column and row of the augmented $\overline{B}$ matrix are added as a separate operation. This allows the objective function transformation to be dropped from the code when it is performed in other subroutines.

The inversion routine works by selecting in sequence each row, from two (2) to ND+1, as pivot rows; finding the largest element of the selected pivot row in a column of number equal to or larger than the current pivot row number as the pivot element; reordering the columns so that the selected pivot element column is now in the corresponding pivot row column number; and performing a Gauss-Jordan pivot on each row and column selected.

```
3909 FOR I=2 TO ND+1
3910 P#=T#(I,I)
3911 K=I
3912 IF I=ND+1 THEN 3918
3913 FOR J=I+1 TO ND+1
3914 IF ABS(T#(I,J))<ABS(P#) THEN 3917
3915 P#=T#(I,J)
3916 K=J
3917 NEXT J
3918 IF P#<>0# THEN 3922
```

If a pivot cannot be found the matrix has failed to reinvert because of numerical precision.

```
3919 PRINT "MATRIX IS SINGULAR"
3920 ER=3
3921 RETURN
3922 IF I=K THEN 3927
3923 FOR J=2 TO ND+1
3924 SWAP B#(J,I),B#(J,K)
3925 SWAP T#(J,I),T#(J,K)
3926 NEXT J
3927 FOR J=2 TO ND+1
3928 P#(J)=T#(I,J)/P#
3929 NEXT J
```

```
3930 FOR J=2 TO ND+1
3931 L#=P#(J)
3932 IF L#=0# OR J=I THEN 3937
3933 FOR K=2 TO ND+1
3934 B#(K,J)=B#(K,J)-(L#*B#(K,I))
3935 T#(K,J)=T#(K,J)-(L#*T#(K,I))
3936 NEXT K
3937 NEXT J
3938 FOR J=2 TO ND+1
3939 B#(J,I)=B#(J,I)/P#
3940 T#(J,I)=T#(J,I)/P#
3941 NEXT J
3942 NEXT I
```

Transforms the first column of the reinverted augmented $\overline{B}$ matrix.

```
3943 FOR I=2 TO ND+1
3944 B#(I,1)=0#
3945 FOR J=2 TO ND+1
3946 B#(I,1)=B#(I,1)+B#(I,J)*B#(J+ND,1)
3947 NEXT J
3948 NEXT I
```

Transforms the objective row of the reinverted augmented $\overline{B}$ matrix.

```
3949 B#(1,1)=0#
3950 FOR I=2 TO ND+1
3951 B#(1,I)=0#
3952 B#(1,1)=B#(1,1)+A#(1,I)*B#(I,1)
3953 NEXT I
3954 FOR I=2 TO ND+1
3955 IF A#(1,I)=0# THEN 3959
3956 FOR J=2 TO ND+1
3957 B#(1,J)=B#(1,J)+A#(1,I)*B#(I,J)
3958 NEXT J
3959 NEXT I
3960 RETURN
```

Program Table of Contents

Table 2 can be used to reconstruct the above computer code from the computer disk and to organize subroutines from previous program listings. Since BASIC code is dependent on program routine line numbers for its subroutine branching, the statement numbers must be maintained

Table 2. Dual simplex BASIC program table of contents

as listed above.

Dual Simplex Method:
Solutions to Example 4 Minimum Project Cost Problem

To expand the example 2 ten (10) activity network model into a linear programming problem requires the definition of several variables not used in the CPM networks.

- c_{ij} - Incremental cost to reduce duration of activity ij
- R_{ij} - Increments of reduction of duration of activity ij
- T_i - Node time for node i
- u_{ij} - Upper limit for duration of activity ij if less than durij
- l_{ij} - Lower limit for duration of activity ij
- n - Maximum node number
- Dur. - Project total duration incrememts
- dur_{ij} - Normal or minimum cost duration increments for activity ij
- fix. - Fixed cost per increment of total project duration

Formulating the problem in algebraic notation the cost function becomes:

$$\text{minimize} \sum_{\text{all } ij} c_{ij}*R_{ij}+fix.*(T_n-T_0)$$

once the constant cost terms are dropped from the equation. The network relationships become the constraints:

$$\text{subject to: } T_j-T_i-(dur_{ij}-R_{ij})>=0 \text{ for all } ij$$

$$T_n-T_0<=Dur.$$

for which the range of the variables are restricted to:

$$dur_{ij}-l_{ij}> R_{ij} >dur_{ij}-u_{ij} \quad \text{for all } ij$$

$$T_i,R_{ij},Dur.>=0 \quad \text{for all } i,ij$$

If this notation is used to label a network, the result is figure 26.

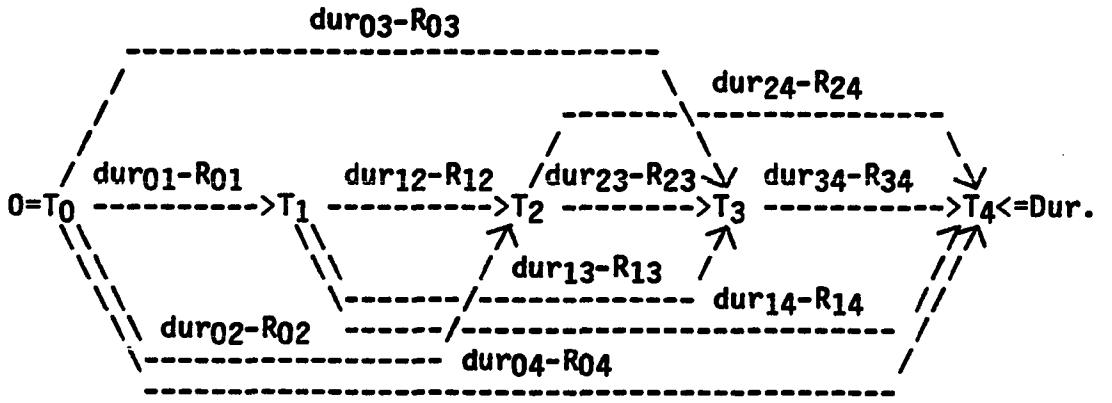


Figure 26. Minimum project cost problem example 4 arrow diagram

To provide data for the example, the arbitrary set of data in table 3 is used. The data are not intended to represent any real or estimated project. The example problem is only to demonstrate the mechanics and performance characteristic of the algorithms.

Table 3. Activity duration and incremental cost table

Activity	Min. dur.	Max. dur.	Dur. var.	Var. range	Inc. cost
Activity #1	1	10	R_{03} or R5	0-9	5
Activity #2	1	20	R_{24} or R6	0-19	10
Activity #3	1	30	R_{01} or R7	0-29	15
Activity #4	1	40	R_{12} or R8	0-39	20
Activity #5	1	50	R_{23} or R9	0-49	25
Activity #6	1	60	R_{34} or R10	0-59	30
Activity #7	1	70	R_{13} or R11	0-69	35
Activity #8	1	80	R_{02} or R12	0-79	40
Activity #9	1	90	R_{14} or R13	0-89	45
Activity #10	1	100	R_{04} or R14	0-99	50

Rewriting the ten (10) activity minimum project cost network in the form of a LP problem results in:

minimize 5*R5+10*R6+15*R7+20*R8+25*R9+30*R10
+35*R11+40*R12+45*R13+50*R14

subject to: T3-(10-R5)-0>=0
T4-(20-R6)-T2>=0
T1-(30-R7)-0>=0
T2-(40-R8)-T1>=0
T3-(50-R9)-T2>=0
T4-(60-R10)-T3>=0
T3-(70-R11)-T1>=0
T2-(80-R12)-0>=0
T4-(90-R13)-T1>=0
T4-(100-T14)-0>=0
T4<=Dur.
R5<=9
R6<=19
R7<=29
R8<=39
R9<=49
R10<=59
R11<=69
R12<=79
R13<=89
R14<=99
T1,T2,.....R14>=0

or as a arrow diagram:

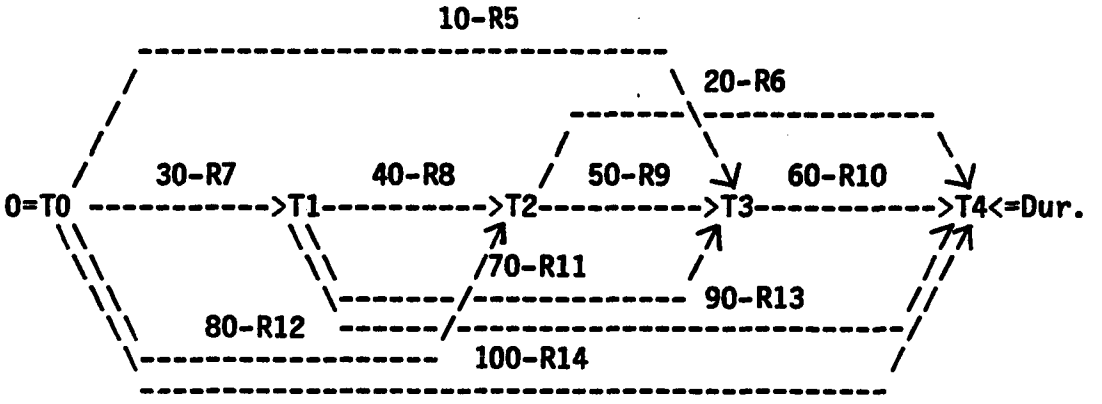


Figure 27. Minimum project cost problem example 4 arrow diagram

Solutions to Minimum Project Cost Problem

To establish a simple bench mark for the execution time of the dual simplex method, the example LP problem was run with predetermined project durations ranging from one hundred sixty (160) increments to four (4) increments. The durations were chosen to correspond to the breaks in the piece wise linear cost curve solution of the out-of-kilter method described in the following chapter. Fixed costs were dropped from the problem because a fixed project duration was used, making the fixed cost, fix.*(T_n-T_0), a constant for each computer run.

The results of the computer runs are displayed in table 4 which lists for each of the ten (10) runs the fixed project duration, the number of iterations required to reach the solution, the seconds in processing time required to reach the solution, the value of the objective function at the solution, and the value of each variable.

These computer runs were executed on a Panasonic Sr. Partner computer with the program code compiled using an IBM BASIC Compiler.

Table 4. Dual simplex method solutions to example 4 minimum project cost problem with linear activity cost functions

Dur.	160	120	101	100	91
Itr.	7	8	10	9	11
Sec.	5	6	7	6	8
Obj.	750	1950	2710	2755	3700
T1	30	30	11	10	1
T2	80	80	80	80	71
T3	100	100	81	81	72
T4	160	120	101	100	91
R5	0	0	0	0	0
R6	0	0	0	0	0
R7	0	0	19	20	29
R8	0	0	0	0	0
R9	30	30	49	49	49
R10	0	40	40	41	41
R11	0	0	0	0	0
R12	0	0	0	0	9
R13	0	0	0	0	0
R14	0	0	0	0	9

Table 4. Continued

Dur.	90	72	43	11	4
Itr.	12	13	13	14	15
Sec.	8	9	9	10	10
Obj.	3835	6265	11195	17275	18640
T1	1	1	1	1	1
T2	70	70	41	9	2
T3	71	71	42	10	3
T4	90	72	43	11	4
R5	0	0	0	0	7
R6	0	18	18	18	18
R7	29	29	29	29	29
R8	0	0	0	32	39
R9	49	49	49	49	49
R10	41	59	59	59	59
R11	0	0	29	61	68
R12	10	10	39	71	78
R13	1	19	48	80	87
R14	10	28	57	89	96

MINIMUM PROJECT COST CURVE PROBLEM WITH LINEAR COST FUNCTIONS

The project duration variable Dur. was fixed at predetermined points on the CPM time scale for the dual simplex solutions to the minimum project cost problem with linear cost functions. If the process were continued for each time scale point between the minimum possible project duration and the maximum project duration, a project "cost curve" of optimal costs could be constructed for every feasible duration.

Fortunately, because of the linear characteristics of the cost curve, the number of points needed to define a cost curve are limited to the points where the linear segments of the piece wise cost curve "break".

In example 4, the minimum project cost was calculated at known break point values (Dur.) from four (4) to one hundred sixty (160) work increments. A total of one hundred twelve (112) dual simplex iterations for a total time of sixty-eight (68) seconds are needed to calculate enough points to define a cost curve. Using a method derived by Fulkerson[11], the same curve can be calculated in two (2) seconds.

Fulkerson's Interpretation of Cost Problem with Linear Costs

The Fulkerson's cost problem is solved using a network flow model and a unique application of the Kuhn-Tucker conditions[40] called the "out-of-kilter" method. With this approach, greater computer processing speed and activity capacity can be attained than with the dual simplex method.

Fulkerson's network flow model is based on the dual problem of the minimum project cost problem with linear costs functions. But before continuing, the LP problem must be changed slightly from the LP defined in the earlier chapter. Defining:

c_{ij} - Incremental cost benefit to increase duration of activity ij
f_{ij} - Fixed cost not related to duration of activity ij
Dur_{ij} - Duration of activity ij (as apposed to durij-Rij)
T_i - Node time for node i
l_{ij} - Minimum duration for activity ij
u_{ij} - Normal duration for activity ij
n - Maximum node number
Dur. - Project total duration

so that the cost of an activity is:

$$\text{Cost of activity ij} = f_{ij} - c_{ij} * Dur_{ij}$$

and the total project cost is:

$$\text{Project cost} = \sum_{\text{all ij}} (f_{ij} - c_{ij} * Dur_{ij})$$

By dropping the fixed costs from the project cost equation, the linear costs LP problem can be written as the following equations.

$$\text{maximize} \sum_{\text{all ij}} c_{ij} * Dur_{ij}$$

				dual variables
subject to:	$Dur_{ij} + T_i - T_j <= 0$	all ij		$<F_{ij}>$
	$T_n - T_0 <= Dur.$			$< V >$
	$Dur_{ij} <= u_{ij}$	all ij		$<G_{ij}>$
	$-Dur_{ij} <= -l_{ij}$	all ij		$<H_{ij}>$

Dur_{ij}, T_i, T_j free

This form of the LP problem demonstrates the one assumption that Fulkerson has made. All the node times and duration variables are assumed to be free variables. Although this is not true, only the

positive solutions are considered.

Using the LP above, the dual of the cost problem can be written as:

$$\text{minimize} \quad \text{Dur.}*V + \sum_{\text{all } ij} u_{ij}*G_{ij} - \sum_{\text{all } ij} l_{ij}*H_{ij}$$

$$\text{subject to:} \quad F_{ij}+G_{ij}-H_{ij}=c_{ij} \quad \text{all } ij$$

$$\sum_{\text{all } j} (F_{ij}-F_{jl}) = \begin{cases} V, j=0 \\ 0, j=1,n-1 \\ -V, j=n \end{cases}$$

$$F_{ij}, G_{ij}, H_{ij} \geq 0$$

where the first set of constraints can be rewritten in the form:

$$G_{ij} = c_{ij}-F_{ij}+H_{ij} \quad \text{all } ij$$

or:

$$H_{ij} = -c_{ij}+F_{ij}+G_{ij} \quad \text{all } ij$$

and the second set of constraints are the node conservation constraints of a flow network with flows F_{ij}.

The first set of constraints, along with the minimum requirement of the objective function, means that at an optimal solution either $G_{ij}=0$ or $H_{ij}=0$, so:

$$G_{ij} = \max (0, c_{ij}-F_{ij})$$

and:

$$H_{ij} = \max (0, -c_{ij}+F_{ij})$$

By substitution the the objective function is then:

$$\text{minimize} \quad \text{Dur.}*V + \sum_{\text{all } ij} u_{ij}*\max (0, c_{ij}-F_{ij}) - \sum_{\text{all } ij} l_{ij}*\max (0, -c_{ij}+F_{ij})$$

and the new objective function is now piece wise linear in the flow variable F_{ij} for a given Dur. and V.

The objective can be drawn for activity ij as figure 28 where the axes are the activity's contribution to the objective versus the

flow variable F_{ij}. When the variable F_{ij} is greater than or equal to

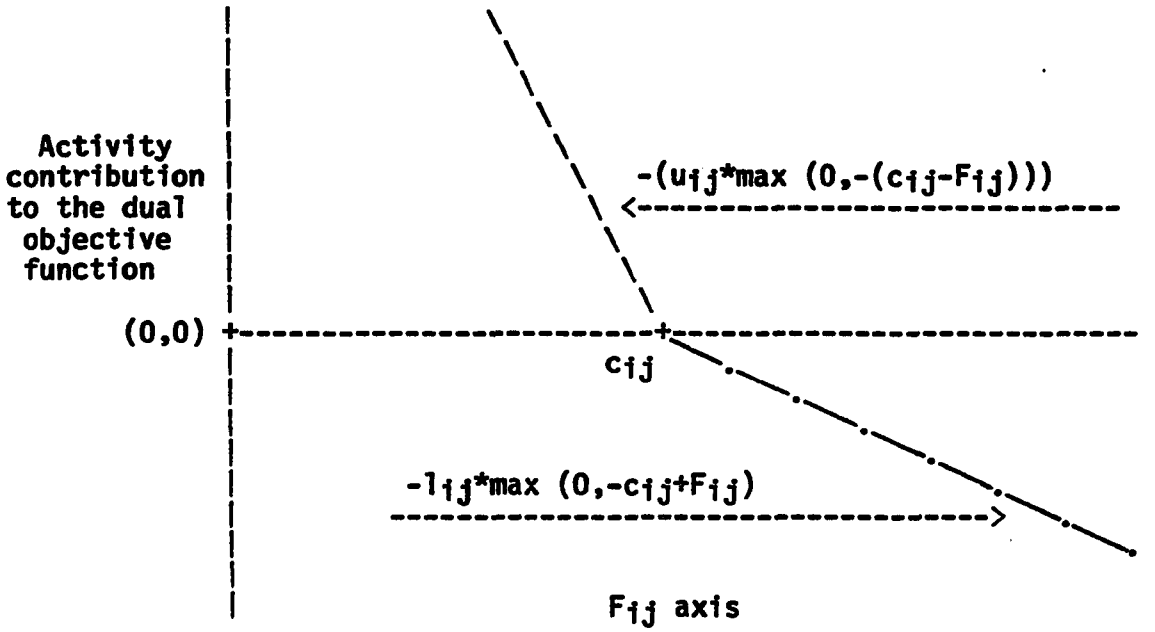


Figure 28. Out-of-kilter method activity piece wise objective function

zero (0) and less than c_{ij}, the incremental cost for activity ij, then the curve has a negative slope of $-u_{ij}$. If the variable F_{ij} is greater than or equal to c_{ij}, then the curve has a negative slope of $-l_{ij}$.

If F_{ij} is split into two (2) components F_{ij1} and F_{ij2} where:

$$0<=F_{ij1}<=c_{ij} \quad \text{all ij}$$

$$0<=F_{ij2}<=\text{inf} \quad \text{all ij}$$

then the dual problem can be rewritten as:

$$\text{minimize Dur.*V} - \sum_{\text{all ij}} u_{ij}*F_{ij1} - \sum_{\text{all ij}} l_{ij}*F_{ij2}$$

$$\text{subject to: } \sum_{\text{all ijk}} F_{ijk}-F_{jlk} = \begin{cases} V, j=0 \\ 0, j=1,n-1 \\ -V, j=n \end{cases}$$

which is a flow problem for a given flow V and duration Dur. where each

activity of the network has two (2) associated flows, F_{ij1},F_{ij2}, and the cost for the first flow is u_{ij} and for the second flow is l_{ij}.

Figure 29 is the ten (10) activity example with flow variables F_{ij1} and F_{ij2}.

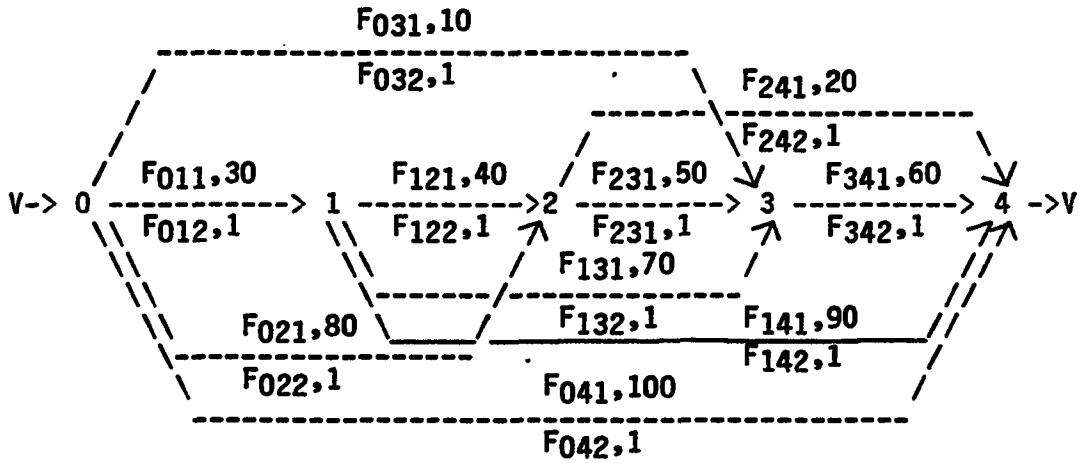


Figure 29. Out-of-kilter method dual flow diagram example 6

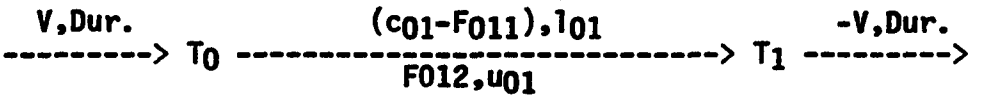
Theory of Out-of-kilter Method

The flow model interpretation of the dual of the LP linear cost problem, given that the duration and flow are fixed, provides a means for finding the optimal points on the cost curve. To demonstrate the method and to avoid the confusion of notation, a one (1) activity network will be used for example 5.

$$T_0 \xrightarrow[\text{Dur.} <= u_{01}]{\text{Dur.} >= l_{01}} T_1$$

Dur.- Project total duration or in this case activity duration
l_{01} - Minimum activity 01 duration
u_{01} - Normal activity 01 duration
c_{01} - Incremental cost to reduce activity 01 duration
T_i - Node time for node i

If the dual of the network is written in the notation of the dual problem, then the flow model is a follows.



$-F_{011} <= c_{01}$

V	- Total network flow
Dur.	- Incremental cost for flow V
$(c_{01}-F_{011})$	- First level of flow in arc 01
F_{012}	- Second level of flow in arc 01
l_{01}	- Incremental cost for flow F_{012}
u_{01}	- Incremental cost for flow $(c_{01}-F_{011})$
c_{01}	- Maximum flow for $-F_{011}$
inf	- infinity

The activity's contribution to the piece wise objective function for the example 5 is then as shown in figure 30.

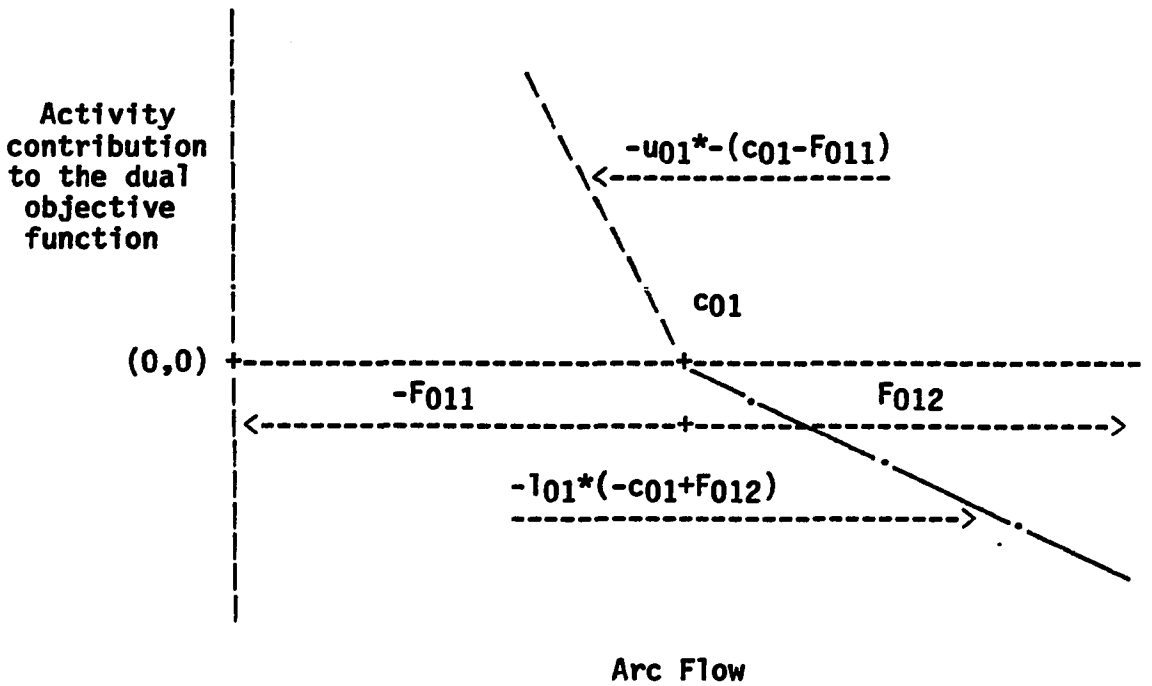


Figure 30. Out-of-kilter method objective function example 5

The objective of the dual problem is now written as:

$$\text{minimize} \quad Dur.*V+u_{01}*(c_{01}-F_{011})-l_{01}*(-c_{01}+F_{012})$$

or

$$\text{minimize} \quad Dur.*V+u_{01}*(-F_{011})-l_{01}*F_{012}$$

and the constraints as:

$$F_{011}<=c_{01}$$

$$V-(c_{01}-F_{011})-(-c_{01}+F_{012})=0$$

$$(c_{01}-F_{011})+(-c_{01}+F_{012})-V=0$$

$$F_{011},F_{012},V>=0$$

or

$$F_{011}<=c_{01}$$

$$V-(-F_{011})-F_{012}=0$$

$$(-F_{011})+F_{012}-V=0$$

$$F_{011},F_{012},V>=0$$

for the constraints.

The equality constraints of the above problem make it ideal for the use of the Lagrangian function to find the optimal solution. By writing the above problem as a Lagrangian function:

$$W(V,F_{011},F_{012},S,T_0,T_1)=$$

$$Dur.*V-u_{01}*F_{011}-l_{01}*F_{012}+S*(-c_{01}+F_{011})+T_0*(V+F_{011}-F_{012})+T_1*(-F_{011}+F_{012}-V)$$

the problem can now be minimized by using the Kuhn-Tucker[40] conditions.

Kuhn-Tucker Conditions

The Kuhn-Tucker theory[40] provides a set of conditions sufficient for the optimal solution to the Lagrangian function assuming that the function is differentiable. For the Lagrangian of the example 5 one (1)

activity network, the Kuhn-Tucker conditions are:

$$\frac{\&W(V,F_{011},F_{012},S,T_0,T_1)}{\&V}=Dur.+T_0-T_1>=0 \quad \text{or} \quad T_1-T_0<=Dur.$$

$$\frac{\&W(V,F_{011},F_{012},S,T_0,T_1)}{\&F_{011}}=u_{01}+S+T_0-T_1>=0 \quad \text{or} \quad T_1-T_0-S<=u_{01}$$

$$\frac{\&W(V,F_{011},F_{012},S,T_0,T_1)}{\&F_{012}}=-l_{01}-T_0+T_1>=0 \quad \text{or} \quad T_1-T_0>=l_{01}$$

$$\frac{\&W(V,F_{011},F_{012},S,T_0,T_1)}{\&S}=-c_{01}-F_{011} \quad \text{or} \quad (-F_{011})<=c_{01}$$

$$\frac{\&W(V,F_{011},F_{012},S,T_0,T_1)}{\&T_0}=V-(-F_{011})-F_{012}=0$$

$$\frac{\&W(V,F_{011},F_{012},S,T_0,T_1)}{\&T_1}=(-F_{011})+F_{012}-V=0$$

$$V*\frac{\&W(V,F_{011},F_{012},S,T_0,T_1)}{\&V}=V*(Dur.+T_0-T_1)=0$$

$$F_{011}*\frac{\&W(V,F_{011},F_{012},S,T_0,T_1)}{\&F_{011}}=F_{011}*(u_{01}+S+T_0-T_1)=0$$

$$F_{012}*\frac{\&W(V,F_{011},F_{012},S,T_0,T_1)}{\&F_{012}}=F_{012}*(-l_{01}-T_0+T_1)=0$$

$$S*\frac{\&W(V,F_{011},F_{012},S,T_0,T_1)}{\&S}=S*(-c_{01}+F_{011})=0$$

$$V,F_{011},F_{012},S>=0 \qquad T_0,T_1=\text{free}$$

These conditions are also the network node flow conditions:

$$V-(-F_{011})-F_{012}=0$$

$$(-F_{011})+F_{012}-V=0$$

and the critical path restrictions:

$$T_1-T_0<=Dur.$$

$$T_1-T_0>=l_{01}$$

$$T_1-T_0<=u_{01}+S$$

and what are called the "out-of-kilter" conditions:

$$S*(-c_{01}+F_{011})=0$$

$$(-F_{011})*(u_{01}+S+T_0-T_1)=0$$

$$F_{012}*(-l_{01}-T_0+T_1)=0$$

The above conditions can be seen more easily in figure 31 as a

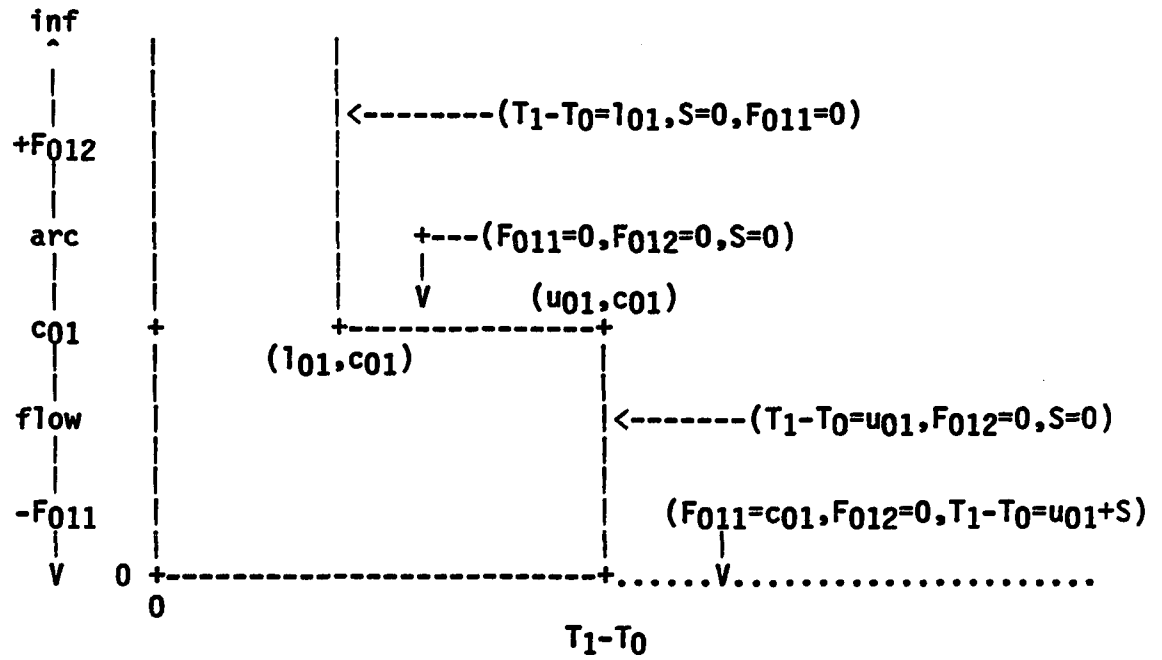


Figure 31. Out-of-kilter method kilter line example 5

"kilter" line. For each activity, a kilter line can be drawn on a set of axes consisting of a flow network axis and a CPM network axis. If a combination of CPM network times T_1-T_0 and flow network flows F_{011}, F_{012} can be found such that the points they define lie on the kilter lines, then the solution satisfies the Kuhn-Tucker conditions and is optimal.

To find a cost curve, a number of points which satisfy the kilter lines must be found which correspond to different project durations Dur.

and network flows V. The search for optimal points starts at the maximum project duration. At this project duration, all activities are at their maximum duration and the optimal solution is known. For example 5, the maximum duration solution is:

$$\text{Dur.}=u_{01}$$

$$T_0=0$$

$$T_1=u_{01}$$

$$F_{011}=0$$

$$F_{012}=0$$

$$V=0$$

$$S=0$$

or in figure 32 a point at (1) on the kilter line.

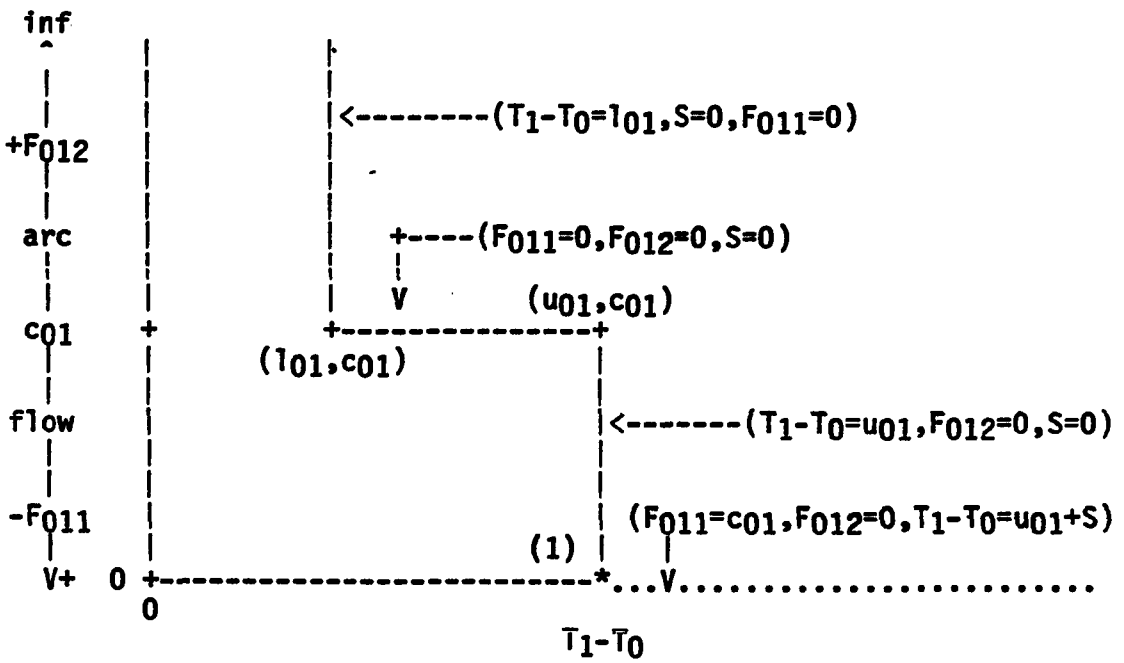


Figure 32. Out-of-kilter method kilter line example 5, iteration one

Staying on the kilter line, the flow (c_{01}-F_{011}) can be increased to c_{01} while still satisfying all flow conservation constraints in the flow net. This condition is called a "breakthrough". (If the network was larger and consisted of more than one (1) activity, the maximum flow that could be passed throught the flow network without violating the kilter lines and without changing the CPM network times is used.)

This now leaves V equal to c_{01}, which is the unit cost for the next reduction in the CPM network's duration on the kilter line. Figure 33 moves the point to (2) where the next move along the kilter line requires a change in the CPM network times T_1-T_0.

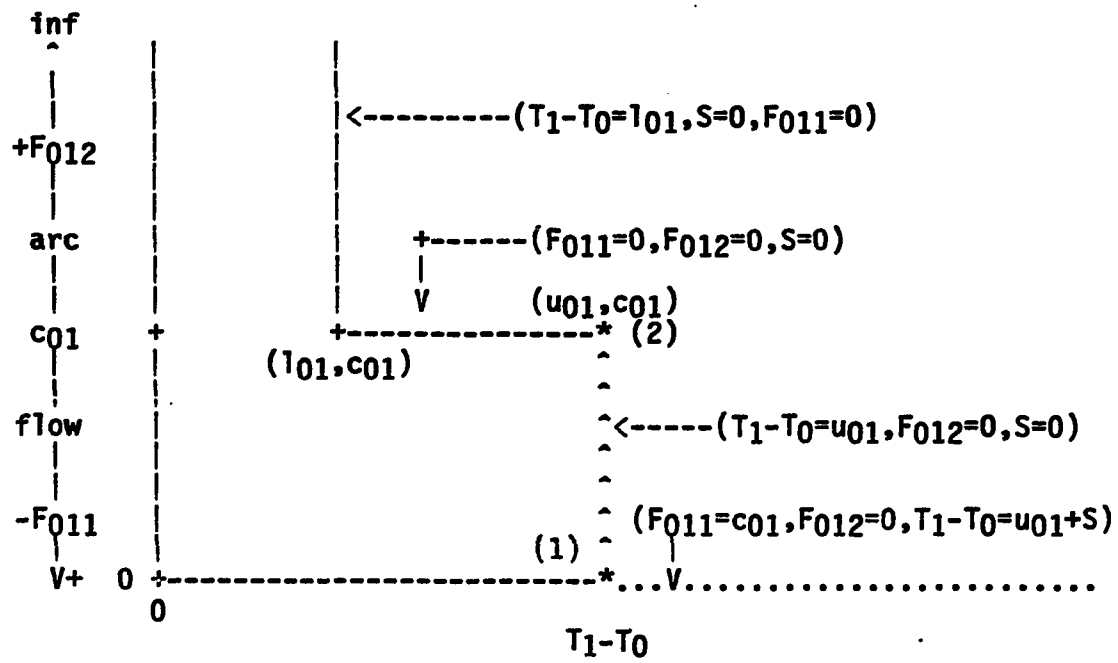


Figure 33. Out-of-kilter method kilter line example 5, iteration one

To move the point from (2) the duration of the CPM network, T_1-T_0, must be changed for the point to stay on the kilter line. The flow V is

the unit cost for this reduction in duration so the cost of the move is $V*(u_{01}-l_{01})$. By moving to point (3) in figure 34 the duration of the CPM

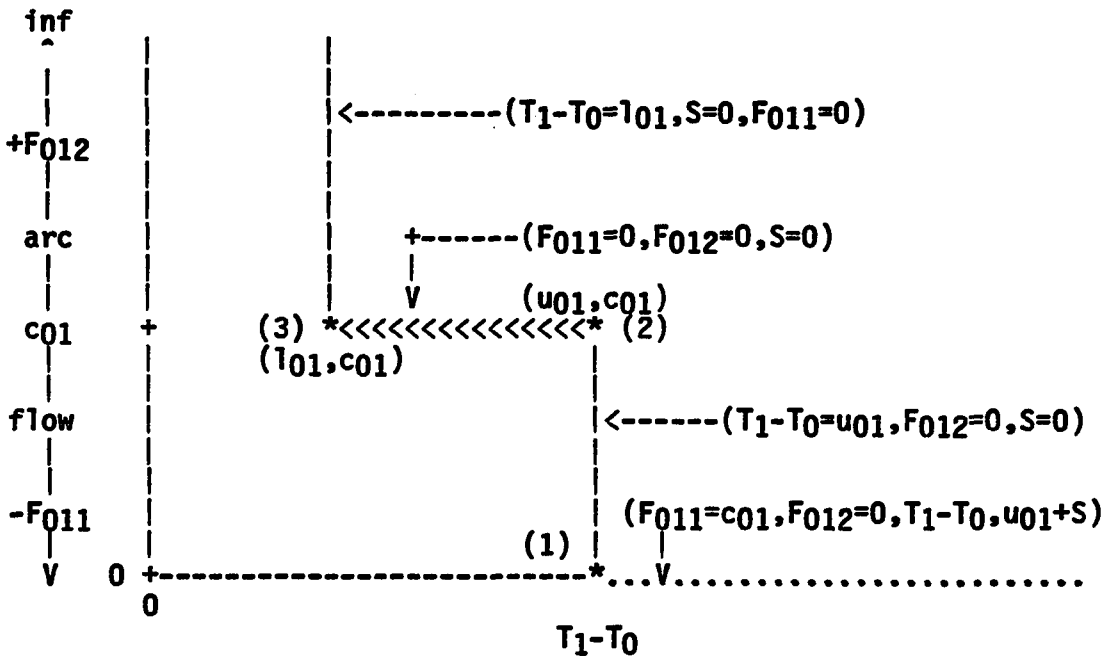


Figure 34. Out-of-kilter method kilter line example 5, iteration one

network Dur. is reduced to l_{01}. If the network were larger and consisted of more than one (1) activity, the maximum reduction in the CPM network's duration without violating the kilter lines and without changing the flow network flows is used.

To move the point from (3) as in figure 35 requires an increase in flow. Breakthrough requires finding the maximum flow that can pass through the flow net without changing the network times; for this move to reach a limit, the point would have to move to an infinite flow. This breakthrough would make any further reduction in CPM network duration an infinite cost, so the algorithm is stopped.

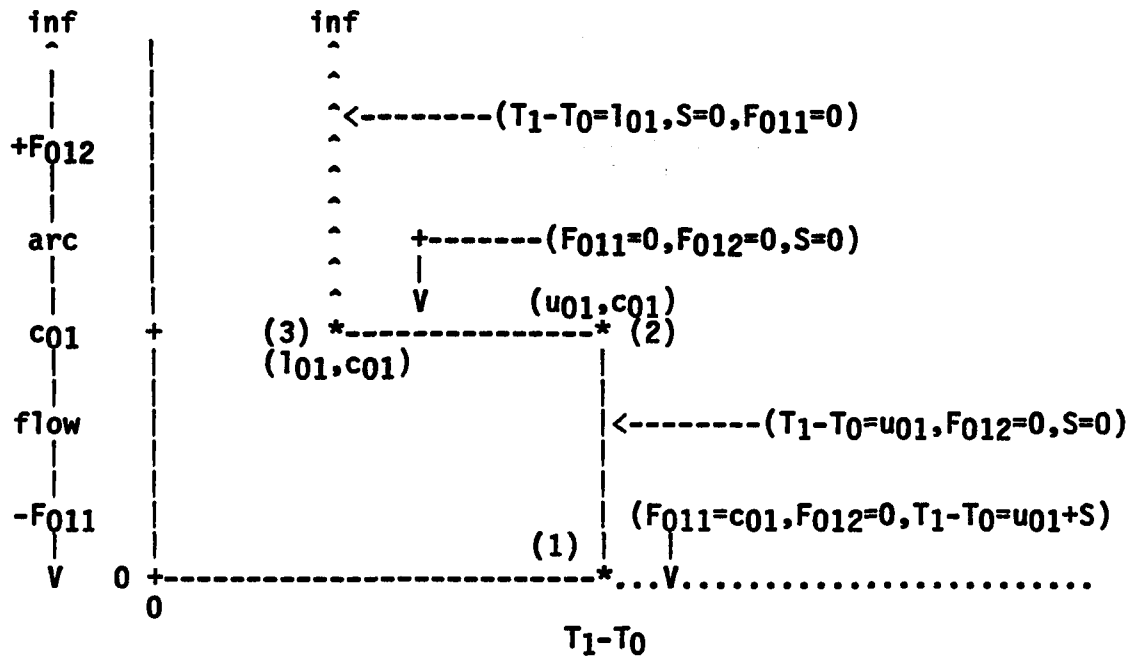


Figure 35. Out-of-kilter method kilter line example 5, iteration two

To translate the results into a cost curve, the first point on the curve is the maximum duration of u_{01} which has a cost of zero (0). The next point is then at duration l_{ij} with a cost of $V*(u_{ij}-l_{ij})$.

Out-of-kilter Method BASIC Code

The out-of-kilter method BASIC code presented here is based on an algorithm described by Salah Elmaghraby[41].

All programs listed in the text consist of a main calling routine and a series of input, output, and processing subroutines. These routines are listed in the text as well as on a copy of a computer disk which is compatible with IBM micro-computers.

Out-of-kilter Main Routine -- File MAIN-KLT

The out-of-kilter main calling routine (MAIN-KLT) dimensions four

(4) arrays; writes the options menu to the screen as shown in figure 36;

```
                    OUT-OF-KILTER METHOD

              NUMBER OF ACTIVITIES          10
              NUMBER OF NODES                5

                    M-RETURN TO MENU

                A-ACTIVITIES
                U-EXECUTE ALGORITHM
                R-REPORT
                S-SAVE  F-FETCH


          OPTION ?
```

Figure 36. Out-of-kilter method main menu screen

calls the utility subroutines UTIL-OPT, UTIL-ERS, UTIL-CON, UTIL-CHX; calls the data input and output routines INPT-ACT and REPT-CRV; calls and times the processing algorithm ALGR-KLT; and saves and fetches the input data to disk file "DATA".

```
1 REM                      * OUT-OF-KILTER METHOD *

2 REM-----------------------------MAIN-KLT-------------------------------
3 REM      AR - NUMBER OF ACTIVITIES OR ARCS
4 REM      BI - MACHINE INFINITE
5 REM      ND - NUMBER OF NODES
6 REM      SM - MACHINE ZERO
7 REM      A(AR,5) - ACTIVITY FILE (I,J,L,U,COST)
8 REM      B(AR,7) - WORK FILE (I,J,L,U,COST,F(i,j,1),F(i,j,2))
9 REM      N(ND,3) - NODE LABLES (Q,I,TE)
10 REM     C#(CYC,2) - COST CURVE (DUR,COST)
11 REM---------------------------------------------------------------
```

Sets AR to the default number of arcs and activities in the network and ND to the default number of nodes in the network. Sets CYC to the maximum number of points on the cost curve. Sets BI# to machine infinite for double precision variables and SM# to machine zero.

```
12 AR=0
13 ND=0
```

```
14 CYC=100
15 BI#=1E+10
16 SM#=1E-10
```

Prompts and reads from the keyboard the number of activities in the network and the number of nodes.

The network node numbering scheme must start at zero (0) and be consecutively integers.

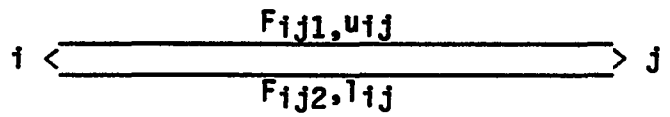
```
17 CLS
18 LOCATE 1,9:PRINT "OUT-OF-KILTER METHOD"
19 LOCATE 3,1:PRINT "NUMBER OF ACTIVITIES":LOCATE 3,31:INPUT   "",L$
20 GOSUB 1870:REM UTIL-CHX
21 IF Z#<>BI# THEN AR=Z#
22 LOCATE 3,30:PRINT AR,"    ":LOCATE 4,1:PRINT "NUMBER OF NODES":LOCATE
   4,31:INPUT "",L$
23 GOSUB 1870:REM UTIL-CHX
24 IF Z#<>BI# THEN ND=Z#
25 LOCATE 4,30:PRINT ND,"    "
```

Dimensions the activity array A(AR,5), the work space array B(AR,7), the node array N(AR,3), and the cost curve array C#(CYC,2).

The activity array elements are:

A(activity number,1) - Start node, i
A(activity number,2) - Start node, j
A(activity number,3) - Minimum duration
A(activity number,4) - Maximum duration
A(activity number,5) - Incremental cost

The work file and the node file elements defined in the notation of



$F_{ij1} <= c_{ij}$

$F_{ij2} <= inf$

the CPM network and the flow network are:

B(activity number,1) - Start node i
B(activity number,2) - Finish node j
B(activity number,3) - l_{ij} minimum duration
B(activity number,4) - u_{ij} maximum duration

```
B(activity number,5)  - cij incremental cost
B(activity number,6)  - Fij1 flow
B(activity number,7)  - Fij2 flow
N(i,1)                - Node flow at node i
N(i,2)                - Node from which node i flow came
N(i,3)                - Ti time of occurrence of node i
```

and the cost curve elements are:

```
C#(curve increment,1) - Project duration at curve break
C#(curve increment,2) - Project cost at curve break
```

```
26 DIM A(AR,5)
27 DIM B(AR,7)
28 DIM N(ND,3)
29 DIM C#(CYC,2)
```

Prints the option menu to the screen; calls the option line routine UTIL-OPT; and pauses for the entry of "M", "O", "A", "C", "B", "U", "R", "N", "S", "F" for the option variable L$.

```
30 LOCATE 8,10:PRINT          "M-RETURN TO MENU"
31 LOCATE 10,5:PRINT       "A-ACTIVITIES"
32 LOCATE 11,5:PRINT       "U-EXECUTE ALGORITHM"
33 LOCATE 12,5:PRINT       "R-REPORT"
34 LOCATE 13,5:PRINT       "S-SAVE  F-FETCH"
35 GOSUB 1800:REM UTIL-OPT
36 LOCATE 21,8:INPUT "",L$
```

Calls either the activity input subroutine INPT-ACT, the processing subroutine ALGR-KLT, or the report subroutine REPT-CRV based on the option variable L$.

```
37 CLS
38 IF L$<>"A" THEN 41
39 GOSUB 1100:REM INPT-ACT
40 GOTO 37
41 IF L$<>"U" THEN 47
42 TM=3600*VAL(MID$(TIME$,1,2))+60*VAL(MID$(TIME$,4,2))+VAL(MID$(TIME$,7
   ,2))
43 GOSUB 3300:REM ALGR-KLT
44 TM=3600*VAL(MID$(TIME$,1,2))+60*VAL(MID$(TIME$,4,2))+VAL(MID$(TIME$,7
   ,2))-TM
45 CLS
46 GOTO 30
47 IF L$<>"R" THEN 50
48 GOSUB 2100:REM REPT-CRV
49 GOTO 37
```

Saves the content of AR, ND, A(AR,5) to disk file "DATA" as an ASCI file if option "S" is selected.

```
50 IF L$<>"S" THEN 60
51 OPEN "O",#1,"DATA"
52 PRINT #1,STR$(AR)
53 PRINT #1,STR$(ND)
54 FOR I=1 TO AR
55 FOR J=1 TO 5
56 PRINT #1,STR$(A(I,J))
57 NEXT J
58 NEXT I
59 CLOSE #1
```

Loads to AR, ND, A(AR,5) the disk file "DATA" if option "F" is selected.

```
60 IF L$<>"F" THEN 74
61 OPEN "I",#1,"DATA"
62 INPUT #1,X$
63 AR=VAL(X$)
64 INPUT #1,X$
65 ND=VAL(X$)
66 FOR I=1 TO AR
67 FOR J=1 TO 5
68 INPUT #1,X$
69 A(I,J)=VAL(X$)
70 NEXT J
71 NEXT I
72 CLOSE #1
73 GOTO 30
74 IF L$="N" THEN RUN
75 GOTO 30
```

Utility Subroutines -- Files UTIL-OPT, UTIL-ERS, UTIL-CON, and UTIL-CHX

Same as for critical path method.

Activity Input Subroutine -- File INPT-ACT

The code and explanation of the activity input subroutine is listed in the critical path BASIC program chapter. The out-of-kilter method uses the same input screen as the CPM method and the same arrow network except for the following restrictions. Beyond the ordered pair

restrictions, the sequence of node numbers must start at zero (0); and in sequential integers, end with the number of nodes ND minus one. (This is true for IBM BASIC where arrays are dimensioned starting at zero (0).)

The information entered for activity input has two added entries. The activities have two durations, a minimum duration MIN DUR and a normal duration NORMAL DUR, rather than just the normal duration of the CPM method. Each activity is also assigned a cost which is incurred when the duration is reduced by one (1) increment of time.

Figure 37 is the input screen for the out-of-kilter method.

ACTIVITIES

NO.	I	J	MIN DUR	NORM DUR	INC COST
1	0	3	1	10	5
2	2	4	1	20	10
3	0	1	1	30	15
4	1	2	1	40	20
5	2	3	1	50	25
6	3	4	1	60	30
7	1	3	1	70	35
8	0	2	1	80	40
9	1	4	1	90	45
10	0	4	1	100	50

OPTION ?

Figure 37. Out-of-kilter method activity input screen

Project Cost Curve Subroutine -- File REPT-CRV

The total project cost curve comprised of optimum solutions to the minimum project cost problem is piece wise linear. The points where the curve breaks are the points listed on the project cost curve report. Up

to one hundred (100) points can be listed in the report. The screen, shown in figure 38, lists the computer processing time to reach the

```
                PROJECT COST CURVE
SOLUTION REACHED IN     2  SEC
     NO.  DURATION  COST
      1     190       0
      2     160       750
      3     120       1950
      4     101       2710
      5     100       2755
      6     91        3700
      7     90        3835
      8     72        6265
      9     43        11195
      10    11        17275
      11    4         18640

     OPTION ?
```

Figure 38. Out-of-kilter method project cost curve screen

solution; the line number of the break point; the duration of the project at the point; and the project cost at that point.

At the end of each screen of output, the OPTION ? line is printed and the program is paused. To continue, enter a blank. Otherwise, any other option.

```
2100                  * COST CURVE REPORT SUBROUTINE *
2101 REM-------------------------REPT-CRV-----------------------------

2102 LOCATE 1,9:PRINT "PROJECT COST CURVE"
2103 IF ER<>2 AND ER<>3 AND ER<>4 THEN 2106
2104 LOCATE 2,5:PRINT "NO SOLUTION FOUND ";ER;"ER"
2105 GOTO 2123
2106 IF ER<>0 THEN 2109
2107 LOCATE 2,1:PRINT "MAXIMUM ITERATIONS REACHED "
2108 GOTO 2111
2109 IF ER<>1 THEN 2123
2110 LOCATE 2,1:PRINT "SOLUTION FOUND IN ";IT;"ITR"
2111 LOCATE 2,30:PRINT TM;"SEC":LOCATE 3,8:PRINT "NO.   DURATION   COST"
```

```
2112 G=4
2113 FOR I=1 TO CYC
2114 LOCATE G,7:PRINT I:LOCATE G,12:PRINT C#(I,1):LOCATE G,22:PRINT C#(I
     ,2)
2115 G=G+1
2116 IF G<20 THEN 2122
2117 GOSUB 1860:REM UTIL-CON
2118 LOCATE 21,8:INPUT "",L$
2119 GOSUB 1870:REM UTIL-CHX
2120 IF L$<>"" AND Z#=BI# THEN RETURN
2121 GOSUB 1864:REM UTIL-CON+4
2122 NEXT I
2123 GOSUB 1800:REM UTIL-OPT
2124 LOCATE 21,8:INPUT "",L$
2125 RETURN
```

Out-of-kilter Algorithm Subroutine -- ALGR-KLT

The out-of-kilter algorithm subroutine (ALGR-KLT) is a flow algorithm combined with a CPM network algorithm. Once the minimum project duration based on normal activity durations is found for the CPM network with the network alogrithm, the flow algorithm starts a series of flow and time adjustments within the restrictions of the kilter diagrams until the minimum duration CPM network is reached.

The algorithm is a labeling routine in which the N(ND,3) is the node array containing the flow network node flows (N(i,1), nodes i=0 to ND-1), the source of the flow or "from" node indicators (N(i,2) nodes i=0 to ND-1), and the CPM networks node times (N(i,3) i=0 to ND-1).

In detail, the algorithm consists of the following steps.

(1) Sort the arcs (B(k,1)->B(k,2) arc k=1 to AR), called activities in the CPM network, in their start node, and then finish node order so that all arcs starting from node zero (0) are grouped together and subordered by their finish nodes. Using the CPM forward pass

algorithm with all arcs set at the normal activity durations (B(k,4)), find the maximum CPM network duration and its early start node times (N(i,3) node i=0 to ND-1). Let the network duration of the last network node (N(ND-1,3)) with a cost of zero (0) be the first point on the cumulative cost curve. Go to step (2).

(2) "Unlabel" all node flows to zero (N(i,1)=0 node i=0 to ND-1) and all "from" node indicators (N(i,2)=-BI node i=0 to ND-1) to negative infinity, and go to step (3).

(3) Starting with the first node, "label" node zero (0) with an infinite flow capacity (N(0,1)=BI) and set the "from" node indicator to zero (N(0,2)=0). Then, for each node in ascending order, say for node i, select all arcs, say k, leaving node i with minimum arc or activity duration equal to the difference between the nodes at the ends of the arc (N(B(k,2),3)-N(B(k,1),3)=B(k,3)) and with a flow assigned to the j node less than that assigned to the i node, label arc k's j node with the flow of node i (N(B(k,2),1)=N(B(k,1),1), and arc k's "from" node indicator to with i (N(B(k,2),2)=i). Continue this procedure until the next to the last node is reached. (Note: In the following steps the same procedure is used except that some of the arcs have limited capacity. In that case, the flow in the arc may be restricted so that the minimum of either the flow at the i node, the flow allowed through the arc, or the flow at the j node are used to determine the flow reaching the j node flow.) If the last node (N(ND-1,1)=BI) has infinite flow, then the unit cost for

the next reduction in duration will be infinite, so stop the algorithm. If not infinite, the CPM network can be reduced further. Go to step (4).

(4) Knowing that an infinite flow can not be passed through the flow network, the next step is to find the maximum flow that can be passed through the flow network in the arcs that are on the vertical segements of the kilter lines. This is done by using the same procedure as in step (3) with the addition of the arcs in which the difference between their node times is equal to the maximum arc or activity duration (N(B(k,2),3)-N(B(k,1),3)=B(k,4) k=1 to AR) and the first level of the stepped flow is less than the incremental arc or activity cost (B(k,6)<B(k,5)). For these arcs, the flow assigned to the arc's j node is further restricted to the incremental arc or activity cost minus the current flow (B(k,5)-B(k,6)). Go to step (5).

(5) If step (3) indicated that a flow can be passed through the network (N(ND-1,1)>0), then a breakthrough has occurred. In this case, the flow network's arcs are updated in step (7). If a breakthrough has not occurred (N(ND-1,1)=0), then step (6) is taken to try to find any reverse flows which might lead to a breakthrough. If step (4) and (6) do not result in a breakthrough, then the CPM network is adjusted in step (8).

(6) With no breakthrough, it might be possible to find a network flow by reducing arc flows or reversing the flows. Starting sequentially

from the first arc, if the arc's finish node, say node j, has been labeled (N(j,2)>=0) and the start node, say node i, has not been labeled (N(i,2)=-BI), then by reversing the flow in the arc, a breakthrough might be found. To find if a reverse flow for say the arc ij or arc k can be found, the difference between the node times must be equal to the upper or lower activity durations (N(B(k,2),3)-N(B(k,1)=B(k,4) or N(B(k,2),3)-N(B(k,1),3)= B(k,3)) and the flow at either level (B(k,6) or B(k,7)) must be greater than zero (0). If a reverse flow is possible, then label the i node with the minimum of either the arc flow or the labeled flow at the j node. After all reverse flows are found by selecting all arcs in ascending ij order, return to step (4).

(7) When a breakthrough has occurred, the flow passed through the net needs to be distributed to the flow network arcs (B(k,6) and B(k,7) k=1 to AR). This is done by starting at the terminal node (ND-1); and in reverse sequence from ND-1 to zero (0), adding to each arcs specified by the nodes (j=ND-1 to 0) and "from" labels (N(i,2), j=ND-1 to 0) the flow passed through the network (N(ND-1,1)). If the nodes define a reversed arc where i>j, then subtract the flow. This is continued for each node until node zero is reached. Return to step (2).

(8) If a breakthrough has not occurred, then a new set of CPM network node times (N(i,3)) must to be found. From step (4), the array N(ND,3) has been labeled with flows, "from" nodes, and CPM early

start times. Each arc or activity is defined by a pair of nodes in this array. All activities that have a labeled start node (N(i,2)>=0) and an unlabeled finish node (N(j,2)=-BI), or an unlabeled start node and a labeled finish node, constitute a "cut" set for which the activity with the least reduction, or increase, in duration (N(j,3)-N(i,3)) from the upper of lower limits, limits the reduction in the total CPM networks duration for the current set of flows. From the kilter lines, any activity with greater than the minimum duration, or less than the maximum duration, is part of the cut set that can be used to reduce the duration of the CPM network. Selecting the arc or activity with the minimum difference from the maximum or minimum durations and subtracting the difference from all unlabeled node times (N(i,2)=-BI, i=0 to ND-1) updates the current CPM network node times (N(i,3)-change for all unlabeled nodes).

(9) The CPM network was changed at a cost which had a unit value equal to the flow through the network at the last breakthrough. This cost is an incremental cost, so the point on the cumulative cost curve is the summation of the last cumulative cost point and the current cost. Return to step (2).

```
3300 REM          *OUT-OF-KILTER ALGORITHM SUBROUTINE*
3301 REM-------------------------ALGR-KLT-------------------------------
```

Sets BI to machine infinite for single precision variables.

```
3302 BI=1E+08
```

Loads the activity data from array A(AR,5) into work array B(AR,7).

```
3303 FOR I=1 TO AR
3304 FOR J=1 TO 5
3305 B(I,J)=A(I,J)
3306 NEXT J
3307 B(I,6)=0
3308 B(I,7)=0
3309 NEXT I
```

Initializes the node array N(ND,3) to zero (0) flows N(ND,1), to null "from" indicators N(ND,2), and to zero (0) early start CPM time N(ND,3).

```
3310 FOR I=0 TO ND-1
3311 N(I,1)=0
3312 N(I,2)=-BI
3313 N(I,3)=0
3314 NEXT I
```

Sets iteration count CYC to one (1) and sorts the work array on the CPM i node B(AR,1) and then j node B(AR,2) using a Schell sort[30].

```
3315 CYC=1
3316 Y=AR
3317 Y=INT(Y/2)
3318 FOR Z=1 TO AR-Y
3319 A=B(Z+Y,1)
3320 B=B(Z+Y,2)
3321 C=B(Z+Y,3)
3322 D=B(Z+Y,4)
3323 E=B(Z+Y,5)
3324 FOR W=Z TO 1 STEP -Y
3325 IF B(W,1)<A THEN 3334
3326 IF B(W,1)>A THEN 3328
3327 IF B(W,2)<B THEN 3334
3328 B(W+Y,1)=B(W,1)
3329 B(W+Y,2)=B(W,2)
3330 B(W+Y,3)=B(W,3)
3331 B(W+Y,4)=B(W,4)
3332 B(W+Y,5)=B(W,5)
3333 NEXT W
3334 B(W+Y,1)=A
3335 B(W+Y,2)=B
3336 B(W+Y,3)=C
3337 B(W+Y,4)=D
3338 B(W+Y,5)=E
3339 NEXT Z
3340 IF Y>1 THEN 3317
```

Sets the start node numbered zero (0) to infinity so that node is not eligible for CPM network time reduction.

The node numbering scheme for the out-of-kilter algorithm is more rigid than the CPM algorithm. Not only do the conventions of the CPM network apply, but also the node numbering must start at zero (0) and be consecutive integers up to ND-1.

```
3341 N(0,1)=BI
3342 N(0,2)=BI
```

Finds the duration of the CPM network when all activities are set at their normal durations.

```
3343 AA=1
3344 FOR N=0 TO ND-2
3345 FOR A=AA TO AR
3346 IF B(A,1)>N THEN 3350
3347 IF B(A,1)<>N THEN 3349
3348 IF N(N,3)+B(A,4)>N(B(A,2),3) THEN N(B(A,2),3)=N(N,3)+B(A,4)
3349 NEXT A
3350 AA=A
3351 NEXT N
```

Sets the first point on the cost curve C#(1,1), C#(1,2) to the maximum duration and zero (0) cost.

```
3352 C#(CYC,1)=N(ND-1,3)
```

Before starting any flow calculations, intializes all nodes not on the minimum duration critical path, or on the upper part of the kilter line, to zero (0) flow N(ND,1) and negative infinity "from" node N(N,2).

```
3353 FOR N=0 TO ND-1
3354 IF N(N,1)=BI THEN 3357
3355 N(N,1)=0
3356 N(N,2)=-BI
3357 NEXT N
```

Sets any node terminating an activity that is at its minimum duration to an infinite flow. This corresponds to the upper part of the out-of-kilter line.

```
3358 AA=1
3359 FOR N=0 TO ND-2
3360 IF N(N,1)<>BI THEN 3369
3361 FOR A=AA TO AR
3362 IF B(A,1)>N THEN 3368
```

```
3363 IF B(A,1)<>N THEN 3367
3364 IF N(N,3)+B(A,3)-N(B(A,2),3)<>0 THEN 3367
3365 N(B(A,2),1)=BI
3366 N(B(A,2),2)=N
3367 NEXT A
3368 AA=A
3369 NEXT N
```

Stops the algorithm if the last node has an infinite flow. This indicates that the CPM network has a critical path in which all the activities on the path have been reduced to minimum duration.

```
3370 IF N(ND-1,1)=BI THEN RETURN
```

Seeks a maximum flow through the flow network which does not violate any of the activity kilter lines.

This can be best visualized on the kilter diagram. The first node is assumed to have an infinite flow. Each arc's directed flow is restricted by its kilter line so that the flow through the net can be found by calculating the maximum flow that can reach each node in succession until the last node is reached. If a node is reached by one (1) or more flows, then the node number from which the maximum flow came is assigned to N(node,2) and the maximum flow is assigned to N(node,1). If no flow reaches the node then N(node,2) is left at negative machine infinite or -BI.

```
3371 AA=1
3372 FOR N=0 TO ND-2
3373 IF N(N,1)=0 THEN 3387
3374 FOR A=AA TO AR
3375 IF B(A,1)>N THEN 3386
3376 IF B(A,1)<>N THEN 3385
3377 IF N(B(A,2),2)<>-BI THEN 3385
3378 IF N(N,3)+B(A,4)-N(B(A,2),3)=0 AND B(A,6)<B(A,5) THEN 3382
3379 IF N(N,3)+B(A,3)-N(B(A,2),3)<>0 THEN 3385
3380 N(B(A,2),1)=N(N,1)
3381 GOTO 3384
3382 N(B(A,2),1)=B(A,5)-B(A,6)
3383 IF N(N,1)<N(B(A,2),1) THEN N(B(A,2),1)=N(N,1)
3384 N(B(A,2),2)=N
3385 NEXT A
3386 AA=A
3387 NEXT N
```

Goes to the CPM time change routine when a breakthrough is found or N(ND-1,2) has a label.

```
3388 IF N(ND-1,2)<>-BI THEN 3405
```

Finds a reverse flow combination which can increase the flow in the flow net.

Although the network is a directed network, if a flow in an arc can be reduced in such a manner as to increase to total flow in the network, then the arc can be considered reversed in flow even though the total flow in the arc is in the proper direction.

```
3389 X=0
3390 FOR N=1 TO ND-2
3391 IF N(N,2)=-BI THEN 3403
3392 FOR A=1 TO AR
3393 IF B(A,2)<>N THEN 3402
3394 IF N(B(A,1),2)<>-BI THEN 3402
3395 IF N(B(A,1),3)+B(A,4)-N(N,3)<>0 AND N(B(A,1),3)+B(A,3)-N(N,3)<>0
     THEN 3402
3396 IF B(A,6)<=0 AND B(A,7)<=0 THEN 3402
3397 X=1
3398 N(B(A,1),2)=N
3399 N(B(A,1),1)=N(N,1)
3400 IF B(A,6)>0 AND N(B(A,1),1)>B(A,6) THEN N(B(A,1),1)=B(A,6)
3401 IF B(A,7)>0 AND N(B(A,1),1)>B(A,7) THEN N(B(A,1),1)=B(A,7)
3402 NEXT A
3403 NEXT N
```

Returns to the flow algorithm to seek an increase in total flow of the flow network if the change switch is set to one (1).

```
3404 IF X=1 THEN 3371
```

Returns to main routine if a breakthrough has occurred with an infinite flow.

```
3405 IF N(ND-1,1)=BI THEN RETURN
3406 IF N(ND-1,1)=0 THEN 3425
```

Changes the flows assigned to each activity or arc in the flow net.

The breakthough algorithm found the maximum flow that could be passed through the network. This algorithm distributes the flow among the activities. Remembering that the kilter line has two (2) flow change segments or that the activity has two (2) flows, one of which is limited to a flow equal to the incremental activity cost, the flow change must be distributed between the two (2) flows B(activity number,6) and B(activity number,7).

```
3407 L=N(ND-1,1)
3408 J=ND-1
3409 I=N(ND-1,2)
3410 FOR N=0 TO ND-1
3411 FOR A=1 TO AR
3412 IF B(A,1)<>I OR B(A,2)<>J THEN 3416
3413 IF B(A,6)=B(A,5) THEN B(A,7)=B(A,7)+L
3414 IF B(A,6)<B(A,5) THEN B(A,6)=B(A,6)+L
3415 GOTO 3419
3416 IF B(A,2)<>I OR B(A,1)<>J THEN 3419
3417 IF B(A,7)=0 THEN B(A,6)=B(A,6)-L
3418 IF B(A,7)>0 THEN B(A,7)=B(A,7)-L
3419 NEXT A
3420 IF I=0 THEN 3353
3421 J=I
3422 I=N(J,2)
3423 NEXT N
3424 GOTO 3353
```

Finds the maximum reduction in CPM network duration within the restrictions of the kilter lines.

```
3425 A1=BI
3426 A2=BI
3427 FL=0
3428 FOR A=1 TO AR
3429 IF B(A,1)=0 THEN FL=FL+B(A,6)+B(A,7)
3430 IF N(B(A,1),2)=-BI OR N(B(A,2),2)<>-BI THEN 3436
3431 S1=N(B(A,1),3)+B(A,4)-N(B(A,2),3)
3432 S2=N(B(A,1),3)+B(A,3)-N(B(A,2),3)
3433 IF S1<0 AND A1>-S1 THEN A1=-S1
3434 IF S2<0 AND A1>-S2 THEN A1=-S2
3435 GOTO 3441
3436 IF N(B(A,1),2)<>-BI OR N(B(A,2),2)=-BI THEN 3441
3437 S1=N(B(A,1),3)+B(A,4)-N(B(A,2),3)
3438 S2=N(B(A,1),3)+B(A,3)-N(B(A,2),3)
3439 IF S1>0 AND A2>S1 THEN A2=S1
3440 IF S2>0 AND A2>S2 THEN A2=S2
3441 NEXT A
3442 IF A1>A2 THEN A1=A2
3443 FOR N=1 TO ND-1
3444 IF N(N,2)=-BI THEN N(N,3)=N(N,3)-A1
3445 NEXT N
```

Finds the cost for the CPM network reduction in duration.

The reduction in duration of the CPM network is at a cost equal to the total flow in the flow network.

```
3446 CYC=CYC+1
3447 C#(CYC,1)=N(ND-1,3)
3448 C#(CYC,2)=A1*FL+C#(CYC-1,2)
3449 GOTO 3353
```

Program Table of Contents

Table 5 can be used to reconstruct the above computer code from the

Table 5. Out-of-kilter BASIC program table of contents

File	Program lines	Page	Routines
MAIN-KLT	0001-0075	100	Out-of-kilter method
INPT-ACT	1100-1133	24	Activity input subroutine
UTIL-OPT	1800-1803	22	Option line subroutine
UTIL-ERS	1850-1853	22	Erase option subroutine
UTIL-CON	1860-1866	22	Continue line subroutine
UTIL-CHX	1870-1882	23	Data check subroutine
REPT-CRV	2100-2125	105	Cost curve report subroutine
ALGR-KLT	3300-3449	110	Out-of-kilter algorithm subroutine

computer disk and to organize subroutines from previous program listings. Since BASIC code is dependent on program line numbers for its subroutine branching, the statement numbers must be maintained as listed above.

Out-of-kilter Method: Solutions to Example 6 Minimum Project Cost Curve Problem

Example 6 for the out-of-kilter method is based on the same example 4 network schedule as shown in figure 26 which was used to demonstrate the linear cost problem. The input screen is shown in figure 9 and the output screen is shown in figure 39.

PROJECT COST CURVE
SOLUTION REACHED IN 2 SEC

NO.	DURATION	COST
1	190	0
2	160	750
3	120	1950
4	101	2710
5	100	2755
6	91	3700
7	90	3835
8	72	6265
9	43	11195
10	11	17275
11	4	18640

OPTION ?

Figure 39. Out-of-kilter method cost curve output screen

The execution time was two seconds (2) when run on the Panasonic Sr. Partner computer using the above code compiled by the IBM BASIC compiler.

MINIMUM PROJECT COST PROBLEM WITH QUADRATIC COST FUNCTIONS

In the minimum project cost problem with linear cost functions the objective function was defined as:

$$\text{Total project cost} = \sum_{\text{all } ij} c_{ij} * R_{ij} + \text{fix.} * (T_n - T_0)$$

were:

- c_{ij} - Incremental cost to reduce duration of activity ij
- R_{ij} - Reduction in duration of activity ij
- dur_{ij} - Normal or minimum cost duration of activity ij
- fix. - Fixed cost per increment of total project duration
- T_i - Node time for node i
- n - Maximum node number

A better aproximation of the project cost would be a quadratic function:

$$\text{Total Project cost} = \sum_{\text{all } ij} (cf_{ij} * (R_{ij} - rmc_{ij})^2 + f_{ij} * (T_j - T_i)) + \text{fix.} * (T_n - T_0)$$

or:

$$\text{Total Project cost} = \sum_{\text{all } ij} (cf_{ij} * (R_{ij} - rmc_{ij})^2 + f_{ij} * (dur_{ij} - R_{ij})) + \text{fix.} * (T_n - T_0)$$

with:

- rmc_{ij} - Reduction in duration of ij at which its variable cost is minimum
- cf_{ij} - Coefficient for curvature of cost curve for activity ij
- f_{ij} - Fixed cost per increment of activity's "window" for completion

and were each activity's variable cost is a quadratic function with a minimum value at duration $dur_{ij} - rmc_{ij}$ and a curvature factor of cf_{ij}.

To find the optimal solution to this problem requires solving a set of linear constraints with a convex quadratic objective function, or a quadratic programming (QP) problem. Fortunately, the QP problem can be solved with the primal-dual method using a series of LP subproblems[42] as derived by Beale from the Kuhn-Tucker conditions[40].

Before describing Beale's method, the primal simplex and primal-dual methods will be presented.

Theory of Primal Simplex Method

The dual simplex method required that the coefficients of the objective function in the simplex tableau be greater than or equal to zero (0) and that the starting current point (0,0) violate at least one (1) constraint. In this form, the current point was an infeasible solution for the primal problem or the point was "primal infeasible".

In many LP problems, the current point (0,0) satisfies all the model's constraints so that the current point is "primal feasible". This point would also satisfy the dual simplex algorithm's condition for an optimal solution.

But if some of the coefficients of the objective function are less than zero (0) and if one (1) of the variables can be increased to a larger positive number while still satisfying the constraints of the model, the current point (0,0) cannot be the minimum value of the objective.

To solve this primal feasible problem, the primal simplex method was developed.

The Two Dimensional Case solved by the Primal Simplex Method

As with the dual simplex method, let the following two (2) variable LP problem be the primal simplex example 7.

minimize (-a)*X+b*Y

subject to:
X >=0
Y>=0
-X >=e
-Y>=f

e,f>=0

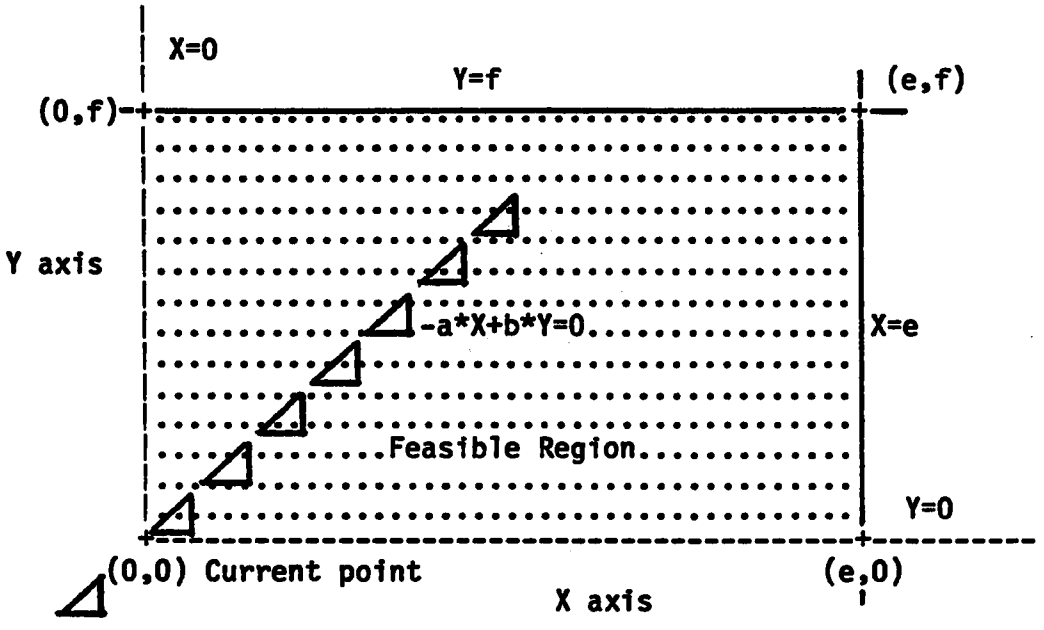


Figure 40. Primal simplex method example 7

In this example, the objective function has a negative coefficient of -a, and the constraint constants are all nonnegative.

Primal Simplex Tableau and Transformation

In a manner similar to the dual simplex method, slack can be added to the above equations and the equations written as tableau 8 with two

	1	X	Y
z=	0	(-a)	b
x=	0	1	0
y=	0	0	1
s1=	e	-1	0
s2=	f	0	-1

Tableau 8. Primal simplex method tableau example 7, iteration one

(2) possible pivot rows. Since all slacks (s1,s2) are nonnegative, the constraints are satisfied and the current point (0,0) is in the feasible region of the LP problem.

In the dual simplex method, this would terminate the algorithm; but in example 7, the objective function is not at a minimum value. This can be seen by inspection of the constraints and the objective function. Since it is possible to increase the value of the X variable to e without violating any constraint while reducing the value of the objective function z by a*e, the current point (0,0) is not optimal.

To find a new point (e,0) a Gauss-Jordan elimination can be used as in the dual simplex method. For the transformation of the tableau, the column headed with the most negative objective function coefficient would be the best choice or the column headed by the element -a.

But for a Gauss-Jordan elimination, a pivot row is also needed. To find a row in example 7 for a Gauss-Jordan elimination, every row of the tableau is tried as a pivot row and the minimum value of the objective selected:

$$\text{minimum } \left(-\frac{-a*0}{1}, -\frac{-a*0}{0}, -\frac{-a*e}{-1}, -\frac{-a*f}{0}\right)$$

or considering only valid operations:

$$\text{minimum } \left(\frac{0}{1}, \frac{e}{-1}\right)$$

Since the -(-a) coefficient is common to all the possible objective function values, the pivot row can be selected from among the pivot rows with a negative element in the pivot column.

It might appear that most negative ratio would determine the pivot

row from among the rows with negative elements in the pivot column; but for the new point to remain a primal feasible solution point, the constraint with the smallest absolute value of ratio of constant column element to pivot column element ratio must be chosen. This can be demonstrated by completing several simplex transformations on negative ratios with other than the minimum absolute value.

The third constraint can be selected for the pivot row using the above rule. In the new pivot row, the pivot element has a negative value, so division by the pivot element changes the sign of the slack. Using example 7, the pivot row is divided by the pivot element in tableau 9 and row reduced in tableau 10.

	1	X	Y
z=	0	-a	b
x=	0	1	0
y=	0	0	1
-s1=	-e	1	0

Tableau 9. Primal simplex method tableau example 7, iteration one

	1	-s1	Y
z=	-a*e	-a	b
x=	e	1	0
y=	0	0	1
s1=	0	1	0

Tableau 10. Primal simplex method tableau example 7, iteration one

This transformation has left a negative slack column in the tableau; or in equation form, the slack coefficient is the negative of its original value. To change this slack to a positive value, the pivot column elements of the tableau are multiplied through by a negative one (-1). Tableau 11 is the final result of the primal simplex method.

	1	-(-s1)	Y
z=	-a*e	-(-a)	b
x=	e	-(1)	0
y=	0	0	1
s1=	0	-(1)	0

Tableau 11. Primal simplex method tableau example 7, iteration one

The new point, shown in figure 41, can be read directly from the primal simplex tableau as in the dual simplex method.

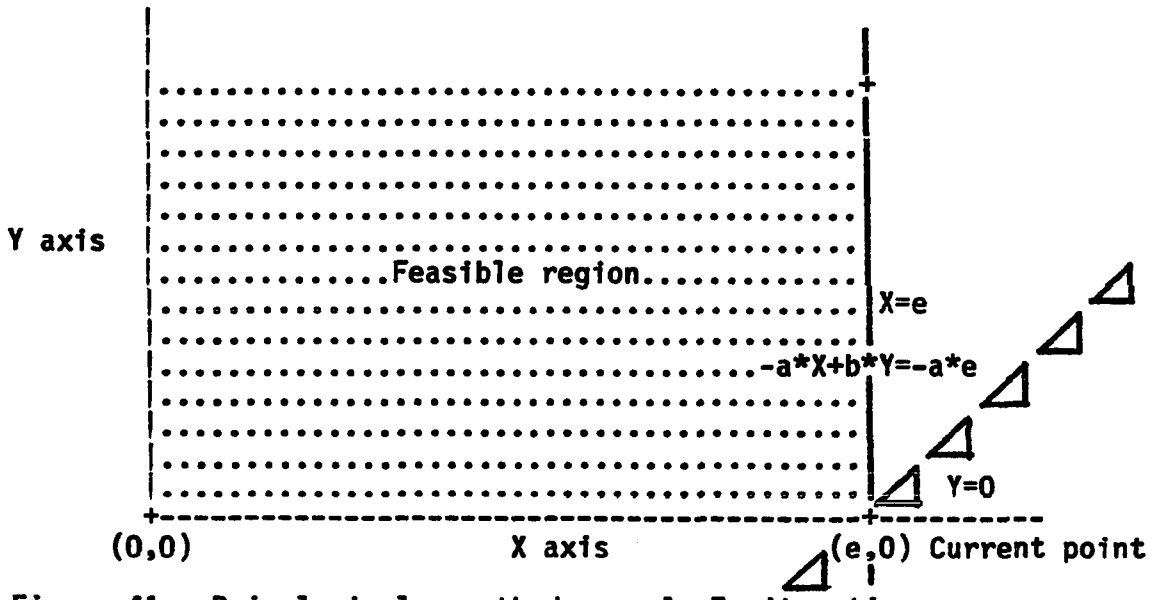


Figure 41. Primal simplex method example 7, iteration one

The transformation has left the first row of the tableau with all nonnegative elements. To reduce the value of objective function further would require a negative element in the objective row since all LP variables are restricted to nonnegative values. When the elements of the objective row are all nonnegative, the current point is optimum and the algorithm is stopped.

If a second pivot were required, the transformation of the constraints is identical to the dual simplex method with the augmented

$$\begin{vmatrix} e, -1, 0 \end{vmatrix} * \begin{vmatrix} 1 & 0 & 0 \\ e & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} 0, -1, 0 \end{vmatrix}$$

$$\begin{vmatrix} f, 0, -1 \end{vmatrix} * \begin{vmatrix} 1 & 0 & 0 \\ e & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} f, 0, -1 \end{vmatrix}$$

$\overline{B}$ inverse matrix transformation resulting in the new tableau 12.

	1	s1	Y
z=	-a*e	a	b
x=	e	-1	0
y=	0	0	1
s1=	0	-1	0
s2=	f	0	-1

Tableau 12. Primal simplex method tableau example 7, iteration one

Theory of Primal-dual Method

A LP problem can be either, or both, primal and dual infeasible for the current point (0,0). In this case, the LP problem can be solved by a combination of the primal and dual methods described earlier.

Although several primal-dual methods are available, such as the criss-cross method[43] or the objective function penalty method[44], the approach used here is to first consider for a transformation a reduced primal feasible problem with only the nonviolated constraints of the orginal problem and then to consider for a transformation a reduced dual feasible problem with only the columns with nonnegative objective function elements.

In general, the method starts at the current point (0,0) and uses a two (2) phase search for a pivot column or row with which to transform the simplex tableau to a new point.

First using the primal algorithm, the simplex tableau is searched for a primal pivot column. If all possible primal pivot columns are exhausted, then the dual algorithm is used to search the simplex tableau for a pivot row. Depending on the pivot found, primal or dual, the simplex tableau is transformed and the method continues to the next iteration to find a new point until point both primal and dual feasible is found.

Primal-dual Method BASIC Code

All programs listed in the text consist of a main calling routine and a series of input, output, and processing subroutines. These

routines are listed in the text as well as on a computer disk compatible with IBM micro-computers.

Primal-Dual Method Main routine -- File MAIN-PDS

The primal-dual main calling routine (MAIN-PDS) dimensions the nine (9) data arrays; writes the option menu to the screen as shown in figure

```
                    PRIMAL-DUAL METHOD

          NUMBER OF CONSTRAINTS          10
          NUMBER OF VARIABLES            14
          MAXIMUM ITERATIONS           1000

                    M-RETURN TO MENU

             O-OBJECTIVE COEFFICIENTS
             A-CONSTRAINT COEFFICIENTS
             C-CONSTRAINT TYPES
             B-BOUNDED VARIABLES
             U-EXECUTE ALGORITHM
             R-REPORT LISTING
             S-SAVE  F-FETCH
             N-NEW PROBLEM

        OPTION ?
```

Figure 42. Primal-dual method main menu screen

42; calls the utility subroutines UTIL-OPT, UTIL-ERS, UTIL-CON, UTIL-CHX; calls the data input and output subroutines INPT-OBJ, INPT-CON, INPT-TYP, INPT-BND, and REPT-SMP; calls and times the processing algorithm ALGR-PDS: and saves and fetches the input data to disk.

```
1 REM              * PRIMAL-DUAL SIMPLEX METHOD *
2 REM-----------------------MAIN-PDS----------------------------

3 REM      BI# - MACHINE INFINITE
4 REM      CO  - PIVOT COLUMN FOR SIMPLEX TRANSFORMATION
5 REM      ER  - ERROR KEY
6 REM      IN  - NUMBER OF ITERATIONS BETWEEN REINVERSIONS
7 REM      IR  - MAXIMUM NUMBER OF ITERATIONS
```

```
8 REM      MD  - NUMBER OF CONSTRAINTS
9 REM      ND  - NUMBER OF VARIABLES
10 REM     PA  - NUMBER OF ATTEMPTS AT PIVOT
11 REM     RO  - PIVOT ROW FOR SIMPLEX TRANSFORMATION
12 REM     SM# - MACHINE ZERO
13 REM     A#(MD+1,ND+8) - ORIGINAL DATA AND KEYS
14 REM     B#(2*ND+1,ND+1) - SIMPLEX MATRIX
15 REM     H#(ND+1) - PAST ITERATION SOLUTION
16 REM     M#(ND+1,2) - UPPER AND LOWER BOUND VALUES
17 REM     P#(ND+1) - WORK VECTOR
18 REM     R(MD+1) - CONSTRAINT TYPE (1->=,0-=,-1-<=)
19 REM     T#(ND+1,ND+1) - INVERSION WORK FILE
20 REM     V#(MD+1+ND+1+ND+1+ND+1) - ROW AND COLUMN ARRAY
21 REM     X#(ND) - SOLUTION VECTOR
22 REM ------------------------------------------------------------
```

Sets MD to the default number of constraints in the LP problem and ND to the number of variables. Sets IN to the number of iterations before the reinversion of the augmented $\overline{B}$ matrix. Sets the default maximum number of iterations to one thousand (1000). Sets BI# to machine infinite and SM# to machine zero.

```
23 MD=0
24 ND=0
25 IN=10
26 IR=1000
27 BI#=1E+10
28 SM#=1E-10
```

Prompts and reads from the keyboard the number of constraints MD in the LP problem, the number of variables ND, and the maximum number of iterations IR allowed before the primal-dual simplex algorithm is stopped.

```
29 CLS
30 LOCATE 1,7:PRINT "PRIMAL-DUAL METHOD"
31 LOCATE 3,1:PRINT "NUMBER OF CONSTRAINTS":LOCATE 3,31:INPUT "",L$
32 GOSUB 1870:REM UTIL-CHX
33 IF Z#<>BI# THEN MD=Z#
34 LOCATE 3,30:PRINT MD,"    ":LOCATE 4,1:PRINT "NUMBER OF VARIABLES":
   LOCATE 4,31:INPUT "",L$
35 GOSUB 1870:REM UTIL-CHX
36 IF Z#<>BI# THEN ND=Z#
37 LOCATE 4,30:PRINT ND,"    ":LOCATE 5,1:PRINT "MAXIMUM ITERATIONS":
   LOCATE 5,31:INPUT "",L$
38 GOSUB 1870:REM UTIL-CHX
39 IF Z#<>BI# THEN IR=Z#
40 LOCATE 5,30:PRINT IR,"    "
```

Dimensions the constraint coefficients array A#(MD+1,ND+8). Dimensions the basis inverse and basis array B#(2*ND+1,ND+1). Dimensions the holding array H#(ND+1), the upper and lower bound array M#(ND+1), the pivot row P#(ND+2), the constraint type array R(MD+1+ND+1), the reinversion working space array T#(ND+1,ND+1), the pivot selection array V#(MD+1+ND+1+ND+1+ND+1), and the solution vector X#(ND).

The array V#() is the summary of the first row and the first column, including the phantom upper and lower bound constraints, of the expanded simplex tableau. This allows for a rapid search of the possible primal and then dual pivots.

```
41 DIM A#(MD+1,ND+8)
42 DIM B#(2*ND+1,ND+1)
43 DIM H#(ND+1)
44 DIM M#(ND+1,2)
45 DIM P#(ND+2)
46 DIM R(MD+1)
47 DIM T#(ND+1,ND+1)
48 DIM V#(MD+1+ND+1+ND+1+ND+1)
49 DIM X#(ND)
```

Initializes the constraint type array to all greater than or equals.

```
50 FOR I=1 TO MD+1
51 R(I)=1
52 NEXT I
```

Prints the option menu to the screen; calls the option line routine UTIL-OPT; and pauses for the entry of "M", "O", "A", "C", "B", "Q", "U", "R", "S", "F", "N" for the option variable L$.

```
53 LOCATE 8,15: PRINT          "M-RETURN TO MENU"
54 LOCATE 10,10:PRINT       "O-OBJECTIVE COEFFICIENTS"
55 LOCATE 11,10:PRINT       "A-CONSTRAINT COEFFICIENTS"
56 LOCATE 12,10:PRINT       "C-CONSTRAINT TYPES"
57 LOCATE 13,10:PRINT       "B-BOUNDED VARIABLES"
58 LOCATE 14,10:PRINT       "U-EXECUTE ALGORITHM"
59 LOCATE 15,10:PRINT       "R-REPORT LISTING"
60 LOCATE 16,10:PRINT       "S-SAVE F-FETCH"
61 LOCATE 18,10:PRINT       "N-NEW PROBLEM"
62 GOSUB 1800:REM UTIL-OPT
63 LOCATE 21,8:INPUT "",L$
```

Calls either the objective function input subroutine INPT-OBJ, the constraint input subroutine INPT-CON, the constraint type input subroutine INPT-TYP, the upper and lower bound input subroutine INPT-

BND, the processing subroutine ALGR-PDS, or the report routine REPT-SMP based on the option variable L$.

```
64 CLS
65 H=0
66 G=2
67 IF L$<>"O" THEN 70
68 GOSUB 1200:REM INPT-OBJ
69 GOTO 64
70 IF L$<>"A" THEN 73
71 GOSUB 1300:REM INPT-CON
72 GOTO 64
73 IF L$<>"C" THEN 76
74 GOSUB 1400:REM INPT-TYP
75 GOTO 64
76 IF L$<>"B" THEN 79
77 GOSUB 1500:REM INPT-BND
78 GOTO 64
```

Sets the optimal solution equal to the current point of the primal-dual algorithm.

```
79 IF L$<>"U" THEN 88
80 TM=3600*VAL(MID$(TIME$,1,2))+60*VAL(MID$(TIME$,4,2))+VAL(MID$(TIME$,7
   ,2))
81 GOSUB 3300:REM ALGR-PDS
82 OB#=B#(I,1)
83 FOR I=1 TO ND
84 X#(I)=B#(I+1,1)
85 NEXT I
86 TM=3600*VAL(MID$(TIME$,1,2))+60*VAL(MID$(TIME$,4,2))+VAL(MID$(TIME$,7
   ,2))-TM
87 GOTO 53
88 IF L$<>"R" THEN 91
89 GOSUB 2200:REM REPT-SMP
90 GOTO 64
```

Saves the content of MD, ND, M#(ND+1,ND+8), A#(MD+1,ND+8), and R(ND+1) to disk file "DATA" as an ASCI file if option "S" is selected.

```
91 IF L$<>"S" THEN 106
92 OPEN "O",#1,"DATA"
93 PRINT #1,STR$(MD)
94 PRINT #1,STR$(ND)
95 FOR I=1 TO ND+1
96 PRINT #1,STR$(M#(I,1))
97 PRINT #1,STR$(M#(I,2))
98 NEXT I
```

```
99 FOR I=1 TO MD+1
100 FOR J=1 TO ND+8
101 PRINT #1,STR$(A#(I,J))
102 NEXT J
103 PRINT #1,STR$(R(I))
104 NEXT I
105 CLOSE #1
```

Loads to MD, ND, M#(ND+1,2), A#(MD+1,ND+8), and R(ND+1) the disk file "DATA" if option "F" is selected.

```
106 IF L$<>"F" THEN 128
107 OPEN "I",#1,"DATA"
108 INPUT #1,X$
109 MD=VAL(X$)
110 INPUT #1,X$
111 ND=VAL(X$)
112 FOR I=1 TO ND+1
113 INPUT #1,X$
114 M#(I,1)=VAL(X$)
115 INPUT #1,X$
116 M#(I,2)=VAL(X$)
117 NEXT I
118 FOR I=1 TO MD+1
119 FOR J=1 TO ND+8
120 INPUT #1,X$
121 A#(I,J)=VAL(X$)
122 NEXT J
123 INPUT #1,X$
124 R(I)=VAL(X$)
125 NEXT I
126 CLOSE #1
127 GOTO 53
```

Restarts program for new run if option "N" is selected.

```
128 IF L$="N" THEN RUN
129 GOTO 53
```

Utility Subroutines -- Files UTIL-OPT, UTIL-ERS, UTIL-CON, and UTIL-CHX

Same as for critical path method.

Input Subroutines -- Files INPT-OBJ, INPT-CON, INPT-TYP, and INPT-BND

Same as for dual simplex method.

Report Subroutine -- File REPT-SMP

Same as for dual simplex method.

Primal-dual Algorithm Subroutine -- File ALGR-PDS

The primal-dual algorithm first initializes a simplex tableau to represents a current point (0,0) at the origin and the constraints that form the zero axes. Then, in an iterative fashion, the algorithm executes the following steps.

Primal Search The first steps follow the primal algorithm as if the simplex tableau were a LP problem consisting of only the constraints satisfied by the current point. This approach assumes that the current point is primal feasible by satisfying a reduced set of constraints; that a new point can be found by increasing the values of the variables associated with negative elements in the simplex objective function row; and that primal feasibility can be maintained for the reduced set of constraints.

Using this reduced set of constraints requires a modification to the primal simplex method because of the possibility that in the reduced tableau all columns eligible for a primal pivot will not provide a pivot row under the rules of the primal algorithm. If a column fails to provide a pivot row, then several eligible pivot columns will have to be tried until all eligible columns have been eliminated.

(1) Find all columns from the simplex tableau eligible as a primal pivot column which have a negative element in the first row. If no column can be found and the simplex tableau is primal feasible then

go to step (5). Otherwise, go to step (2)

(2) From the still eligible primal pivot columns, select the column with the most negative element in the first row. If no pivot column can be found, then go to step (5). Otherwise, go to step (3).

(3) From the pivot column selected in step (2), allow any satisfied constraint from the LP problem to be a candidate for the primal pivot row if the constraint's transformed row element corresponding to the pivot column is less than zero (0). Go to step (4)

(4) If more than one (1) transformed constraint qualifies as a pivot row, then select as the pivot row the row with the minimum absolute value of the ratio of the constant column element and the pivot column element. If no row is found for a pivot, then remove the current column as an eligible pivot column and return to step (2). Otherwise, go to step (8).

Dual Search The next two (2) steps assume that the simplex tableau consists only of columns with positive first row elements and that a new point can be found satisfying one of the violated constraints of the simplex tableau.

(5) If all columns of the simplex tableau have been eliminated as candidates for a primal pivot columns, then find all the LP problem violated constraints which are eligible for a dual simplex pivot row. If no constraint is violated and the tableau is primal and dual feasible, then set ER=1 and return to the main routine.

Otherwise go to step (6).

(6) From the eligible pivot rows, select the constraint row which is the greatest geometric distance from the current point. If the simplex tableau is either dual or primal infeasible and no pivot row or column has been found, then set ER=2 and return to main routine. Otherwise, go to step (7).

(7) In the transformed constraint selected in step (6) as a candidate for a dual pivot row, select as a pivot element the row element which is greater than zero (0) and corresponds to a positive element in the first row of the tableau which has the minimum ratio of the first row element and pivot row element. If no pivot element is found, then remove the row as an eligible dual pivot row and return to step (6). Otherwise, go to step (8).

(8) At this point, either a primal pivot or a dual pivot has been found. Transform the simplex part of the tableau using a Gauss-Jordan elimination with the pivot row selected in steps (2),(3), and (4); or steps (6) and (7). Go to step (9).

(9) The tranformed augmented simplex tableau now represents the new point which becomes the current point. Increment the iteration count by one. If the iterations count is greater than the maximum allowed, set ER=0 and go to main routine. Otherwise, go to step (1).

```
3300 REM                 * PRIMAL-DUAL ALGORITHM *
3301 REM-----------------------ALGR-PDS-------------------------------
```

Initializes the simplex tableau to represent the point at the origin and the zero axes or the lower bounds of all variables.

```
3302 FOR I=1 TO 2*ND+1
3303 FOR J=1 TO ND+1
3304 B#(I,J)=0#
3305 NEXT J
3306 NEXT I
3307 FOR I=2 TO ND+1
3308 H#(I)=BI#
3309 B#(I,I)=1#:B#(I,1)=M#(I,2)
3310 B#(ND+I,I)=1#:B#(ND+I,1)=M#(I,2)
3311 B#(1,I)=A#(1,I)
3312 NEXT I
```

Sets the pointers for the V#(MD+1+ND+1+ND+1+ND+1) array where LOW is the start of the lower bounds, UP is the start of the upper bounds, and CON is the start of the constraints. Initializes the iteration count IT to zero (0). Initializes the pass counter PA, or the number number of tries at finding a pivot column or row, to one (1). Sets the error code ER=0.

```
3313 LOW=ND+1
3314 UP=ND+1+ND+1
3315 CON=ND+1+ND+1+ND+1
3316 IT=0
3317 PA=1
```

Increments the iteration counter by one (1) and returns to the main routine if the limit on the iterations IR is exceeded.

```
3318 IT=IT+1
3319 IF IT<IR THEN 3322
3320 ER=0
3321 RETURN
```

Transforms the linear objective function rather than using the simplex tableau first row transformation.

This step is optional if the size of the LP problem is small or the accuracy required is not important.

```
3322 GOSUB 3550:REM TRAN-OBJ
```

Loads the V#(MD+1+ND+1+ND+1+ND+1) array.

This array contains the first row of the simplex tableau, the constant elements of the transformed lower bound constraints, the constant elements of the transformed upper bound constraints, the constant elements of the transformed LP problems constraints.

```
3323 FOR H=2 TO ND+1
3324 V#(H)=B#(1,H)
3325 NEXT H
3326 FOR I=2 TO ND+1
3327 V#(I+LOW)=B#(I,1)-M#(I,2)
3328 NEXT I
3329 FOR J=2 TO ND+1
3330 IF M#(J,1)<=0# THEN 3334
3331 A#=M#(J,1)
3332 IF A#=BI# THEN A#=0#
3333 V#(J+UP)=A#-B#(J,1)
3334 NEXT J
3335 FOR K=2 TO MD+1
3336 REM-------------------------------------------------------
3337 REM
3338 REM      SUPPORTING PLANE AND DEEP CUT SUBROUTINES
3339 REM
3340 REM-------------------------------------------------------
3341 Z#=0#
3342 B#=-A#(K,1)
3343 FOR J=2 TO ND+1
3344 B#=B#+A#(K,J)*B#(J,1)
3345 Z#=Z#+A#(K,J)*A#(K,J)
3346 NEXT J
3347 SN=R(K)
3348 IF SN=0 THEN SN=-SGN(B#)
3349 P#(K)=CDBL(SN)
3350 V#(K+CON)=B#*P#(K)
3351 IF V#(K+CON)<0# AND Z#<>0# THEN V#(K+CON)=V#(K+CON)/CDBL(SQR(Z#))
3352 NEXT K
```

Searches for a primal pivot using the V#() array to find eligible pivot columns.

```
3353 REM-------------- PRIMAL SIMPLEX --------------------
3354 MI#=-SM#
3355 CO=0
3356 FOR H=2 TO ND+1
3357 IF M#(H,1)=BI# OR MI#<=V#(H) THEN 3360
3358 MI#=V#(H)
3359 CO=H
3360 NEXT H
```

Goes to the dual simplex algorithm if no primal pivot column is found.

```
3361 IF CO=0 THEN 3404
```

Searches for a primal pivot row from the upper and lower bounds and the constraints of the LP problem.

```
3362 PA=PA+1
3363 V#(CO)=0#
3364 MN#=BI#
3365 MA#=0#
3366 RO=0
3367 FOR I=2 TO ND+1
3368 IF V#(I+LOW)<0# OR B#(I,CO)>-SM#/100# THEN 3375
3369 B#=-(V#(I+LOW)/B#(I,CO))
3370 IF MN#<B# THEN 3375
3371 IF MN#=B# AND MA#>-(B#(I,CO)) THEN 3375
3372 MN#=B#
3373 MA#=-(B#(I,CO))
3374 RO=I
3375 NEXT I
3376 FOR J=2 TO ND+1
3377 IF M#(J,1)<=0# OR V#(J+UP)<0# OR -B#(J,CO)>-SM#/100# THEN 3384
3378 B#=-(V#(J+UP)/(-B#(J,CO)))
3379 IF MN#<B# THEN 3384
3380 IF MN#=B# AND MA#>B#(J,CO) THEN 3384
3381 MN#=B#
3382 MA#=B#(J,CO)
3383 RO=J+LOW
3384 NEXT J
3385 FOR K=2 TO MD+1
3386 IF V#(K+CON)<0# THEN 3399
3387 A#=0#
3388 FOR J=2 TO ND+1
3389 A#=A#+A#(K,J)*B#(J,CO)
3390 NEXT J
3391 A#=A#*P#(K)
3392 IF A#>=-SM#/100# THEN 3399
3393 B#=-(V#(K+CON)/A#)
3394 IF MN#<B# THEN 3399
3395 IF MN#=B# AND MA#>-A# THEN 3399
3396 MN#=B#
3397 MA#=-A#
3398 RO=K+UP
3399 NEXT K
```

If no pivot row is found, returns to continue the search for a new pivot column.

```
3400 IF RO=0 THEN 3354
```

Transforms the pivot row constraint and the simplex tableau.

```
3401 GOSUB 3600:REM CONSTRAINT TRANSFORMATION SUBROUTINE
3402 GOSUB 3700:REM INVERSION SUBROUTINE
3403 IF ER=3 THEN RETURN
3404 GOTO 3318
```

Searches for a dual pivot row in the V#() array.

```
3405 REM-----------------DUAL SIMPLEX-------------------------
3406 MI#=-SM#
3407 RO=0
3408 FOR I=2+LOW TO CON+MD+1
3409 IF MI#<=V#(I) THEN 3412
3410 MI#=V#(I)
3411 RO=I-LOW
3412 NEXT I
```

If a dual pivot row is found, searches for a pivot column. Otherwise, returns to main routine

```
3413 IF RO>0 THEN 3417
3414 ER=2
```

Sets error code ER=1 if no column or row has been set ineligible for a pivot at the time the algorithm was terminated. Otherwise, sets ER=2 for an infeasible solution.

```
3415 IF PA=1 THEN ER=1
3416 RETURN
3417 V#(RO+LOW)=0#
3418 PA=PA+1
3419 GOSUB 3600:REM TRANSFORMATION SUBROUTINE
3420 MN#=BI#
3421 MA#=0#
3422 CO=0
3423 FOR I=2 TO ND+1
3424 IF M#(I,1)=BI# OR B#(1,I)<0# OR P#(I)<SM#/100# THEN 3431
3425 B#=(B#(1,I)/P#(I))
3426 IF MN#<B# THEN 3431
3427 IF MN#=B# AND MA#>P#(I) THEN 3431
3428 MN#=B#
3429 MA#=P#(I)
3430 CO=I
3431 NEXT I
3432 IF CO=0 THEN 3405
```

Transforms simplex tableau.

```
3433 GOSUB 3700:REM INVERSION SUBROUTINE
```

```
3434 IF ER=3 THEN RETURN
3435 GOTO 3318
```

Objective Function Transformation -- File TRAN-OBJ

The use of the transformation of the objective function in the primal-dual algorithm is optional since the objective is also transformed as the first row of the simplex tableau. In actual practice, the precision maintained in the simplex tableau is not adequate when solving problems that have many variables or running many iterations between reinversions.

```
3550 REM           * OBJECTIVE TRANSFORMATION SUBROUTINE *
3551 REM-----------------------TRAN-OBJ-------------------------------

3552 B#(1,1)=0#
3553 FOR I=2 TO ND+1
3554 B#(1,I)=0#
3555 B#(1,1)=B#(1,1)+A#(1,I)*B#(I,1)
3556 NEXT I
3557 FOR I=2 TO ND+1
3558 IF A#(1,I)=0# THEN 3562
3559 FOR J=2 TO ND+1
3560 B#(1,J)=B#(1,J)+A#(1,I)*B#(I,J)
3561 NEXT J
3562 NEXT I
3563 RETURN
```

Constraint Transformation Subroutine -- File TRAN-CON

Same as for dual simplex method.

Inversion Subroutine -- File TRAN-INV

Same as for dual simplex method.

Reinversion Subroutine -- File TRAN-RIV

Same as for dual simplex method.

Program Table of Content

Table 6 can be used to reconstruct the primal-dual simplex method

Table 6. Primal-dual method BASIC program table of contents

File	Program lines	Page	Routines
MAIN-BEA	0001-0147	127	Primal-dual method
INPT-OBJ	1200-1220	61	Objective coefficient input subroutine
INPT-CON	1300-1328	63	Constraint coefficient input subroutine
INPT-TYP	1400-1440	65	Constraint type input subroutine
INPT-BND	1500-1523	67	Variable bounds input subroutine
UTIL-OPT	1800-1803	22	Option line subroutine
UTIL-ERS	1850-1853	22	Erase option subroutine
UTIL-CON	1860-1866	22	Continue line subroutine
UTIL-CHX	1870-1882	23	Data check subroutine
REPT-SMP	2200-2224	69	Simplex report subroutine
ALGR-PDS	3300-3433	134	Primal-dual simplex algorithm subroutine
TRAN-OBJ	3550-3563	139	Objective function transformation subr.
TRAN-CON	3600-3633	74	Constraint transformation subroutine
TRAN-INV	3700-3743	76	Basis inversion subroutine
TRAN-RIV	3900-3960	78	Basis reinversion subroutine

from the above computer code and to organize subroutines from previous program listings. Since BASIC code is dependent on program line numbers for its subroutine branching, the statement numbers must be maintained as listed above.

Theory of Beale's Method

Beale's method is a method for finding the optimal solution to a linear set of constraints and a strictly convex quadratic objective function. A key to the method is the manner in which the quadratic objective function is expressed as a set of linear equations.

Beale's Method Tableau and Transformation of Objective Function

As in example 7, let the following example 8 be a two (2) dimensional QP problem where the quadratic function is strictly convex

with all coefficients in lower case letters positive.

minimize $l*X^2+m*Y^2+2*n*X*Y+2*r*X+2*s*Y$

subject to:

$$X \geq 0 \quad (4)$$
$$Y \geq 0 \quad (5)$$
$$c*X+d*Y-e \geq 0 \quad (6)$$
$$g*X+h*Y-f \geq 0 \quad (7)$$

To find the minimum value of the unconstrained quadratic function, the partial derivatives are set equal to zero (0) resulting in a set of simultaneous equations which can be solved for X and Y. In example 8, if the quadratic function:

$$F(X,Y)=l*X^2+m*Y^2+2*n*X*Y+2*r*X+2*s*Y$$

is differentiated with respect to the X variable and set equal to zero (0):

$$\frac{\&F(X,Y)}{\&X}=2*r+2*l*X+2*n*Y=0$$

and with respect to the Y variable and set equal to zero (0):

$$\frac{\&F(X,Y)}{\&Y}=2*s+2*n*X+2*m*Y=0$$

The resulting linear functions, divided by two (2), can be written in tableau form as the second and third row of tableau 13 where s3 and s4

	1	X	Y	
z=	0	r	s	1
s3=	r	l	n	X
s4=	s	n	m	Y

Tableau 13. Beale's method tableau example 8, objective function

are the "slack-surplus" constants indicating the displacement, either

positive or negative, of the current point (0,0) from the minimal unconstrained solution point of the quadratic function and where the first row of the tableau is the transpose of the first column.

Example 8 is constrained, so the objective tableau must be enlarged to include a simplex tableau as in tableau 14. The objective function

	1	X	Y	
z=	0	r	s	
x=	0	1	0	
y=	0	0	1	
s1=	-e	c	d	$\bar{p}'$
z=	0	r	s	1
s3=	r	1	n	X
s4=	s	n	m	Y

Tableau 14. Beale's method augmented simplex tableau example 8

elements of the simplex tableau are taken from the first row of quadratic tableau.

To demonstrate a transformation of the augmented simplex tableau from one current point to another current point with the quadratic tableau appended to the bottom, a typical pivot row will be appended to the tableau; and tableau 14 will be transformed to tableau 16.

The violated constraint is appended to the simplex part of tableau 14 for a pivot row, and a Gauss-Jordan elimination is used to transform the simplex part of the tableau 14 to tableau 15.

	1	s1	Y	
z=	-	-	-	
x=	$\frac{e}{c}$	1	$-\frac{d}{c}$	
y=	0	0	1	
s1=	0	1	0	$\bar{p}'$

Tableau 15. Beale's method simplex tableau example 8, iteration one

The transformation of the quadratic part of the augmented tableau requires a double pivot. Algebraically, the X variable can be replaced by:

$$x=\frac{e}{c}+s1-\frac{d}{c}*Y$$

and substituted into the original quadratic function:

$$F(s1,Y)=(\frac{2*r*e}{c}+\frac{1*e\hat{}2}{c\hat{}2})+1*s1\hat{}2+(\frac{1*d\hat{}2}{c\hat{}2}+m-\frac{2*n*d}{c})*Y+(2*n-\frac{2*1*d}{c})*s1*Y+$$

$$(2*r+\frac{2*1*e}{c})*s1+(2*s-\frac{2*r*d}{c}+\frac{2*n*e}{c}-\frac{2*e*d*1}{c\hat{}2})*Y=z$$

which when differentiated with respect to s1 and set equal to zero (0):

$$\frac{\&F(s1,Y)}{\&S1}=(2*r+\frac{2*1*e}{c})+2*1*s1+(2*n-\frac{2*1*d}{c})*Y=0$$

and with respect to Y and set equal to zero (0):

$$\frac{\&F(s1,Y)}{\&Y}=(2*s-\frac{2*r*d}{c}+\frac{2*n*e}{c})+(2*n-\frac{2*1*d}{c})*s1+(\frac{2*1*d\hat{}2}{c\hat{}2}+2*m-\frac{4*n*d}{c})*Y=0$$

If the objective function is evaluated at the current point $(\frac{e}{c},0)$ then:

$$z=1*\frac{e\hat{}2}{c\hat{}2}+m*0\hat{}2+2*n*e*0+2*r*\frac{e}{c}+2*s*0=1*\frac{e\hat{}2}{c\hat{}2}+2*r*\frac{e}{c}$$

Reconstructing the tableau using the transformed components results in:

	1	s1	Y	
z=	$\frac{2*r*e+l*e^2}{c}\ \frac{}{c^2}$	$\frac{r+l*e}{c}$	$\frac{s-r*d+n*e-e*d*l}{c}\ \frac{}{c}\ \frac{}{c^2}$	
x=	$\frac{e}{c}$	1	$\frac{-d}{c}$	
y=	0	0	1	
s1=	0	1	0	$\bar{p}'$
z=	$\frac{2*r*e+l*e^2}{c}\ \frac{}{c^2}$	$\frac{r+l*e}{c}$	$\frac{s-r*d+n*e-e*d*l}{c}\ \frac{}{c}\ \frac{}{c^2}$	1
s5=	$r+\frac{l*e}{c}$	1	$n-\frac{l*d}{c}$	s1
s6=	$\frac{s-r*d+n*e-e*d*l}{c}\ \frac{}{c}\ \frac{}{c^2}$	$n-\frac{l*d}{c}$	$\frac{l*d^2+m-2*n*d}{c^2}\ \frac{}{c}$	Y

Tableau 16. Beale's method tableau example 8 objective function, iteration one

The transformation of the quadratic tableau can also be done in tableau form with a series row and column operations using the components of the transformed simplex tableau 15. Starting with the current quadratic tableau 17 and using the rules of matrix multiplication to multiply tableau 17 times the augmented $\bar{B}$ inverse of the simplex tableau as shown in tableau 18 results in tableau 20.

	1	X	Y	
z=	0	r	s	1
s3=	r	l	n	X
s4=	s	n	m	Y

Tableau 17. Beale's method tableau example 8 objective function

1	0	0
$\frac{e}{c}$	1	$\frac{-d}{c}$
0	0	1

Tableau 18. Simplex method tableau example 8 basis inverse component, iteration one

$\frac{r*e}{c}$	r	$s-\frac{d*r}{c}$
$\frac{1*e+r}{c}$	1	$n-\frac{d*1}{c}$
$\frac{e*n+s}{c}$	n	$m-\frac{d*n}{c}$

Tableau 19. Beale's method tableau example 8 objective function, iteration one

Tableau 19 is then transposed to tableau 20. Tableau 20 is again matrix multiplied times the augmented $\overline{B}$ inverse resulting in tableau 21. If tableau 21 is compared with tableau 16 which was obtained by algebraic substitution, the two (2) tableau are identical.

$\frac{r*e}{c}$	$\frac{1*e+r}{c}$	$\frac{e*n+s}{c}$
r	1	n
$s-\frac{d*r}{c}$	$n-\frac{d*1}{c}$	$m-\frac{d*n}{c}$

Tableau 20. Beale's method tableau example 8 objective function, iteration one

	1	s1	Y	
z=	$\frac{2*r*e+1*e\hat{}2}{c \quad c\hat{}2}$	$r+\frac{1*e}{c}$	$s-\frac{r*d}{c}+\frac{n*e}{c}-\frac{e*d*1}{c\hat{}2}$	1
s5=	$r+\frac{1*e}{c}$	1	$n-\frac{1*d}{c}$	s1
s6=	$s-\frac{r*d}{c}+\frac{n*e}{c}-\frac{e*d*1}{c\hat{}2}$	$n-\frac{1*d}{c}$	$\frac{1*d\hat{}2}{c\hat{}2}+m-\frac{2*n*d}{c}$	Y

Tableau 21. Beales method tableau example 8 objective function, iteration one

Beale's Method Tableau in Matrix Notation

To expand the two (2) dimensional example 8 and to give a more rigorous definition of the components of the augmented simplex tableau, the QP problem can be written in matrix notation as:

$$\text{maximize} \quad -z$$

$$\text{subject to:} \quad \underset{-\bar{a}+}{z+2*\bar{c}'\bar{x}+\bar{x}'\bar{Q}\bar{x}} = 0$$

$$\underset{\bar{A}\,\bar{x}}{} \geq \bar{0}$$

$$\bar{x} \geq \bar{0}$$

The quadratic objective function can be written as two (2) augmented variable vectors and an augmented $\bar{Q}$ matrix as follows:

$$\left| 1 , \bar{x}' \right| * \left| \begin{array}{c|c} z & \bar{c}' \\ \hline \bar{c} & \bar{Q} \end{array} \right| * \left| \begin{array}{c} 1 \\ \hline \bar{x} \end{array} \right| = z+2*\bar{c}'\bar{x}+\bar{x}'\bar{Q}\bar{x}$$

The transformation for the quadratic function can be written as:

$$(\underline{\bar{Q}\bar{B}}-1)'\underline{\bar{B}}-1=\underline{\bar{B}}-1'\underline{\bar{Q}\bar{B}}-1$$

where $\underline{\bar{Q}}$ and $\underline{\bar{B}}$ are the augmented $\bar{Q}$ and $\bar{B}$ matrices, respectively. This

can also be written as the following matrix operations:

$$\begin{vmatrix} 1 & \bar{B}^{-1}\bar{b}' \\ \bar{0} & \bar{B}^{-1}{}' \end{vmatrix} * \begin{vmatrix} 0 & \bar{c}' \\ \bar{c} & \bar{Q} \end{vmatrix} * \begin{vmatrix} 1 & \bar{0}' \\ \bar{B}^{-1}\bar{b} & \bar{B}^{-1} \end{vmatrix}$$

Or in two (2) steps, the first two (2) matrices are mutiplied and then multiplied again.

$$\begin{vmatrix} \bar{B}^{-1}\bar{b}'\bar{c} & \bar{c}'+\bar{B}^{-1}\bar{b}'\bar{Q} \\ \bar{B}^{-1}\bar{c} & \bar{B}^{-1}\bar{Q} \end{vmatrix} * \begin{vmatrix} 1 & \bar{0}' \\ \bar{B}^{-1}\bar{b} & \bar{B}^{-1} \end{vmatrix}$$

The resulting symmetric matrix is:

$$\begin{vmatrix} \bar{B}^{-1}\bar{b}'\bar{c}+\bar{B}^{-1}\bar{b}'\bar{c}+\bar{B}^{-1}\bar{b}'\bar{Q}\bar{B}^{-1}\bar{b} & \bar{c}'\bar{B}^{-1}+\bar{B}^{-1}\bar{b}'\bar{Q}\bar{B}^{-1} \\ \bar{B}^{-1}\bar{c}+\bar{B}^{-1}\bar{Q}\bar{B}^{-1}\bar{b} & \bar{B}^{-1}\bar{Q}\bar{B}^{-1} \end{vmatrix}$$

where the second half of the final matrix is the inversion of the partial derivatives.

From LP theory:

$$\bar{x}=\bar{B}^{-1}\bar{b}$$

so the value of the objective is:

$$\bar{B}^{-1}\bar{b}'\bar{c}+\bar{B}^{-1}\bar{b}'\bar{c}+\bar{B}^{-1}\bar{b}'\bar{Q}\bar{B}^{-1}\bar{b}=2*\bar{c}'\bar{x}+\bar{x}'\bar{Q}\bar{x}=z$$

which corresponds to the upper left hand corner of the final matrix.

Derivation of Beale's Method

Again, using a two (2) dimensional example 9 which in this case has been simplified further to demonstrate the method, a QP problem can be written as the following set of equations where the quadratic function

minimize $Z(X,Y)=X^2-2*a*X+Y^2$

subject to: $X \geq 0$

$Y \geq 0$

$b*X+b*Y \geq c$

$a,b,c>0$

is strictly convex and the lower case letter are positive coefficients.

Figure 43 is the same example 9 in graphic form.

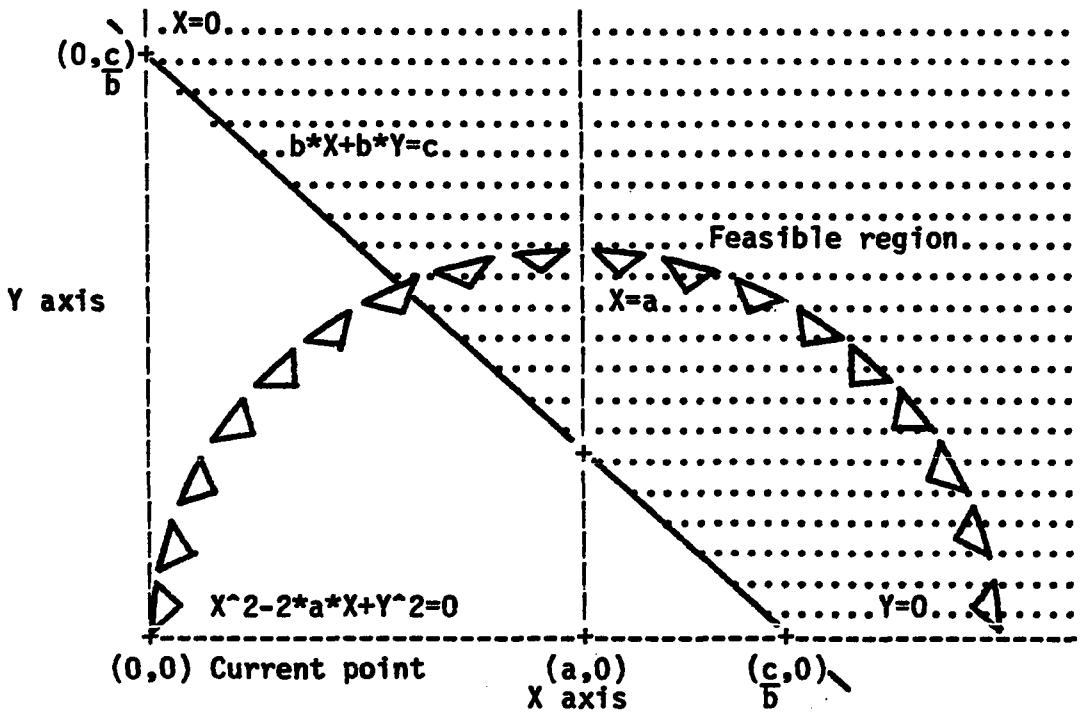


Figure 43. Dual simplex method example 9

As with the out-of-kilter method, the QP problem can be written as a Lagrangian function:

$$F(X,Y,W)=X^2-2a*X+Y^2+W*(c-b*X-b*Y)$$

where Z is the Lagrangian multiplier.

The Kuhn-Tucker conditions sufficient for an optimal solution are:

$$\frac{\&F(X,Y,W)}{\&X}=2*X-2*a-b*W>=0$$

$$\frac{\&F(X,Y,W)}{\&Y}=2*Y-b*W>=0$$

$$\frac{\&F(X,Y,W)}{\&Z}=c-b*X-b*Y<=0$$

$$X*\frac{\&F(X,Y,W)}{\&X}=X*(2*X-2*a-b*W)=0$$

$$Y*\frac{\&F(W,Y,W)}{\&Y}=Y*(2*Y-b*W)=0$$

$$W*\frac{\&F(X,Y,W)}{\&W}=Z*(c-b*X-b*Y)=0$$

Let:

$$s2=2*X-2*a=\frac{\&Z(X,Y)}{\&X}$$

and:

$$s3=2*Y=\frac{\&Z(X,Y)}{\&Y}$$

where s2,s3 are slack-surplus, then:

$$-s2+b*W<=0$$
$$-s3+b*W<=0$$
$$-c+b*X+b*Y>=0$$

$$X*(s2-b*W)=0$$
$$Y*(s3-b*W)=0$$
$$W*(c-b*X-b*Y)=0$$

which are now the Kuhn-Tucker conditions for the LP "subproblem":

minimize s2*X+s3*Y

subject to: X >=0
Y>=0
b*X +b*Y>=c

s2=X-a
s3=Y

s2,s3 free

Primal-dual Method to Solve Beale's LP Subproblem

The LP subproblem can be either, or both, primal and dual infeasible. The subproblem can be solved by a combination of the primal and dual methods described earlier. Although several primal-dual methods are available, such as the criss-cross method or the objective function penalty method, the approach used in the following code is to first use the primal algorithm and then the dual algorithm.

In general, the method starts at the current point (0,0) and uses a two (2) phase search for a pivot column or row with which to transform the augmented simplex tableau to a new point. First, using the primal algorithm, the simplex tableau is searched for a primal pivot column in order of the most negative first row elements; and then the rows of the simplex tableau, as well as the objective tableau, are searched for a primal pivot element. If all possible primal pivot columns fail to have a pivot element, then the dual algorithm is used to search only simplex tableau, in order of most geometric distant constraint form the current point, for a dual pivot row and element. Depending on the pivot found, the augmented simplex tableau is transformed.

The LP subproblem derived from the Kuhn-Tucker conditions can be written as the augmented simplex tableau with one additional row called a "switch" row. Since slack-surplus is not restricted in sign like the simplex variables, the tableau, which has only nonnegative variables, would have to have a negative and positive column to represent the slack-surplus. By using the switch row set to zero (0)

for positive slack and one (1) for negative surplus, only a positive column need to be displayed in the tableau.

Applying the algorithm in greater detail, example 9 can be written as tableau 22. Tableau 22 shows a negative element in the

	1	X	Y	
z=	-	s2=-a	s3=0	
x=	0	1	0	
y=	0	0	1	
s1=	-c	b	b	
s2=	-a	1	0	$\bar{p}'$
z=	0	-a	0	1
s2=	-a	1	0	X
s3=	0	0	1	Y
		0	0	switch

Tableau 22. Beale's method tableau example 9

objective function row. Because the second column of the tableau has all positive pivot elements, the primal subproblem has no eligible pivot rows and does not provide a transformation. (If the third column had a negative first element, the search for primal pivot would continue through the third column.)

Beale's algorithm solves this infeasibility with a pivot row in the quadratic part of the tableau. Recalling that the rows of the quadratic part of the tableau are the equation coefficients for the partials of

the variables which correspond to the quadratic tableau's diagonal elements, the rows of the quadratic tableau are eligible as pivot rows if the element in the diagonal of the quadratic tableau which corresponds to the pivot column is positive. In the case where both the simplex part of the tableau and the quadratic part of the tableau have a possible pivot row, then the row with the minimum absolute value of the ratio of the constant element to the pivot column element is chosen for the pivot row.

By using the quadratic part of the tableau as if the quadratic function were unconstrained, the second row can be used as a pivot row

	1	s2	Y	
z=	-	0	0	
x=	a	1	0	
y=	0	0	1	
s1=	a*b-c	b	b	
s1=	a*b-c	b	b	$\bar{p}'$
z=	$-a^2$	0	0	$\bar{1}$
s4=	0	1	0	s2
s5=	0	0	1	Y
		1	0	switch

Tableau 23. Beale's method tableau example 9, iteration one

for a dual simplex pivot resulting in tableau 23 and the new point shown in figure 44.

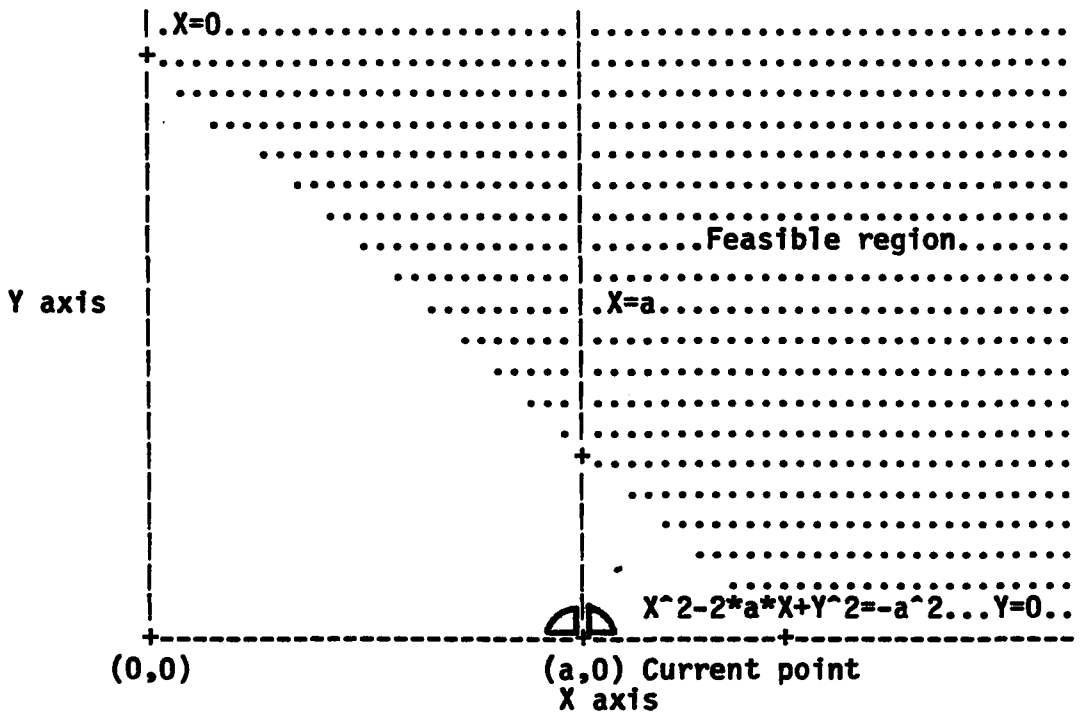


Figure 44. Beale's method example 9, iteration one

The new tableau now has the slack-surplus s2 in column two (2). When the tableau was transformed, s2 entered the transformed constraints, so the column switch was changed from zero (0) to one (1) to indicate that the elements of column two (2) are now of the opposite sign than in the tableau. This change in the switch also allows a higher priority to be assigned to pivoting on columns with slacks-surplus from the quadratic part of the tableau in order to force the partials of the quadratic objective function as close to zero (0) as possible within the restrictions of the feasible region.

A primal pivot column can no longer be found in tableau 23, so a dual simplex pivot is made on the fourth and only primal infeasible row

of the tableau. Two of the elements of the row are positive and both have the same pivot ratio of zero (0). In the dual simplex algorithm, this tie would be broken with the perturbation method; but in this case, the slack-surplus generated by the quadratic tableau is given priority so that column two is selected as the pivot column.

If the tableau were not primal feasible and all possible primal pivots had been eliminated, then only the positive row element in columns headed by a positive element would be candidates for a dual pivot elements.

Tableau 24 is the result of the transformation of the simplex and

	1	s1	Y	
z=	–	$\frac{c-a*b}{b}$	$\frac{a*b-c}{b}$	
x=	$\frac{c}{b}$	1	-1	
y=	0	0	1	
s1=	0	1	0	
s7=	$\frac{a*b-c}{b}$	-1	2	$\bar{p}$:
z=	$\frac{c\hat{}2-2*a*c}{b\hat{}2 \quad b}$	$\frac{c-a*b}{b}$	$\frac{a*b-c}{b}$	1
s6=	$\frac{c-a*b}{b}$	1	-1	s1
s7=	$\frac{a*b-c}{b}$	-1	2	Y
		0	0	switch

Tableau 24. Beale's method tableau example 9, iteration two

and quadratic tableau. In this case the slack s1 is not a free so the switch row element corresponding to the pivot column is set to zero (0).

The result of the pivot is shown in figure 45. The new point $(\frac{c}{b},0)$ is at the intersection of the third constraint and the X axis.

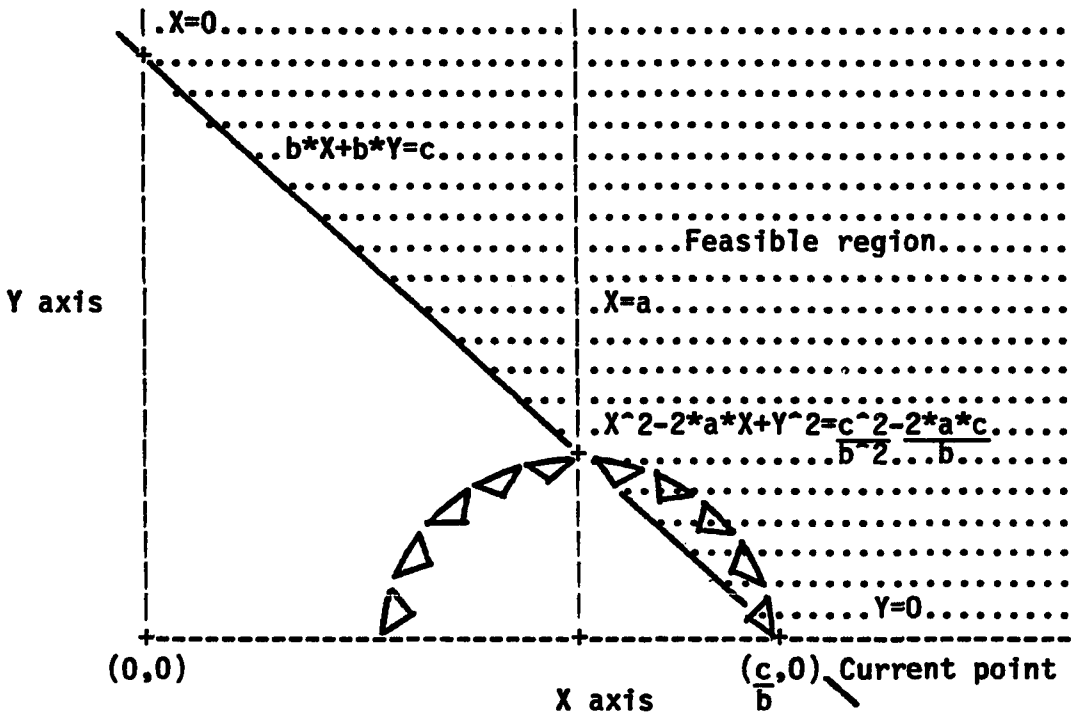


Figure 45. Beale's method example 9, iteration two

The new tableau has a primal pivot. Since $\frac{c-a*b}{b}>0$, $\frac{a*b-c}{b}<0$, the third column of the tableau is chosen for the primal pivot column. Searching the pivot column and assuming that the absolute value of $\frac{a*b-c}{2*b}<\frac{c}{b}$, the third row of the quadratic tableau is selected for the pivot row.

Transforming tableau 24 results in tableau 25.

	1	s1	s7	
z=	–	$\frac{c-a*b}{2*b}$	0	
x=	$\frac{c+a*b}{2*b}$	$\frac{1}{2}$	-1	
y=	$\frac{c-a*b}{2*b}$	$\frac{1}{2}$	1	
s1=	0	0	1	
				$\overline{p}'$
z=	$\frac{c\hat{}2-2*a*b*c-a\hat{}2*b\hat{}2}{2*b\hat{}2}$	$\frac{c-a*b}{2*b}$	0	1
s8=	$\frac{c-a*b}{2*b}$	1	0	s1
s9=	0	0	2	s7
		0	1	switch

Tableau 25. Beale's method tableau example 9, solution

Graphically, the new point $(\frac{c+a*b}{2*b},\frac{c-a*b}{2*b})$ is shown in figure 46. This new point is intersection of the third constraint and the partial of the quadratic function, now expressed in terms of s2 and s1, with respect to s2.

The new tableau is now primal and dual feasible so the subproblem is optimal. The coefficients of the transformed objective function in the subproblem are equal to the partial derivatives of the transformed quadratic with respect to s7 and s1. If the tableau were reconstructed in the original X and Y variables by back substitution, the Kuhn-Tucker

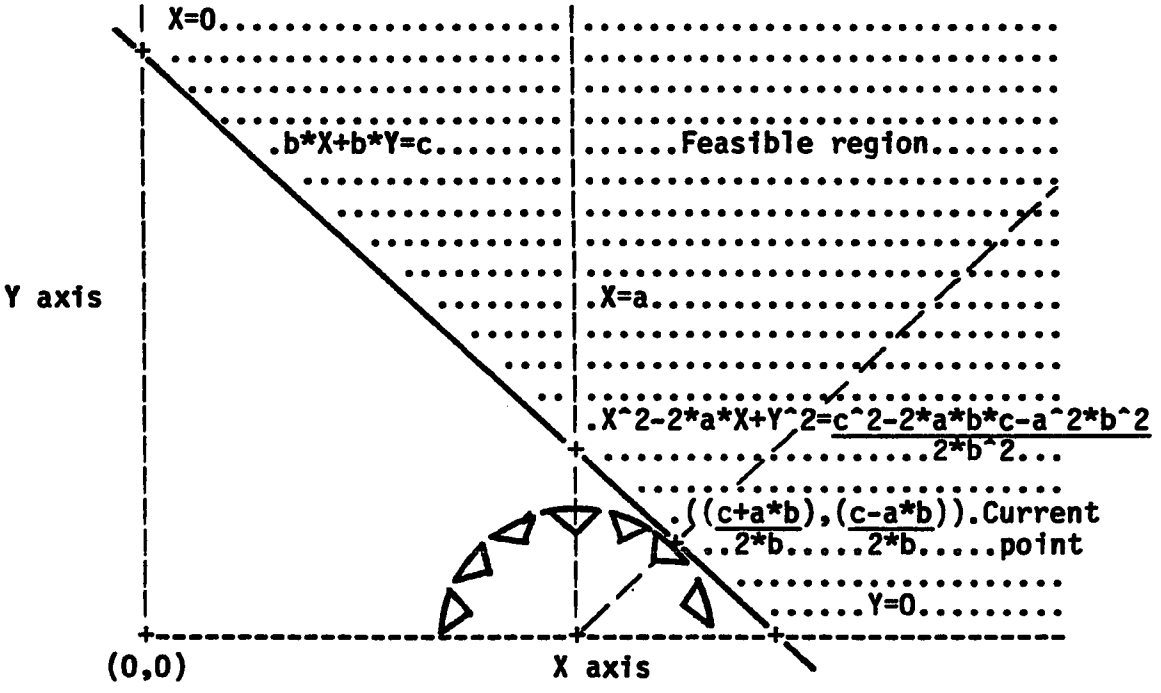


Figure 46. Beale's method example 9, solution

conditions sufficient for the optimal solution of the QP problem would be satisfied by the point $(\frac{c+a*b}{2*b},\frac{c-a*b}{2*b})$.

Beale's Method in Matrix Notation

To expand the two (2) dimension example, the QP problem can be written as:

$$\text{minimize} \quad 2\bar{c}'\bar{x}+\bar{x}'\bar{Q}\bar{x}$$

$$\text{subject to:} \quad \bar{A}\bar{x}\geq\bar{a}$$
$$\bar{x}\geq\bar{0}$$

As before, the Kuhn-Tucker conditions are:

$$2\bar{Q}\bar{x}+2\bar{c}-\bar{A}'\bar{w}\geq\bar{0}$$
$$\bar{a}-\bar{A}\bar{x}\leq\bar{0}$$
$$\bar{x}'*(2\bar{Q}\bar{x}+2\bar{c}-\bar{A}'\bar{w})=0$$
$$\bar{w}'*(\bar{a}-\bar{A}\bar{x})=0$$

Letting:

$$\bar{o}=2\bar{Q}\bar{x}+2\bar{c}$$

the subproblem can be written as:

$$\text{minimize} \quad \bar{o}'\bar{x}$$

$$\text{subject to:} \quad \bar{A}\bar{x}\geq\bar{a}$$
$$\bar{x}\geq\bar{0}$$

and:

$$\bar{o}=\bar{Q}\bar{x}+\bar{c}$$

The complete augmented dual simplex tableau used in the computer program can now be constructed:

			program variables
	0	$\bar{c}'$	Q#(1,ND+1)
	$\bar{c}$	$\bar{Q}$	Q#(2 to ND+1,ND+1)
	0	$2\bar{c}'$	A#(1,ND+1)
	$-\bar{a}$	$\bar{A}$	A#(2 to MD+1,ND+1) (column one negative)
	$-\bar{B}^{-1}\bar{c}$	$\bar{B}^{-1}\bar{Q}$	A#(MD+2 to MD+2+ND,ND+1) (column one negative)
z=	$\bar{B}^{-1}\bar{b}'\bar{c}+\bar{B}^{-1}\bar{b}'\bar{c}+\bar{B}^{-1}\bar{b}'\bar{Q}\bar{B}^{-1}\bar{b}$	$\bar{c}'\bar{B}^{-1}+\bar{B}^{-1}\bar{b}'\bar{Q}\bar{B}^{-1}$	B#(1,ND+1) & V#(ND+1)
x=	$\bar{B}^{-1}\bar{b}$	$\bar{B}^{-1}$	B#(2 to ND+1,ND+1)
	$-\bar{b}$	$\bar{B}$	B#(ND+2 to ND+2+ND,ND+1) (column one negative)
s=	$-a_p+\bar{a}_p'\bar{B}^{-1}\bar{b}$	$\bar{a}_p'\bar{B}^{-1}$	P#(ND+1)
	0	switch	S#(ND+1)

This tableau has an added set of constraints in the constraint rows. These constraints are the result of the first step of the transformation of the quadratic tableau. Because the complete transformation of the quadratic tableau requires one (1) more step or

transformation, these half transformed constraints can be treated in the same manner as the original constraints of the LP problem[45].

For the starting point (0,0), the augmented simplex tableau is initialized to:

			program variables
	0	$\bar{c}'$	Q#(1,ND+1)
	$\bar{c}$	$\bar{Q}$	Q#(2 to ND+1,ND+1)
	0	$2\bar{c}'$	A#(1,ND+1)
	$-\bar{a}$	$\bar{A}$	A#(2 to MD+1,ND+1) (A#(-,1) is negative)
	$\bar{c}$	$\bar{Q}$	A#(MD+2 to MD+2+ND,ND+1)
z=	0	$\bar{c}'$	B#(1,ND+1) & V#(ND+1)
x=	$\bar{0}$	$\bar{I}$	B#(2 to ND+1,ND+1)
	$\bar{0}$	$\bar{I}$	B#(ND+2 to ND+1+ND,ND+1) (B#(-,1) is negative)
s=	$-a_p$	$\bar{a_p}'$	P#(ND+1)
	0	$\bar{0}$	S#(ND+1)

Beale's Method BASIC Code

The BASIC program code presented here is Beale's method modified to solve both primal and dual infeasible models.

All programs listed in the text consist of a main calling routine and a series of input, output, and processing subroutines. These routines are listed in the text as well as on a computer disk compatible with IBM micro-computers.

Beale's Method Main routine -- File MAIN-BEA

Beale's main calling routine (MAIN-BEA) dimensions the eleven (11) data arrays; writes the option menu to the screen as shown in figure 47;

```
                    BEALE'S METHOD

          NUMBER OF CONSTRAINTS          10
          NUMBER OF VARIABLES            14
          MAXIMUM ITERATIONS           1000

                  M-RETURN TO MENU

             O-OBJECTIVE COEFFICIENTS
             A-CONSTRAINT COEFFICIENTS
             C-CONSTRAINT TYPES
             B-BOUNDED VARIABLES
             Q-QUADRATIC COEFFICIENTS
             U-EXECUTE ALGORITHM
             R-REPORT LISTING
             S-SAVE  F-FETCH
             N-NEW PROBLEM

      OPTION ?
                                  O'X+X'QX=z
```

Figure 47. Beale's method main menu screen

calls the utility subroutines UTIL-OPT, UTIL-ERS, UTIL-CON, UTIL-CHX; calls the data input and output subroutines INPT-OBJ, INPT-CON, INPT-TYP, INPT-BND, INPT-QUD, and REPT-SMP; calls and times the processing algorithm ALGR-BEA: and saves and fetches the input data to disk.

```
1 REM                        *BEALES METHOD*
2 REM----------------------------MAIN-BEA-----------------------

3 REM      BI# - MACHINE INFINITE
4 REM      CO  - PIVOT COLUMN FOR SIMPLEX TRANSFORMATION
5 REM      ER  - ERROR KEY
6 REM      IN  - NUMBER OF ITERATIONS BETWEEN REINVERSIONS
7 REM      IR  - MAXIMUM NUMBER OF ITERATIONS
8 REM      MD  - NUMBER OF CONSTRAINTS
9 REM      ND  - NUMBER OF VARIABLES
```

```
10 REM     PA  - NUMBER OF ATTEMPTS AT PIVOT
11 REM     PM  - SIGN KEY (+-)
12 REM     RO  - PIVOT ROW FOR SIMPLEX TRANSFORMATION
13 REM     SM# - MACHINE ZERO
14 REM     A#(MD+2+ND,ND+8) - ORIGINAL DATA AND KEYS
15 REM     B#(2*ND+1,ND+1) - PRIMAL-DUAL MATRIX
16 REM     H#(ND+1) - PAST ITERATION SOLUTION
17 REM     M#(ND+1,2) - UPPER AND LOWER BOUND VALUES
18 REM     P#(ND+2) - WORK VECTOR
19 REM     Q#(ND,ND) - QUADRATIC OBJECTIVE MATRIX
20 REM     R(MD+1) - CONSTRAINT TYPE (1->=,0-=,-1-<=)
21 REM     S#(ND+1) - FREE VARIABLE COLUMN SWITCH
22 REM     T#(ND+1,ND+1) - INVERSION WORK FILE
23 REM     V#(ND+1+ND+1+ND+1+MD+1) - ROW AND COLUMN ARRAY
24 REM     X#(ND) - SOLUTION VECTOR
25 REM ------------------------------------------------------------
```

Sets MD to the default number of constraints in the LP problem and ND to the number of variables. Sets IN to the number of iterations before the reinversion of the augmented $\overline{B}$ basis. Sets the default maximum number of iterations to one thousand. Sets BI# to a number considered machine infinite and SM# to a number considered machine zero.

```
26 MD=0
27 ND=0
28 IN=10
29 IR=1000
30 BI#=1E+10
31 SM#=1E-09
```

Prompts and reads from the keyboard the number of constraints MD in the QP problem, the number of variables ND, and the maximum number of iterations IR allowed before Beale's algorithm is stopped.

```
32 CLS
33 LOCATE 1,10:PRINT "BEALE'S METHOD"
34 LOCATE 3,1:PRINT "NUMBER OF CONSTRAINTS":LOCATE 3,31:INPUT "",L$
35 GOSUB 1870:REM UTIL-CHX
36 IF Z#<>BI# THEN MD=Z#
37 LOCATE 3,30:PRINT MD,"    ":LOCATE 4,1:PRINT "NUMBER OF VARIABLES"
   :LOCATE 4,31:INPUT "",L$
38 GOSUB 1870:REM UTIL-CHX
39 IF Z#<>BI# THEN ND=Z#
40 LOCATE 4,30:PRINT ND,"    ":LOCATE 5,1:PRINT "MAXIMUM ITERATIONS"
   :LOCATE 5,31:INPUT "",L$
41 GOSUB 1870:REM UTIL-CHX
42 IF Z#<>BI# THEN IR=Z#
43 LOCATE 5,30:PRINT IR,"    "
```

Dimensions the array A#(MD+2+ND+1,ND+8) which contains the linear portions of the quadratic objective function, the constraint coefficients, and the first step of the transformation of the quadratic partials. Dimensions the basis inverse and basis array B#(2*ND+1,ND+1). Dimensions the holding array H#(ND+1), the upper and lower bound array M#(ND+1), the pivot row P#(ND+2), the augmented Q matrix array Q#(ND+1,ND+1), the enlarged constraint type array R(MD+1+ND+1), the switch array S#(ND+1), the reinversion working space array T#(ND+1,ND+1), the pivot selection array V#(MD+1+ND+1+ND+1+ND+1), and the solution vector X#(ND).

The array A#(MD+2+ND+1,ND+8) has been modified from the array used in the dual simplex algorithm by adding at each iteration a set of rows to the bottom of the constraint tableau which are the first step of the transformation of the quadratic tableau. These added rows are used as if they are constraints in their original coefficients since the last step in the quadratic transformation is the same as the simple simplex transformation. The constraint type array R(MD+1+ND+1) has been enlarged to include the rows added with the quadratic tableau transformation to the constraint matrix A#(MD+2+ND+1,ND+8). The array V#() is the summary of the first row and the first column, including the phantom upper and lower bound constraints, of the augmented simplex tableau. This allows for a rapid search of the possible primal and then dual pivots.

```
44 DIM A#(MD+2+ND+1,ND+8)
45 DIM B#(2*ND+1,ND+1)
46 DIM H#(ND+1)
47 DIM M#(ND+1,2)
48 DIM P#(ND+1)
49 DIM Q#(ND+1,ND+1)
50 DIM R(MD+1+ND+1)
51 DIM S#(ND+1)
52 DIM T#(ND+1,ND+1)
53 DIM V#(MD+1+ND+1+ND+1+ND+1)
54 DIM X#(ND)
```

Initializes the constraint type array to all greater than or equals.

```
55 FOR I=1 TO MD+1+ND+1
56 R(I)=1
57 NEXT I
```

Prints the option menu to the screen; calls the option line routine UTIL-OPT; and pauses for the entry of "M", "O", "A", "C", "B", "Q", "U", "R", "S", "F", "N" for the option variable L$.

```
58 LOCATE 8,15: PRINT          "M-RETURN TO MENU"
```

```
59 LOCATE 10,10:PRINT       "O-OBJECTIVE COEFFICIENTS"
60 LOCATE 11,10:PRINT       "A-CONSTRAINT COEFFICIENTS"
61 LOCATE 12,10:PRINT       "C-CONSTRAINT TYPES"
62 LOCATE 13,10:PRINT       "B-BOUNDED VARIABLES"
63 LOCATE 14,10:PRINT       "Q-QUADRATIC COEFFICIENTS"
64 LOCATE 15,10:PRINT       "U-EXECUTE ALGORITHM"
65 LOCATE 16,10:PRINT       "R-REPORT LISTING"
66 LOCATE 17,10:PRINT       "S-SAVE F-FETCH"
67 LOCATE 18,10:PRINT       "N-NEW PROBLEM"
68 LOCATE 22,1:PRINT "                  OX'+XQX'=z"
69 REM LOCATE 23,1:PRINT "1*X*2*Y>=c  1*X*(2*Y-3*Z)>=c  1*Y+a*(X-b)^2>=c"
70 GOSUB 1800:REM UTIL-OPT
71 LOCATE 21,8:INPUT "",L$
```

Calls either the objective function input subroutine INPT-OBJ, the constraint input subroutine INPT-CON, the constraint type input subroutine INPT-TYP, the upper and lower bound input subroutine INPT-BND, the quadratic $\overline{Q}$ matrix input subroutine INPT-QUD, the processing subroutine ALGR-BEA, or the report routine REPT-SMP based on the option variable L$.

```
72 CLS
73 H=0
74 G=2
75 IF L$<>"O" THEN 78
76 GOSUB 1200:REM INPT-OBJ
77 GOTO 72
78 IF L$<>"A" THEN 81
79 GOSUB 1300:REM INPT-CON
80 GOTO 72
81 IF L$<>"C" THEN 84
82 GOSUB 1400:REM INPT-TYP
83 GOTO 72
84 IF L$<>"B" THEN 87
85 GOSUB 1500:REM INPT-BND
86 GOTO 72
87 IF L$<>"Q" THEN 90
88 GOSUB 1600:REM INPT-QUD
89 GOTO 72
90 IF L$<>"U" THEN 99
91 TM=3600*VAL(MID$(TIME$,1,2))+60*VAL(MID$(TIME$,4,2))+VAL(MID$(TIME$,7
   ,2))
92 GOSUB 3300:REM ALGR-BEA
```

Sets the optimal solution equal to the current point of Beale's algorithm.

```
93 OB#=B#(1,1)
```

```
94 FOR I=1 TO ND
95 X#(I)=B#(I+1,1)
96 NEXT I
97 TM=3600*VAL(MID$(TIME$,1,2))+60*VAL(MID$(TIME$,4,2))+VAL(MID$(TIME$,7
   ,2))-TM
98 GOTO 58
99 IF L$<>"R" THEN 102
100 GOSUB 2200:REM REPT-SMP
101 GOTO 72
```

Saves the content of MD, ND, M#(ND+1,ND+8), A#(MD+1,ND+8), Q#(ND+1,ND+1), and R(ND+1) to disk file "DATA" as an ASCI file if option "S" is selected.

```
102 IF L$<>"S" THEN 120
103 OPEN "O",#1,"DATA"
104 PRINT #1,STR$(MD)
105 PRINT #1,STR$(ND)
106 FOR I=1 TO ND+1
107 FOR J=1 TO ND+1
108 PRINT #1,STR$(Q#(I,J))
109 NEXT J
110 PRINT #1,STR$(M#(I,1))
111 PRINT #1,STR$(M#(I,2))
112 NEXT I
113 FOR I=1 TO MD+1
114 FOR J=1 TO ND+8
115 PRINT #1,STR$(A#(I,J))
116 NEXT J
117 PRINT #1,STR$(R(I))
118 NEXT I
119 CLOSE #1
```

Loads to MD, ND, M#(ND+1,2), A#(MD+1,ND+8), Q#(ND+1,ND+1), and R(ND+1) the disk file "DATA" if option "F" is selected.

```
120 IF L$<>"F" THEN 146
121 OPEN "I",#1,"DATA"
122 INPUT #1,X$
123 MD=VAL(X$)
124 INPUT #1,X$
125 ND=VAL(X$)
126 FOR I=1 TO ND+1
127 FOR J=1 TO ND+1
128 INPUT #1,X$
129 Q#(I,J)=VAL(X$)
130 NEXT J
131 INPUT #1,X$
```

```
132 M#(I,1)=VAL(X$)
133 INPUT #1,X$
134 M#(I,2)=VAL(X$)
135 NEXT I
136 FOR I=1 TO MD+1
137 FOR J=1 TO ND+8
138 INPUT #1,X$
139 A#(I,J)=VAL(X$)
140 NEXT J
141 INPUT #1,X$
142 R(I)=VAL(X$)
143 NEXT I
144 CLOSE #1
145 GOTO 58
```

Restarts program for new run if option "N" is selected.

```
146 IF L$="N" THEN RUN
147 GOTO 58
```

Utility Subroutines -- Files UTIL-OPT, UTIL-ERS, UTIL-CON, and UTIL-CHX

Same as for critical path method.

Input Subroutines -- Files INPT-OBJ, INPT-CON, INPT-TYP, and INPT-BND

The input subroutines used for the Beale's method are the same as for the dual simplex method. The linear portion of the quadratic objective function is entered through the subroutine INPT-OBJ.

Qaudratic Matrix Subroutine -- INPT-QUD

The program stores the quadratic objective function in two arrays. The linear part of the objective or the $\bar{c}$ vector is stored in the first

$$\left|\ 1\ ,\ \bar{x}'\ \right| * \left|\begin{array}{c|c} z & \bar{c}' \\ \hline \bar{c} & \bar{Q} \end{array}\right| * \left|\begin{array}{c} 1 \\ \hline \bar{x} \end{array}\right| = z+2*\bar{c}'\bar{x}+\bar{x}'\bar{Q}\bar{x}$$

row of the A#(MD+2+ND+1,ND+8) array by the subroutine INPT-OBJ. The Q matrix, which is symmetric and positive definite, is stored in the last ND columns and ND rows of the Q#(ND+1,ND+1) array by the quadratic

matrix subroutine.

The quadratic matrix subroutine (INPT-QUD) is the interactive screen input of the Q#(ND+1,ND+1) array. To reach the screen from the main menu, type "Q" as the OPTION ?. The screen, as shown in figure 48,

```
NON-0 COEFFICIENTS Q MATRIX
  ROW COLUMN  COEFFICIENTS
  5   5       10
  6   6       9
  7   7       8
  8   8       7
  9   9       6
  10  10      5
  11  11      4
  12  12      3
  13  13      2
  14  14      1

OPTION ?
```

Figure 48. Quadratic input

consists of a ROW column, a COLUMN column and the $\overline{Q}$ matrix COEFFICIENT column. When the screen is first entered the cursor "_" will be located in the ROW column. When a $\overline{Q}$ matrix row number or a blank is typed and entered, the cursor will move to the COLUMN column so that the column number of the $\overline{Q}$ matrix can be typed and entered. The cursor will then move to the COEFFICIENT column so that the $\overline{Q}$ matrix entry can be typed and entered.

If an error is made, then the row and column numbers are reentered and the coefficient retyped. To select other options and leave the screen, the option letter is typed in the ROW column and entered.

```
1600 REM              * QUADRATIC INPUT SUBROUTNE *
1601 REM-------------------------INPT-QUD-----------------------------
```

```
1602 LOCATE 1,1:PRINT "NON-ZERO COEFFICIENTS Q MATRIX":LOCATE 2,3:PRINT
     "ROW COLUMN COEFFICIENTS"
1603 H=1
1604 R=0
1605 G=G+1
1606 R=R+1
1607 GOSUB 1800:REM UTIL-OPT
1608 LOCATE G,4:INPUT "",L$
1609 GOSUB 1870:REM UTIL-CHX
1610 IF L$<>"" AND Z#=BI# THEN RETURN
1611 IF Z#<>BI# THEN H=Z#
1612 IF R<=ND THEN 1615
1613 R=1
1614 IF L$="" THEN H=H+1
1615 IF H>ND OR H<=0 THEN 1607
1616 GOSUB 1850:REM UTIL-ERS
1617 LOCATE G,3:PRINT H:LOCATE G,10:INPUT "",L$
1618 GOSUB 1870:REM UTIL-CHX
1619 IF Z#<>BI# THEN R=Z#
1620 IF R>ND OR R<=0 THEN 1617
1621 LOCATE G,9:PRINT R,"  ":LOCATE G,16:PRINT Q#(H+1,R+1):LOCATE G,17
     :INPUT "",L$
1622 GOSUB 1870:REM UTIL-CHX
```

Loads the augmented $\overline{Q}$ matrix of the quadratic objective function.

The array Q#(ND+1,ND+1) is actually the augmented $\overline{Q}$ matrix. When the $\overline{Q}$ matrix elements are entered into the augmented $\overline{Q}$ matrix, the row and column must be offset by one. Also the $\overline{Q}$ matrix is symmetrical so the elements in the transposed positions must be equal.

The linear part of the quadratic objective function is first entered into the first row of the A#(MD+2+ND+1,ND+8) to utilize the simplex program code. These data are transferred in the first steps of the Beale's algorithm to the Q#(ND+1,ND+1) array.

```
1623 IF Z#<>BI# THEN Q#(H+1,R+1)=Z#
1624 IF Z#<>BI# THEN Q#(R+1,H+1)=Z#
1625 LOCATE G,16:PRINT Q#(H+1,R+1),"   "
1626 IF G<18 THEN 1605
1627 GOSUB 1860:REM UTIL-CON
1628 GOTO 1606
```

Report Subroutine -- File REPT-SMP

Same as for dual simplex method.

Beale's Algorithm Subroutine -- File ALGR-BEA

Beale's algorithm first initializes a simplex tableau to represents a current solution point at the origin and the constraints that form the zero axes, a quadratic tableau to represents the quadratic function partials evaluated at the origin, and a switch row set all to zeros (0).

In an iterative fashion, the algorithm then executes the following steps.

Primal Search The first steps follow the primal algorithm as if the augmented simplex tableau were for a LP problem consisting of only the objective function derived from the quadratic tableau and constraints satisfied by the current point.

This approach assumes that the current point is primal feasible satisfying a reduced set of constraints; that a new point can be found by increasing the values of the variables associated with negative elements in the simplex objective function row; and that primal feasiblity can be maintained for the reduced set of constraints.

Using this reduced set of constraints requires a modification to the primal simplex method because of the possibility that in the reduced tableau all columns eligible for a primal pivot will not provide a pivot element under the rules of the primal algorithm. If a column fails to provide a pivot element, then several eligible pivot columns will have to be tried until all eligible columns have been eliminated.

(1) Find all columns from the simplex tableau eligible as a primal pivot column which have a negative element in the first row or for

which the switch row element is set to one (1). If no column can be found and the simplex tableau is primal feasible, then go to step (5). Otherwise, go to step (2).

(2) From the eligible primal pivot columns which have the switch row elements set to one (1), select as a pivot column the column which has the greatest absolute value first row element of all eligible columns that have not failed to provide a pivot element in the current iteration. If this fails, then select from the remaining eligible pivot columns the column with the most negative element in the first row. If no pivot column can be found, then go to step (5). Otherwise, go to step (3).

Geometrically, the priority given to pivoting on the slack-surplus (switch row element set to one (1)) from the partial constraints forces the new point closer to the unconstrained minimum point of the quadratic function.

(3) From the pivot column selected in step (1), allow any satisfied constraint from the LP subproblem to be a candidate for the primal pivot row if the element in both the transformed constraint and the candidate column has the same sign as the first element of the pivot column. (This use of the sign of the first element of the pivot column is another way of taking into account the switch row.) In addition to the candidate constraints, use as a candidate row, the row from the quadratic tableau which corresponds to the diagonal element of the quadratic tableau in the candidate pivot

column if the diagonal element is greater than zero (0).

(4) If more than one transformed constraint or quadratic tableau row qualifies as a pivot row, then select as the pivot row the row with the minimum absolute value of the ratio of the first column element and the pivot element. If no row is found for a pivot element, then remove the current column as a possible pivot column and return to step (2). Otherwise, go to step (8).

Geometrically, the selection of the minimum pivot ratio quarantees that the new point is at an intersection within the bounds of the feasible region of the reduced simplex tableau. If the partial constraint from the quadratic tableau forms an intersection point within the reduced set of constraints and yields the greatest reduction in the objective function, then it is selected for a pivot row.

Dual Search The next two (2) steps assume that the simplex tableau consists only of columns headed by positive elements, and that a new point can be found by satisfying one (1) of the violated constraints of the LP subproblem.

(5) If all columns of the simplex tableau have been eliminated as candidates for primal pivot columns, then find all the violated constraint of the LP subproblem which are eligible for a dual simplex pivot row. If no constraint is violated and the tableau is primal and dual feasible, then set ER=1 and return to the main routine. Otherwise, go to step (6).

(6) From the eligible rows, select the constraint row which is the

greatest geometric distance from the current point. If the simplex tableau is either dual or primal infeasible and no pivot row or column has been found, then set ER=2 and return to main routine. Otherwise, go to step (7).

(7) In the transformed constraint selected in step (6) as a candidate for a dual pivot row, select as a pivot element the row element corresponding to a column which has a switch row element set to one (1) and which has the same sign as the first row element and the minimum absolute value for the ratio of the first row element and the pivot row element. If no such pivot columns exists, then select from the remaining columns with positive first row elements and positive corresponding pivot row elements. If still no pivot element is found, then remove the row as a possible dual pivot row and return to step (6). Otherwise, go to step (8).

Geometrically, the priority given the slack-surplus rather than the slacks of the constraints forces the new point to be within the feasible region of the constraints before moving to the minimum point of the unconstrained quadratic objective function.

(8) At this point, either a primal pivot or a dual pivot has been found. The simplex part of the tableau is transformed using a Gauss-Jordan elimination and the pivot row selected in steps (2),(3), and (4); or steps (6) and (7). Go to step (9).

(9) Set the switch row element to one (1) for the pivot column if the pivot row is from the quadratic tableau. If the pivot row is not

from the quadratic tableau, then set the switch to zero (0). Go to step (10).

(10) The quadratic tableau has yet to be transformed. To transform the quadratic tableau, the tableau is matrix multiplied times the augmented $\overline{B}$ inverse from the transformed simplex tableau. Then, the product matrix is transposed. (This matrix is the constraints stored in the lower half of the A#(MD+2+ND+1,ND+8) array, rows MD+3 to MD+2+ND+1.) The transposed matrix is then matrix multiplied times the augmented $\overline{B}$ matrix again resulting in the transformed quadratic tableau of the augmented $\overline{Q}$ matrix.

(11) The transformed augmented simplex tableau now represents a new point which becomes the current point. Increment the iteration count by one. If the iterations count is greater than the maximum allowed, set ER=0 and go to main routine. Otherwise, go to step (1).

```
3300 REM                  *BEALE'S ALGORITHM*
3301 REM---------------------ALGR-BEA-----------------------------
```

Initializes the augmented simplex tableau to represent the point at the origin and the zero axes or the lower bound of all varibles.

```
3302 FOR I=2 TO 2*ND+1
3303 FOR J=1 TO ND+1
3304 B#(I,J)=0#
3305 NEXT J
3306 NEXT I
3307 FOR I=2 TO ND+1
3308 S#(I)=0#
3309 H#(I)=BI#
3310 B#(I,I)=1#:B#(I,1)=M#(I,2)
3311 B#(ND+I,I)=1#:B#(ND+I,1)=M#(I,2)
3312 B#(1,I)=A#(1,I)
3313 NEXT I
```

Completes the augmented $\overline{Q}$ matrix.

```
3314 FOR I=2 TO ND+1
3315 Q#(1,I)=A#(1,I)/2#
3316 Q#(I,1)=A#(1,I)/2#
3317 NEXT I
```

Sets the pointers for the V#(MD+1+ND+1+ND+1+ND+1) array where LOW is the start of the lower bounds, UP is the start of the upper bounds, CON is the start of the LP subproblem constraints, OBJ is the start of the quadratic partial constraints. Initializes the iteration count IT to zero (0). Initializes the pass counter, or the number of tries at finding a pivot column or row, to one (0). Sets maximum iterations ER=0.

```
3318 LOW=ND+1
3319 UP=ND+1+ND+1
3320 CON=ND+1+ND+1+ND+1
3321 OBJ=ND+1+ND+1+MD+1
3322 IT=0
3323 PA=1
3324 ER=0
```

Increments the iteration counter by one (1) and returns to the main routine if the limit on iterations IR is exceeded.

```
3325 IT=IT+1
3326 IF IT>IR THEN RETURN
```

Transforms the quadratic tableau using the current augmented $\overline{B}$ inverse.

```
3327 GOSUB 3550:REM TRAN-QUD
```

Loads the V#(MD+1+ND+1+ND+1+ND+1) array.

The array V#() contains the first row of the augmented simplex tableau, and the constant elements of the transformed lower bound constraints, the transformed upper bound constraints, the transformed LP constraints.

```
3328 FOR H=2 TO ND+1
3329 V#(H)=B#(1,H)
3330 NEXT H
3331 FOR I=2 TO ND+1
3332 V#(I+LOW)=B#(I,1)-M#(I,2)
3333 NEXT I
3334 FOR J=2 TO ND+1
3335 IF M#(J,1)<=0# THEN 3339
3336 A#=M#(J,1)
3337 IF A#=BI# THEN A#=0#
```

```
3338 V#(J+UP)=A#-B#(J,1)
3339 NEXT J
3340 FOR K=2 TO MD+1
3341 REM------------------------------------------------------------
3342 REM
3343 REM          SUPPORTING PLANE AND DEEP CUT SUBROUTINES
3344 REM
3345 REM
3346 REM------------------------------------------------------------
3347 Z#=0#
3348 B#=-A#(K,1)
3349 FOR J=2 TO ND+1
3350 B#=B#+A#(K,J)*B#(J,1)
3351 Z#=Z#+A#(K,J)*A#(K,J)
3352 NEXT J
3353 V#(K+CON)=0#:IF Z#=0# THEN 3359
3354 SN=R(K)
3355 IF SN=0 THEN SN=-SGN(B#)
3356 P#(K)=CDBL(SN)
3357 V#(K+CON)=B#*P#(K)
3358 IF V#(K+CON)<0# THEN V#(K+CON)=V#(K+CON)/CDBL(SQR(Z#)))
3359 NEXT K
3360 REM-------------- PRIMAL SIMPLEX --------------------
```

Searches for a primal pivot using the V#() array to find dual infeasible columns.

```
3361 MI#=-SM#/10#
3362 CO=0
```

Searches for a primal pivot among columns with switches S#(ND+1) set to one (1).

This gives priority to the slack-surplus from the quadratic objective function.

```
3363 FOR H=2 TO ND+1
3364 IF S#(H)=0# OR M#(H,1)=BI# OR MI#<=-ABS(V#(H)) THEN 3367
3365 MI#=-ABS(V#(H))
3366 CO=H
3367 NEXT H
3368 IF CO>0 THEN 3375
```

Searches for a primal pivot among columns with switches set to zero (0).

```
3369 FOR H=2 TO ND+1
3370 IF S#(H)>0# OR M#(H,1)=BI# OR MI#<=V#(H) THEN 3373
```

```
3371 MI#=V#(H)
3372 CO=H
3373 NEXT H
3374 IF CO=0 THEN 3439
```

Increments the pass switch by one (1).

In order to tell if the algorithm tried a number of pivot columns and failed to find a pivot or if no primal pivot columns were found, a count is kept of the number of times a primal pivot column was found.

```
3375 PA=PA+1
```

Sets PM to the sign of the primal pivot column first row element.

```
3376 PM=SGN(V#(CO))
```

Sets the first row element from V#(ND+1) corresponding to the pivot column to zero (0) so it is no longer eligible for a pivot column.

```
3377 V#(CO)=0#
```

Searches the quadratic tableau for a pivot row.

In this program, the elements of the quadratic tableau are calculated only for the first row and the diagonal elements; and then only if they are in a pivot column or row. To do this, the first half of the transformation of the tableau is done completely and stored in the A#() array. The final transformation is done in the following set of code.

The selection of the pivot row is based on the minimum absolute value of the ratio of the heading of the pivot column and the pivot element. If a tie is found, then, for precision, the largest divisor is used.

```
3378 MN#=BI#
3379 MA#=0#
3380 RO=0
3381 A#=0#
3382 FOR H=2 TO ND+1
3383 A#=A#+A#(MD+1+CO,H)*B#(H,CO)
3384 NEXT H
3385 IF A#<SM#/100# THEN 3389
3386 MN#=ABS(B#(1,CO)/A#)
3387 MA#=A#
3388 RO=OBJ+CO
```

```
     Searches lower bound constraints for a pivot row.

3389 FOR I=2 TO ND+1
3390 IF V#(I+LOW)<0# OR ABS(B#(I,CO))<SM#/100# THEN 3398
3391 IF PM<>SGN(B#(I,CO)) THEN 3398
3392 B#=ABS(V#(I+LOW)/B#(I,CO))
3393 IF MN#<B# THEN 3398
3394 IF MN#=B# AND MA#>ABS(B#(I,CO)) THEN 3398
3395 MN#=B#
3396 MA#=ABS(B#(I,CO))
3397 RO=I
3398 NEXT I

     Searches upper bound constraints for a pivot row.

3399 FOR J=2 TO ND+1
3400 IF M#(J,1)<=0# OR V#(J+UP)<0# OR ABS(B#(J,CO))<SM#/100# THEN 3408
3401 IF PM=SGN(B#(J,CO)) THEN 3408
3402 B#=ABS(V#(J+UP)/B#(J,CO))
3403 IF MN#<B# THEN 3408
3404 IF MN#=B# AND MA#>ABS(B#(J,CO)) THEN 3408
3405 MN#=B#
3406 MA#=ABS(B#(J,CO))
3407 RO=J+ND+1
3408 NEXT J

     Searches LP subproblem constraints for pivot row.

3409 FOR K=2 TO MD+1
3410 IF V#(K+CON)<0# THEN 3424
3411 A#=0#
3412 FOR J=2 TO ND+1
3413 A#=A#+A#(K,J)*B#(J,CO)
3414 NEXT J
3415 A#=A#*P#(K)
3416 IF ABS(A#)<SM#/100# THEN 3424
3417 IF PM<>SGN(A#) THEN 3424
3418 B#=ABS(V#(K+CON)/A#)
3419 IF MN#<B# THEN 3424
3420 IF MN#=B# AND MA#>ABS(A#) THEN 3424
3421 MN#=B#
3422 MA#=ABS(A#)
3423 RO=K+ND+1+ND+1
3424 NEXT K

     Returns to find a new pivot column if no pivot row is found.

3425 IF RO=0 THEN 3361
```

Complete the transformation of the selected pivot row.

```
3426 GOSUB 3600:REM TRAN-CON
```

Set switch to zero (0).

```
3427 S#(CO)=0#
```

Checks accuracy of constraint selected from quadratic tableau by using the symmetry of the quadratic tableau.

The quadratic tableau is symmetric, so a test of precision can be made by calculating the transpose of the transposed column and comparing the elements.

```
3428 IF RO<>OBJ+CO THEN 3436
3429 FOR H=1 TO ND+1
3430 FOR I=2 TO ND+1
3431 P#(H)=P#(H)+A#(H+MD+1,I)*B#(I,RO-OBJ)
3432 NEXT I
3433 P#(H)=P#(H)/2#
3434 NEXT H
```

Sets the switch to one (1) when the pivot row is from the quadratic tableau.

```
3435 S#(CO)=1#
```

Transforms the simplex tableau.

```
3436 GOSUB 3700:REM TRAN-INV
```

Returns to main routine and sets ER=3 if the transformation fails because of numerical accuracy.

```
3437 IF ER=3 THEN RETURN
```

Returns to start next iteration.

```
3438 GOTO 3323
3439 REM----------------DUAL SIMPLEX----------------------
```

Searches for a dual pivot using the V#() array to find primal infeasible constraints.

```
3440 MI#=-SM#
3441 RO=0
3442 FOR I=2+LOW TO CON+MD+1
```

```
3443 IF MI#<=V#(I) THEN 3446
3444 MI#=V#(I)
3445 RO=I-LOW
3446 NEXT I
```

Sets ER=1 if no pivot column or pivot row was found on the first pass through the tableau. Sets ER=2 for an infeasible problem if no pivot column or pivot row was found after more than one pass through the tableau.

```
3447 IF RO>0 THEN 3451
3448 ER=2
3449 IF PA=1 THEN ER=1
3450 RETURN
```

Sets the element of V#() corresponding to the pivot row to zero (0) so it is no longer eligible for a pivot row. Increments pass counter by one (1).

```
3451 V#(RO+LOW)=0#
3452 PA=PA+1
```

Transforms the candidate pivot row.

```
3453 GOSUB 3600:REM TRAN-CON
```

Searches for a pivot column from among dual feasible columns with switchs S#(ND+1) set to one (1).

This gives priority to the slack-surplus from the quadratic objective function.

```
3454 MN#=BI#
3455 MA#=0#
3456 CO=0
3457 FOR I=2 TO ND+1
3458 IF S#(I)=0# OR M#(I,1)=BI# OR ABS(P#(I))<SM#/100# THEN 3466
3459 IF SGN(B#(1,I))<>0 AND SGN(B#(1,I))<>SGN(P#(I)) THEN 3466
3460 B#=B#(1,I)/P#(I)
3461 IF MN#<B# THEN 3466
3462 IF MN#=B# AND MA#>ABS(P#(I))THEN 3466
3463 MN#=B#
3464 MA#=ABS(P#(I))
3465 CO=I
3466 NEXT I
3467 IF CO>0 THEN 3478
```

Searches for a dual feasible column from among the remaining

eligible columns.

```
3468 FOR I=2 TO ND+1
3469 IF S#(I)>0# OR M#(I,1)=BI# OR B#(1,I)<0# OR P#(I)<SM#/100# THEN 3476
3470 B#=B#(1,I)/P#(I)
3471 IF MN#<B# THEN 3476
3472 IF MN#=B# AND MA#>P#(I) THEN 3476
3473 MN#=B#
3474 MA#=P#(I)
3475 CO=I
3476 NEXT I
3477 IF CO=0 THEN 3439
```

Sets the switch row element of the column corresponding to the pivot column to zero (0).

```
3478 S#(CO)=0#
```

Transforms the simplex tableau using the pivot row selected.

```
3479 GOSUB 3700:REM TRAN-INV
```

Returns to main routine and sets ER=3 if the transformation fails because of numerical accuracy.

```
3480 IF ER=3 THEN RETURN
```

Returns to start next iteration.

```
3481 GOTO 3323
```

Quadratic Tableau Transformation -- File TRAN-QUD

The quadratic tableau transformation subroutine (TRAN-QUD) transforms the quadratic tableau in two (2) steps. The first step

$$\begin{vmatrix} 0 & \bar{c}' \\ \bar{c} & \bar{Q} \end{vmatrix}' * \begin{vmatrix} 1 & \bar{0}' \\ \bar{B}^{-1}\bar{b} & \bar{B}^{-1} \end{vmatrix}' = \begin{vmatrix} \bar{B}^{-1}{}'\bar{b}'\bar{c} & \bar{c}'+\bar{B}^{-1}{}'\bar{b}'\bar{Q} \\ \bar{B}^{-1}{}'\bar{c} & \bar{B}^{-1}{}'\bar{Q} \end{vmatrix}$$

transforms the quadratic tableau or the augmented $\bar{Q}$ matrix by matrix multiplying the tableau times the augmented $\bar{B}$ inverse from the transformed simplex tableau and then transposing the resulting matrix. This half transformed and transposed matrix is then stored in the

constraint matrix A#(MD+1+ND+1,ND+8) as if it were a set of constraints in the original variables.

The last step of the transformation is the same as the simplex transformation in which the transformated objective function row used in the simplex tableau on the next iteration is the result of the multiplication of the first row of the half transformed quadratic tableau stored as a constraint row times the augmented $\overline{B}$ inverse.

$$\begin{array}{|c|c|}\hline \overline{B}^{-1}\overline{b}'\overline{c} & \overline{c}'+\overline{B}^{-1}\overline{b}'\overline{Q} \\ \hline \overline{B}^{-1}\overline{c} & \overline{B}^{-1}\overline{Q} \\ \hline \end{array} * \begin{array}{|c|c|}\hline 1 & \overline{0}' \\ \hline \overline{B}^{-1}\overline{b} & \overline{B}^{-1} \\ \hline \end{array} =$$

$$\begin{array}{|c|c|}\hline \overline{B}^{-1}\overline{b}'\overline{c}+\overline{B}^{-1}\overline{b}'\overline{c}+\overline{B}^{-1}\overline{b}'\overline{Q}\overline{B}^{-1}\overline{b} & \overline{c}'\overline{B}^{-1}+\overline{B}^{-1}\overline{b}'\overline{Q}\overline{B}^{-1} \\ \hline \overline{B}^{-1}\overline{c}+\overline{B}^{-1}\overline{Q}\overline{B}^{-1}\overline{b} & \overline{B}^{-1}\overline{Q}\overline{B}^{-1} \\ \hline \end{array}$$

```
3550 REM          * QUADRATIC OBJECTIVE TRANSFORMATION *
3551 REM------------------------TRAN-QUD-------------------------------
```

Multiplies the quadratic tableau times the augmented $\overline{B}$ inverse matrix from the simplex tableau and then transforms the resulting matrix.

```
3552 FOR I=1 TO ND+1
3553 A#(MD+2,I)=Q#(I,1)
3554 FOR J=2 TO ND+1
3555 A#(MD+1+J,I)=0#
3556 A#(MD+2,I)=A#(MD+2,I)+Q#(I,J)*B#(J,1)
3557 NEXT J
3558 FOR J=2 TO ND+1
3559 IF Q#(I,J)=0# THEN 3563
3560 FOR K=2 TO ND+1
3561 A#(MD+1+K,I)=A#(MD+1+K,I)+Q#(I,J)*B#(J,K)
3562 NEXT K
3563 NEXT J
3564 A#(I+MD+1,1)=-A#(I+MD+1,1)
3565 NEXT I
```

Multiplies the first row of the half transformed quadratic matrix

times the augmented B inverse matrix resulting in the objective function row of the transformed dual tableau.

Since the objective function row of the dual tableau was also calculated in the reinversion subroutine TRAN-RIV, it is possible to save computer processing time by dropping lines 3949-3959 from TRAN-RIV.

```
3566 B#(1,1)=-A#(MD+2,1)
3567 FOR J=2 TO ND+1
3568 B#(1,J)=0#
3569 B#(1,1)=B#(1,1)+A#(MD+2,J)*B#(J,1)
3570 NEXT J
3571 FOR J=2 TO ND+1
3572 IF A#(MD+2,J)=0# THEN 3576
3573 FOR K=2 TO ND+1
3574 B#(1,K)=B#(1,K)+A#(MD+2,J)*B#(J,K)
3575 NEXT K
3576 NEXT J
```

Recalculates the objective function row and checks for accuracy.

Since the quadratic tableau is always symmetrical, the first row of the tableau can be calculated and compared to the first column.

```
3577 FOR K=1 TO ND+1
3578 B#(1,K)=B#(1,K)-A#(K+MD+1,1)
3579 FOR J=2 TO ND+1
3580 B#(1,K)=B#(1,K)+A#(K+MD+1,J)*B#(J,1)
3581 NEXT J
3582 B#(1,K)=B#(1,K)/2#
3583 NEXT K
3584 RETURN
```

Constraint Transformation Subroutine -- File TRAN-CON

Same as for dual simplex method.

Inversion Subroutine -- File TRAN-INV

Same as for dual simplex method.

Reinversion Subroutine -- File TRAN-RIV

Same as for dual simplex method.

Program Table of Contents

Table 7 can be used to reconstruct the above computer code from the

Table 7. Beale's BASIC program table of contents

File	Program lines	Page	Routines
MAIN-BEA	0001-0147	160	Beale's method
INPT-OBJ	1200-1220	61	Objective coefficient input subroutine
INPT-CON	1300-1328	63	Constraint coefficient input subroutine
INPT-TYP	1400-1440	65	Constraint type input subroutine
INPT-BND	1500-1523	67	Variable bounds input subroutine
INPT-QUD	1600-1628	166	Quadratic input subroutine
UTIL-OPT	1800-1803	22	Option line subroutine
UTIL-ERS	1850-1853	22	Erase option subroutine
UTIL-CON	1860-1866	22	Continue line subroutine
UTIL-CHX	1870-1882	23	Data check subroutine
REPT-SMP	2200-2224	69	Simplex report subroutine
ALGR-BEA	3300-3481	172	Beale's algorithm subroutine
TRAN-QUD	3550-3584	180	Quadratic tableau tranformation subroutine
TRAN-CON	3600-3633	74	Constraint transformation subroutine
TRAN-INV	3700-3743	76	Basis inversion subroutine
TRAN-RIV	3900-3960	78	Basis reinversion subroutine

computer disk and to organize subroutines from previous program listings. Since BASIC code is dependent on program line numbers for its subroutine branching, the statement numbers must be maintained as listed above.

Beale's Method:
Solutions to Example 10 Minimum Project Cost Problem

Using the previously defined QP problem:

$$\text{minimize} \sum_{\text{all } ij} (cf_{ij}*(R_{ij}-rmc_{ij})\hat{}2+f_{ij}*(T_j-T_i)) + fix.*(T_n-T_0)$$

$$\text{subject to: } T_j-(dur_{ij}-R_{ij})-T_i>=0 \text{ for all } ij$$

$$T_n-T_0<=Dur.$$

$$dur_{ij}-l_{ij}> R_{ij} >dur_{ij}-u_{ij} \text{ for all } ij$$

$$T_i,R_{ij}>=0 \text{ for all } i,ij$$

where:

cf_{ij} - Coefficient for the curvature of the cost for activity ij
rmc_{ij} - Duration reduction of ij at which the variable cost is minimum
f_{ij} - Fixed cost per increment of activity ij potential duration
R_{ij} - Reduction in duration of activity ij
T_i - Node time for node i
u_{ij} - Upper limit for duration of activity ij if less than durij
l_{ij} - Lower limit for duration of activity ij
n - Maximum node number
fix. - Fixed cost per increment of project duration
dur_{ij} - Upper limit or cost reference duration of activity ij
Dur. - Project duration

the ten (10) activity linear cost problem of example 4 can be rewritten

as a QP problem for example 10 as:

```
minimize  10*R5^2+9*R6^2+8*R7^2+7*R8^2+6*R9^2+5*R10^2
             +4*R11^2+3*R12^2+2*R13^2+1*R14^2
           -R5-R6-R7-R8-R9-R10-R11-R12-R13-R14

subject to: T3-(10-R5)-0>=0
            T4-(20-R6)-T2>=0
            T1-(30-R7)-0>=0
            T2-(40-R8)-T1>=0
            T3-(50-R9)-T2>=0
            T4-(60-R10)-T3>=0
            T3-(70-R11)-T1>=0
            T2-(80-R12)-0>=0
            T4-(90-R13)-T1>=0
            T4-(100-R14)-0>=0

                 T4<=Dur.
                 R5<=9
                 R6<=19
                 R7<=29
                 R8<=39
                 R9<=49
                 R10<=59
                 R11<=69
                 R12<=79
                 R13<=89
                 R14<=99

                T1,T2,.....R14>=0
```

and solved by Beale's method.

Minimum Project Cost Problem with Quadratic Cost Functions Solutions

Using Beale's method, the example 10 problem as defined above was solved with the computer algorithm for ten (10) predetermined project duration ranging from one hundred (100) to ten (10) time increments in steps of ten (10) increments. Since the duration of the project is predetermined, the fixed cost fix.*(T_n-T_0), is not included in the example problem objective function.

The results of the computer runs as executed on the Panasonic Sr. Partner are displayed in table 8 which lists for each fixed project duration the number of iterations required to reach a solution, the seconds required to reach a solution, the value of the objective function at the solution, and the value of each variable.

Table 8. Beale's method solutions to the Example 10 minimum project cost problem with quadratic cost functions

Dur.	100	90	80	70	60
Itr.	41	28	31	31	31
Sec.	133	87	103	103	103
Obj.	14780.31524	19121.24837	24378.06271	30551.00826	37640.08503
T1	16.89093	13.71880	10.54666	7.37453	4.20240
T2	49.74721	46.27071	42.79421	39.31772	35.84122
T3	76.36980	70.79996	65.23013	59.66029	54.09046
T4	100	90	80	70	60
R5	0.05	0.05	0.05	0.05	0.05
R6	0.05555	0.05555	0.05555	0.05555	0.05555
R7	13.10906	16.28119	19.45333	22.62546	25.79759
R8	7.14377	7.44808	7.75245	8.05681	8.36117
R9	23.37740	25.47074	27.56408	29.65742	31.75076
R10	36.36980	40.79996	45.23013	49.66029	54.09046
R11	10.52113	12.91883	15.31653	17.71423	20.11193
R12	30.25278	33.72928	37.20578	40.68227	44.15877
R13	6.89093	13.71880	20.54666	27.37453	34.20240
R14	0.50000	10	20	30	40

Table 8. Continued

Dur.	50	40	30	20	10
Itr.	49	37	32	28	44
Sec.	164	122	100	83	138
Obj.	45673.70877	55545.735	68007.735	83069.735	100740.76
T1	0.99999	1.00000	1	1	1
T2	31.43362	25.64	19.64	13.64	7.64
T3	48.14011	39	29	19	9
T4	50	40	30	20	10
R5	0.05	0.05	0.05	0.05	1
R6	1.43362	5.64	9.64000	13.64	17.64
R7	29	29	29	29	29
R8	9.56637	15.36	21.36	27.36	33.36
R9	33.29351	36.64	40.64	44.64	48.64
R10	58.14011	59	59	59	59
R11	22.85988	32	42	52	62
R12	48.56637	54.36	60.36	66.36	72.36
R13	41	51	61	71	81
R14	50	60	70	80	90

MINIMUM PROJECT MAN COUNT PROBLEM WITH HYPERBOLIC AND PARABOLIC MAN COUNT FUNCTIONS

The simplex methods could optimize only problems with linear objective functions and linear constraints. Unfortunately, since most cost functions are nonlinear, the linearity restriction restricted the method to simplified models with limited actual applications.

Beale's method could optimize a quadratic cost problem if all the cost functions could be reduced to a single quadratic objective function. Again, in many cases, costs can not be expressed as a single quadratic function.

The variable costs of most interest to contractors are related to manpower. In estimating the cost of a project, the number of manhours or mandays required to complete an activity determine the fixed direct labor costs. From the mandays is calculated the number of men and days needed for each activity (a hyperbolic function is assumed) to complete the project within the alloted time. The number of men or man count then determine the variable direct labor or supervision costs.

To reduce supervision costs, the schedule which minimizes the total activity man count is generally selected. Using the notation of previous sections, let:

T_i	- Node time for node i
Dur_{ij}	- Duration of activity ij
M_{ij}	- Men assigned to complete activity ij
h_{ij}	- Mandays to complete activity ij
Dur.	- Project duration
n	- Maximum node number

Then, a nonlinear convex programming (CP) problem for the minimum total

activity man count can be written with objective function:

$$\text{minimize} \sum_{\text{all } ij} M_{ij}$$

and schedule constraints:

$$\text{subject to:} \quad T_j - T_i - Dur_{ij} >= 0 \quad \text{for all } ij$$

$$T_n - T_0 <= Dur.$$

and hyperbolic man count constraints:

$$M_{ij} * Dur_{ij} >= h_{ij} \quad \text{for all } ij$$

$$T_i, Dur_{ij}, M_{ij} >= 0 \quad \text{for all } i, ij$$

This constraint set can be reduced further to a set of hyperbolic and hyperbolic of two sheets constraints:

$$\text{subject to:} \quad M_{ij} * (T_j - T_i) >= h_{ij} \quad \text{for all } ij$$

$$T_n - T_0 <= Dur.$$

$$T_i, M_{ij} >= 0 \text{ for all } ij$$

Theory of Supporting Plane

To solve the minimum man count CP problem, the dual simplex method must be modified to include nonlinear convex constraints in the constraint set. In the dual simplex method, a constraint is eligible for a pivot row if it is violated by the simplex current point. Of the violated constraint, the constraint that is the greatest geometric distance from the simplex current point is then selected for the pivot row.

This same procedure can be followed even if the constraint is convex nonlinear. To find if the constraint is "violated", it is a simple matter to substitute the current point's coordinate values into

the nonlinear constraints function. The geometric distance to the nonlinear constraint, assuming the constraint is differentiable, can be calculated iteratively for almost any convex function.

This leaves only the pivot row to be defined. But what ever pivot row is used, it will have to be linear to be compatible with the simplex tableau; it will have to be selected so that any point that satisfies the convex constraint will not violate the pivot row; and it will have to be violated by the simplex current point.

Witzgall's Supporting Plane

In the case of the parabolic constraint, the substitute linear constraint, or "supporting plane" derived from the parabolic constraint which will provide for a pivot row in the simplex method, can be taken directly from the parabolic equation[46].

If a nonlinear constraint can be written in the following format:

$$a_{i0} - \sum_{\text{all } j} a_{ij} * X_{ij} - \sum_{\text{all } k} b_k * \left(\sum_{\text{all } j} (a_{ijk} * X_{ijk})^2 \right) >= 0 \qquad (11)$$

it is parabolic. The parabolic equation has a linear component:

$$a_{i0} - \sum_{\text{all } j} a_{ij} * X_{ij} >= 0 \qquad (12)$$

and k squared components:

$$b_k * \left(\sum_{\text{all } j} (a_{ijk} * X_{ijk})^2 \right) >= 0$$

Geometrically, if a point satisfies the parabolic equation (11) then, since the squared terms can only have a positive value, the point cannot violate the linear component (12) of the parabolic equation. In fact, the linear equation and the parabolic equation share only a single point

and that is the point at which both equations are equal to zero (0).

The convention of the simplex method is to start with the point at the origin; so for the first iteration of the dual simplex method, the linear component of the parabolic equation (12) and the parabolic equation (11) are both violated or both satisfied depending on the value of the constant a_{10}.

The parabolic equation can now be written in vector notation as:

$$a_0 - 2\bar{a}'\bar{x} - \bar{x}'\overline{Qx}$$

or in the simplex tableau as:

$$\left|\begin{array}{c|c|c|c|c} 1 & \bar{x}' & * & a_0 \;|\; -\bar{a}' & * & 1 \\ & & & -\bar{a} \;|\; -\bar{Q} & & \bar{x} \end{array}\right|$$

In this form, a parabolic constraint can be transposed by using the same transformation:

$$(\underline{\overline{QB}}^{-1})'\underline{\bar{B}}^{-1} = \underline{\bar{B}}^{-1}{}'\underline{\bar{Q}}'\underline{\bar{B}}^{-1}$$

as used for the quadratic tableau in Beale's method or:

$$\begin{array}{c|c} 1 & \bar{s} \\ \hline a_0 - 2\bar{a}'\bar{B}^{-1}\bar{b} - \bar{b}'\bar{B}^{-1}{}'\overline{QB}^{-1}\bar{b} \;|\; -\bar{a}'\bar{B}^{-1} - \bar{b}'\bar{B}^{-1}{}'\overline{QB}^{-1} & 1 \\ -\bar{B}^{-1}{}'\bar{a} - \bar{B}^{-1}{}'\overline{QB}^{-1}\bar{b} \;|\; -\bar{B}^{-1}{}'\overline{QB}^{-1} & \bar{s} \end{array}$$

where the linear component of the transformed parabolic is now:

$$\left| a_0 - 2\bar{a}'\bar{B}^{-1}\bar{b} - \bar{b}'\bar{B}^{-1}{}'\overline{QB}^{-1}\bar{b} \;\middle|\; -2\bar{a}'\bar{B}^{-1} - 2\bar{b}'\bar{B}^{-1}{}'\overline{QB}^{-1} \right| = \bar{p}'$$

In the transformed parabolic, the variables of the constraint include slacks set to zero (0) as well as the remaining original variables also set to zero (0). As in the original equation in which the variables are set to zero (0) at the origin, the linear component of

the transformed constraint provides a supporting plane for the simplex method at each iteration.

Parabolic Constraints with Two Variables

The linear component of the parabolic constraint provides a supporting plane from which to derive a pivot row. If a multivariate parabolic must be used, Witzgall's method provides a means of generating supporting planes for the constraint. Unfortunately, this method also requires a substantial amount of tableau space.

Take the two (2) variable parabolic constraint:

$$Y-a*(X-b)^2>=c$$

This constraint when expanded becomes:

$$-a*b^2-c+2*a*b*X+Y-a*X^2>=0$$

which when written in tableau form is:

1	X	Y	
$-a*b^2-c$	$-a*b$	$\frac{1}{2}$	1
$-a*b$	$-a$	0	X
$\frac{1}{2}$	0	0	Y

Tableau 26. Parabolic constraint tableau

with the supporting plane:

$$(a*b^2-c)-a*b*X+\frac{Y}{2}>=0$$

To represent this constraint in the simplex tableau requires nine (9) tableau 26 entries.

By taking advantage of the geometry of two (2) and three (3) dimensional constraints, the tableau size requirements for the two (2) variate constraint can be reduced to three (3) entries and the "strength" of the supporting plane, or the distance of the supporting plane from the current point, can be increased over that obtained in Witzgall's method.

Parabolic, Hyperbolic, and Hyperbolic of Two Sheets Constraints

Three basic types of nonlinear constraints are used in the following problems. The parabolic constraint in the form Y+a*(X-b)^2>=c where the coefficient of Y is one (1), a is always negative and b and c are positive. The hyperbolic constraint in the form Y*X>=c where the coefficients of X and Y are one (1) and c is greater than zero (0). And the hyperbolic of two sheets or "sheet" constraint in the form X*(Y-Z)>=c where the coefficients of X, Y and Z are one (1) and c is greater than zero (0).

Tangent Plane Through a Point on a Differentiable Constraint

For any differentiable simplex constraint, for any point satisfying the constraint as a equality, a plane can be found through the point which is tangent to the constraint. If the constraint is convex, any point that satisfies the constraint will also satisfy the tangent plane; and the tangent plane will meet the requirements of a supporting plane.

To algebraically derive in two (2) dimensions the tangent line at the point (x,y) on the function F(X,Y), take the partials with respect to the X variable and the Y variable. These partials give a parameterized rate of change at the point (x,y) of the function in the X

direction and the Y direction where the parameter in terms of X is:

$$T=X*\frac{\&F(x,y)}{\&X}$$

or in terms of Y is:

$$T=Y*\frac{\&F(x,y)}{\&Y}$$

To find the intersection points of the tangent line and the axes, the parameterized slopes in both directions along the line yield the points:

$$(0,y+T*\frac{\&Y}{\&F(x,y)})$$

and:

$$(x+T*\frac{\&X}{\&F(x,y)},0)$$

which is graphically shown in figure 49.

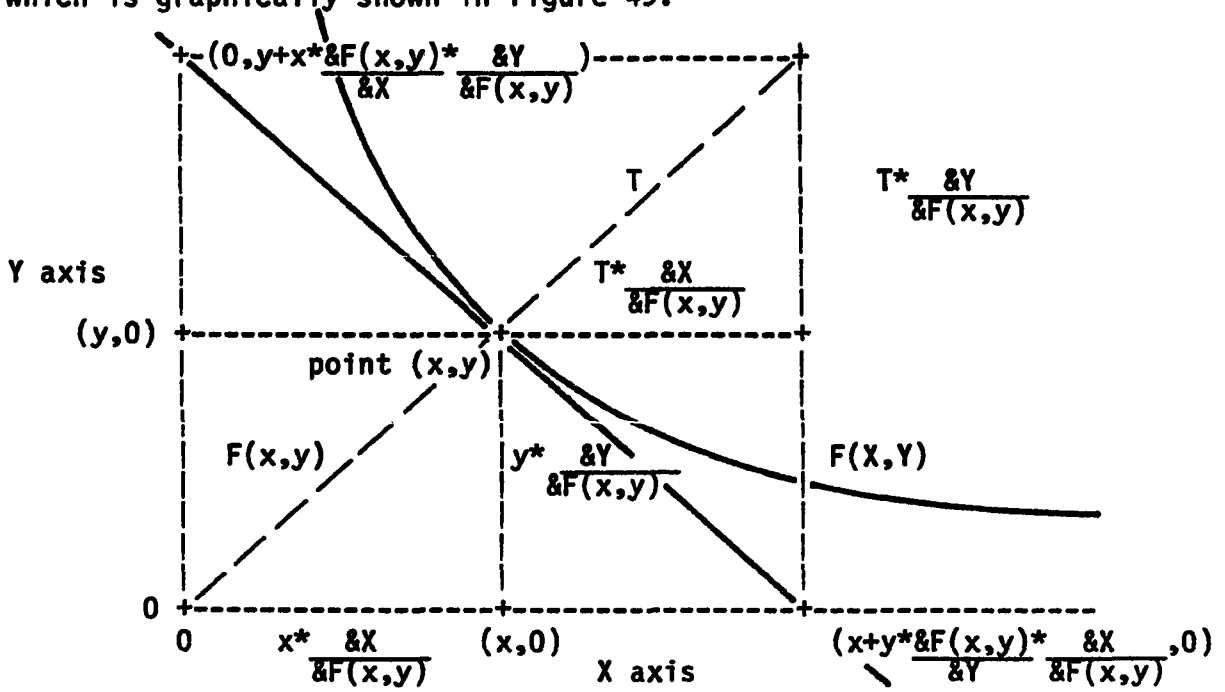


Figure 49. Tangent line to a curve at point (x,y)

The line passing through these two (2) points can be found by solving the two (2) simultaneous equations for i and j[47].

$$i*(x+y*\frac{\&F(x,y)}{\&Y}*\frac{\&X}{\&F(x,y)})+j*(0)=F(x,y)$$

$$i*(0)+j*(y+x*\frac{\&F(x,y)}{\&X}*\frac{\&Y}{\&F(x,y)})=F(x,y)$$

resulting in:

$$i=\frac{F(x,y)}{x+y*\frac{\&F(x,y)}{\&Y}*\frac{\&X}{\&F(x,y)}} \qquad j=\frac{F(x,y)}{y+x*\frac{\&F(x,y)}{\&X}*\frac{\&Y}{\&F(x,y)}}$$

Back substitution gives:

$$\frac{\&F(x,y)}{\&X}*X+\frac{\&F(x,y)}{\&Y}*Y=\frac{\&F(x,y)}{\&X}*x+\frac{\&F(x,y)}{\&Y}*y$$

which can be reduced to:

$$\frac{\&F(x,y)}{\&X}*(X-x)+\frac{\&F(x,y)}{\&Y}*(Y-y)=0$$

If the formula is expanded to three (3) dimensions then:

$$\frac{\&F(x,y,z)}{\&X}*(X-x)+\frac{\&F(x,y,z)}{\&Y}*(Y-y)+\frac{\&F(x,y,z)}{\&Z}*(Z-z)=0$$

Nonlinear Convex Constraints Used in Scheduling Problems

Given the point (x,y,F(x,y)) or (x,y,z), a supporting plane can be found through the point for any differentiable convex constraint. Several constraints are repeatedly required in the following scheduling models and will be defined in more detail.

A Supporting Line Through a Point on a Parabolic Constraint To find the supporting line to the parabolic constraint defined by:

$$Y-a*(X-b)^2>=c,\ Y>=0,\ X>=0,\ a>=0,\ b>=0,\ c>=0$$

the function must first be proven convex. Writing the function as:

$$Y=a*X^2-2*a*b*X+c+a*b^2$$

and differentiated with respect to X:

$$\frac{\&F(X)}{\&X}=2*a*X-2*a*b$$

then if the function is convex, two (2) points on the function ((X',F(X')) and (X",F(X"))) should always lie above or on the tangent line at (X',F(X')) or:

$$F(X")>=F(X')*\frac{\&F(X')}{\&X}*(X"-X')$$

should hold where X"-X'>0. Expanding the equation:

a*X"^2-2*a*b*X"+c+a*b^2>=a*X'^2-2*a*b*X'+c+a*b^2+(2*a*X'-2*a*b)*(X"-X')

and reducing:

$$X"^2>=X'^2-2*X'*X"$$

or using the fact that X'>=0 and X">=0 and dividing through by X"^2:

$$1>=\frac{X'^2}{X"^2}-\frac{2*X'*X"}{X"^2}$$

which is true since:

$$\frac{X'}{X"}<1$$

Using the formula developed above, the supporting line at (x,y) is:

$$(-2*a*x+2*a*b)*(X-x)+(Y-y)>=0$$

or:

$$-2*a*x*X+Y>=2*a*b*x+y$$

A Supporting Line Through a Point on a Hyperbolic Constraint To find a supporting line to the hyperbolic constraint defined by:

$$X*Y>=c,\ X>0,\ Y>0,\ c>0$$

the function must first be proven convex. First, select two points X',Y' and X",Y" such that:

$$X'*Y'=c,\ X"*Y"=c,\ X'>0,\ Y'>0,\ X">0,\ Y">0$$

then any linear combination of the two (2) points on the function must

lie above or on the function if the function is convex. With 0<Q<1 then:

$$(Q*X'+(1-Q)*X")*(Q*Y'+(1-Q)*Y")=$$

$$c*(Q^2+(1-Q)^2+Q*(1-Q)*(\frac{X"*Y'+X'*Y"}{X'*Y'\ X"*Y"}))=$$

$$c*(Q^2+(1-Q)^2+Q*(1-Q)*(\frac{X"+X'}{X'\ X"}))$$

Since by inspection:

$$\frac{X"+X'}{X'\ X"}=A+\frac{1}{A}>=2 \quad 0<A<inf \quad \text{minimum at } A=1$$

then:

$$c*(Q^2+(1-Q)^2+Q*(1-Q)*(\frac{X"+X'}{X'\ X"}))>=c*(Q^2+(1-Q)^2+Q*(1-Q)*2)=c$$

Using the formula developed above, the supporting line at (x,y) is:

$$y*(X-x)+x*(Y-y)>=0$$

or:

$$y*X+x*Y>=2*x*y$$

A Tangent Plane Through a Point on a Hyperbolic of Two Sheets

To find a supporting plane to the sheet constraint the function defined by:

$$X*(Y-Z)=c,\ X>0,\ Y>0,\ Z>=0,\ (Y-Z)>0,\ c>0$$

the function must be proven convex. First, select two (2) points X',Y', Z' and X",Y",Z" such that:

$$X'*(Y'-Z')=c,\ X"*(Y"-Z")=c,\ X'>0,\ Y'>0,\ Z'>=0,\ X">0,\ Y">0,\ Z">=0$$

then any linear combination of the two (2) points on the function must lie above or on the function if the function is convex. With 0<Q<1 then:

$$(Q*X'+(1-Q)*X")*(Q*Y'+(1-Q)*Y"-(Q*Z'+(1-Q)*Z"))=$$

$$(Q*X'+(1-Q)*X")*(Q*(Y'-Z')+(1-Q)*(Y"-Z"))=$$

which can be reduced to:

$$c*(Q^2+(1-Q)^2+Q*(1-Q)*(\frac{X'+X"}{X"\ X'}))>=c$$

as in the hyperbolic case.

Using the formula developed above, the supporting plane at (x,y,z) is:

$$(y-z)*(X-x)+x*(Y-y)-x*(Z-z)>=0$$

or:

$$(y-z)*X+x*Y-x*Z>=2*x*y-2*x*z$$

A Supporting Line Through a Point on a Cubic Hyperbolic Constraint

A variation of two (2) of the above constraints is useful in combining a quadratic objective function with a hyperbolic or sheet constraint. To find a supporting line to the "cubic hyperbolic" constraint defined by the function:

$$X*Y^2=c,\ X>0,\ Y>0,\ c>0$$

the function must first be proven convex. Select two (2) points X',Y' and X",Y" such that:

$$X'*Y'^2=c,\ X"*Y"^2=c,\ X'>0,\ Y'>0,\ X">0,\ Y">0$$

then any linear combination of the two (2) points on the function must lie above or on the function if the function is convex. With 0<Q<1 then:

$$(Q*X'+(1-Q)*X")*(Q*Y'+(1-Q)*Y")^2=$$

$$c*(Q^3+(1-Q)^3+Q*(1-Q)^2*(\frac{X'}{X"}+2*\frac{Y'}{Y"})+Q^2*(1-Q)*(\frac{X"}{X'}+2*\frac{Y"}{Y'}))$$

Now take:

$$\frac{X'}{X"}+2*\frac{Y'}{Y"}=\frac{X'}{X"}+2*\frac{\sqrt{X"}}{\sqrt{X'}}=A+\frac{2}{A}>=3 \quad 0<A<inf \quad \text{minimum at } A=1$$

and by inspection:

$$\frac{X"}{X'}+2*\frac{Y"}{Y'}>=3$$

so that:

$$c*(Q^3+(1-Q)^3+3*Q*(1-Q)^2+Q^2*(1-Q)*3)=c$$

Using the formula developed above, the supporting line at (x,y) is:

$$y^2*(X-x)+2*x*y*(Y-y)>=0$$

or:

$$y^2*X+2*x*y*Y>=3*x*y^2$$

A Supporting Plane Through a Point on a Cubic Sheet Constraint

Another combined constraint is the "cubic sheet" constraint defined by the function:

$$X*(Y-Z)^2=c$$

To prove the function convex, select two (2) points X',Y',Z' and X",Y", Z" such that:

$$X'*(Y'-Z')^2=c,\ X"*(Y"-Z")^2=c,\ X'>0,\ Y'>0,\ Z'>=0,\ X">0,\ Y">0,\ Z">=0$$

then any linear combination of the two (2) points on the function must lie above or on the function if the function is convex. With $0<Q<1$ then:

$$(Q*X'+(1-Q)*X")*(Q*Y'+(1-Q)*Y"-(Q*Z'+(1-Q)*Z"))^2=$$

$$(Q*X'+(1-Q)*X")*(Q*(Y'-Z')+(1-Q)*(Y"-Z"))^2=$$

which can be reduced to:

$$c*(Q^3+(1-Q)^3+Q*(1-Q)^2*(\frac{X'}{X"}+2*\frac{\sqrt{X"}}{\sqrt{X'}})+Q^2*(1-Q)*(\frac{X"}{X'}+2*\frac{\sqrt{X'}}{\sqrt{X"}}))>=c$$

as in the hyperbolic cubic case. Using the formula developed above, the supporting plane at (x,y,z) is:

$$(y-z)^2*(X-x)+(2*x*y-2*x*z)*(Y-y)+(-2*x*y+2*x*z)*(Z-z)>=0$$

or:

$$(y-z)*X+2*x*Y-2*x*Z>=3*x*(y-z)$$

Dual Simplex Method with Nonlinear Constraints BASIC Code

The nonlinear constraint algorithms generate for the simplex methods linear supporting planes to replace the original nonlinear constraint. The supporting plane is stored in the constraint matrix of the simplex tableau like a normal simplex constraint, and the nonlinear

convex constraint is coded into "keys" in the last seven columns of the simplex constraint matrix A#(MD+1,ND+8).

Because a majority of the code is identical to the dual simplex algorithm, most of the routines used in the dual simplex algorithm with nonlinear constraints are taken directly from the dual simplex algorithm.

Dual Simplex Method Main Routine -- File MAIN-SMP

Same as for the dual simplex method with the exception that the constraint matrix A#(MD+1,ND+8) is enlarged by seven (7) columns to accommodate keys for the nonlinear constraints, and the main program of the dual simplex method (MAIN-SMP) is changed to call the constraint keys subroutine by changing line:

79 GOSUB 3000:REM ALGR-KEY

Utility Subroutines -- Files UTIL-OPT, UTIL-ERS, UTIL-CON, and UTIL-CHX

Same as for critical path method.

Input Subroutines -- Files INPT-OBJ, INPT-CON, INPT-TYP, and INPT-BND

Same as for dual simplex method.

To represent the nonlinear constraints in the linear constraint matrix, the following programming conventions have been followed. The parabolic constraint is represented by the linear constraint Y+a*X>=b with c entered as a PARABOLIC constant in constraint type input routine; the hyperbolic constraint is represented by the linear constraint Y+X>=0 with c entered as a HYPERBOLIC constant; and the sheet constraint by the linear constraint X+2*Y-3*Z>=0 with c entered as a SHEET constant.

Report Subroutine -- File REPT-SMP

Same as for dual simplex method.

Constraint Keys Subroutine -- File ALGR-KEY

The constraint keys subroutine (ALGR-KEY) sets the keys as shown in table 9 (the "X" or "Y" represents the column numbers in the $\bar{A}$ matrix in which the X or Y coefficient are located) before the dual simplex algorithm is processed. When the processing ends, the simplex tableau is returned to its original preprocessed form with the keys.

Table 9. Nonlinear constraint key in $\bar{A}$ matrix

	$\bar{A}$ matrix column number						
Constraint type	ND+2	ND+3	ND+4	ND+5	ND+6	ND+7	ND+8
Parabolic	b	0	0	"X"	"Y"	0	c
Hyperbolic	0	c	0	"X"	"Y"	0	0
Sheet	0	0	c	"X"	"Y"	"Z"	0

```
3000 REM                *CONSTRAINT KEYS ALGORITHM*
3001 REM-----------------------ALGR-KEY-----------------------------

3002 FOR I=2 TO MD+1
3003 IF A#(I,ND+2)=0 THEN 3011
3004 A#(I,ND+8)=A#(I,1)
3005 FOR K=2 TO ND+1
3006 IF A#(I,K)<0 THEN A#(I,ND+5)=K-1
3007 IF A#(I,K)>0 THEN A#(I,ND+6)=K-1
3008 IF A#(I,K)<0 THEN A#(I,ND+7)=ABS(A#(I,K))
3009 NEXT K
```

Sets the parabolic constraint keys in columns ND+5,6,7 of the constraint matrix. Column ND+5 is the Y variable number. Column ND+6 is the X variable number. Column ND+7 is the coefficient of the X variable.

```
3010 GOTO 3023
```

```
3011 IF A#(I,ND+3)=0 THEN 3017
3012 FOR K=2 TO ND+1
3013 IF A#(I,K)=1 THEN A#(I,ND+5)=K-1
3014 IF A#(I,K)=2 THEN A#(I,ND+6)=K-1
3015 NEXT K
```

Sets the hyperbolic constraint keys in columns ND+5,6 of the constraint matrix. Column ND+5 is then X variable number. Column ND+6 is then Y variable number.

```
3016 GOTO 3023
3017 IF A#(I,ND+4)=0 THEN 3023
3018 FOR K=2 TO ND+1
3019 IF A#(I,K)=1 THEN A#(I,ND+5)=K-1
3020 IF A#(I,K)=2 THEN A#(I,ND+6)=K-1
3021 IF A#(I,K)=-3 THEN A#(I,ND+7)=K-1
3022 NEXT K
```

Sets the sheet constraint keys in columns ND+5,6,7 of the constraint matrix. Column ND+5 is the X variable number. Column ND+6 is the Y variable number. Column ND+7 is the Z variable number.

```
3023 NEXT I
3024 GOSUB 3300:REM ALGR-SMP;ALGR-BEA;ALGR-GOF;ALGR-CRN;ALGR-GOM
```

Executes the processing algorithm,

```
3025 REM-----------------------------------------------------------
3026 REM GOSUB 3100:REM ALGR-RST;ALGR-BND;ALGR-DBK;ALGR-PRS
3027 REM FOR I=2 TO ND+1
3028 REM M#(I,1)=N%(1,I)
3029 REM M#(I,2)=N%(2,I)
3030 REM NEXT I
```

In the branch and bound algorithms, the upper and lower bound array is used to set subproblem bounds. To restore the original bounds, the first node of the original problem is used to restore the upper and lower bound array to its original values.

```
3031 REM-----------------------------------------------------------
3032 FOR K=2 TO MD+1
3033 IF A#(K,ND+2)=0 THEN 3041
3034 A#(K,1)=A#(K,ND+8)
3035 FOR J=2 TO ND+1
3036 A#(K,J)=0
3037 IF J=A#(K,ND+5)+1 THEN A#(K,J)=-A#(K,ND+7)
3038 IF J=A#(K,ND+6)+1 THEN A#(K,J)=1
3039 NEXT J
```

Reconstructs the original parabolic coefficients from the keys located in ND+5,6,7,8 of the constraint matrix.

```
3040 GOTO 3055
3041 IF A#(K,ND+3)=0 THEN 3048
3042 FOR J=1 TO ND+1
3043 A#(K,J)=0
3044 IF J=A#(K,ND+5)+1 THEN A#(K,J)=1
3045 IF J=A#(K,ND+6)+1 THEN A#(K,J)=2
3046 NEXT J
```

Reconstructs the original hyperbolic coefficients from the keys located in ND+5,6 of the constraint matrix.

```
3047 GOTO 3055
3048 IF A#(K,ND+4)=0 THEN 3055
3049 FOR J=1 TO ND+1
3050 A#(K,J)=0
3051 IF J=A#(K,ND+5)+1 THEN A#(K,J)=1
3052 IF J=A#(K,ND+6)+1 THEN A#(K,J)=2
3053 IF J=A#(K,ND+7)+1 THEN A#(K,J)=-3
3054 NEXT J
```

Reconstructs the original sheet coefficients from the keys located in ND+5,6,7 of the constraint matrix.

```
3055 NEXT K
3056 CLS
3057 RETURN
```

Dual Simplex Algorithm Subroutine -- File ALGR-SMP

Same as for the dual simplex method except that the nonlinear constraint subroutines must be called from the dual simplex algorithm by adding the lines:

```
3333 IF A#(K,ND+2)<>0# THEN GOSUB 4000:REM PAR-TANL
3334 IF A#(K,ND+3)<>0# THEN GOSUB 4200:REM HYP-TANL
3336 IF A#(K,ND+4)<>0# THEN GOSUB 4400:REM SHT-TANL
```

Transformation Subroutines -- Files TRAN-CON, TRAN-INV, and TRAN-RIV

Same as for dual simplex method.

Parabolic Subroutines -- Files PAR-TANA, PAR-TANL, or PAR-TANG

Although an equation for a supporting line was derived in terms of any tangent point to the constraint function, finding the best supporting line which will meet all of the requirements of a pivot row is dependent on finding the best tangent point on the constraint function.

To find a point on the constraint which will result in the strongest possible pivot row requires that the supporting line, or in this case the tangent line, be the greatest possible geometric distance from the simplex current point. This distance is measured by the length along a normal to the tangent line which passes through the simplex current point. The line tangent to the constraint that yields the strongest simplex pivot row for the given nonlinear constraint has as its tangent point, a point on the normal line passing through the simplex current point.

This point can also be shown to correspond to the point on the constraint function that is closest to the simplex current point. This point can be found for some constraints with a closed form solution; but in most cases, an iterative approach is used. Several algorithms will be presented for finding this point, each having advantages in different applications.

Axial Algorithm The axial algorithm for the parabolic constraint starts with a point (b,D) on a line through the axis of the parabolic curve as shown in figure 50. (The constraint is only in two (2)

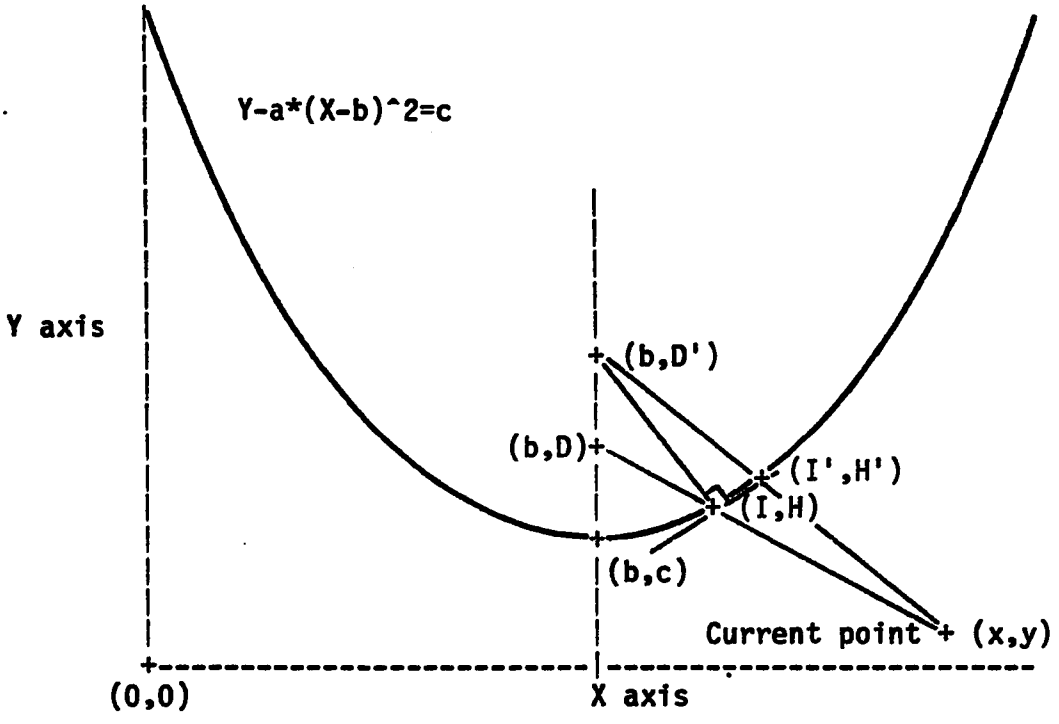


Figure 50. Parabolic constraint supporting line, axial algorithm

dimensions, so only the corresponding dimensions of the simplex current point need be considered.) The equation of the line through the point (b,D) and the current point of the simplex method (x,y) is:

$$X=\frac{(Y*(x-b)+b*y-x*D)}{(y-D)}$$

The intersection of the line and the curve:

$$(Y-c)=a*(X-b)^2$$

can be found by substitution as:

$$Y=\frac{(x-D)^2+K-\sqrt{((x-D)^2+K)^2-4*G*c*(x-D)^2-K^2}}{2*G}=H$$

where:

$$G=a*(x-b)^2$$
$$K=2*D*G$$

Say the point of intersection is the point (I,H). The I is not needed in further calculations so its value is not calculated.

If the point (I,H) were on the normal line from the current point (x,y) to the strongest pivot row or tangent line, then the normal to the tangent line at point (I,H) passing through (I,H) would intersect the axis at (b,D).

To find the actual point of intersection, the parabolic function:

$$F(X,Y)=Y-a*X\hat{}2+2*a*b*X-a*b\hat{}2-c=0$$

is differentiated with respect to X and Y:

$$\frac{\&F(X,Y)}{\&Y}=1 \quad \frac{\&F(X,Y)}{\&X}=-2*a*X+2*a*b$$

and the parametric equation of the normal line through (I,H) is found:

$$Y=H+T \qquad X=I+(2*a*b-2*a*I)*T$$

or rewriting:

$$Y=\frac{X}{(2*a*b-2*a*I)}-\frac{(I-2*a*b*H+2*a*g*H)}{(2*a*b-2*a*I)}$$

The point of intersection of the normal through (I,H) and the axis:

$$X=b$$

is then by substitution:

$$Y=H+\frac{1}{2*a}=D'$$

for a new point (b,D') which is assumed not (b,D) so the search continues.

This leads back to finding the intersection of the line between (b,D') and (x,y) and the parabolic curve by the formula:

$$Y=\frac{(x-D')\hat{}2+K-\sqrt{((x-D')\hat{}2+K)\hat{}2-4*G*c*(x-D')\hat{}2-K\hat{}2}}{2*G}=H'$$

where:

$$G=A*(x-b)^2$$
$$K=2*D'*G$$

If the point (I',H') is as close to point (I,H) as needed by the computer tolerances then the stop at (I'H'). Otherwise, set:

$$H=H'$$

and continue the iteration until a closer tolerance is reached.

Supporting Line for Parabolic Using Axial Algorithm The supporting line for parabolic constraint using the axial algorithm subroutine (PAR-TANA) finds the point on the parabolic constraint, a constraint whose parameters are stored in the last seven (7) columns of the constraint matrix, which is closest to the current simplex point by using the axial algorithm and then constructs the tangent line through the point and loads the line's coefficients into the linear portion of the constraint matrix.

```
4000 REM    * SUPPORTING LINE FOR PARABOLIC USING AXIAL ALGORITHM *
4001 REM-------------------------PAR-TANA-----------------------------

4002 ZX=A#(K,ND+5)+1
4003 ZY=A#(K,ND+6)+1
4004 A#=A#(K,ND+7)
4005 B#=A#(K,ND+2)
4006 IF B#<0# THEN B#=0#
4007 C#=A#(K,ND+8)
4008 Y#=B#(ZY,1)
4009 X#=B#(ZX,1)
4010 IF X#=H#(ZX) AND Y#=H#(ZY) THEN RETURN
4011 IF X#>=0# AND Y#>=0# AND Y#-A#*(X#-B#)^2>=C# THEN RETURN
4012 IF X#<>B# THEN 4015
4013 Y#=C#
4014 GOTO 4027
4015 G#=A#*(B#-X#)*(B#-X#)
4016 D#=C#+1#/(2#*A#)
4017 IF D#<=Y# THEN D#=Y#+1#/(2#*A#)
4018 FOR W=1 TO 50
4019 K#=2#*D#*G#
```

```
4020 H#=(((Y#-D#)^2#+K#)-CDBL(SQR(((Y#-D#)^2#+K#)^2#-4#*G#*C#*(Y#-D#)^2#
     -K#^2#)))/(2#*G#)
4021 IF ABS(D#-H#-1#/(2#*A#))<.00001 THEN 4024
4022 D#=H#+1#/(2#*A#)
4023 NEXT W
4024 Y#=H#
4025 IF Y#<C# THEN Y#=C#
4026 X#=(B#*A#+SGN(X#-B#)*CDBL(SQR(A#*(Y#-C#))))/A#
4027 REM A#(K,1)=Y#+2#*A#*X#*(B#-X#)
4028 A#(K,1)=A#*B#*B#-A#*X#*X#+C#
4029 FOR I=2 TO ND+1
4030 A#(K,I)=0#
4031 IF I=ZX THEN A#(K,I)=2#*A#*(B#-X#)
4032 IF I=ZY THEN A#(K,I)=1#
4033 NEXT I
4034 RETURN
```

Line Search Algorithm The line search algorithm for the parabolic constraint starts with two (2) points on the parabolic curve.

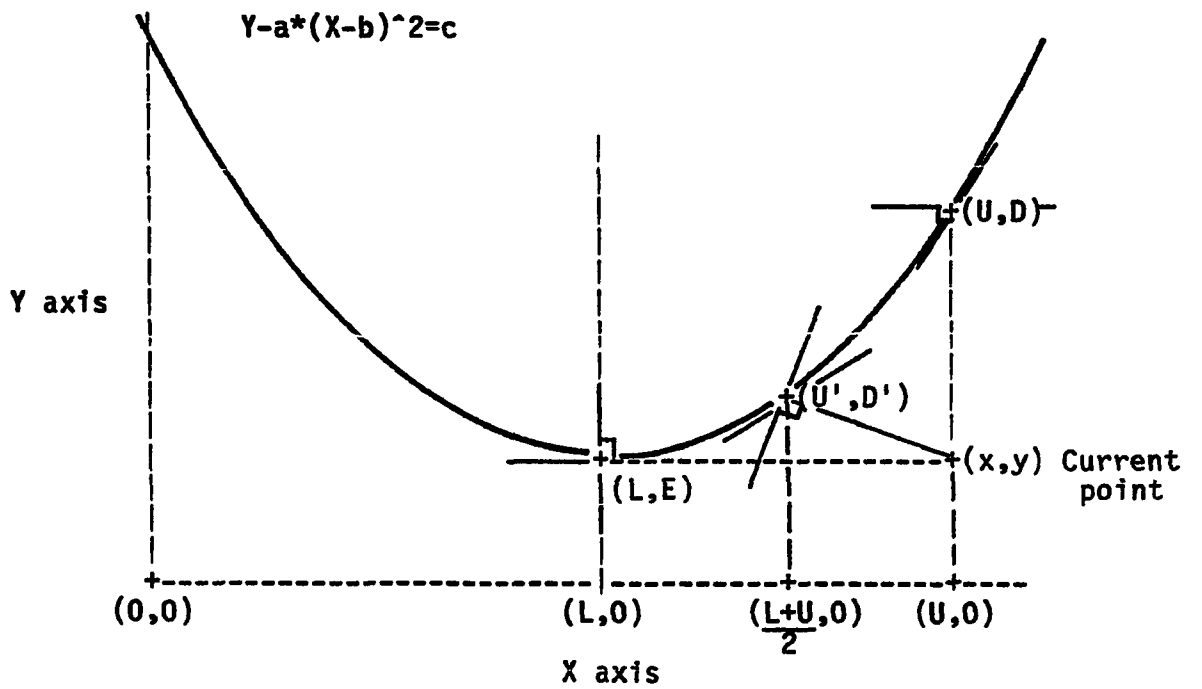


Figure 51. Parabolic constraint supporting line, line search algorithm

The first point (u,d) is the projection of the simplex current point up to the parabolic curve as shown in figure 51 or when:

S=sign(b-x) where s=1 if >0, s=0 if =0 s=-1 if <0

then:

U=x+S*small amount
D=c+a*(x-b)^2

The second point (L,E) is either the apex of the parabolic:

L=b
E=c

or if y greater than c then:

$$L=b+\sqrt{\frac{y-c}{a}}$$
E=y

The algorithm cuts the X axis interval or |U-L| in half and selects as the new set of points the set of points which are on the side of the original interval which reduces the differences in the slopes of the normal line to the curve at the point (U,D) and the line through the points (U,D) and (x,y). To do this, first let G equal the slope of the normal line to the curve at the point (U,D):

$$G=\frac{1}{(2*a*b-2*a*U)}$$

and K equal the slope of the line through (U,D) and (x,y):

$$K=\frac{(D-y)}{(U-x)}=\frac{(a*(U-b)^2+c-y)}{(U-x)}$$

so that:

S=sign(G-K) S=(1,-1)

Then, move to a new point at half interval distance in the direction which reduces the differences in slopes with:

$$U'=\frac{U-(S*|U-L|)}{2}$$

resulting in a new interval on the X axis of only half the width. (After the first iteration of the search, the upper and lower point might not coincide with the U and L variable name.) When the interval |U-L| is within a preset tolerance, then stop the search for the tangent point. Otherwise, repeat the algorithm with another half interval by setting:

L=U
U=U'

Supporting line for Parabolic Using Line Search Algorithm The supporting line for parabolic constraint using the line search algorithm subroutine (PAR-TANL) finds the point on the parabolic constraint, a constraint whose parameters are stored in the last seven (7) columns of the constraint matrix, which is closest to the simplex current point by using the line search algorithm and then constructs the tangent line through the point and loads the line's coefficients into the linear portion of the constraint matrix.

```
4000 REM * SUPPORTING LINE FOR PARABOLIC USING LINE SEARCH ALGORITHM *
4001 REM------------------------PAR-TANL-----------------------------

4002 ZX=A#(K,ND+5)+1
4003 ZY=A#(K,ND+6)+1
4004 A#=A#(K,ND+7)
4005 B#=A#(K,ND+2)
4006 IF B#<0# THEN B#=0#
4007 C#=A#(K,ND+8)
4008 Y#=B#(ZY,1)
4009 X#=B#(ZX,1)
4010 IF X#=H#(ZX) AND Y#=H#(ZY) THEN RETURN
4011 IF X#>=0# AND Y#>=0# AND Y#-A#*(X#-B#)^2>=C# THEN RETURN
4012 U#=X#
4013 IF X#=B# THEN 4026
4014 L#=B#
4015 S#=SGN(B#-X#)
4016 U#=U#+S#*1E-08
```

```
4017 IF Y#>C# THEN L#=B#+S#*CDBL(SQR((Y#-C#)/A#))
4018 FOR W=1 TO 100
4019 IF ABS(U#-L#)<1E-08 THEN 4025
4020 T#=U#
4021 S#=((X#-U#)-2#*A#*(B#-U#)*(Y#-A#*U#*U#+2#*A#*B#*U#-A#*B#*B#-C#))/(2
     #*A#*(B#-U#)*(X#-U#))
4022 U#=U#-((SGN(S#)*ABS(L#-U#))/2#)
4023 L#=T#
4024 NEXT W
4025 X#=U#
4026 Y#=A#*(U#-B#)*(U#-B#)+C#
4027 REM A#(K,1)=Y#+2#*A#*X#*(B#-X#)
4028 A#(K,1)=A#*B#*B#-A#*X#*X#+C#
4029 FOR I=2 TO ND+1
4030 A#(K,I)=0#
4031 IF I=ZX THEN A#(K,I)=2#*A#*(B#-X#)
4032 IF I=ZY THEN A#(K,I)=1#
4033 NEXT I
4034 RETURN
```

Gordian Algorithm A simpler and more efficient means of finding a point through which to pass the tangent line is to simply project up in the Y direction to the curve from the simplex current point. This crude, but simple, means of cutting the "Gordian" knot expands the use of the supporting plane to more complex functions such as the cubics used later. The price that is paid for the simpler algorithm is that the resulting supporting line or pivot row is not the strongest pivot row that can be found for the given constraint.

The algorithm simply evaluates the Y variable based on the x value of the simplex current point. Since the parabolic function never approaches the vertical or infinite slope, there is always a corresponding Y value. Figure 52 shows graphically the projection of the simplex current point onto the parabolic curve.

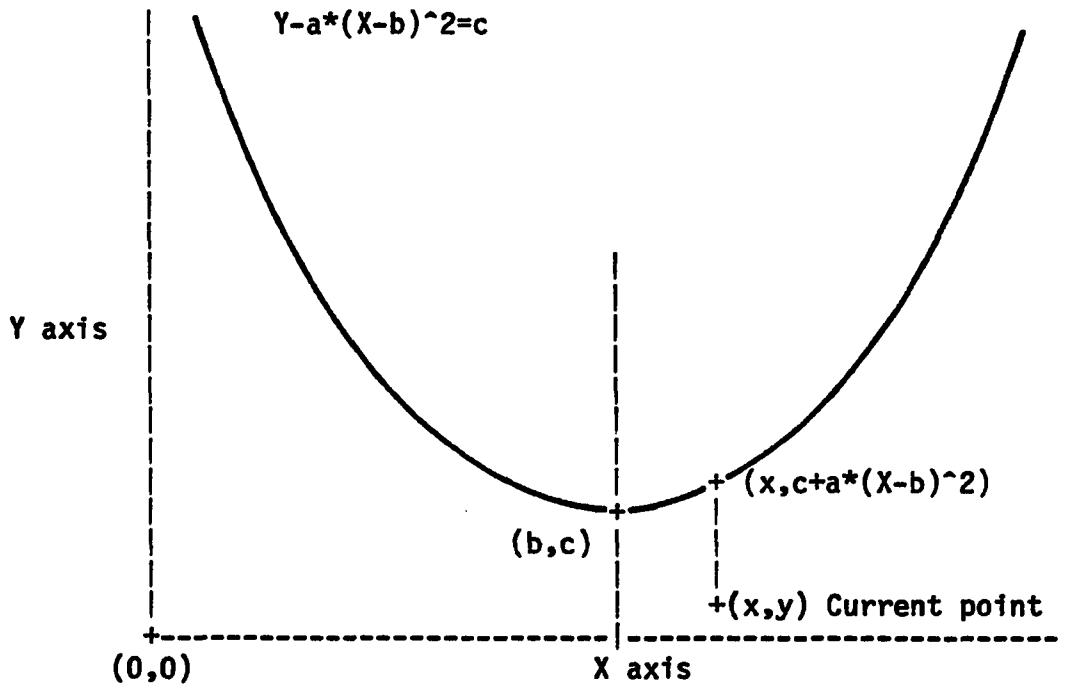


Figure 52. Parabolic constraint supporting line, Gordian algorithm

Supporting line for Parabolic Using Gordian Algorithm The supporting line for parabolic constraint using the Gordian algorithm subroutine (PAR-TANG) finds the point on the parabolic constraint, a constraint whose parameters are stored in the last seven (7) columns of the constraint matrix, which is the vertical projection of X coordinate of the simplex current point on the parabolic function and then constructs the tangent line through the point and loads the line's coefficients into the linear portion of the constraint matrix.

```
4000 REM * SUPPORTING LINE FOR PARABOLIC USING GORDIAN ALGORITHM *
4001 REM-------------------------PAR-TANG----------------------------

4002 ZX=A#(K,ND+5)+1
4003 ZY=A#(K,ND+6)+1
4004 A#=A#(K,ND+7)
```

```
4005 B#=A#(K,ND+2)
4006 IF B#<0# THEN B#=0#
4007 C#=A#(K,ND+8)
4008 Y#=B#(ZY,1)
4009 X#=B#(ZX,1)
4010 IF X#=H#(ZX) AND Y#=H#(ZY) THEN RETURN
4011 IF X#>=0# AND Y#>=0# AND Y#-A#*(X#-B#)^2>=C# THEN RETURN
4012 A#(K,1)=A#*B#*B#-A#*X#*X#+C#
4013 FOR I=2 TO ND+1
4014 A#(K,I)=0#
4015 IF I=ZX THEN A#(K,I)=2#*A#*(B#-X#)
4016 IF I=ZY THEN A#(K,I)=1#
4017 NEXT I
4018 RETURN
```

Hyperbolic Subroutines -- Files HYP-TANA, HYP-TANL, or HYP-TANG

The same three approaches used with the parabolic constraint for finding a tangent point can be used with the hyperbolic constraint. The details of the algorithms are different due to the differences in the functions, but in general the approaches utilize the intersections on an axis line, the distance on the curve between two (2) points, and the projection of the simplex current point to the curve.

Axial Algorithm The axial algorithm first constructs an axis line for the hyperbolic function X*Y>=c which is the line of slope one (1) through the origin. Any line that is the normal to the tangent to the curve that goes through the tangent point would work as an axis, but the forty-five (45) degree line is mathematically the simplest.

On the axis line, select as a starting point the point (D,D) where:

$$D=c+1$$

or:

$$D=x+1$$

if D>x where x is the X coordinate of the current simplex point (x,y)

Graphically, the axes and the point (D,D) are shown in figure 53.

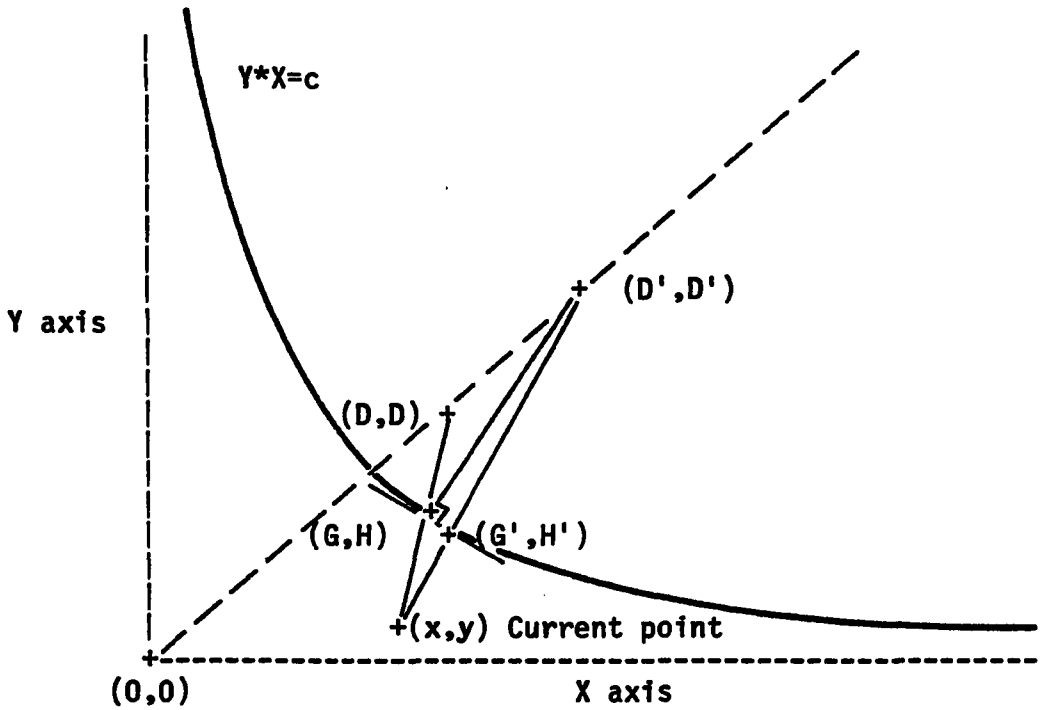


Figure 53. Hyperbolic constraint supporting line, axial algorithm

As with the parabolic function, the line through (d,d) and (x,y) is:

$$Y=\frac{((D-y)*X+D*(y-x))}{(D-x)}$$

and the interection of the line with:

$$Y*X>=c$$

is:

$$X=\frac{D*(x-y)+\sqrt{(D*(y-x))\hat{}2+4*c*(D-x)*(D-y)}}{2*(D-y)}=G$$

or say the point (G,H). If the point (G,H) were on the normal line from the current point (x,y) to the strongest pivot row or tangent line, then the normal to the tangent line at point (G,H) passing through (G,H) would intersect the axis at (D,D).

To find if this is true, the normal line to the tangent of the

function:

$$F(X,Y)=X*Y-c=0$$

differentiated with respect to X and Y:

$$\frac{\&F(X,Y)}{\&X}=Y \qquad \frac{\&F(X,Y)}{\&Y}=X$$

can be written in parametric form for the point (g,h):

$$X=G+H*T \qquad Y=H+G*T$$

or as:

$$Y=\frac{(H\hat{}2+G*X-G\hat{}2)}{H}$$

The intersection of the line with the axis:

$$X=Y$$

is then:

$$X=\frac{(H\hat{}2-G)}{(H+G)}=\frac{(c+G\hat{}2)}{G}=D'$$

for a new point (D',D'), which is assumed not to be (D,D), so the search continues.

This leads back to finding the intersection of the line between (D',D') and (x,y) and the hyperbolic curve by the formula:

$$X=\frac{D'*(x-y)+\sqrt{(D'*(y-x))\hat{}2+4*c*(D'-x)*(D'-y)}}{2*(D'-y)}=G'$$

If the point (G',H') is as close to point (G,H) as needed for the computer tolerances, then the stop at (G'H'). Otherwise, set:

$$G=G'$$

and continue the iteration until a closer tolerance is reached.

Supporting Line for Hyperbolic Using Axial Algorithm The supporting line for the hyperbolic constraint using the axial algorithm

subroutine (HYP-TANA) finds the point on the hyperbolic constraint, a constraint whose parameters are stored in the last seven (7) columns of the constraint matrix, which is closest to the simplex current point by using the axial algorithm and then constructs the tangent line through the point and loads the line's coefficients into the linear portion of the constraint matrix.

```
4200 REM   * SUPPORTING LINE FOR HYPERBOLIC USING AXIAL ALGORITHM *
4201 REM-------------------------HYP-TANA-------------------------------

4202 ZX=A#(K,ND+5)+1
4203 ZY=A#(K,ND+6)+1
4204 C#=A#(K,ND+3)
4205 Y#=B#(ZY,1)
4206 X#=B#(ZX,1)
4207 IF X#=H#(ZX) AND Y#=H#(ZY) THEN RETURN
4208 IF X#>0# AND Y#>0# AND X#*Y#>=C# THEN RETURN
4209 D#=(C#+1#)
4210 IF X#>=D# THEN D#=(X#+1#)
4211 FOR W=1 TO 20
4212 G#=(X#*D#-Y#*D#+SQR(((Y#*D#-X#*D#)^2)+4#*(D#-X#)*(D#-Y#)*C#))/(2#*(
     D#-Y#))
4213 D#=(G#*G#+C#)/G#
4214 NEXT W
4215 A#(K,1)=2#*C#
4216 FOR I=2 TO ND+1
4217 A#(K,I)=0#
4218 IF I=ZX THEN A#(K,I)=C#/G#
4219 IF I=ZY THEN A#(K,I)=G#
4220 NEXT I
4221 RETURN
```

Line Search Algorithm The line search algorithm starts with the current simplex point (x,y) and the point (G,H) on the hyperbolic curve as shown in figure 54 where:

$$G = \sqrt{C}$$

$$H = \sqrt{C}$$

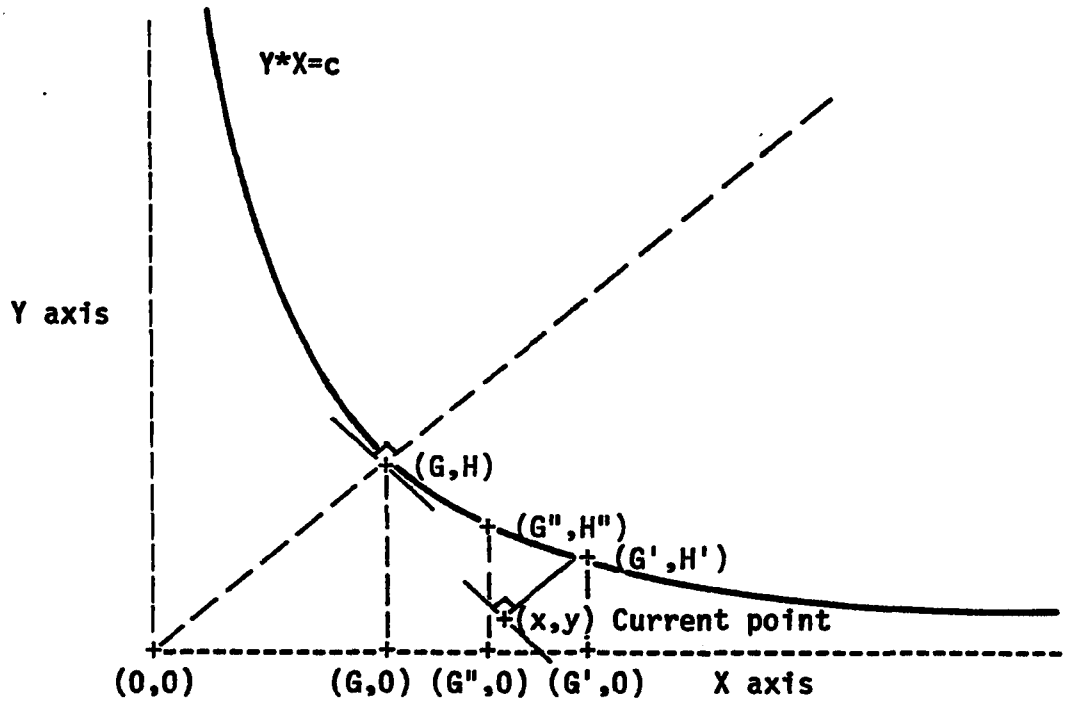


Figure 54. Hyperbolic constraint supporting plane, line search algorithm

The normal to the tangent line at the point (G,H) on the function:

$$F(X,Y)=X*Y-c=0$$

through the point (x,y) can be found by differentiating with respect to X and Y:

$$\frac{\&F(X,Y)}{\&X}=Y \qquad \frac{\&F(X,Y)}{\&Y}=X$$

and written in parametric form as:

$$X=x+H*T \qquad Y=y+G*T$$

or by substitution as:

$$Y=y+\frac{((X-x)*G)}{H}$$

Since:

$$H=\frac{c}{G}$$

then:

$$Y=y+\frac{((X-x)*G\hat{}2)}{c}$$

The intersection of this line passing through the point (x,y) and the hyperbolic function:

$$X*Y=c$$

results in:

$$X=\frac{x*G\hat{}2-y*C+\sqrt{(y*c-x*G\hat{}2)\hat{}2+4*c\hat{}2*G\hat{}2}}{2*G\hat{}2}=G'$$

Saying that the intersection point is (G',H'), if the point (G',H') is as close to point (G,H) as needed for the computer tolerances then stop at (G',H'). Otherwise, a new point (G",H") from which to start the iteration again can be found by averaging the two (2) previous intersections by:

$$G''=\frac{(G'+G)}{2}$$

Supporting Line for Hyperbolic Using Line Search Algorithm The supporting line for the hyperbolic constraint using the line search algorithm subroutine (HYP-TANL) finds the point on the hyperbolic constraint, a constraint whose parameters are stored in the last seven (7) columns of the constraint matrix, which is closest to the simplex current point by using the line search algorithm and then constructs the tangent line through the point and loads the line's coefficients into the linear portion of the constraint matrix.

```
4200 REM * SUPPORTING LINE FOR HYPERBOLIC USING LINE SEARCH ALGORITHM *
4201 REM------------------------HYP-TANL-------------------------------

4202 ZX=A#(K,ND+5)+1
4203 ZY=A#(K,ND+6)+1
4204 C#=A#(K,ND+3)
4205 Y#=B#(ZY,1)
4206 X#=B#(ZX,1)
4207 IF X#=H#(ZX) AND Y#=H#(ZY) THEN RETURN
4208 IF X#>0# AND Y#>0# AND X#*Y#>=C# THEN RETURN
```

```
4209 G#=(C#*X#-C#*Y#+CDBL(SQR(((C#*Y#-C#*X#)^2)+4#*C#*C#*C#)))/(2#*C#)
4210 FOR W=1 TO 50
4211 T#=(G#*G#*X#-C#*Y#+CDBL(SQR(((C#*Y#-G#*G#*X#)^2)+4#*C#*C#*G#*G#)))/
     (2#*G#*G#)
4212 IF ABS(G#-T#)<.0000001 THEN 4215
4213 G#=(G#+T#)/2#
4214 NEXT W
4215 A#(K,1)=2#*C#
4216 FOR I=2 TO ND+1
4217 A#(K,I)=0#
4218 IF I=ZX THEN A#(K,I)=C#/G#
4219 IF I=ZY THEN A#(K,I)=G#
4220 NEXT I
4221 RETURN
```

Gordian Algorithm The Gordian algorithm for the hyperbolic constraint projects the current simplex point (x,y) onto the hyperbolic curve in either the X or Y direction depending on the location of the current dual simplex point. The hyperbolic function asymptotically approaches both the X and Y axes, so trying to project the simplex point onto the curve in only one (1) direction would not result in a point on the curve in all cases. To solve this problem, the region of points violating the constraint is divided into three (3) regions as shown in figure 55. If the simplex current point lies below the line $Y=\frac{\sqrt{c}}{2}$ and is less than x then the point ($\sqrt{c}$, $\sqrt{c}$) is always use the point through which the tangental line passes. If the current dual simplex point lies above the line or on the line, and $y>=\frac{\sqrt{c}}{2}$, then point on the curve to be used as the tangent point is found by the projection in the Y direction. Otherwise, the projection in the X direction is used.

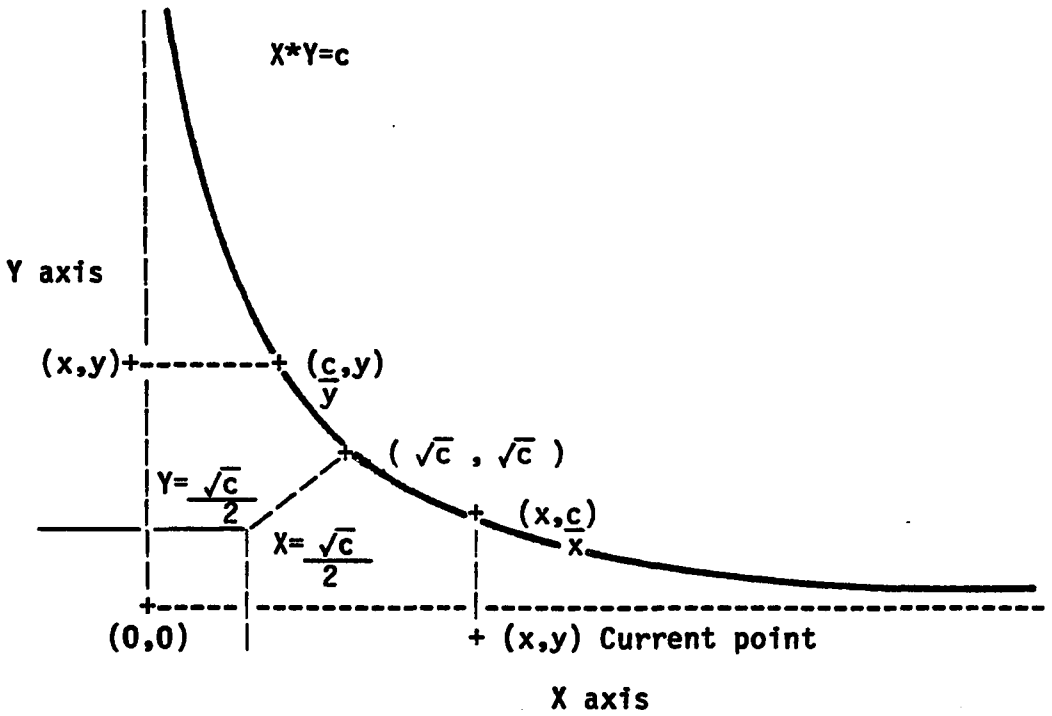


Figure 55. Hyperbolic constraint supporting line, Gordian algorithm

Supporting Line for Hyperbolic Using Gordian Algorithm The supporting line for the hyperbolic constraint using the Gordian algorithm subroutine (HYP-TANG) finds the point on the hyperbolic constraint, a constraint whose parameters are stored in the last seven (7) columns of the constraint matrix, by using the Gordian algorithm and then constructs the tangent line through the point and loads the line's coefficients into the linear portion of the constraint matrix.

```
4200 REM * SUPPORTING LINE FOR HYPERBOLIC USING GORDIAN ALGORITHM *
4201 REM----------------------HYP-TANG-----------------------------

4202 ZX=A#(K,ND+5)+1
4203 ZY=A#(K,ND+6)+1
4204 C#=A#(K,ND+3)
4205 Y#=B#(ZY,1)
```

```
4206 X#=B#(ZX,1)
4207 IF X#=H#(ZX) AND Y#=H#(ZY) THEN RETURN
4208 IF X#>0# AND Y#>0# AND X#*Y#>=C# THEN RETURN
4209 G#=CDBL(SQR(C#))
4210 IF Y#>X# THEN GOTO 4219
4211 IF X#>=G#/2# THEN G#=X#
4212 A#(K,1)=2#*G#*C#
4213 FOR L=2 TO ND+1
4214 A#(K,L)=0#
4215 IF ZX=L THEN A#(K,L)=C#
4216 IF ZY=L THEN A#(K,L)=G#*G#
4217 NEXT L
4218 RETURN
4219 IF Y#>=G#/2# THEN G#=Y#
4220 A#(K,1)=2#*C#
4221 FOR L=2 TO ND+1
4222 A#(K,L)=0#
4223 IF ZX=L THEN A#(K,L)=G#
4224 IF ZY=L THEN A#(K,L)=C#/G#
4225 NEXT L
4226 RETURN
```

Hyperbolic Sheet Subroutine -- Files SHT-TANA, SHT-TANG, or SHT-TANG

The sheet constraint adds a third dimension to the search for a point on the surface which will result in the strongest pivot row. As before, the point on the surface which is closest to the simplex current point will provide that point.

A modification of three approaches used previously for the hyperbolic function will work for the sheet constraint. This can be attributed to the fact that the sheet function is linear in the Y and Z axes plane and hyperbolic like only in the X and Y axes plane.

Axial Algorithm The axial algorithm first constructs an axis line. In three (3) dimensions, this line is the intersection of the plane formed by the forty-five (45) degree axis of all the hyperbolic like sections in X and Y axes plane and the plane through the simplex

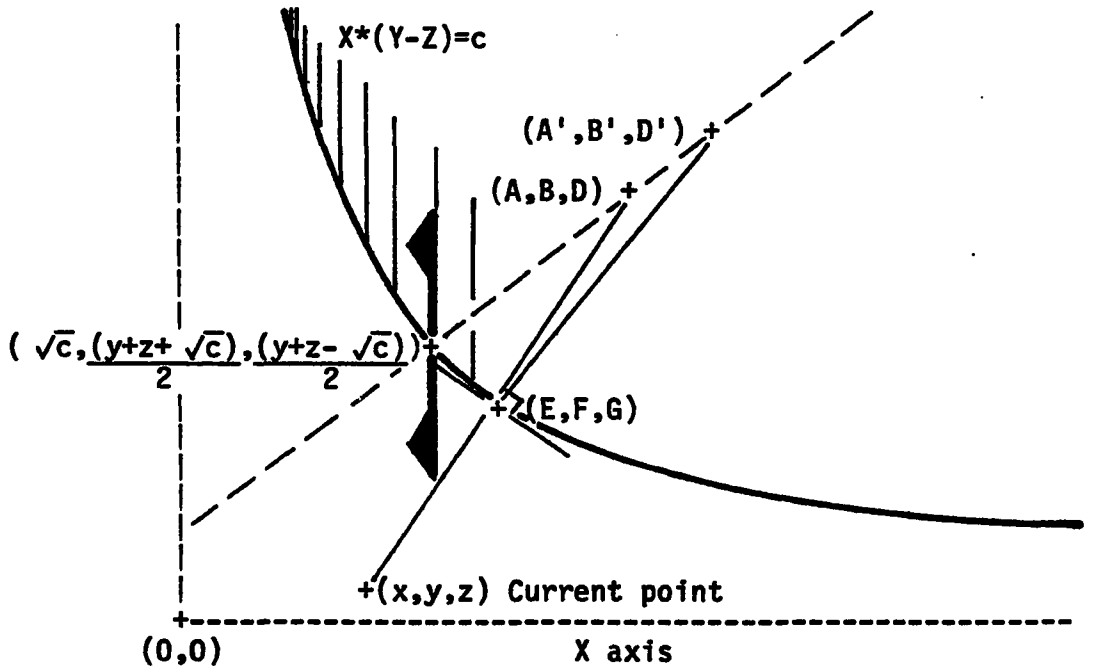


Figure 56. Hyperbolic sheet constraint supporting plane, axial algorithm X and Y axes view

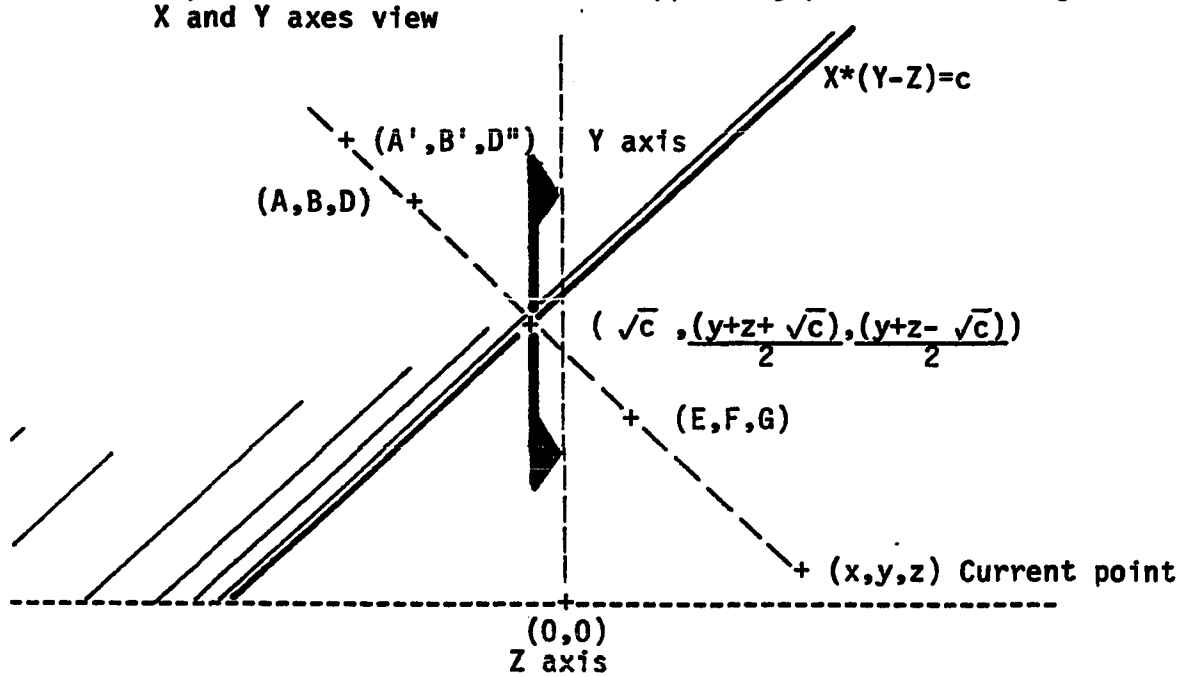


Figure 57. Hyperbolic sheet constraint supporting plane, axial algorithm Y and Z axes view

current point forming a line with a slope of negative one (-1) in the Z and Y axes plane as shown in figures 56 and 57.

Algebraically, the parametric line for the axis can be found from the point on the surface of the function which is at the intersection of:

$$Y+Z=y+z$$

and:

$$Y-Z=\sqrt{c}$$

in the plane:

$$X=\sqrt{c}$$

or:

$$X=\sqrt{c} \qquad Y=\frac{(y+z+\sqrt{c})}{2} \qquad Z=\frac{(y+z-\sqrt{c})}{2}$$

If the function:

$$F(X,Y)=X*Y-X*Z-c=0$$

is differentiated with respect to X, Y and Z:

$$\frac{\&F(X,Y)}{\&X}=(Y-Z) \qquad \frac{\&F(X,Y)}{\&Y}=X \qquad \frac{\&F(X,Y)}{\&Z}=-X$$

the parametric line through the point and normal to the tangent to the surface at that point is:

$$X=\sqrt{c}+\sqrt{c}*T \qquad Y=\frac{(y+z+\sqrt{c})}{2}+\sqrt{c}*T \qquad Z=\frac{(y+z-\sqrt{c})}{2}-\sqrt{c}*T$$

The parameter T can be solved for in terms of X as

$$T=\frac{(X-\sqrt{c})}{\sqrt{c}}$$

then by back substituted in the parametric equation:

$$Y=X+\frac{(y+z-\sqrt{c})}{2}$$

and:

$$Z=-X+\frac{(y+z+\sqrt{c})}{2}$$

As with the hyperbolic axial algorithm, a point (A,B,D) on the axis where:

$$A=c$$

or if:

$$A<x$$

then:

$$A=x+1$$

$$B=A+\frac{(y+z-\sqrt{c})}{2}$$

$$D=-A+\frac{(y+z+\sqrt{c})}{2}$$

The line through the points (A,B,D) and (x,y,z) can be written as the parametric line:

$$E=x+(A-x)*T \quad F=y+(A-y)*T \quad G=z+(D-z)*T$$

where (E,F,G) is on the curve X*(Y-Z)=c. Back substituting into E*(F-G)=c and solving for T:

$$T=\frac{-K+\sqrt{K\hat{}2-4*(A-x)*(B-D-(y-z))*(x*(y-z)-c)}}{2*(A-x)*(B-D-(y-z))}$$

where:

$$K=((y-z)*(A-x)+x*(B-D)-(y-z))$$

results in the intersection point (E,F,G):

$$E=x+(A-x)*T$$
$$F=y+(B-y)*T$$
$$G=z+(D-z)*T$$

Through the point on the curve (E,F,G) the normal to the curve can be written as the parametric line:

$$X=E+\frac{c}{E}*T \quad Y=F+E*T \quad Z=F-E*T$$

where T is:

$$T=\frac{((X-E)*E)}{c}$$

With back substitution into the parametric equations:

$$Y=\frac{(F*c+E\hat{}2*X-E\hat{}3)}{c}$$

The intersection of the line through the points (E,F,G) and (x,y,z) and the axis line is then found by back substitution:

$$Y=X+\frac{(y+z-\sqrt{c})}{2}=\frac{(F*c+E\hat{}2*X-E\hat{}3)}{c}$$

and solving for X:

$$X=\frac{(2*F*c-2*E\hat{}3-c*G+c*\sqrt{c})}{(2*c-2*E\hat{}2)}=A'$$

Say that X is equal to A' then:

$$B'=A'+\frac{(y+z-\sqrt{c})}{2}$$

and:

$$D'=-A'+\frac{(y+z+\sqrt{c})}{2}$$

If the point (A,B,D) and (A',B',D') are the same point then the algorithm would stop. Otherwise, let A',B',D' be the new intersection point for the next iteration.

Supporting plane for Hyperbolic Sheet Using Axial Algorithm The supporting plane for the hyperbolic sheet constraint using the axial algorithm subroutine (TANA-SHT) finds the point on the hyperbolic sheet constraint, a constraint whose parameters are stored in the last seven (7) columns of the constraint matrix, which is closest to the simplex current point by using the axial algorithm and then constructs the tangent plane through the point and loads the plane's coefficients into the linear portion of the constraint matrix.

```
4400 REM  * SUPPORTING PLANE FOR SHEET USING AXIAL ALGORITHM *
```

```
4401 REM-------------------------SHT-TANA----------------------------

4402 C#=A#(K,ND+4)
4403 ZX=1+A#(K,ND+5)
4404 ZY=1+A#(K,ND+6)
4405 ZZ=1+A#(K,ND+7)
4406 Y#=B#(ZY,1)
4407 X#=B#(ZX,1)
4408 Z#=B#(ZZ,1)
4409 IF X#=H#(ZX) AND Y#=H#(ZY) AND Z#=H#(ZZ) THEN RETURN
4410 G#=(Y#-Z#)
4411 IF X#>0# AND G#>0# AND X#*G#>=C# THEN RETURN
4412 A#=C#
4413 IF X#>A# THEN A#=(C#+1#)
4414 FOR W=1 TO 20
4415 B#=X#*(2#*A#-CDBL(SQR(C#))-Y#+Z#)+(A#-X#)*(Y#-Z#)
4416 K#=2#*(A#-X#)*(2#*A#-CDBL(SQR(C#))-Y#+Z#)
4417 T#=(-B#+CDBL(SQR((B#^2#)-2#*K#*((X#*(Y#-Z#))-C#))))/K#
4418 E#=X#+(A#-X#)*T#
4419 A#=(2#*(Y#+((2#*A#-Y#+Z#-CDBL(SQR(C#)))*T#/2#))*C#-2#*E#*E#*E#-C#*(
     Y#+Z#)+C#*CDBL(SQR(C#)))/(2#*C#-2#*E#*E#)
4420 NEXT W
4421 A#(K,1)=2#*C#
4422 FOR J=2 TO ND+1
4423 A#(K,J)=0#
4424 IF ZX=J THEN A#(K,J)=C#/E#
4425 IF ZY=J THEN A#(K,J)=E#
4426 IF ZZ=J THEN A#(K,J)=-E#
4427 NEXT J
4428 RETURN
```

Line Search Algorithm The line search algorithm for the hyperbolic sheet constraint depends on the fact that the function is a cylinder sloping upward in the Z direction at a forty-five (45) degree angle. If a plane is passed through the simplex current point at a forty-five (45) degree angle to the Y axis and parallel to the X axis so that it intersects the function at a ninety (90) degree angle as shown in figures 58 and 59 then the surface at its intersection with the plane will form a two (2) dimensional two (2) variable (y+z)*X-2*X*Y=c curve that is then used for the line search.

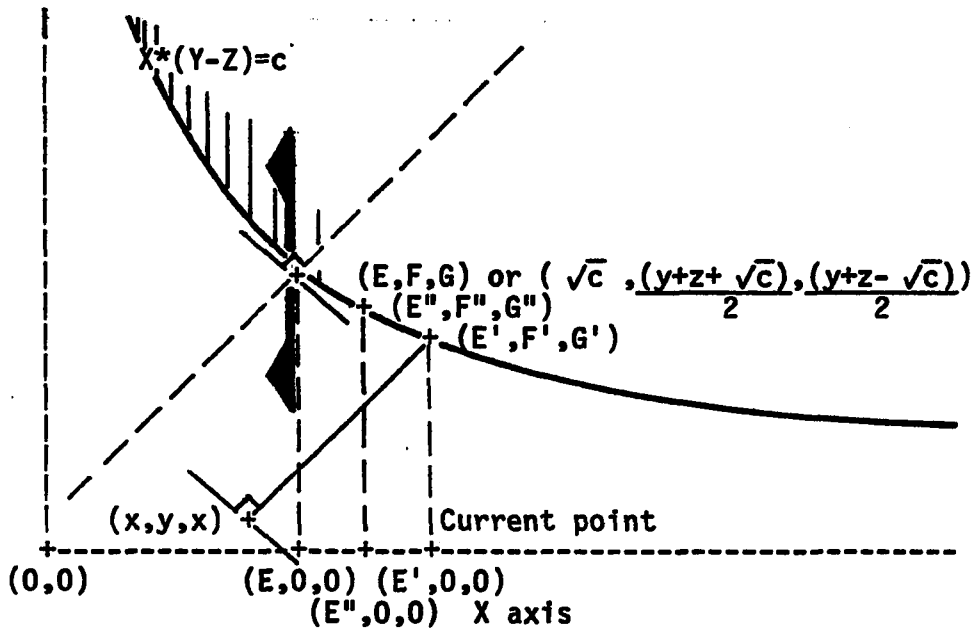


Figure 58. Hyperbolic sheet constraint supporting plane, line search algorithm X and Y axes view

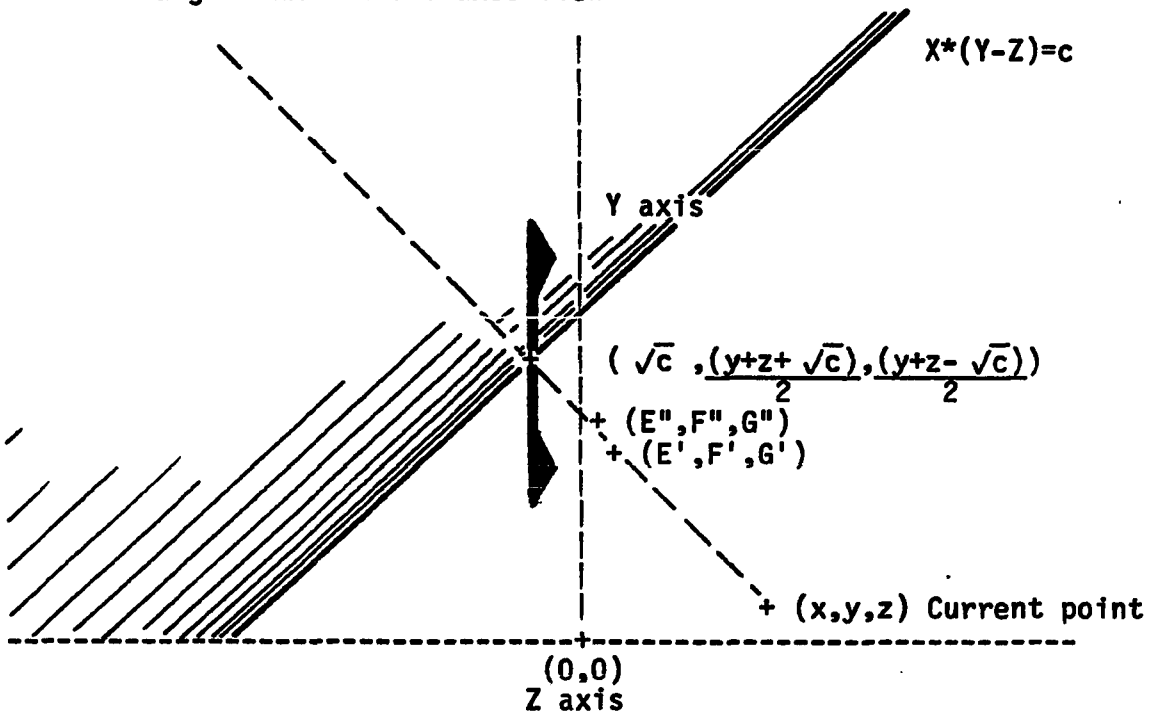


Figure 59. Hyperbolic sheet constraint supporting plane, line search algorithm Y and Z axes view

As in the hyperbolic case, a point ($\sqrt{c}$,$\frac{(y+z+\sqrt{c})}{2}$,$\frac{(y+z-\sqrt{c})}{2}$) or say (E,F,G) is selected as the starting point. This point is the same point picked as the intersection of the axis line with the function in the axial algorithm. The normal to the tangent plane at the point (E,F,G) on the function:

$$F(X,Y)=X*Y-X*Z-c=0$$

through the point (x,y,z) can be found by differentiating with respect to X and Y and Z:

$$\frac{\&F(X,Y)}{\&X}=Y-Z \qquad \frac{\&F(X,Y)}{\&Y}=X \qquad \frac{\&F(X,Y)}{\&Z}=-X$$

and writing the parametric equation:

$$X=x+(F-G)*T \qquad Y=y+E*T \qquad Z=z-E*T$$

Since the point (E,F,G) is on the surface, then:

$$F-G=\frac{c}{E}$$

so by back substitution into the parametric equation:

$$T=\frac{((X-x)*E)}{c}$$

which can be used to express Y and Z in terms of X:

$$Y=y+\frac{(E\hat{}2*(X-x))}{c} \qquad Z=z-\frac{(E\hat{}2*(X-x))}{c}$$

With these equations the intersection of the normal line throught point (x,y,z) and the surface:

$$X*(Y-Z)>=c$$

is:

$$X=\frac{(2*E\hat{}2*x+z*c-y*c)+\sqrt{(y*c-z*c-2*E\hat{}2*x)\hat{}2+8*E\hat{}2*c\hat{}2}}{4*E\hat{}2}=E'$$

Say that the new intersection point is (E',F',G'). If the point (E,F,G) and (E',F',G') are the same or are as close to the same point as required for computer tolerances, then stop the iterations. Otherwise, the point to start the next iteration is:

$$E"=\frac{(E+E')}{2}$$

$$F"=y+\frac{(E"^2*(E"-x))}{c} \qquad G"=z-\frac{(E"^2*(E"-x))}{c}$$

Supporting Plane for Hyperbolic Sheet Using Line Search Algorithm

The supporting plane for the hyperbolic sheet constraint using the line search algorithm subroutine (TANL-SHT) finds the point on the sheet constraint, a constraint whose parameters are stored in the last seven (7) columns of the constraint matrix, which is closest to the simplex current point by using the line search algorithm and then constructs the tangent plane through the point and loads the plane's coefficients into the linear portion of the constraint matrix.

```
4400 REM * SUPPORTING PLANE FOR SHEET USING LINE SEARCH ALGORITHM *
4401 REM------------------------SHT-TANL------------------------------

4402 C#=A#(K,ND+4)
4403 ZX=1+A#(K,ND+5)
4404 ZY=1+A#(K,ND+6)
4405 ZZ=1+A#(K,ND+7)
4406 Y#=B#(ZY,1)
4407 X#=B#(ZX,1)
4408 Z#=B#(ZZ,1)
4409 IF X#=H#(ZX) AND Y#=H#(ZY) AND Z#=H#(ZZ) THEN RETURN
4410 G#=(Y#-Z#)
4411 IF X#>0# AND G#>0# AND X#*G#>=C# THEN RETURN
4412 E#=((2#*C#*X#-Y#*C#+Z#*C#)+CDBL(SQR(((Y#*C#-Z#*C#-2#*C#*X#)^2#)+8#*
     C#*C#*C#)))/(4#*C#)
4413 FOR W=1 TO 100
4414 T#=((2#*E#*E#*X#-Y#*C#+Z#*C#)+CDBL(SQR(((Y#*C#-Z#*C#-2#*E#*E#*X#)^2
     #)+8#*E#*E#*C#*C#)))/(4#*E#*E#)
```

```
4415 IF ABS(T#-E#)<.000001 THEN 4418
4416 E#=(T#+E#)/2#
4417 NEXT W
4418 A#(K,1)=2#*C#
4419 FOR J=2 TO ND+1
4420 A#(K,J)=0#
4421 IF ZX=J THEN A#(K,J)=C#/E#
4422 IF ZY=J THEN A#(K,J)=E#
4423 IF ZZ=J THEN A#(K,J)=-E#
4424 NEXT J
4425 RETURN
```

Gordian Algorithm The Gordian algorithm for the hyperbolic sheet constraint again utilizes the fact that function is a cylinder in the Z axis direction. The algorithm for the sheet constraint projects the simplex current point (x,y,z) onto the hyperbolic sheet surface in either the X or Y axis direction depending on the location of the current simplex point. The sheet function asymptotically approaches both the X and Y axes plane, so trying to project the simplex point onto the surface in only one direction would not result in a point on the surface in all cases. To solve this problem, the region of points violating the function is divided into three (3) regions as shown in figures 60 and 61. If the simplex current point lies below the planes $Y=Z=\frac{\sqrt{c}}{2}$ and $X=\frac{\sqrt{c}}{2}$ then the point ($\sqrt{c}$, $\sqrt{c}$,0) is always used as the point through which the tangental plane passes. If the simplex current point lies above or on the plane, and $Y-Z>=\frac{\sqrt{c}}{2}$ and y-x>x, then the tangent point is found by the projection of the current point to the surface in the Y direction in the X and Y axes plane. Otherwise, the projection to the surface in the X direction is used.

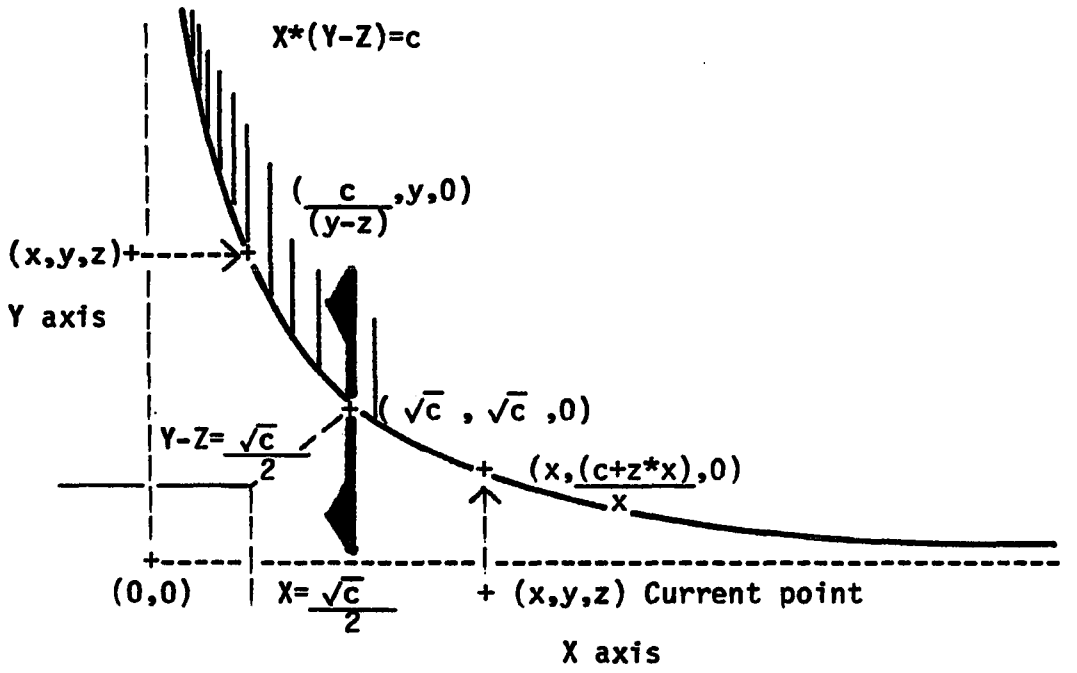


Figure 60. Hyperbolic sheet constraint supporting plane, Gordian algorithm X and Y axes view

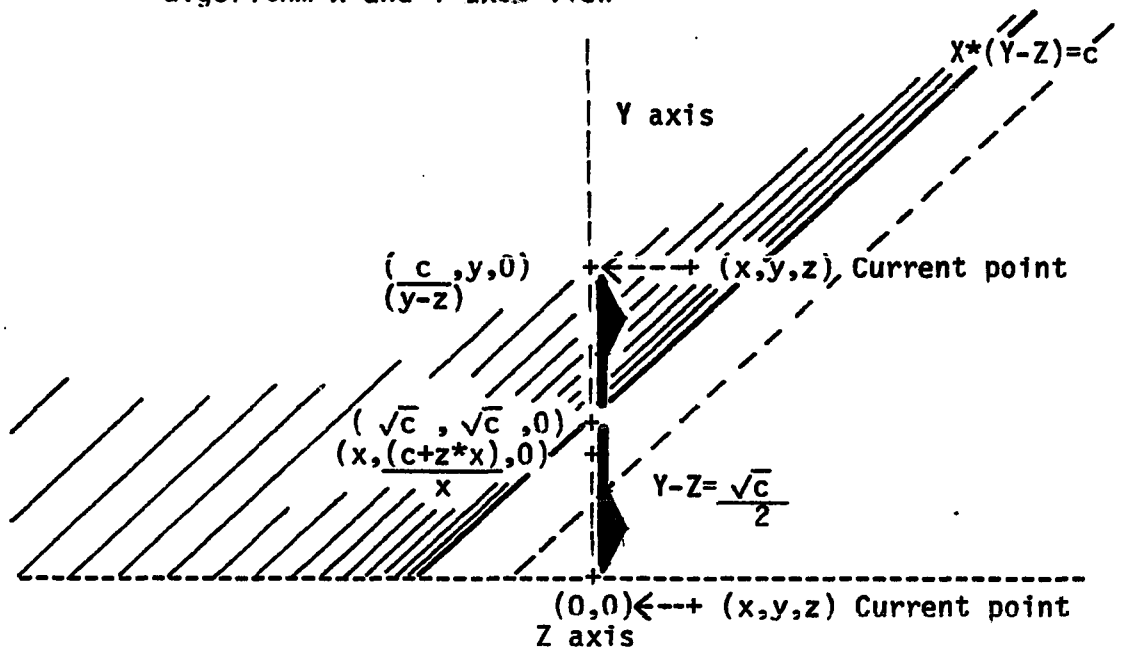


Figure 61. Hyperbolic sheet constraint supporting plane, Gordian algorithm Y and Z axes view

Supporting Plane for Hyperbolic Sheet Using Gordian Algorithm

The supporting plane for the hyperbolic sheet constraint using the Gordian algorithm subroutine (TANG-SHT) finds the point on the sheet constraint, a constraint whose parameters are stored in the last seven (7) columns of the constraint matrix, by using the Gordian algorithm and then constructs the tangent plane through the point and loads the plane's coefficients into the linear portion of the constraint matrix.

```
4400 REM       * SUPPORTING PLANE FOR SHEET SUBROUTINE *
4401 REM----------------------SHT-TANG-------------------------------

4402 ZX=A#(K,ND+5)+1
4403 ZY=A#(K,ND+6)+1
4404 ZZ=A#(K,ND+7)+1
4405 C#=A#(K,ND+4)
4406 X#=B#(ZX,1)
4407 Y#=B#(ZY,1)
4408 Z#=B#(ZZ,1)
4409 IF X#=H#(ZX) AND Y#=H#(ZY) AND Z#=H#(ZZ) THEN RETURN
4410 IF X#>0# AND Z#>=0# AND Y#-Z#>0# AND X#*(Y#-Z#)>=C# THEN RETURN
4411 G#=SQR(C#)
4412 IF Y#-Z#<=X# THEN 4422
4413 IF Y#-Z#>=G#/2# THEN G#=(Y#-Z#)
4414 A#(K,1)=2#*C#
4415 FOR L=2 TO ND+1
4416 A#(K,L)=0#
4417 IF ZX=L THEN A#(K,L)=G#
4418 IF ZY=L THEN A#(K,L)=C#/G#
4419 IF ZZ=L THEN A#(K,L)=-C#/G#
4420 NEXT L
4421 RETURN
4422 IF X#>=G#/2# THEN G#=X#
4423 A#(K,1)=2#*C#
4424 FOR L=2 TO ND+1
4425 A#(K,L)=0#
4426 IF ZX=L THEN A#(K,L)=C#/G#
4427 IF ZY=L THEN A#(K,L)=G#
4428 IF ZZ=L THEN A#(K,L)=-G#
4429 NEXT L
4430 RETURN
```

Program Table of Content

The dual simplex method utilizes the pivot rows generated by the nonlinear constraint subroutines. As far as program coding is concerned, the nonlinear constraint subroutines are an extension of the dual simplex method.

Table 10 can be used to reconstruct the dual simplex method with the

Table 10. Dual simplex with nonlinear constraints BASIC program table of contents

File	Program lines	Pages	Routines
MAIN-SMP	0001-0132	56	Dual simplex method
INPT-OBJ	1200-1220	61	Objective coefficient input subroutine
INPT-CON	1300-1328	63	Constraint coefficient input subroutine
INPT-TYP	1400-1440	65	Constraint type input subroutine
INPT-BND	1500-1523	67	Variable bounds input subroutine
UTIL-OPT	1800-1803	22	Option line subroutine
UTIL-ERS	1850-1853	22	Erase option subroutine
UTIL-CON	1860-1866	22	Continue line subroutine
UTIL-CHX	1870-1882	23	Data check subroutine
REPT-SMP	2200-2224	69	Simplex report subroutine
ALGR-KEY	3000-3057	200	Constraint key subroutine
ALGR-SMP	3300-3398	70	Dual simplex algorithm subroutine
TRAN-CON	3600-3633	74	Constraint transformation subroutine
TRAN-INV	3700-3743	76	Basis inversion subroutine
TRAN-RIV	3900-3960	78	Basis reinversion subroutine
PAR-TANL	4000-4034	209	Supporting line for parabolic subroutine
HYP-TANL	4200-4221	217	Supporting line for hyperbolic subroutine
SHT-TANL	4400-4425	228	Supporting plane for sheet subroutine

above computer code from the computer disk and to organize subroutines from previous program listings. Since BASIC code is dependent on program line numbers for its subroutine branching, the statement numbers must be maintained as listed above.

A line in the main program of the dual simplex method (MAIN-SMP) must be changed to call the constraint keys subroutine:

```
79 GOSUB 3000:REM ALGR-KEY
```

and the nonlinear constraint subroutines must be called from the dual simplex algorithm ALGR-SMP by adding the lines:

```
3333 IF A#(K,ND+2)<>0# THEN GOSUB 4000:REM PAR-TANL
3334 IF A#(K,ND+3)<>0# THEN GOSUB 4200:REM HYP-TANL
3336 IF A#(K,ND+4)<>0# THEN GOSUB 4400:REM SHT-TANL
```

Dual Simplex Method with Nonlinear Constraints: Solutions to Example 11 Minimum Project Man Count Problem

The most obvious application of the nonlinear version of the dual simplex method is to the solution of the minimum project man count CP problem. If the example ten (10) activity network as shown in figure 61 is used as the schedule logic for a man count problem, then the following

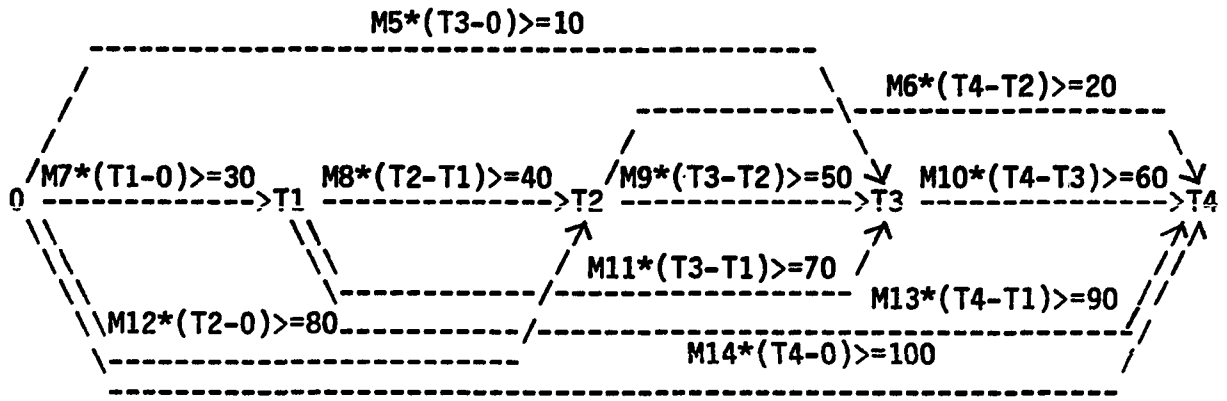


Figure 62. Dual simplex method with nonlinear constraints example 11 minimum project man count problem arrow diagram

example 11 minimum project man count CP problem can be constructed in which the total activity man count for ten (10) scheduled activities

for which the mandays requirements ranging from ten (10) to (100) mandays is minimized.

minimize M5+M6+M7+M8+M9+M10+M11+M12+M13+M14

subject to:
M5*(T3-0)>=10
M6*(T4-T2)>=20
M7*(T1-0)>=30
M8*(T2-T1)>=40
M9*(T3-T2)>=50
M10*(T4-T3)>=60
M11*(T3-T1)>=70
M12*(T2-0)>=80
M13*(T4-T1)>=90
M14*(T4-0)>=100
T4<=Dur.
T1,T2,....M14>=0

Minimum Project Man Count Solutions

The dual simplex method can be combined with hyperbolic and sheet constraints subroutines based on the axial, line search, or Gordian algorithms. The supporting planes derived using the line search algorithm are stronger than the planes derived using the Gordian algorithm, but less time is required to execute the Gordian algorithm. To provide a comparison between the two (2) algorithms, example 11 was solved for ten (10) predetermined project durations at ten (10) work increment intervals using both algorithms.

The computer runs were executed on a Panasonic Sr. Partner computer with code compiled by the IBM BASIC Compiler. The results are displayed in table 11 which lists for each of the ten (10) fixed project duration, the number of iterations required to reach the solution, the seconds required to reach the solution, the value of the objective function at the solution, and the value of each variable.

Table 11. Dual simplex method, using supporting planes derived with the line search algorithm, solutions to the example 11 minimum project man count problem with hyperbolic man count functions

Dur.	100	90	80	70	60
Itr.	115	114	107	115	116
Sec.	205	198	186	206	216
Obj.	12.75378	14.17086	15.94222	18.21968	21.25630
T1	19.12823	17.21547	15.30255	13.38980	11.47698
T2	48.67236	43.80524	38.93799	34.07078	29.20354
T3	75.02409	67.52155	60.01935	52.51685	45.01448
T4	100	90	80	70	60
M5	0.13329	0.14810	0.16661	0.19041	0.22215
M6	0.38965	0.43294	0.48706	0.55665	0.64942
M7	1.56836	1.74261	1.96045	2.24051	2.61392
M8	1.35390	1.50433	1.69237	1.93414	2.25650
M9	1.89741	2.10825	2.37176	2.71060	3.16236
M10	2.40231	2.66922	3.00290	3.43187	4.00386
M11	1.25232	1.39148	1.56540	1.78904	2.08721
M12	1.64364	1.82626	2.05454	2.34805	2.73939
M13	1.11287	1.23652	1.39109	1.58981	1.85478
M14	1	1.11111	1.25	1.42857	1.66666

Table 11. Continued

Dur.	50	40	30	20	10
Itr.	109	112	114	107	112
Sec.	190	194	209	193	209
Obj.	25.50756	31.88445	42.51260	63.76891	127.53782
T1	9.56413	7.65130	5.73848	3.82564	1.91282
T2	24.33627	19.46899	14.60177	9.73447	4.86725
T3	37.51203	30.00964	22.50723	15.00481	7.50241
T4	50	40	30	20	9.99999
M5	0.26658	0.33322	0.44430	0.66645	1.33290
M6	0.77931	0.97413	1.29885	1.94826	3.89654
M7	3.13671	3.92089	5.22785	7.84182	15.68357
M8	2.70780	3.38475	4.51299	6.76952	13.53902
M9	3.79484	4.74353	6.32474	9.48705	18.97414
M10	4.80462	6.00579	8.00772	12.01157	24.02320
M11	2.50466	3.13082	4.17443	6.26164	12.52328
M12	3.28727	4.10909	5.47878	8.21821	16.43638
M13	2.22574	2.78218	3.70957	5.56436	11.12873
M14	2	2.49999	3.33333	5	9.99999

To execute the dual simplex method with the nonlinear subroutines using the Gordian algorithm, the subroutines must be called from the dual simplex algorithm ALGR-SMP by adding the lines:

```
3333 IF A#(K,ND+2)<>0# THEN GOSUB 4000:REM PAR-TANG
3334 IF A#(K,ND+3)<>0# THEN GOSUB 4200:REM HYP-TANG
3336 IF A#(K,ND+4)<>0# THEN GOSUB 4400:REM SHT-TANG
```

Table 12 contains the results of the ten (10) computer runs of the example 11 minimum project man count CP problem using the nonlinear subroutines derived with the Gordian algorithm. Although the average number of iterations required to solve the problems was about five percent (5%) higher, the time per iteration was about thirty percent (30%) less, resulting in a reduction of overall processing time.

Table 12. Dual simplex method, using supporting planes derived with the Gordian algorithm, solutions to the example 11 minimum project man count problem with hyperbolic man count functions

Dur.	100	90	80	70	60
Itr.	117	129	117	119	111
Sec.	141	156	137	138	130
Obj.	12.75378	14.17086	15.94222	18.21968	21.25630
T1	19.12829	17.21536	15.30263	13.38981	11.47694
T2	48.67257	43.80516	38.93797	34.07073	29.20351
T3	75.02410	67.52164	60.01927	52.51686	45.01448
T4	100	90	80	70	60
M5	0.13329	0.14810	0.16661	0.19041	0.22215
M6	0.38965	0.43294	0.48706	0.55664	0.64942
M7	1.56835	1.74262	1.96044	2.24050	2.61393
M8	1.35389	1.50433	1.69238	1.93414	2.25649
M9	1.89742	2.10823	2.37176	2.71059	3.16236
M10	2.40231	2.66923	3.00289	3.43187	4.00386
M11	1.25232	1.39147	1.56541	1.78904	2.08721
M12	1.64363	1.82626	2.05454	2.34805	2.73939
M13	1.11287	1.23652	1.39109	1.58981	1.85478
M14	0.99999	1.11111	1.25	1.42857	1.66666

Table 12. Continued

Dur.	50	40	30	20	10
Itr.	124	112	113	115	121
Sec.	149	133	135	139	153
Obj.	25.50756	31.88445	42.51260	63.76891	127.53782
T1	9.56411	7.65129	5.73846	3.82564	1.91281
T2	24.33623	19.46899	14.60174	9.73448	4.86724
T3	37.51202	30.00963	22.50723	15.00481	7.50241
T4	50	40	30	20	10
M5	0.26658	0.33322	0.44430	0.66645	1.33290
M6	0.77930	0.97413	1.29884	1.94827	3.89654
M7	3.13672	3.92090	5.22788	7.84181	15.68367
M8	2.70780	3.38475	4.51300	6.76952	13.53899
M9	3.79483	4.74354	6.32471	9.48706	18.97411
M10	4.80462	6.00578	8.00772	12.01157	24.02321
M11	2.50465	3.13082	4.17442	6.26164	12.52326
M12	3.28727	4.10909	5.47879	8.21820	16.43639
M13	2.22574	2.78218	3.70957	5.56436	11.12872
M14	2	2.5	3.33333	5	10

MINIMUM PROJECT SUPERVISION COST PROBLEM WITH HYPERBOLIC MAN COUNT AND PARABOLIC COST FUNCTIONS

By approximating supervision costs as a parabolic function of man counts, man counts can be converted to direct supervision costs. This results in a CP problem with a quadratic objective function which can be solve by Beale's method with nonlinear constraints.

With the notation of the previous chapter, let:

f_{ij} - Fixed overhead cost per day of activity ij
Dur_{ij} - Duration of activity ij
Sc_{ij} - Supervisory cost per man per day
cs_{ij} - Supervisory cost curve parameter
M_{ij} - Men assigned to activity ij
T_i - Node time for node i
h_{ij} - Mandays for activity ij
Dur. - Project maximum allowable duration
fix. - Fixed overhead cost per day of project duration
n - Final node number of CPM network

then direct supervisory cost is approximately related to the activity man count, or men per day, as a parabolic function as shown in figure 63.

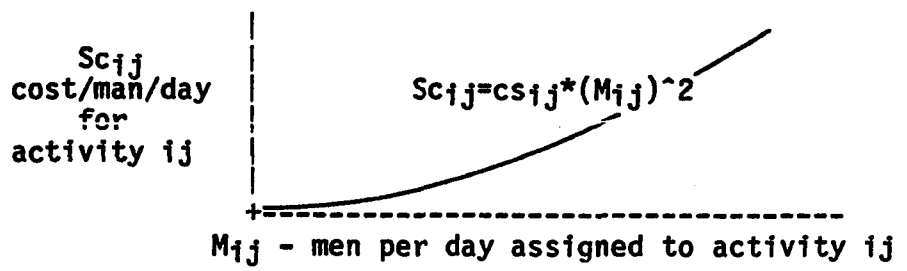


Figure 63. Supervisory cost versus activity man count per day

The parabolic curve represents the supervisory cost per man per day Sc_{ij} for each man assigned to an activity, and cs_{ij} is the curvature parameter of the parabolic curve.

Total supervisory cost for activity ij can then be written as:

Total activity ij supervisory cost = $M_{ij}*Sc_{ij}*Dur_{ij}+f_{ij}*(T_j-T_i)$

$= M_{ij}*cs_{ij}*(M_{ij})^2*Dur_{ij}+f_{ij}*(T_j-T_i)$

$= h_{ij}*cs_{ij}*(M_{ij})^2*Dur_{ij}+f_{ij}*(T_j-T_i)$

$\overline{Dur}_{ij}$

$= h_{ij}*cs_{ij}*(M_{ij})^2+f_{ij}*(T_j-T_i)$

and for the total project as the quadratic function:

$$\text{Project cost} = \sum_{\text{all } ij} (h_{ij}*cs_{ij}*(M_{ij})^2+f_{ij}*(T_j-T_i)) + fix.(T_n-T_0)$$

or:

$$\text{Project cost} = \sum_{\text{all } ij} (h_{ij}*cs_{ij}*(M_{ij})^2+f_{ij}*(Dur_{ij})) + fix.(T_n-T_0)$$

with the schedule restrictions:

subject to: $T_j-T_i-Dur_{ij}>=0$ for all ij

$T_n-T_0<=Dur.$

and the hyperbolic mandays constraints:

$M_{ij}*Dur_{ij})>=h_{ij}$ for all ij

$T_i,Dur_{ij},M_{ij}>=0$ for all i,ij

For the first project cost formulation this constraint set can be reduced further to:

subject to: $M_{ij}*(T_j-T_i)>=h_{ij}$ for all ij

$T_n-T_0<=Dur.$

$T_i,Dur_{ij},M_{ij}>=0$ for all ij

Beale's Method with Nonlinear Constraints BASIC Code

Beale's method can utilized the supporting planes generated by the nonlinear constraint subroutines so that the nonlinear coding is an extension of program code for Beale's method.

Beale's Method Main Routine -- File MAIN-BEA

Same as for Beale's method except for a line in the main routine which must be changed to call the constraint keys subroutine:

```
94 GOSUB 3000:REM ALGR-KEY
```

Utility Subroutines -- Files UTIL-OPT, UTIL-ERS, UTIL-CON, and UTIL-CHX

Same as for critical path method.

Input Subroutines -- Files INPT-OBJ, INPT-CON, INPT-TYP, and INPT-BND

The input subroutines used for the Beale's method are the same as for the dual simplex method. The linear portion of the quadratic objective function is entered through the subroutine INPT-OBJ.

Quadratic Matrix Subroutine -- INPT-QUD

Same as for Beale's method.

Report Subroutine -- File REPT-SMP

Same as for dual simplex method.

Constraint Keys Subroutine -- File ALGR-KEY

Same as for dual simplex method with nonlinear constraints.

Beale's Algorithm Subroutine -- File ALGR-BEA

Same as for Beale's method except for the nonlinear constraint subroutines which must be called from Beale's algorithm ALGR-BEA by adding the lines:

```
3342 IF A#(K,ND+2)<>0# THEN GOSUB 4000:REM PAR-TANL
3344 IF A#(K,ND+3)<>0# THEN GOSUB 4200:REM HYP-TANL
3346 IF A#(K,ND+4)<>0# THEN GOSUB 4400:REM SHT-TANL
```

Quadratic Tableau Transformation -- File TRAN-QUD

Same as for Beale's method.

Constraint Transformation Subroutine -- File TRAN-CON

Same as for dual simplex method.

Inversion Subroutine -- File TRAN-INV

Same as for dual simplex method.

Reinversion Subroutine -- File TRAN-RIV

Same as for dual simplex method.

Parabolic Subroutines -- Files PAR-TANA, PAR-TANL, or PAR-TANG

Same as for dual simplex method with nonlinear constraints.

Hyperbolic Subroutines -- Files HYP-TANA, HYP-TANL, or HYP-TANG

Same as for dual simplex method with nonlinear constraints.

Hyperbolic Sheet Subroutines -- Files SHT-TANA, SHT-TANL, or SHT-TANG

Same as for dual simplex method with nonlinear constraints.

Program Table of Content

Table 13 can be used to reconstruct the Beale's method with the nonlinear constraints from the computer disk files and to organize subroutines from previous program listings. Since BASIC code is dependent on program line numbers for its subroutine branching, the statement numbers must be maintained as listed below.

A line in the main program of Beale's method (MAIN-BEA) must be changed to call the constraint keys subroutine:

```
94 GOSUB 3000:REM ALGR-KEY
```

and the nonlinear constraint subroutines must be called from Beale's algorithm (ALGR-BEA) by adding the lines:

```
3342 IF A#(K,ND+2)<>0# THEN GOSUB 4000:REM PAR-TANL
3344 IF A#(K,ND+3)<>0# THEN GOSUB 4200:REM HYP-TANL
```

```
3346 IF A#(K,ND+4)<>0# THEN GOSUB 4400:REM SHT-TANL
```

Table 13. Beale's with nonlinear constraints BASIC program table of contents

Beale's Method with Nonlinear Constraints: Solutions to Example 12 Minimum Project Supervision Cost Problem

Using the example 11 ten (10) activity network for the minimum project man count problem with the variables as shown in figure 62, a supervision cost CP problem can be constructed and solved using Beale's method with nonlinear constraints.

A minimum supervision cost CP problem for example 12 can be written as follows:

$$\text{minimize} \quad 10*M5^2+9*M6^2+8*M7^2+7*M8^2+6*M9^2+5*M10^2 +4*M11^2+3*M12^2+2*M13^2+M14^2 +(T3-0)+(T4-T2)+(T1-0)+(T2-T1)+(T3-T2)+(T4-T3)+(T3-T1) +(T2-0)+(T4-T1)+(T4-0)$$

$$\text{subject to:} \quad M5*(T3-0)>=10$$
$$M6*(T4-T2)>=20$$
$$M7*(T1-0)>=30$$
$$M8*(T2-T1)>=40$$
$$M9*(T3-T2)>=50$$
$$M10*(T4-T3)>=60$$
$$M11*(T3-T1)>=70$$
$$M12*(T2-0)>=80$$
$$M13*(T4-T1)>=90$$
$$M14*(T4-0)>=100$$
$$T4<=Dur.$$

$$T1,T2,.....M14>=0$$

Minimum Supervision Cost Problem Solutions

Beale's method can be combined with hyperbolic and hyperbolic sheet constraint subroutines utilizing either the axial, the line search, or the Gordian algorithms. The pivot rows derived using the line search algorithm are stronger than the pivot rows derived using the Gordian algorithm, but less time is required to execute the Gordian subroutine. To provide a comparison between the two (2) algorithms, example 12 was run for ten (10) project durations Dur. spaced at ten (10) work increment intervals using both algorithms.

The results of the computer runs are displayed for the line search algorithm in table 14 which lists for each of ten (10) runs the fixed project duration, the number of iterations required to reach the solution, the seconds required to reach the solution, the value of the objective function at the solution, and the value of each variable.

Table 14. Beale's method, using supporting planes derived with the line search algorithm, solutions to the example 12 minimum project supervision cost problem with parabolic cost and hyperbolic man count functions

Dur.	100	90	80	70	60
Itr.	240	240	215	202	184
Sec.	1333	1333	1200	1049	974
Obj.	554.78018	554.78018	554.78018	556.79123	581.59037
T1	16.84936	16.84936	16.84919	15.63239	13.04579
T2	34.59311	34.59311	34.58314	32.57042	27.95595
T3	52.98392	52.98392	52.98395	50.08515	43.30467
T4	74.39570	74.39570	74.39585	70	60
M5	0.18873	0.18873	0.18873	0.19965	0.23092
M6	0.50247	0.50247	0.50247	0.53433	0.62414
M7	1.78048	1.78048	1.78050	1.91909	2.29959
M8	2.25431	2.25431	2.25428	2.36154	2.68273
M9	2.71874	2.71874	2.71874	2.85474	3.25760
M10	2.80219	2.80219	2.80217	3.01282	3.59382
M11	1.93720	1.93720	1.93719	2.03176	2.31337
M12	2.31259	2.31259	2.31259	2.45621	2.86164
M13	1.56394	1.56394	1.56394	1.65539	1.91676
M14	1.34416	1.34416	1.34416	1.42857	1.66666

Table 14. Continued

Dur.	50	40	30	20	10
Itr.	199	162	164	174	177
Sec.	1031	820	846	872	931
Obj.	654.85698	831.36710	1270.72451	2619.40329	10124.15667
T1	10.66004	8.41532	6.26032	4.15609	2.07484
T2	23.32245	18.67304	14.01192	9.34371	4.67229
T3	36.31425	29.17392	21.93619	14.64317	7.32508
T4	50	40	30	20	10
M5	0.27537	0.34277	0.45586	0.68291	1.36517
M6	0.74969	0.93778	1.25093	1.87682	3.75396
M7	2.81424	3.56492	4.79208	7.21831	14.45889
M8	3.15895	3.89950	5.16022	7.71067	15.39970
M9	3.84858	4.76150	6.30972	9.43491	18.84811
M10	4.38412	5.54217	7.44065	11.20066	22.43061
M11	2.72859	3.37209	4.46546	6.67488	13.33273
M12	3.43017	4.28425	5.70942	8.56190	17.12218
M13	2.28775	2.84948	3.79112	5.68041	11.35624
M14	2	2.49999	3.33333	5	10

To execute Beale's method with the nonlinear subroutines based on the Gordian algorithm, the subroutines must be called from the Beale's algorithm ALGR-BEA by adding the lines:

```
3342 IF A#(K,ND+2)<>0# THEN GOSUB 4000:REM PAR-TANG
3344 IF A#(K,ND+3)<>0# THEN GOSUB 4200:REM HYP-TANG
3346 IF A#(K,ND+4)<>0# THEN GOSUB 4400:REM SHT-TANG
```

Table 15 contains the results of the ten (10) runs using the nonlinear subroutines based on the Gordian algorithm.

Both constraint types were executed on a Panasonic Sr. Partner computer with code compiled with the IBM BASIC Compiler.

Table 15. Beale's method, using supporting planes derived with the Gordian algorithm, solutions to the example 12 minimum project supervision cost problem with parabolic cost and hyperbolic man count functions

Dur.	100	90	80	70	60
Itr.	203	203	213	206	189
Sec.	842	842	908	854	766
Obj.	554.78018	554.78018	554.78018	556.79123	581.59037
T1	16.84926	16.84926	16.84930	15.63227	13.04571
T2	34.59319	34.59319	34.59339	32.57047	27.95585
T3	52.98393	52.98393	52.98429	50.08523	43.30466
T4	74.39584	74.39584	74.39591	70	60
M5	0.18873	0.18873	0.18873	0.19965	0.23092
M6	0.50247	0.50247	0.50248	0.53433	0.62413
M7	1.78049	1.78049	1.78048	1.91910	2.29960
M8	2.25429	2.25429	2.25427	2.36152	2.68273
M9	2.71876	2.71876	2.71873	2.85473	3.25758
M10	2.80217	2.80217	2.80221	3.01284	3.59381
M11	1.93719	1.93719	1.93718	2.03175	2.31336
M12	2.31259	2.31259	2.31257	2.45621	2.86165
M13	1.56395	1.56395	1.56394	1.65539	1.91675
M14	1.34416	1.34416	1.34415	1.42857	1.66666

Table 15. Continued

Dur.	50	40	30	20	10
Itr.	181	169	157	163	171
Sec.	743	706	645	694	733
Obj.	654.85698	831.36710	1270.72451	2619.40329	10124.15667
T1	10.66014	8.41515	6.26029	4.15617	2.07484
T2	23.32262	18.67296	14.01188	9.34378	4.67233
T3	36.31437	29.17383	21.93621	14.64321	7.32509
T4	50	40	30	20	10
M5	0.27537	0.34277	0.45586	0.68291	1.36517
M6	0.74969	0.93777	1.25092	1.87683	3.75398
M7	2.81421	3.56499	4.79210	7.21817	14.45891
M8	3.15893	3.89946	5.16023	7.71068	15.39949
M9	3.84859	4.76151	6.30967	9.43496	18.84829
M10	4.38416	5.54212	7.44067	11.20075	22.43066
M11	2.72859	3.37208	4.46544	6.67490	13.33270
M12	3.43014	4.28426	5.70944	8.56184	17.12207
M13	2.28775	2.84946	3.79111	5.68044	11.35624
M14	2	2.5	3.33333	5	10

Cubic Hyperbolic and Cubic Hyperbolic of Two Sheets Constraints

The quadratic tableau transformation in Beale's method requires not only an array space equivalent to the augmented $\overline{B}$ and augmented $\overline{B}$ inverse matrix, but also the extra transformation step with its inherent lack of precision. In many cases, the objective function can be rewritten in the form of nonlinear convex constraints and can be minimized using the primal-dual simplex routine, thus avoiding the time and precision handicaps of Beale's method.

In the minimum project supervison cost problem let:

$$W_{ij}=(M_{ij})^2 \text{ for all } ij$$

Then, the objective function can be written as a linear function:

$$\text{Project cost} = \sum_{\text{all } i,j} (cs_{ij} * W_{ij} + f_{ij} * (T_j - T_i)) + fix. * (T_n - T_0)$$

and the constraints as:

$$\text{subject to: } W_{ij} * (T_j - T_i)\hat{}2 >= (h_{ij})\hat{}2 \text{ for all } ij$$

$$T_n - T_0 <= Dur.$$

The nonlinear constraints needed to solve the supervision cost problem can now be written in general terms for X,Y,Z as:

$$X * Y\hat{}2 >= h\hat{}2 = c$$

and:

$$X * (Y-Z)\hat{}2 >= h\hat{}2 = c$$

Primal-dual Method with Cubic Hyperbolic Constraints BASIC Code

The primal-dual method with nonlinear constraints utilizes the supporting planes generated by the nonlinear constraint subroutines. As far as program coding is concerned, the nonlinear subroutines are an extension of the primal-dual program.

Primal-dual Method Main routine -- File MAIN-PDS

Same as for primal-dual method except the main routine must call the constraint keys subroutine by changing line:

```
81 GOSUB 3000:REM ALGR-KEY
```

Utility Subroutines -- Files UTIL-OPT, UTIL-ERS, UTIL-CON, and UTIL-CHX

Same as for critical path method.

Input Subroutines -- Files INPT-OBJ, INPT-CON, INPT-TYP, and INPT-BND

Same as for dual simplex method with the exception of the input of constraint coefficients and types. In the case of the cubic hyperbolic

constraint X*Y^2>=c, the constraint is entered as if it were a simple linear constraint of the form X+Y>=0, and the c constant is entered in the HYPERBOLIC column of the constraint type input screen. For the cubic sheet constraint X*(Y-Z)^2>=c, the input is X+2*Y-3*Z>=0, and the c constant is entered in the SHEET column of the constraint type input screen.

Report Subroutine -- File REPT-SMP

Same as for dual simplex method.

Constraint Keys Subroutine -- File ALGR-KEY

Same as dual simplex method with nonlinear constraints.

Primal-dual Algorithm Subroutine -- File ALGR-PDS

Same as for primal-dual method except the nonlinear subroutines must be called by adding lines:

```
3338 IF A#(K,ND+3)<>0# THEN GOSUB 4600:REM CUBIC HYPERBOLIC
3339 IF A#(K,ND+4)<>0# THEN GOSUB 4800:REM CUBIC SHEET
```

Objective Function Transformation -- File TRAN-OBJ

Same as for primal-dual method.

Constraint Transformation Subroutine -- File TRAN-CON

Same as for dual simplex method.

Inversion Subroutine -- File TRAN-INV

Same as for dual simplex method.

Reinversion Subroutine -- File TRAN-RIV

Same as for dual simplex method.

Cubic Hyperbolic Subroutine -- File CBH-TANG

The Gordian algorithm provides a simple means of finding a point

on the hyperbolic cubic curve X*Y^2=c with which to generate a tangent point for a supporting plane.

Gordian Algorithm The Gordian algorithm for the cubic hyperbolic function projects the simplex current point (x,y) to the curve in either the X or Y direction, depending on the location of the simplex current point, from which is generated a supporting line. Since the cubic hyperbolic function asymptotically approaches both the X and Y axes, trying to project the simplex point onto the curve in only one (1) direction would not result in a point on the curve in all cases. To solve this problem, the region of points which violates the function is divided into three (3) regions as shown in figure 64. If the current

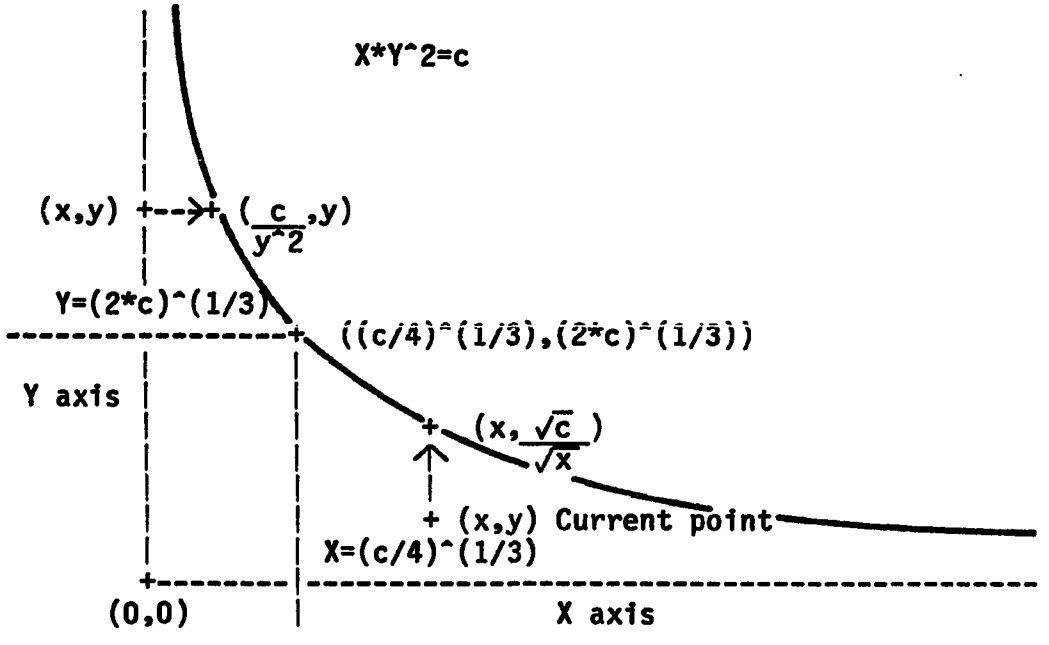


Figure 64. Cubic hyperbolic constraint supporting line, Gordian algorithm

simplex point lies below the line Y=(2*c)^(1/3) and left of the line X=(c/4)^(1/3), then the point ((2*c)^(1/3),((c/2)^(1/3)), or the point on the curve where the tangent line has a slope of negative one (-1), is always used as the point through which the supporting line passes. If the simplex current point lies above or on the line Y=(2*c)^(1/3), then the projection in the Y direction is used as the point on the curve for the tangent point. Otherwise, the projection in the X direction is used.

Supporting line for Cubic Hyperbolic Using Gordian Algorithm The supporting line for the cubic hyperbolic constraint using the Gordian algorithm subroutine (HCB-TANG) finds the point on the cubic hyperbolic constraint, a constraint whose parameters are stored in the last seven (7) columns of the constraint matrix, by using the Gordian algorithm and then constructs the tangent line through the point and loads the line's coefficients into the linear portion of the constraint matrix.

```
4600 REM    * SUPPORTING LINE FOR CUBIC HYPERBOLIC SUBROUTINE *
4601 REM-----------------------CBH-TANG-----------------------------

4602 ZX=A#(ND+5)+1
4603 ZY=A#(ND+6)+1
4604 C#=A#(ND+3)
4605 Y#=B#(ZY,1)
4606 X#=B#(ZX,1)
4607 IF X#=H#(ZX) AND Y#=H#(ZY) THEN RETURN
4608 IF X#>0# AND Y#>0# AND X#*(Y#*Y#)>=C# THEN RETURN
4609 H#=(2#*C#)^(1#/3#)
4610 G#=C#/(H#*H#)
4611 IF Y#<H# AND X#<G# THEN GOTO 4621
4612 IF Y#>=X# THEN 4620
4613 A#(1)=3#*C#
4614 FOR L=2 TO ND+1
4615 A#(L)=0#
4616 IF ZX=L THEN A#(L)=C#/X#
```

```
4617 IF ZY=L THEN A#(L)=2#*CDBL(SQR(C#*X#))
4618 NEXT L
4619 RETURN
4620 H#=Y#
4621 A#(1)=3#*C#
4622 FOR L=2 TO ND+1
4623 A#(L)=0#
4624 IF ZX=L THEN A#(L)=H#*H#
4625 IF ZY=L THEN A#(L)=2#*C#/H#
4626 NEXT L
4627 RETURN
```

Cubic Hyperbolic of two Sheets Subroutine -- File CBS-TANG

Again, the Gordian algorithm can be used to find a tangent point on the cubic hyperbolic of two sheets function for a supporting plane.

Gordian Algorithm The Gordian algorithm for the cubic sheet constraint utilizes the fact that function is a cylinder in the Z axis. The algorithm for the cubic sheet constraint projects the simplex current point (x,y,z) onto the surface in either the X or Y axis direction depending on the location of the simplex current point. The function asymptotically approaches both the X and Y axes, so trying to project the simplex point onto the surface in only one (1) direction would not result in a point on the surface in all cases. To solve this problem, the region of points violated by the constraint is divided into three (3) regions as shown in figures 65 and 66. If the simplex current point lies below the plane Y-Z=(2*c/4)^(1/3) and to the left of the plane X=(c/4)^(1/3) then the point ((c/4)^(1/3),(2*c)^(2/3),0) is always used as the point through which the supporting plane passes. If the simplex current point lies above or on the plane Y-Z=(2*c)^(1/3), then the point on the surface used as the tangent point is found by the

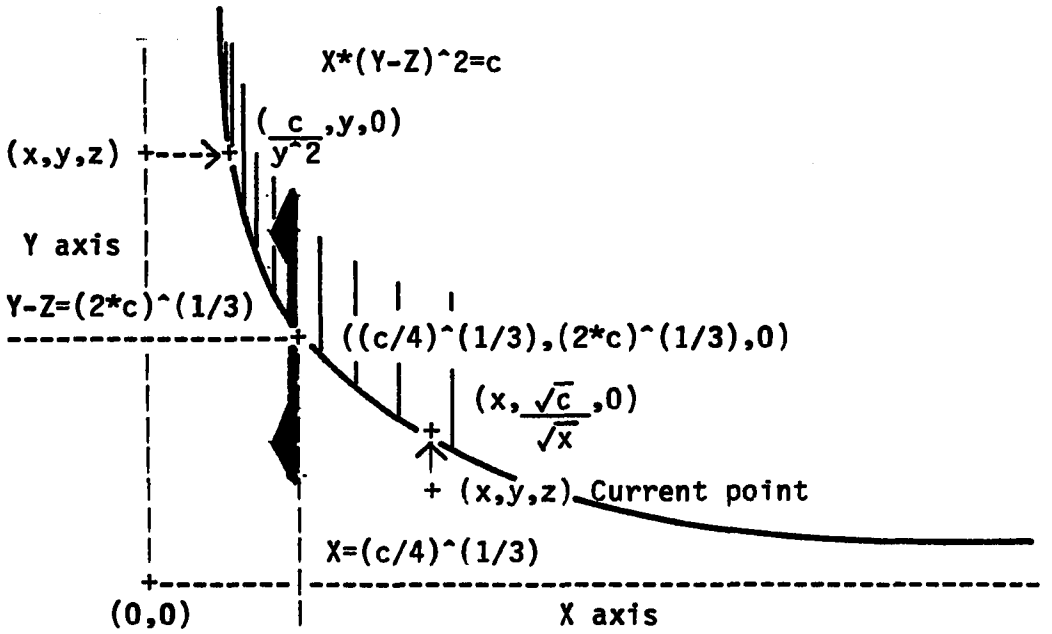


Figure 65. Cubic hyperbolic of two sheets constraint supporting plane, Gordian algorithm X and Y axes view

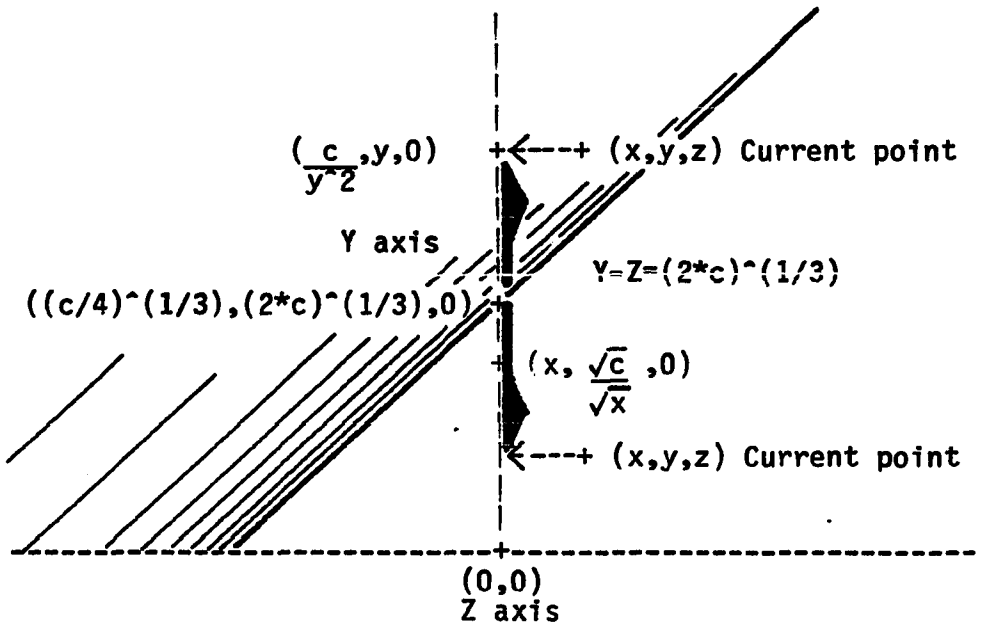


Figure 66. Cubic hyperbolic of two sheets constraint supporting plane Gordian algorithm Y and Z axes view

projecting the current point to the surface in the Y direction in the X and Y axes plane. Otherwise, the projection in the X direction is used.

Supporting Plane for Cubic Sheet Using Gordian Algorithm The supporting plane for cubic hyperbolic of two sheet constraint using the Gordian algorithm subroutine (CBS-TANG) finds the point on the cubic sheet constraint, a constraint whose parameters are stored in the last seven (7) columns of the constraint matrix, by using the Gordian algorithm and then constructs the tangent plane through the point and loads the plane's coefficients into the linear portion of the constraint matrix.

```
4800 REM       * SUPPORTING PLANE FOR CUBIC SHEET SUBROUTINE *
4801 REM-----------------------CBS-TANG---------------------------------

4802 ZX=A#(K,ND+5)+1
4803 ZY=A#(K,ND+6)+1
4804 ZZ=A#(K,ND+7)+1
4805 C#=A#(K,ND+4)
4806 X#=B#(ZX,1)
4807 Y#=B#(ZY,1)
4808 Z#=B#(ZZ,1)
4809 IF X#=H#(ZX) AND Y#=H#(ZY) AND Z#=H#(ZZ) THEN RETURN
4810 IF X#>0# AND Z#>=0# AND Y#-Z#>0# AND X#*(Y#-Z#)^2#>=C# THEN RETURN
4811 H#=(2#*C#)^(1#/3#)
4812 G#=C#/(H#*H#)
4813 IF Y#-Z#<H# AND X#<G# THEN 4816
4814 IF X#>=G# THEN 4824
4815 H#=(Y#-Z#)
4816 A#(K,1)=3#*C#
4817 FOR L=2 TO ND+1
4818 A#(K,L)=0#
4819 IF ZX=L THEN A#(K,L)=H#*H#
4820 IF ZY=L THEN A#(K,L)=2#*C#/H#
4821 IF ZZ=L THEN A#(K,L)=-2#*C#/H#
4822 NEXT L
4823 RETURN
4824 A#(K,1)=3#*SQR(C#*X#)
```

```
4825 FOR L=2 TO ND+1
4826 A#(K,L)=0#
4827 IF ZX=L THEN A#(K,L)=CDBL(SQR(C#/X#))
4828 IF ZY=L THEN A#(K,L)=2#*X#
4829 IF ZZ=L THEN A#(K,L)=-2#*X#
4830 NEXT L
4831 RETURN
```

Program Table of Content

Table 16 can be used to reconstruct the primal-dual simplex method with cubic contraints from the computer disk files and to organize

Table 16. Primal-dual method with nonlinear constraints BASIC program table of contents

subroutines from previous program listings. Since BASIC code is dependent on program line numbers for subroutine branching, the statement numbers must be maintained as listed above.

The primal-dual method main routine is a modification of the routine MAIN-BEA which must be changed to call the constraint keys subroutine by changing line:

94 GOSUB 3000:REM ALGR-KEY

and the primal-dual algorithm ALGR-PDS requires adding the lines:

```
3338 IF A#(K,ND+3)<>0# THEN GOSUB 4600:REM CBH-TANG
3339 IF A#(K,ND+4)<>0# THEN GOSUB 4800:REM CBS-TANG
```

for the nonlinear subroutines.

Primal-dual Method with Nonlinear Constraints: Solutions to Example 13 Minimum Project Supervision Cost Problem

The example 12 minimum project supervision cost problem which was solved with Beale's method with nonlinear constraints can be rewritten as a the following CP problem in which the objective function is linear and can solved with the primal-dual method with cubic constraints.

```
minimize      10*W5+9*W6+8*W7+7*W8+6*W9+5*W10+4*W11
                    +3*W12+2*W13+W14
   +(T3-0)+(T4-T2)+(T1-0)+(T2-T1)+(T3-T2)+(T4-T3)+(T3-T1)
                 +(T2-0)+(T4-T1)+(T4-0)

        subject to:  W5*(T3-0)^2>=100
                     W6*(T4-T2)^2>=400
                     W7*(T1-0)^2>=900
                     W8*(T2-T1)^2>=1600
                     W9*(T3-T2)^2>=2500
                     W10*(T4-T3)^2>=3600
                     W11*(T3-T1)^2>=4900
                     W12*(T2-0)^2>=6400
                     W13*(T4-T1)^2>=8100
                     W14*(T4-0)^2>=10000
                     T4<=Dur.

                     T1,T2,.....W14>=0
```

Minimum Project Supervision Cost Problem Solutions

With this problem, ten (10) computer runs were made for project durations of one hundred (100) to ten (10) time increments at ten (10) time increment intervals using the computer code for the primal-dual method with cubic hyperbolic constraints.

The results of the computer runs are displayed in table 17 which lists for each of the ten (10) fixed project duration, the number of iterations required to reach the solution, the seconds required to reach the solution, the value of the objective function at the solution, and the value of each variable.

Table 17. Primal-dual method, using supporting planes derived with the Gordian algorithm, solutions to example 12 minimum project supervision cost problem with cubic hyperbolic cost functions

Dur.	100	90	80	70	60
Itr.	119	123	119	101	108
Sec.	210	225	209	172	190
Obj.	554.78018	554.78018	554.78018	556.79123	581.59037
T1	16.84926	16.84926	16.84925	15.63227	13.04573
T2	34.59319	34.59320	34.59324	32.57041	27.95588
T3	52.98403	52.98408	52.98410	50.08518	43.30469
T4	74.39589	74.39594	74.39595	70	60
W5	0.03562	0.03562	0.03562	0.03986	0.05332
W6	0.25248	0.25248	0.25248	0.28551	0.38955
W7	3.17015	3.17015	3.17015	3.68296	5.28816
W8	5.08183	5.08182	5.08179	5.57684	7.19707
W9	7.39157	7.39154	7.39155	8.14951	10.61183
W10	7.85224	7.85224	7.85225	9.07715	12.91558
W11	3.75271	3.75270	3.75269	4.12804	5.35165
W12	5.34808	5.34808	5.34807	6.03299	8.18904
W13	2.44593	2.44593	2.44593	2.74032	3.67396
W14	1.80676	1.80676	1.80676	2.04081	2.77777

Table 17. Continued

Dur.	50	40	30	20	10
Itr.	111	109	118	126	140
Sec.	203	202	226	230	264
Obj.	654.85698	831.36710	1270.72451	2619.40329	10124.15667
T1	10.66012	8.41523	6.26032	4.15612	2.07485
T2	23.32253	18.67301	14.01188	9.34375	4.67232
T3	36.31433	29.17386	21.93616	14.64318	7.32510
T4	50	40	30	20	10
W5	0.07583	0.11749	0.20781	0.46636	1.86368
W6	0.56204	0.87942	1.56482	3.52250	14.09241
W7	7.91986	12.70894	22.96405	52.10329	209.05862
W8	9.97901	15.20595	26.62821	59.45413	237.14635
W9	14.81157	22.67207	39.81260	89.01866	355.25479
W10	19.22076	30.71534	55.36290	125.45552	503.13922
W11	7.44524	11.37100	19.94043	44.55416	177.76084
W12	11.76599	18.35486	32.59769	73.30559	293.16559
W13	5.23382	8.11950	14.37270	32.26727	128.96453
W14	4	6.25	11.11111	24.99999	99.99999

MINIMUM PROJECT COST CURVE PROBLEM WITH HYPERBOLIC MAN COUNT AND PARABOLIC COST FUNCTIONS

Just as when the dual simplex method was used to minimize the linear cost problem, the nonlinear modifications of the simplex method found only one of the cost versus duration points on the project cost curve. In the case of the linear cost problem, the out-of-kilter method provided a very efficient means of solving for all the cost versus duration points on the project cost curve. In the nonlinear case, the out-of-kilter method proves too complex and problem dependent for a straight forward computer[12] algorithm.

The project cost curve in the nonlinear case is a continuous function rather than the piece wise linear function of the linear cost problem. To approximate the curve in the nonlinear case, a set of durations defining preset intervals must first be defined and then the optimal cost for each interval point found.

One possible method for finding the cost curve is to run the nonlinear simplex method or Beale's method for every interval point on the curve. But rather than rerunning the nonlinear modification of the simplex method from the starting point (0,0) for every point on the curve, the method can be restarted from the last optimal solutiuon point by adding a new and lower duration constraint corresponding to the next duration point on the project cost curve to the optimal simplex tableau as each successive optimal cost verses duration solution point is found.

Theory of Restart Method

This method restarts the Beale's algorithm from the current point of the last optimal solution by adding a new constraint which makes the simplex tableau dual infeasible until a new optimal solution is found. This approach simply takes advantage of the fact that the current simplex point at the last optimal solution is closer to the new optimal solution than the usual simplex starting point at the origin.

Restart Method BASIC Code

Restart Main routine -- File MAIN-RST

The restart method main routine (MAIN-RST) dimensions eleven (11) arrays; writes the options menu to the screen as shown in figure 67;

```
                    RESTART METHOD

NUMBER OF CONSTRAINTS              10
NUMBER OF VARIABLES                14
MAXIMUM ITERATIONS                 1000
DURATION VARIABLE                  4
CURVE POINT INTERVAL               10

                  M-RETURN TO MENU

          O-OBJECTIVE COEFFICIENTS
          A-CONSTRAINT COEFFICIENTS
          C-CONSTRAINT TYPES
          B-BOUNDED VARIABLES
          Q-QUADRATIC COEFFICIENTS
          U-EXECUTE ALGORITHM
          R-REPORT LISTING
          S-SAVE  F-FETCH
          N-NEW PROBLEM

OPTION ?
                          O'X+X'QX=z
   1*X*2*Y>=c   1*X*(2*Y-3*Z)>=c   1*Y+a*(X-b)^2>=c
```

Figure 67. Restart method main menu screen

calls the utility subroutines UTIL-OPT, UTIL-ERS, UTIL-CON, UTIL-CHX; calls the data input and output routines INPT-OBJ, INPT-CON, INPT-TYP, INPT-BND, INPT-QUD, and REPT-CRV; calls and times the processing subroutines ALGR-KEY, ALGR-RST, and ALGR-BEA; and saves and fetches the input data to the disk file "DATA".

```
1 REM                          * RESTART METHOD *
2 REM-----------------------------MAIN-RST-------------------------

3 REM      BI# - MACHINE INFINITE
4 REM      CO  - PIVOT COLUMN FOR SIMPLEX TRANSFORMATION
5 REM      DV -  VARIABLE GOVERNING DURATION OF NETWORK
6 REM      ER  - ERROR KEY
7 REM      IN  - NUMBER OF ITERATIONS BETWEEN REINVERSIONS
8 REM      INV - DURATION INTERVAL OF COST CURVE
9 REM      IR  - MAXIMUM NUMBER OF ITERATIONS
10 REM     MD  - NUMBER OF CONSTRAINTS
11 REM     ND  - NUMBER OF VARIABLES
12 REM     PA  - NUMBER OF ATTEMPTS AT PIVOT
13 REM     PM  - SIGN KEY (+-)
14 REM     RO  - PIVOT ROW FOR SIMPLEX TRANSFORMATION
15 REM     SM# - MACHINE ZERO
16 REM     A#(MD+2+ND,ND+8) - ORIGINAL DATA AND KEYS
17 REM     B#(2*ND+1,ND+1) - PRIMAL-DUAL MATRIX
18 REM     C#(CYC,2) - COST CURVE
19 REM     H#(ND+1) - PAST ITERATION SOLUTION
20 REM     M#(ND+1,2) - UPPER AND LOWER BOUND VALUES
21 REM     P#(ND+2) - WORK VECTOR
22 REM     Q#(ND,ND) - QUADRATIC OBJECTIVE MATRIX
23 REM     R(MD+1) - CONSTRAINT TYPE (1->=,0-=,-1-<=)
24 REM     S#(ND+1) - FREE VARIABLE COLUMN SWITCH
25 REM     T#(ND+1,ND+1) - INVERSION WORK FILE
26 REM     V#(ND+1+ND+1+ND+1+MD+1) - ROW AND COLUMN ARRAY
27 REM     X#(ND) - SOLUTION VECTOR
28 REM ------------------------------------------------------------
```

Sets MD to the default number of constraints in the LP problem and ND to the number of constraints. Sets IN to the number of iterations before a reinversion of the augmented $\bar{B}$ matrix. Sets CYC to the maximum number of duration points on the cost curve. Sets IR to the default maximum number of iterations. Sets BI# to a number considered machine infinite and SM# to a number considered machine zero.

```
29 MD=0
```

```
30 ND=0
31 IN=10
32 CYC=50
33 IR=1000
34 BI#=1E+10
35 SM#=1E-09
```

Prompts and reads from the keyboard MD the number of constraints in the CP problem; ND the number of constraints; IR the maximum number of iterations allowed before the restart algorithm is stopped; DV the starting project duration; and INV the duration interval between points on the project cost curve.

```
36 CLS
37 LOCATE 1,10:PRINT "RESTART METHOD"
38 LOCATE 3,1:PRINT "NUMBER OF CONSTRAINTS":LOCATE 3,31:INPUT "",L$
39 GOSUB 1870:REM UTIL-CHX
40 IF Z#<>BI# THEN MD=Z#
41 LOCATE 3,30:PRINT MD,"    ":LOCATE 4,1:PRINT "NUMBER OF VARIABLES"
   :LOCATE 4,31:INPUT "",L$
42 GOSUB 1870:REM UTIL-CHX
43 IF Z#<>BI# THEN ND=Z#
44 LOCATE 4,30:PRINT ND,"    ":LOCATE 5,1:PRINT "MAXIMUM ITERATIONS"
   :LOCATE 5,31:INPUT "",L$
45 GOSUB 1870:REM UTIL-CHX
46 IF Z#<>BI# THEN IR=Z#
47 LOCATE 5,30:PRINT IR,"    ":LOCATE 6,1:PRINT "DURATION VARIABLE"
   :LOCATE 6,31:INPUT "",L$
48 GOSUB 1870:REM UTIL-CHX
49 IF Z#<>BI# THEN DV=Z#
50 LOCATE 6,30:PRINT DV,"    ":LOCATE 7,1:PRINT "CURVE POINT INTERVAL"
   :LOCATE 7,31:INPUT "",L$
51 GOSUB 1870:REM UTIL-CHX
52 IF Z#<>BI# THEN INV=Z#
53 LOCATE 7,30:PRINT INV,"    "
```

Dimensions the A#(MD+2+ND+1,ND+8) array which contains the linear portions of the quadratic objective function, the constraint coefficients, and the first step of the transformation of the quadratic partials. Dimensions the augmented $\overline{B}$ inverse and augmented $\overline{B}$ array B#(2*ND+1,ND+1). Dimensions the holding array H#(ND+1), the upper and lower bound array M#(ND+1), the pivot row P#(ND+2), the augmented $\overline{Q}$ matrix array Q#(ND+1,ND+1), the enlarged constraint type array R(MD+1+ND+1), the switch array S#(ND+1), the reinversion working array T#(ND+1,ND+1), the pivot selection array V#(MD+1+ND+1+ND+1+ND+1), the solution vector X#(ND), and the cost curve array C#(CYC,2).

The restart method utilizes Beale's algorithm, so the arrays used

are the same for both methods. The only array which is added is the cost curve array C#(CYC,2). This array contains all the points of the cost curve where C#(CYC,1) is the duration and C#(CYC,2) is the cost.

```
54 DIM A#(MD+2+ND+1,ND+8)
55 DIM B#(2*ND+1,ND+1)
56 DIM C#(CYC,2)
57 DIM H#(ND+1)
58 DIM M#(ND+1,2)
59 DIM P#(ND+1)
60 DIM Q#(ND+1,ND+1)
61 DIM R(MD+1+ND+1)
62 DIM S#(ND+1)
63 DIM T#(ND+1,ND+1)
64 DIM V#(MD+1+ND+1+ND+1+ND+1)
```

Initializes the constraint type array to all greater than or equals.

```
65 FOR I=1 TO MD+1+ND+1
66 R(I)=1
67 NEXT I
```

Prints the option menu to the screen; calls the option line routine UTIL-OPT; and pauses for the entry of "M", "O", "A", "C", "B", "Q", "U", "R", "S", "F", "N" for the option variable L$.

```
68 LOCATE 9,15: PRINT          "M-RETURN TO MENU"
69 LOCATE 11,10:PRINT        "O-OBJECTIVE COEFFICIENTS"
70 LOCATE 12,10:PRINT        "A-CONSTRAINT COEFFICIENTS"
71 LOCATE 13,10:PRINT        "C-CONSTRAINT TYPES"
72 LOCATE 14,10:PRINT        "B-BOUNDED VARIABLES"
73 LOCATE 15,10:PRINT        "Q-QUADRATIC COEFFICIENTS"
74 LOCATE 16,10:PRINT        "U=EXECUTE ALGORITHM"
75 LOCATE 17,10:PRINT        "R-REPORT LISTING"
76 LOCATE 18,10:PRINT        "S-SAVE F-FETCH"
77 LOCATE 19,10:PRINT        "N-NEW PROBLEM"
78 LOCATE 22,1:PRINT "                  OX'+XQX'=z"
79 LOCATE 23,1:PRINT "1*X*2*Y>=c  1*X*(2*Y-3*Z)>=c  1*Y+a*(X-b)^2>=c"
80 GOSUB 1800:REM UTIL-OPT
81 LOCATE 21,8:INPUT "",L$
```

Calls either the objective function input subroutine INPT-OBJ, the constraint input subroutine INPT-CON, the constraint type input subroutine INPT-TYP, the upper and lower bound input subroutine INPT-BND, the quadratic $\bar{Q}$ matrix input subroutine INPT-QUD, the processing subroutine ALGR-KEY, or the report subroutine REPT-CRV based on the option variable L$.

```
82 CLS
83 H=0
84 G=2
85 IF L$<>"O" THEN 88
86 GOSUB 1200:REM INPT-OBJ
87 GOTO 82
88 IF L$<>"A" THEN 91
89 GOSUB 1300:REM INPT-CON
90 GOTO 82
91 IF L$<>"C" THEN 94
92 GOSUB 1400:REM INPT-TYP
93 GOTO 82
94 IF L$<>"B" THEN 97
95 GOSUB 1500:REM INPT-BND
96 GOTO 82
97 IF L$<>"Q" THEN 100
98 GOSUB 1600:REM INPT-QUD
99 GOTO 82
100 IF L$<>"U" THEN 109
101 TM=3600*VAL(MID$(TIME$,1,2))+60*VAL(MID$(TIME$,4,2))+VAL(MID$(TIME$,
    7,2))
102 GOSUB 3000:REM ALGR-KEY
103 TM=3600*VAL(MID$(TIME$,1,2))+60*VAL(MID$(TIME$,4,2))+VAL(MID$(TIME$,
    7,2))-TM
104 GOTO 68
105 IF L$<>"R" THEN 108
106 GOSUB 2100:REM REPT-CRV
107 GOTO 82
```

Saves the content of MD, ND, M#(ND+1,ND+8), A#(MD+1,ND+8), Q#(ND+1,ND+1), and R(MD+1+ND+1) to disk file "DATA" as an ASCI file if option "S" is selected.

```
108 IF L$<>"S" THEN 126
109 OPEN "O",#1,"DATA"
110 PRINT #1,STR$(MD)
111 PRINT #1,STR$(ND)
112 FOR I=1 TO ND+1
113 FOR J=1 TO ND+1
114 PRINT #1,STR$(Q#(I,J))
115 NEXT J
116 PRINT #1,STR$(M#(I,1))
117 PRINT #1,STR$(M#(I,2))
118 NEXT I
119 FOR I=1 TO MD+1
120 FOR J=1 TO ND+8
121 PRINT #1,STR$(A#(I,J))
122 NEXT J
```

```
123 PRINT #1,STR$(R(I))
124 NEXT I
125 CLOSE #1
```

Loads to MD, ND, M#(ND+1,2), A#(MD+1,ND+8), Q#(ND+1,ND+1), and R(MD+1+ND+1) the disk file "DATA" if option "F" is selected.

```
126 IF L$<>"F" THEN 152
127 OPEN "I",#1,"DATA"
128 INPUT #1,X$
129 MD=VAL(X$)
130 INPUT #1,X$
131 ND=VAL(X$)
132 FOR I=1 TO ND+1
133 FOR J=1 TO ND+1
134 INPUT #1,X$
135 Q#(I,J)=VAL(X$)
136 NEXT J
137 INPUT #1,X$
138 M#(I,1)=VAL(X$)
139 INPUT #1,X$
140 M#(I,2)=VAL(X$)
141 NEXT I
142 FOR I=1 TO MD+1
143 FOR J=1 TO ND+8
144 INPUT #1,X$
145 A#(I,J)=VAL(X$)
146 NEXT J
147 INPUT #1,X$
148 R(I)=VAL(X$)
149 NEXT I
150 CLOSE #1
151 GOTO 68
```

Restarts program for a new run if option "N" is selected.

```
152 IF L$="N" THEN RUN
153 GOTO 68
```

Utility Subroutines -- Files UTIL-OPT, UTIL-ERS, UTIL-CON, and UTIL-CHX

Same as for critical path method.

Input Subroutines -- Files INPT-OBJ, INPT-CON, INPT-TYP, and INPT-BND

The input subroutines used for the Beale's method are the same as for the dual simplex method. The linear portion of the quadratic

objective function is entered through the subroutine INPT-OBJ.

Qaudratic Matrix Subroutine -- INPT-QUD

Same as for Beale's method.

Report Subroutine -- File REPT-CRV

Same as for out-of-kilter method.

Constraint Keys Subroutine -- File ALGR-KEY

Same as for dual simplex method with nonlinear constraints except for calling the restart subroutine by changing lines:

```
3024 REM
3026 GOSUB 3100:REM ALGR-RST
```

Restart Algorithm Subroutine -- File ALGR-RST

The restart algorithm subroutine (ALGR-RST) saves the value of the current optimal solution in array C#(CYC,1) and then restarts Beale's method from its current optimal point by adding a duration constraint to the augmented simplex tableau which makes the tableau primal infeasible.

```
3100 REM                    * RESTART ALGORITHM *
3101 REM-------------------------ALGR-RST-----------------------------
```

Runs Beale's algorithm

For the first run of Beale's method, the simplex tableau is initialized to the point at the origin. For all remaining runs of the subroutine, the initialization section of the code is skipped.

```
3102 ITR=0
3103 IT=0
3104 CYC=1
3105 ER=0
3106 IF IT>IR THEN RETURN
3107 IF CYC>1 THEN 3110
3108 GOSUB 3300:REM ALGR-BEA
3109 GOTO 3111
3110 GOSUB 3318:REM ALGR-BEA+18
```

Totals the iteration count for all restarts.

```
3111 ITR=ITR+IT
3112 IT=ITR
```

Continues to next point on the cost curve if the error code indicates a optimal solution. Otherwise returns to main routine.

```
3113 IF ER=1 THEN 3117
3114 IF CYC=1 THEN RETURN
3115 ER=1
3116 RETURN
```

Sets C#(CYC,2) to the current optimal point on the cost curve.

```
3117 C#(CYC,1)=M#(DV+1,1)
3118 C#(CYC,2)=B#(1,1)
```

Sets the upper bound on the duration variable to the next duration point on the cost curve.

```
3119 M#(DV+1,1)=M#(DV+1,1)-INV
```

Stops the restart of Beale's method if the duration point is less than or equal to zero (0). Otherwise, increments cycle count and restarts algorithm.

```
3120 IF M#(DV+1,1)<=0 THEN 3115
3121 CYC=CYC+1
3122 GOTO 3105
```

Beale's Algorithm Subroutine -- File ALGR-BEA

Same as for Beale's Method except the nonlinear constraint subroutines must be called from Beale's algorithm (ALGR-BEA) by adding the lines:

```
3342 IF A#(K,ND+2)<>0# THEN GOSUB 4000:REM PAR-TANL
3344 IF A#(K,ND+3)<>0# THEN GOSUB 4200:REM HYP-TANL
3346 IF A#(K,ND+4)<>0# THEN GOSUB 4400:REM SHT-TANL
```

Quadratic Tableau Transformation -- File TRAN-QUD

Same as for Beale's Method.

Constraint Transformation Subroutine -- File TRAN-CON

Same as for dual simplex method.

Inversion Subroutine -- File TRAN-INV

Same as for dual simplex method.

Reinversion Subroutine -- File TRAN-RIV

Same as for dual simplex method.

Parabolic Subroutines -- Files PAR-TANA, PAR-TANL, or PAR-TANG

Same as for dual simplex method with nonlinear constraints.

Hyperbolic Subroutines -- Files HYP-TANA, HYP-TANL, or HYP-TANG

Same as for dual simplex method with nonlinear constraints.

Hyperbolic Sheet Subroutines -- Files SHT-TANA, SHT-TANL, or SHT-TANG

Same as for dual simplex method with nonlinear constraints.

Program Table of Content

Table 18 can be used to reconstruct the restart method with the above computer code and to organize subroutines from previous program listings. Since BASIC code is dependent on program line numbers for its subroutine branching, the statement numbers must be maintained as listed below.

The constraint keys subroutine ALGR-KEY must be changed to call the restart subroutine by changing lines:

```
3024 REM
3026 GOSUB 3100:REM ALGR-RST
```

and the nonlinear constraint subroutines must be called from Beale's algorithm (ALGR-BEA) by adding the lines:

```
3342 IF A#(K,ND+2)<>0# THEN GOSUB 4000:REM PAR-TANL
```

```
3344 IF A#(K,ND+3)<>0# THEN GOSUB 4200:REM HYP-TANL
3346 IF A#(K,ND+4)<>0# THEN GOSUB 4400:REM SHT-TANL
```

Table 18. Restart method BASIC program table of contents

File	Program lines	Page	Routines
MAIN-RST	0001-0153	261	Restart method
INPT-OBJ	1200-1220	61	Objective coefficient input subroutine
INPT-CON	1300-1328	63	Constraint coefficient input subroutine
INPT-TYP	1400-1440	65	Constraint type input subroutine
INPT-BND	1500-1523	67	Variable bounds input subroutine
INPT-QUD	1600-1628	166	Quadratic input subroutine
UTIL-OPT	1800-1803	22	Option line subroutine
UTIL-ERS	1850-1853	22	Erase option subroutine
UTIL-CON	1860-1866	22	Continue line subroutine
UTIL-CHX	1870-1882	23	Data check subroutine
REPT-CRV	2100-2125	105	Cost curve report subroutine
ALGR-KEY	3000-3057	200	Constraint key subroutine
ALGR-RST	3100-3122	266	Restart algorithm
ALGR-BEA	3300-3481	172	Beale's algortihm subroutine
TRAN-QUD	3550-3584	180	Quadratic tableau transformation subroutine
TRAN-CON	3600-3633	74	Constraint transformation subroutine
TRAN-INV	3700-3743	76	Basis inversion subroutine
TRAN-RIV	3900-3960	78	Basis reinversion subroutine
PAR-TANL	4000-4034	209	Supporting line for parabolic subroutine
HYP-TANL	4200-4221	217	Supporting line for hyperbolic subroutine
SHT-TANL	4400-4425	228	Supporting plane for sheet subroutine

Restart Method: Solutions to Example 12 Minimum Project Supervision Cost Curve Problem

With the prompts of restart method main routine screen, the project duration variable number (4) of example 12 and the ten (10) increment interval for the points on the project cost curve were entered.

Starting with an upper bound of the duration variable which was one hundred (100) increments, the method in increments of ten (10) reduced

the project duration from one hundred (100) to ten (10) increments for the costs shown in figure 68.

PROJECT COST CURVE

SOLUTION REACHED IN 1321 ITR 4577 SEC

NO.	DURATION	COST
1	74.39595	554.78018
2	74.39595	554.78018
3	74.39595	554.78018
4	70	556.79123
5	60	581.59037
6	50	654.85698
7	40	831.36106
8	30	1270.72451
9	20	2619.40329
10	10	10124.15667

OPTION ?

Figure 68. Restart method, using supporting planes derived with the line search algorithm, cost curve solution to example 12 minimum project supervision cost problem

SECTION II. INTEGER SOLUTION METHODS

No consideration was given to integer solutions for the models and problems derived in section I. In actual practice, construction crews work one (1) day at a time and consist of an integer number of people.

If the solutions found in the previous section are used for an actual project, the actual results would not be comparable with the results predicted by the models because the requirement of the job site would haphazardly round the the solution to a nonoptimal integer alternative.

Even if the solutions of section I are systematically round to an integer solution, the result would still not be the optimal integer solution. To find the integer solutions, some of the problems presented in section I must be solved by alternate methods.

In section II, the problems of section I will be re-solved with integer methods. In the case of the CPM and the simplex methods with linear constraints, the methods are inherently integer. For the variations of the simplex method with nonlinear constraints, the solutions are not inherently integer. For these methods, alternate methods will be presented.

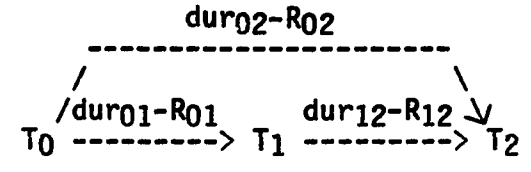
PROBLEMS WITH UNIMODULARITY

The CPM method was based on a time scale of integer points, so node times, activity durations, activity times, and floats were all restricted to integer values. The dual simplex method, in the general case, is not restricted to integer solutions, and yet the solutions to the examples are consistent with the CPM method in that they were all integer values.

The integer solutions to the examples are not an exception. The dual simplex method will always reach integer optimal solutions to the network schedule LP problem with linear cost functions.

Unimodularity of Simplex Tableau

To describe the characteristic of the minimum project cost problem with linear cost functions, the three (3) activity network of example 1 will be used. The example 1 problem has three (3) activities and three (3) nodes in the arrow diagram:



where:

T_i - Node time of node i
dur_{ij} - Normal duration of activity ij
R_{ij} - Increments of reduction of duration of activity ij
u_{ij} - Upper limit for reduction of duration of activity ij
l_{ij} - Lower limit for reduction of duration of activity ij
dur. - Project duration

The constraints of the minimum project cost problem:

$$
\begin{aligned}
T_2-(dur_{02}-R_{02})-T_0 &>= 0\\
T_1-(dur_{01}-R_{01})-T_0 &>= 0\\
T_2-(dur_{12}-R_{12})-T_1 &>= 0\\
-T_2 &>= -dur.\\
R_{02} &>= dur_{02}-u_{02}\\
-R_{02} &>= -dur_{02}+l_{02}\\
R_{01} &>= dur_{01}-u_{01}\\
-R_{01} &>= -dur_{01}+l_{01}\\
R_{12} &>= dur_{12}-u_{12}\\
-R_{12} &>= -dur_{12}+l_{12}
\end{aligned}
$$

$$T_0, T_1, \ldots\ldots R_{12} >= 0$$

can be written in matrix form as:

$$\overline{Ax} >= \overline{a}$$
$$\overline{x} >= \overline{0}$$

Tableau 27 is the $\overline{A}$ matrix of the example three (3) activity problem in tableau format.

T_0	T_1	T_2	R_{02}	R_{01}	R_{12}	
-1	0	1	1	0	0	$>= dur_{02}$
-1	1	0	0	1	0	$>= dur_{01}$
0	-1	1	0	0	1	$>= dur_{12}$
0	0	-1	0	0	0	$>= -dur.$
0	0	0	1	0	0	$>= dur_{02}-u_{02}$
0	0	0	-1	0	0	$>= -dur_{02}+l_{02}$
0	0	0	0	1	0	$>= dur_{01}-u_{01}$
0	0	0	0	-1	0	$>= -dur_{01}+l_{01}$
0	0	0	0	0	1	$>= dur_{12}-u_{12}$
0	0	0	0	0	-1	$>= -dur_{12}+l_{12}$

Tableau 27. Simplex unimodular $\overline{A}$ matrix

This tableau contains three (3) rows for each activity in the network. The network constraint row contains three (3) nonzero entries, two (2) positive one (1) entries and one (1) negative one (-1) entry. The second row and third rows are the upper and lower bounds on the duration reduction variable, each containing only one (1) nonzero entry.

In every case, the bounds row entry are directly below the constraint row entry, so that if the tableau were expanded to the general case, the above conditions would still hold.

With a simple row operation, the tableau elements corresponding to duration reduction variables can be reduced to zero (0) in every network constraint row. This operation has been performed in tableau 28.

T_0	T_1	T_2	R_{02}	R_{01}	R_{12}	
-1	0	1	0	0	0	$\geq -l_{02}$
-1	1	0	0	0	0	$\geq -l_{01}$
0	-1	1	0	0	0	$\geq -l_{12}$
0	0	-1	0	0	0	$\geq -dur.$
0	0	0	1	0	0	$\geq dur_{02}-u_{02}$
0	0	0	-1	0	0	$\geq -dur_{02}+l_{02}$
0	0	0	0	1	0	$\geq dur_{01}-u_{01}$
0	0	0	0	-1	0	$\geq -dur_{01}+l_{01}$
0	0	0	0	0	1	$\geq dur_{12}-u_{12}$
0	0	0	0	0	-1	$\geq -dur_{12}+l_{12}$

Tableau 28. Simplex unimodular $\overline{A}$ matrix

If $\overline{A}$ matrix is transformed and set equal to the matrix $\overline{B}$ then

$$\overline{B}=\overline{A}'$$

the tableau appears as shown in tableau 29.

T'	-1	-1	0	0	0	0	0	0	0	0
	0	1	-1	0	0	0	0	0	0	0
	1	0	1	-1	0	0	0	0	0	0
T"	0	0	0	0	1	-1	0	0	0	0
	0	0	0	0	0	0	1	-1	0	0
	0	0	0	0	0	0	0	0	1	-1

Tableau 29. Simplex unimodular $\overline{B}$ matrix

The $\bar{B}$ matrix can now be proven to have a "unimodular" structure by:

"Let $\bar{B}$ be an m by n matrix whose rows can be partitioned into two (2) disjoint sets, T' and T", such that $\bar{B}$, T', and T" have the following properties

(1) Every entry in $\bar{B}$ is 0,+1,-1.
(2) Every column contains at most two nonzero entries.
(3) If a column of $\bar{B}$ contains two nonzero entries and they are of opposite sign, then both are in T' or T"

then $\bar{B}$ has the unimodular property. If $\bar{B}$ is unimodular then $\bar{B}$ transpose in unimodular."[48]

The importance of unimodular matrices lies in the fact that the vertices of the following polyhedron:

$$\begin{array}{c} \bar{A}\bar{x}>=\bar{a} \\ \bar{d}-\bar{u}<=\bar{x}<=\bar{d}-\bar{l} \end{array}$$

are integers values, whenever $\bar{b},\bar{a},\bar{l},\bar{u},\bar{d}$ are integer values. The solutions for the dual simplex method lie on the vertices of the $\bar{A}$ matrix or the intersections of the constraints, so if the upper and lower limits on the duration reduction variables, the project duration, and all activity durations are integer, then all optimal solutions to the linear activity cost problems are also integer.

GOMORY'S ALL INTEGER METHOD WITH NONLINEAR CONSTRAINTS

In the minimum project cost problem with linear cost functions, the LP problem's constraint coefficient matrix was unimodular. This unimodularity guaranteed integer solutions. In the minimum project man count or cost problems with nonlinear constraints or with quadratic objective functions, integer solutions are no longer guaranteed.

Although dollar amounts are not normally restricted to integer values, man counts and project days are almost always integer values in construction schedules. One way to find an integer solution with dual simplex method with nonlinear constraints is to round up the noninteger solution. In the case of the minimum man count CP problem, the coefficient matrix is at least partially unimodular, so the round up method will give a good solution as demonstrated in Appendix B.

The more complex minimum project supervision cost problem can not be solved by the round up method. In the case where the round up method will not work, several integer methods provide integer solutions to the nonlinear problems.

An integer method which is closely related to the dual simplex method and which is compatible with the nonlinear convex constraints of the modified dual simplex method is Gomory's all integer method[49]. Although Gomory's method is usually not very efficient, in the cases where the optimum real solution is close to integer or the constraint coefficient matrix is at least partially unimodular, the method is competitive with other integer methods.

Theory of Gomory's Cut

Gomory's method is closely related to the dual simplex method in that it uses a simplex tableau and transforms the tableau using a Gauss-Jordan elimination. The primary difference between the two methods is that Gomory's method starts with and maintains an all integer tableau after every transformation.

Gomory's Tableau

In the dual simplex method, the tableau at any iteration is:

$$\begin{array}{rc|c} & 1 & \bar{s} \\ z= & \bar{c}'\bar{B}^{-1}\bar{b} & \bar{c}'\bar{B}^{-1} \\ \hline \bar{x}= & \bar{B}^{-1}\bar{b} & \bar{B}^{-1} \\ \hline s= & -\bar{a}_p+\bar{a}_p'\bar{B}^{-1}\bar{b} & \bar{a}_p'\bar{B}^{-1} \end{array}$$

In Gomory's method, the LP problem is changed slightly from the formulation used earlier and is written as:

$$\begin{aligned} &\text{minimize} && z \\ &\text{subject to:} && \begin{matrix} z+\bar{c}'(-\bar{x}) = 0 \\ -\bar{a}-\bar{A}\,(-\bar{x}) \geq \bar{0} \end{matrix} \\ & && \bar{x} \geq \bar{0} \text{ and integer} \\ & && \bar{c} \geq \bar{0} \end{aligned}$$

or as:

$$\begin{aligned} &\text{maximize} && -z \\ &\text{subject to:} && \begin{matrix} -z-\bar{c}'(-\bar{x}) = 0 \\ \bar{a}+\bar{A}\,(-\bar{x}) \leq \bar{0} \end{matrix} \\ & && \bar{x} \geq \bar{0} \text{ and integer} \\ & && \bar{c} \geq \bar{0} \end{aligned}$$

where all coefficients of the LP problem are restricted to integers; and

the variables, and therefore, the variable columns of the tableau, are the negative of the previous dual simplex tableau.

Assuming that Gomory's method transforms the augmented $\bar{B}$ matrix while still maintaining an all integer elements, the augmented $\bar{B}$ inverse for the Gomory's method becomes:

$$\begin{array}{c|c} 1 & -\bar{s} \\ \hline -1 & \bar{0}' \\ \hline \bar{B}^{-1}\bar{b} & -\bar{B}^{-1} \end{array}$$

so that the transformation of the objective function is:

$$\left| 0 , -\bar{c}' \right| * \left| \begin{array}{c|c} -1 & \bar{0}' \\ \hline \bar{B}^{-1}\bar{b} & -\bar{B}^{-1} \end{array} \right| = \left| -\bar{c}'\bar{B}^{-1}\bar{b} , \bar{c}'\bar{B}^{-1} \right|$$

and of any constraint i:

$$\left| a_i , \bar{a}_i' \right| * \left| \begin{array}{c|c} -1 & \bar{0}' \\ \hline \bar{B}^{-1}\bar{b} & -\bar{B}^{-1} \end{array} \right| = \left| -a_i+\bar{a}_i'\bar{B}^{-1}\bar{b} , -\bar{a}_i'\bar{B}^{-1} \right|$$

resulting in the Gomory tableau at any iteration of:

$$\begin{array}{c|c|c} & 1 & -\bar{s} \\ \hline z= & -\bar{c}'\bar{B}^{-1}\bar{b} & \bar{c}'\bar{B}^{-1} \\ \hline \bar{x}= & \bar{B}^{-1}\bar{b} & -\bar{B}^{-1} \\ \hline s= & -a_p+\bar{a}_p'\bar{B}^{-1}\bar{b} & -\bar{a}_p'\bar{B}^{-1} \end{array}$$

Any pivot row can now be written in the form:

$$\left| -a_p+\bar{a}_p'\bar{B}^{-1}\bar{b}, -\bar{a}_p'\bar{B}^{-1} \right|$$

or as the equation:

$$s=b_{p0}*1+\sum_{\substack{j=1,n \\ b_{p0}<0}} b_{pj}*(-X_j)$$

where the pivot row equation has all integer coefficients again assuming an all integer augmented $\overline{B}$ inverse.

Gomory's Cut

The transformation of the simplex tableau requires a division of the pivot row by the selected pivot element which would more than likely result in a noninteger transformed tableau. To eliminate this problem, Gomory devised an alternate pivot row or "cut"[36] which has all integer elements and a pivot element equal to one (1).

The coefficients of the original pivot row can be factored by first setting:

$$f_j=\frac{b_{pj}}{L}-[b_{pj}/L] \qquad 0<=f_j<1 \qquad L \text{ a positive number}$$

where $[b_{pj}/L]$ means the largest integer value less than or equal to $\frac{b_{pj}}{L}$. Let $r_j=L*f_j$ so that r_j is the noninteger component of $\frac{b_{pj}}{L}$ then:

$$\frac{r_j}{L}=\frac{L*f_j}{L}=\frac{b_{pj}}{L}-[b_{pj}/L] \text{ or } b_{pj}=[b_{pj}/L]*L+r_j$$

Since $0<=f_j<1$ and $L>0$, then $0<=r_j<L$. Now by substitution into the original pivot row the factors of the coefficients:

$$\{[1/L]*L+r\}*s=\{[b_{p0}/L]*L+r_0\}+\sum_{j=1,n}\{[b_{pj}/L]*L+r_j\}*(-X_j)$$

or rewritten as:

$$\sum_{j=1,n} r_j*X_j+r*s=r_0+L*\{[b_{p0}/L]+\sum_{j=1,n}[b_{pj}/L]*(-X_j)+[1/L]*(-s)\}$$

Let s be a positive integer value since we are seeking an all integer solution in the final tableau and the pivot row is all integer. This allows a portion of the factored pivot row to be set to the integer slack s% in:

$$s\%=\{\lceil b_{p0}/L\rceil+\sum_{j=1,n}\lceil b_{pj}/L\rceil*(-X_j)+\lceil 1/L\rceil*(-s)\}$$

If s% is greater than or equal to zero (0) in a feasible solution of the LP problem, then the slack equation can be used as a pivot row.

Say s% is a negative integer so that the equation cannot be used as a pivot row then:

$$\sum_{j=1,n} r_j*X_j + r*s=r_0+L*s\%<0$$

since $r_0<L$, $r_0>=0$, $L>0$ and $L*s\%<0$ by definition and:

$$\sum_{j=1,n} r_j*X_j + r*s<0$$

which would contradict the feasibility of the original pivot row at the optimal solution.

The slack equation can now be rewritten as:

$$[1/L]*s+s\%=\lceil b_{p0}/L\rceil+\sum_{j=1,n}\lceil b_{pj}/L\rceil*(-X_j)$$

or if L>1 then:

$$s\%=[b_{p0}/L]+\sum_{j=1,n}[b_{pj}/L]*(-X_j)$$

The slack equation can be used as an integer cut or pivot row for the Gomory transformation in which all the coefficients of the equation are integer and the pivot element b_{pp} is one (1) when L is chosen to be greater than the pivot element.

Selecting a Dual Simplex Pivot Row from Which to Derive a Gomory Cut

The Gomory cut was derived from a dual simplex pivot row. The dual simplex method of selecting the constraint with the most negative slack or which is the most distant from the current simplex point as the pivot

row would then appear to be the best source of a Gomory cut. Unfortunately, the simplex rules fail to converge in every case, so a weaker choice, which can be proven to converge[36], is used.

The choice is simply to use the first constraint in the Gomory tableau which has a negative slack as the constraint from which to generate the Gomory cut.

Gomory's Transformation

After finding a pivot row, the pivot element and a L which will result in the strongest Gomory cut must be selected so as to maintain primal feasiblity and lexicographic positivity after each transformation. The added requirement of lexicographic positivity, as in the perturbation method of the dual simplex method, is needed in Gomory's method to avoid cycling.

Gomory's method assumes that the starting simplex tableau is lexicographically positive so that in every column the first nonzero element is a positive number. If the tableau is not lexicographically positive, then the tableau has to be row reordered or a constraint must be added to force lexicographic positivity.

(In the dual feasible dual simplex tableau which has positive variable columns, the objective function coefficients and the basis inverse matrix form a lexicographically positive tableau in the first ND+1 rows of the tableau. Since the tableau is lexicographically positive at the first iteration and the basis inverse matrix remains nonsingular, the lexicographically smallest column of the tableau can

always be identified from just the first ND+1 rows of the tableau.)

Let tableau 30 represent the Gomory tableau in which first row is the objective row, rows one (1) through n are the augmented $\overline{B}$ inverse, and the row p is the pivot row selected for generating the Gomory cut. The tableau is assumed to be lexicographically positive. From the pivot

	1	$-s_1$	$-s_j$	$-s_p$	$-s_n$	
-z=	b_{00}	b_{01}	.	b_{0p}	b_{0n}	
x_k=	.	b_{k1}	.	b_{kp}	.	k
x_j=	b_{0j}		b_{ij}		b_{mj}	
x_n=	b_{0n}	.	.	.	b_{mn}	
s%=	b_{p0}	b_{p1}	b_{pj}	b_{pp}	b_{pn}	p pivot row
		1		p		
		max		pivot		
		$[\lvert b_{pj}\rvert/u_j]$		selected		

Tableau 30. Gomory's method tableau

row, among the $b_{pj}<0$, select the column which is lexicographically smallest and call it column p. Find integers $u_j>=1$ for each column in which $b_{pj}<0$ such that if the column elements of j are divided by the integer u_j the column will still be lexicographically larger than column p. Algebraically, that is:

$$\frac{1}{u_j}*\overline{b}_j \text{ lex}> \overline{b}_p$$

Now define $L_g=\max(|b_{pj}|/u_j)$ $j=1,b$ where b is the number of columns where $b_{pj}<0$, $|b_{pj}|$ is the absolute value of the pivot row element in column j, and g is the column with the maximum ratio L_g. This results

in a Gomory cut of:

$$s\% = [b_{p0}/L_g] + \sum_{j=1,n} [b_{pj}/L_g]*(-X_j)$$

Lexicographic Positivity After Transformation with Gomory Cut

With the Gomory cut derived above, a Gauss-Jordan elimination is used to transform the Gomory tableau. The tableau resulting from the transformation using the Gomory cut will remain lexicographically positive.

To show that this is true, consider the cases where the first positive element of the lexicographically positive column g occurs in row k. The elements of the pivot row corresponding to the g and p columns are:

$$[b_{pg}/L_g] = [b_{pg}*u_g/|b_{pg}|] = -u_g \qquad [b_{pp}/L_g] = [b_{pp}*u_g/|b_{pg}|] = -1$$

If the element $b_{kp}=0$ then the g column element b_{kg} will remain positive after the transformation and the column will remain lexicographically positive. But taking the worst case for column g in which $b_{kp}>0$, and pivoting, the partial tableau:

b_{kg}	b_{kp}	k row in which first positive element of column g occurs
.	.	
$-u_g$	-1	p
g	p	

is transformed to:

$b_{kg}-u_g*b_{kp}$	b_{kp}	k
.	.	
0	-1	p
g	p	

but $\frac{b_{kg}}{u_g} > \frac{b_{kp}}{}$ by definition and $u_g < b_{kg}$ since $b_{kp} \geq 0$, $b_{kg} > 0$, $u_g > 0$.

so: $$b_{kg} - u_g * b_{kp} > b_{kg} - b_{kg} * \frac{b_{kp}}{b_{kp}} = 0$$

and g column is still lexicographically positive.

For the other columns where $b_{pj} < 0$ and the 1 row is the row in which the first positive element of column j occurs the pivot row elements are:

$$\lceil b_{pj}/L_g \rceil \leq b_{pj} * u_g / |b_{pg}| \leq b_{pj} * b_{kg} / |b_{pg}| * b_{kp}$$

Again transforming the column element b_{kj}:

$$b_{1j} - \lceil a_{pj}/L_g \rceil * b_{1p} \geq b_{1j} - a_{pj} * b_{kg} * b_{1p} / |a_{pg}| * b_{kp} > 0$$

since $b_{1j} > 0$ and $b_{pj} < 0$ by definition and $b_{kg} > 0$, $b_{1p} \geq 0$, $b_{kp} > 0$.

For the other columns where $b_{pj} \geq 0$ the pivot element b_{1j} transforms to:

$$b_{1j} + \lceil b_{pj}/L_g \rceil * b_{1p} > 0$$

since $b_{1j} > 0$, $\lceil b_{pj}/L_g \rceil \geq 0$, $b_{1p} \geq 0$.

Theory of Wilson's Cut

Gomory's cut can be written as:

$$s\% = \lceil b_{p0}/L_g \rceil + \sum_{b_{jp} > 0} \lceil b_{jp}/L_g \rceil * (-X_j) + \sum_{b_{jp} \leq 0} \lceil b_{jp}/L_g \rceil * (-X_j)$$

where the positive and negative coefficients have been segregated and L_g is chosen by the Gomory algorithm as the minimum L which will maintain lexicographic positivity of the columns of the Gomory tableau. Actually, any larger L can be chosen or $\inf\{L \geq L_g\}$. Wilson's cut[50] is derived by finding an L that strengthens Gomory's cut.

If Gomory's cut is rewritten as:

$$s\%' = -\lceil |-b_{0p}|/L \rceil - \sum_{b_{jp} > 0} \lceil |+b_{jp}|/L \rceil * X_j + \sum_{b_{jp} \leq 0} \lceil |-b_{jp}|/L \rceil * X_j)$$

where s%' is now the slack for differing values of L assuming a constant set of X variable values then s%'is more negative than s% if:

$[|-b_{0p}|/L]$

is increased or:

$[|+b_{jp}|/L]$

is increased or:

$[|-b_{jp}|/L]$

is decreased.

Since decreasing s%' strengthens the cut by making the slack more negative, increasing the first two (2) components of the equation would require a smaller L than the minimum Lg from the Gomory's cut. But if a L larger than Lg can be found such that:

$$[|-b_{0p}|/L]=[|-b_{0p}|/L_g]$$

$$[|+b_{jp}|/L]=[|+b_{jp}|/L_g] \text{ for all } b_{jp}>=0$$

$$[|-b_{jp}|/L]<[|-b_{jp}|/L_g] \text{ for all } b_{jp}<0$$

then s%'<s% for the same set of X_j values.

To find Wilson's L_w, define the set of j=0,J as:

$$L_0=b_{0p}/(1+[b_{0p}/L_g])-\text{small amount}$$

$$L_j=b_{jp}/[b_{jp}/L_g] \text{ for all } b_{jp}>0$$

then let:

$$L_w=\text{maximum } \{L_g,\min(L_j),j=0,J\}$$

so that Wilson's cut is now:

$$s\%'=[b_{0p}/L_w]+\sum_{b_{jp}>0}[b_{jp}/L_w]*(-X_j)+\sum_{b_{jp}<=0}[b_{jp}/L_w]*(-X_j)$$

Selecting a Dual Simplex Pivot Row from Which to Derive a Wilson Cut

In Gomory's method the first constraint of the LP problem with a negative slack was used as the pivot row. This method can be proven to converge. Unfortunately, the method has a poor rate of convergence.

Utilizing the convergence proof of Gomory's method, another method for selecting a pivot row can be derived that usually converges faster.

If the pivot column is found using the Gomory method as described earlier for each transformed constraint of the LP problem that has a negative slack, then a pivot row can be is selected as the row which has the lexicographically largest pivot column from all the potential rows.

This approach favors the rows of the tableau which have the greatest impact on the first column of the Gomory tableau; and in that manner, hopefully leads to a primal feasible tableau at a faster rate.

Gomory's Method with Nonlinear Constraints BASIC Code

The following code is a modification of Gomory's all integer method incorporating suggestions by Wilson[50] and Zoints[51]. The algorithm is coded to take advantage of any unimodularity of the simplex tableau by seeking any simplex transformation which would result in an all integer tableau as well as the supporting planes of the simplex methods.

All the programs listed in the text consist of a main calling routine, and a series of input, output, and processing subroutines. The BASIC routines are listed in the text and as BASIC files on a computer disk compatible with IBM micro-computers.

Gomory's Method Main Routine -- File MAIN-GOM

The Gomory's method main routine (MAIN-GOM) dimensions eleven (11) data arrays; writes the options menu as shown in figure 69; calls the utility subroutines UTIL-OPT, UTIL-ERS, UTIL-CON,UTIL-CHX; calls the input and output routines INPT-OBJ, INPT-CON, INPT-TYP, INPT-BND, and

```
                    GOMORY'S METHOD

          NUMBER OF CONSTRAINTS                10
          NUMBER OF VARIABLES                  14
          MAXIMUM ITERATIONS                 1000

                    M-RETURN TO MENU

             O-OBJECTIVE COEFFICIENTS
             A-CONSTRAINT COEFFICIENTS
             C-CONSTRAINT TYPE
             B-BOUNDED VARIABLES
             U-EXECUTE ALGORITHM
             R-REPORT LISTING
             S-SAVE  F-FETCH
             N-NEW PROBLEM

        OPTION ?

          1*X*2*Y>=c   1*X*(2*Y-3*Z)>=c   1*Y*a*(X-b)^2>=c
```

Figure 69. Gomory's method main menu screen

REP-SMP; calls and times the processing algorithm ALGR-GOM; and saves and fetches the input data to a disk file.

```
1 REM                        *GOMORY'S METHOD*
2 REM-----------------------------MAIN-GOM-------------------------
3 REM     BI# - MACHINE INFINITE
4 REM     CO  - PIVOT COLUMN FOR SIMPLEX TRANSFORMATION
5 REM     ER  - ERROR KEY
6 REM     IR  - MAXIMUM NUMBER OF ITERATIONS
7 REM     MD  - NUMBER OF CONSTRAINTS
8 REM     ND  - NUMBER OF VARIABLES
9 REM     RO  - PIVOT ROW FOR SIMPLEX TRANSFORMATION
10 REM    SM# - MACHINE ZERO
11 REM    A#(MD+1,ND+8) - ORIGINAL DATA AND KEYS
12 REM    B#(ND+2,ND+1) - PRIMAL-DUAL MATRIX
13 REM    C(ND+1) - COLUMNS ELIGIBLE FOR GOMORY PIVOT
14 REM    H#(ND+1) - SOLUTION FROM LAST ITERATION
15 REM    M#(ND+1,2) - LOWER AND UPPER BOUNDS OF VARIABLES
16 REM    P#(ND+1) - PIVOT ROW VECTOR
17 REM    R(MD+1) - CONSTRAINT TYPE (1-">=",0-"=",-1-"<=")
18 REM    S#(ND+1) - TEMPORARY GOMORY PIVOT ROW
19 REM    U(ND+1) - MAX DIVISOR MAINTAINING LEX MIN COLUMN
```

```
20 REM    V#(ND+1) - TEMPORARY SIMPLEX PIVOT ROW
21 REM    X#(ND) - SOLUTION VECTOR
22 REM ------------------------------------------------------------
```

Sets MD to the default number of constraints in the integer CP problem and ND to the number of variables. Sets IR to the default maximum number of iterations. Sets BI# to a number considered machine infinite and SM# to a number considered machine zero.

```
23 MD=0
24 ND=0
25 IR=1000
26 BI#=1E+10
27 SM#=1E-10
```

Prompts and reads from the keyboard the number of constraints MD in the CP model; the number of variables ND; and the maximum number of iterations IR allowed before Gomory's algorithm is stopped.

```
28 CLS
29 LOCATE 1,10:PRINT "GOMORY'S METHOD"
30 LOCATE 3,1:PRINT "NUMBER OF CONSTRAINTS":LOCATE 3,31:INPUT "",L$
31 GOSUB 1870:REM UTIL-CHX
32 IF Z#<>BI# THEN MD=Z#
33 LOCATE 3,30:PRINT MD,"    ":LOCATE 4,1:PRINT "NUMBER OF VARIABLES"
   :LOCATE 4,31:INPUT "",L$
34 GOSUB 1870:REM UTIL-CHX
35 IF Z#<>BI# THEN ND=Z#
36 LOCATE 4,30:PRINT ND,"    ":LOCATE 5,1:PRINT "MAXIMUM ITERATIONS"
   :LOCATE 5,31:INPUT "",L$
37 GOSUB 1870:REM UTIL-CHX
38 IF Z#<>BI# THEN IR=Z#
39 LOCATE 5,30:PRINT IR,"    "
```

Dimensions the A#(MD+1,ND+8) array which contains the constraint coefficients. Dimensions the augmented $\overline{B}$ inverse and augmented $\overline{B}$ matrix B#(ND+2,ND+1). Dimensions the holding array H#(ND+1), the upper and lower bound array M#(ND+1,2), the pivot row P#(ND+1), the constraint type array R(MD+1), and the Gomory pivot row selection array S#(ND+1). Dimensions the column selection array C(ND+1), the largest integer divisor array U(ND+1), the simplex pivot row selection array V#(ND+1), and the solution vector X#(ND).

The arrays S#(ND+1) and V#(ND+1) hold the pivot rows from which the Wilson cut is derived and a simplex transformation is made. If a simplex transformation can be made while still maintaining a integer tableau the simplex pivot row is used. If not, then the Wilson cut is used.

```
40 DIM A#(MD+1,ND+8)
41 DIM B#(ND+2,ND+1)
42 DIM H#(ND+1)
43 DIM M#(ND+1,2)
44 DIM P#(ND+1)
45 DIM R(MD+1)
46 DIM S#(ND+1)
47 DIM C(ND+1)
48 DIM U(ND+1)
49 DIM V#(ND+1)
50 DIM X#(ND)
```

Initializes the constraint array type to all greater than or equals.

```
51 FOR I=1 TO MD+1
52 R(I)=1
53 NEXT I
```

Prints the option menu to the screen; calls the option line routine UTIL-OPT; and pauses for the entry of "M", "A", "C", "B", "U", "R", "S", "F", "N" for the option variable L$.

```
54 LOCATE 8,15: PRINT          "M-RETURN TO MENU"
55 LOCATE 10,10:PRINT       "O-OBJECTIVE COEFFICIENTS"
56 LOCATE 11,10:PRINT       "A-CONSTRAINT COEFFICIENTS"
57 LOCATE 12,10:PRINT       "C-CONSTRAINT TYPES"
58 LOCATE 13,10:PRINT       "B-BOUNDED VARIABLES"
59 LOCATE 14,10:PRINT       "U-EXECUTE ALGORITHM"
60 LOCATE 15,10:PRINT       "R-REPORT LISTING"
61 LOCATE 16,10:PRINT       "S-SAVE F-FETCH"
62 LOCATE 17,10:PRINT       "N-NEW PROBLEM"
63 LOCATE 23,1:PRINT "1*X*2*Y>=c  1*X*(2*Y-3*Z)>=c  1*Y*a*(X-b)^2>=c"
64 GOSUB 1800:REM UTIL-OPT
65 LOCATE 21,8:INPUT "",L$
```

Calls either the objective function input subroutine INPT-OBJ, the constraint input subroutine INPT-CON, the constraint type input subroutine, the upper and lower bound input subroutine INPT-BND, the processing subroutine ALGR-GQM, or the report subroutine based on the option variable L$.

```
66 CLS
67 H=0
68 G=2
69 IF L$<>"O" THEN 72
70 GOSUB 1200:REM INPT-OBJ
71 GOTO 66
```

```
72 IF L$<>"A" THEN 75
73 GOSUB 1300:REM INPT-CON
74 GOTO 66
75 IF L$<>"C" THEN 78
76 GOSUB 1400:REM INPT-TYP
77 GOTO 66
78 IF L$<>"B" THEN 81
79 GOSUB 1500:REM INPT-BND
80 GOTO 66
81 IF L$<>"U" THEN 90
82 TM=3600*VAL(MID$(TIME$,1,2))+60*VAL(MID$(TIME$,4,2))+VAL(MID$(TIME$,7
   ,2))
83 GOSUB 3000:REM ALGR-KEY
84 OB#=B#(1,1)
85 FOR I=1 TO ND
86 X#(I)=B#(I+1,1)
87 NEXT I
88 TM=3600*VAL(MID$(TIME$,1,2))+60*VAL(MID$(TIME$,4,2))+VAL(MID$(TIME$,7
   ,2))-TM
89 GOTO 54
90 IF L$<>"R" THEN 93
91 GOSUB 2200:REM REPT-SMP
92 GOTO 66
```

Saves the content of MD, ND, M#(ND+1,2), A#(MD+1,ND+8), and R(MD+1) to the disk file "DATA" as an ASCI file if option "S" is selected.

```
93 IF L$<>"S" THEN 111
94 OPEN "O",#1,"DATA"
95 PRINT #1,STR$(MD)
96 PRINT #1,STR$(ND)
97 FOR I=1 TO ND+1
98 FOR J=1 TO ND+1
99 PRINT #1,""
100 NEXT J
101 PRINT #1,STR$(M#(I,1))
102 PRINT #1,STR$(M#(I,2))
103 NEXT I
104 FOR I=1 TO MD+1
105 FOR J=1 TO ND+8
106 PRINT #1,STR$(A#(I,J))
107 NEXT J
108 PRINT #1,STR$(R(I))
109 NEXT I
110 CLOSE #1
```

Loads to MD, ND, M#(ND+1,2), A#(MD+1,ND+8), and R(ND+1) to disk file "DATA" if option "F" is selected.

```
111 IF L$<>"F" THEN 136
112 OPEN "I",#1,"DATA"
113 INPUT #1,X$
114 MD=VAL(X$)
115 INPUT #1,X$
116 ND=VAL(X$)
117 FOR I=1 TO ND+1
118 FOR J=1 TO ND+1
119 INPUT #1,X$
120 NEXT J
121 INPUT #1,X$
122 M#(I,1)=VAL(X$)
123 INPUT #1,X$
124 M#(I,2)=VAL(X$)
125 NEXT I
126 FOR I=1 TO MD+1
127 FOR J=1 TO ND+8
128 INPUT #1,X$
129 A#(I,J)=VAL(X$)
130 NEXT J
131 INPUT #1,X$
132 R(I)=VAL(X$)
133 NEXT I
134 CLOSE #1
135 GOTO 54
```

Restarts program for a new run if option "N" is selected.

```
136 IF L$="N" THEN RUN
137 GOTO 54
```

Utility Subroutines -- Files UTIL-OPT, UTIL-ERS, UTIL-CON, and UTIL-CHX

Same as for critical path method.

Input Subroutines -- Files INPT-OBJ, INPT-CON, INPT-TYP, and INPT-BND

Same as for dual simplex method.

Output Subroutine -- File REPT-SMP

Same as for dual simplex method.

Constraint Keys Subroutine -- File ALGR-KEY

Same as for dual simplex method with nonlinear constraints.

Gomory's Algorithm Subroutine -- File ALGR-GOM

Gomory's algorithm uses an all integer dual feasible tableau which is the same as the simplex tableau intialized to represent the current point at the origin and the set of constraints that form the zero axes.

Although Gomory's cut was derived using a tableau in which the variables were the negative of the simplex tableau, this form restricted the use of computer subroutines from the dual simplex method. To compensate for the change in sign of the variable coefficients, the negative of the Gomory cut is actually used as the pivot row.

Starting with the initial tableau, the algorithm in an iterative fashion proceeds through the following steps.

(1) From the constraints of the integer CP problem, select the constraint violated by the current point which is geometrically farthest from the current point. If no constraints are violated, set ER=1 and return to the main routine. Otherwise, go to step (2).

(2) From the new points of intersection formed by the constraint selected in step (1) and the constraints defining the current point, select a point which minimizes the value of the objective function and does not violate the current constraints. This must be done using the perturbation algorithm described for the dual simplex method. If no pivot element can be found the problem is infeasible, so set ER=2 and return to the main routine. Otherwise, go to step (3).

(3) With the pivot element found in step (2), divide the elements of

the selected constraint. If the constraint remains all integer, then use the constraint as a simplex pivot row and go to step (6). Otherwise, go to step (4).

(4) For all the constraints which have negative slack, transform the constraint. Find the column for each transformed constraint which is lexicographically smallest. Of all the the constraints for which the lexicographically smallest column have been determined, select the constraint with the lexicographically largest pivot column for the pivot row. Go to step (5).

(5) From the constraint selected in step (4) derive a Gomory cut and then strengthen the cut with Wilson's method presented above. Use the negative of Wilson's cut as the pivot row and go to step (6).

(6) Use a Gauss-Jordan elimination to transform the current tableau to represent a new integer point and the constraints which define it. If the transformation fails, then set ER=3 and return to the main routine. Otherwise, go to step (7).

(7) Set the current point equal to the new point and return to step (1). If the cycle or iteration of the above steps is repeated more than a preset limit, set ER=0 and return to the main routine.

```
3300 REM              *GOMORY'S ALGORITHM SUBROUTINE*
3301 REM------------------------ALGR-GOM-------------------------------
```

Initializes the simplex tableau to represent the origin by setting the augmented $\bar{B}$ matrix to the identity matrix and the first row of the simplex tableau to objective function. Sets the iteration count IT to zero (0) and the error code ER to zero (0).

By using the simplex tableau in which the starting tableau is the point at the origin, the tableau is guaranteed to be

lexicographically positive in every column.

```
3302 FOR I=2 TO ND+1
3303 FOR J=1 TO ND+1
3304 B#(I,J)=0#
3305 NEXT J
3306 NEXT I
3307 IT=0
3308 FOR I=2 TO ND+1
3309 B#(I,I)=1#:B#(I,1)=M#(I,2)
3310 B#(1,I)=A#(1,I)
3311 H#(I)=BI#
3312 NEXT I
```

Increments the iteration count by one (1). Sets ER=0 if the iteration count is greater than the preset limit IR.

```
3313 IT=IT+1
3314 ER=0
3315 IF IT>IR THEN RETURN
```

Sets the maximum first positive column element MA# to zero (0). Sets the maximum pivot element of the constraint to be used for a Gomory cut MG# to zero (0). Sets the greatest distance of a constraint from the current point MS# to zero (0). Sets the minimum number of zero elements at the top of a Gomory pivot column RA to infinity.

```
3316 MA#=0#
3317 MG#=0#
3318 MS#=0#
3319 RA=BI#
```

Searches the upper and lower bounds constraints and the integer CP problem constraints for both a dual simplex pivot row and a Gomory pivot row.

```
3320 FOR N=1 TO ND+ND+MD
```

Searches the lower bounds for a constraint that has a negative slack and sets S#(ND+1) equal to the transformation of the constraint.

```
3321 Z#=1#
3322 IF N>ND THEN 3330
3323 I=N+1
3324 IF B#(I,1)-M#(I,2)>=0# THEN 3417
3325 S#(1)=B#(I,1)-M#(I,2)
3326 FOR J=2 TO ND+1
3327 S#(J)=-B#(I,J)
```

```
3328 NEXT J
3329 GOTO 3367
```

Searches the upper bounds for a constraint that has a negative slack and sets S#(ND+1) equal to the transformation of the constraint.

```
3330 IF N>ND+ND THEN 3341
3331 J=N-ND+1
3332 IF M#(J,1)<=0# THEN 3417
3333 A#=M#(J,1)
3334 IF A#=BI# THEN A#=0#
3335 IF A#-B#(J,1)>=0# THEN 3417
3336 S#(1)=A#-B#(J,1)
3337 FOR K=2 TO ND+1
3338 S#(K)=B#(J,K)
3339 NEXT K
3340 GOTO 3367
```

Searches the integer CP problem constraints for a constraint that has a negative slack and sets S#(ND+1) equal to the transformation of the constraint.

```
3341 K=N-ND-ND+1
3342 REM-----------------------------------------------------------------
3343 IF A#(K,ND+2)<>0# THEN GOSUB 4100:REM PAR-DEPL
3344 IF A#(K,ND+3)<>0# THEN GOSUB 4300:REM HYP-DEPL
3345 IF A#(K,ND+4)<>0# THEN GOSUB 4500:REM SHT-DEPL
3346 REM            NONLINEAR CONSTRAINT SUBROUTINES
3347 REM-----------------------------------------------------------------
3348 S#(1)=-A#(K,1)
3349 Z#=0#
3350 FOR I=2 TO ND+1
3351 S#(I)=S#(I)+A#(K,I)*B#(I,I)
3352 Z#=Z#+A#(K,I)*A#(K,I)
3353 S#(I)=0#
3354 NEXT I
3355 Z#=CDBL(SQR(Z#))
3356 SN=R(K)
3357 IF SN=0 THEN SN=-SGN(S(1))
3359 S#(1)=S#(1)*SN
3360 IF S#(1)>=0# THEN 3417
3361 FOR I=2 TO ND+1
3362 IF A#(K,I)=0# THEN 3366
3363 FOR J=2 TO ND+1
3364 S#(J)=S#(J)-A#(K,I)*B#(I,J)*SN
3365 NEXT J
3366 NEXT I
```

Selects the constraint whose transformation is in S#(ND+1) which is the greatest distance from the simplex current point and sets V#(ND+1) equal to it as the simplex pivot row.

The variable Z# is the square root of the sum of the squares of the coefficients of the orginal constraint.

```
3367 IF S#(1)/Z#>MS# THEN 3372
3368 FOR I=1 TO ND+1
3369 V#(I)=S#(I)
3370 NEXT I
3371 MS#=S#(1)/Z#
```

Sets the array C(ND+1) to the column numbers eligible for a Gomory pivot column for the constraint in S#(ND+1).

```
3372 CN=0
3373 FOR J=2 TO ND+1
3374 IF S#(J)>=0# THEN 3378
3375 CN=CN+1
3376 C(CN)=J
3377 CC=J
3378 NEXT J
```

Sets ER=2 and returns to the main routine if no column is found for a Gomory pivot column for the constraint now saved in S#(ND+1).

```
3379 ER=2
3380 IF CN=0 THEN RETURN
```

Finds the number of zero (0) elements heading the Gomory pivot column R for the constraint now saved in S#(ND+1)

```
3381 IF CN=1 THEN 3401
3382 FOR I=1 TO ND+2
3383 IF I=ND+2 THEN STOP
3384 MI#=BI#
3385 FOR J=1 TO CN
3386 IF C(J)=0 THEN 3389
3387 IF MI#<=B#(I,C(J)) THEN 3389
3388 MI#=B#(I,C(J))
3389 NEXT J
3390 C=0
3391 FOR J=1 TO CN
3392 IF C(J)=0 THEN 3398
3393 IF MI#<B#(I,C(J)) THEN 3397
3394 CC=C(J)
3395 C=C+1
```

```
3396 GOTO 3398
3397 C(J)=0
3398 NEXT J
3399 IF C=1 THEN 3401
3400 NEXT I
3401 FOR J=1 TO ND+1
3402 IF B#(J,CC)=0# THEN 3406
3403 IF B#(J,CC)<0# THEN STOP
3404 R=J
3405 GOTO 3407
3406 NEXT J
```

Sets P#(ND+1) equal to the constraint whose Gomory pivot column is lexicographically most positive and whose slack is also negative.

```
3407 IF R>RA THEN 3417
3408 IF R=RA AND MA#>B#(R,CC) THEN 3417
3409 IF MA#=B#(R,CC) AND MG#>S#(1) THEN 3417
3410 RA=R
3411 MA#=B#(R,CC)
3412 MG#=S#(1)
3413 CO=CC
3414 FOR I=1 TO ND+1
3415 P#(I)=S#(I)
3416 NEXT I
3417 NEXT N
```

Sets ER=1 and returns to the main routine if no Gomory pivot constraints are found.

This is the same as finding no constraints eligible for a Gomory constraint or the tableau is primal feasible.

```
3418 ER=1
3419 IF RA=BI# THEN RETURN
```

Finds a simplex pivot element in the transformed simplex pivot Z#(ND+1) using the perturbation method.

```
3420 CN=0
3421 FOR I=2 TO ND+1
3422 IF V#(I)>=0# THEN 3425
3423 CN=CN+1
3424 C(CN)=I
3425 NEXT I
3426 CC=C(1)
3427 IF CN=1 THEN 3446
3428 FOR I=1 TO ND+2
```

```
3429 IF I=ND+2 THEN STOP
3430 MI#=BI#
3431 FOR J=1 TO CN
3432 IF C(J)=0 THEN 3434
3433 IF MI#>B#(I,C(J))/-V#(C(J)) THEN MI#=B#(I,C(J))/-V#(C(J))
3434 NEXT J
3435 C=0
3436 FOR J=1 TO CN
3437 IF C(J)=0 THEN 3443
3438 IF B#(I,C(J))/-V#(C(J))>MI# THEN 3442
3439 C=C+1
3440 CC=C(J)
3441 GOTO 3443
3442 C(J)=0
3443 NEXT J
3444 IF C=1 THEN 3446
3445 NEXT I
```

Divides the simplex pivot row by the pivot element. Sets the pivot row to be used for the transformation of the simplex tableau to the simplex pivot row if all the elements of the simplex row when divided by the pivot element are integer.

```
3446 FOR I=1 TO ND+1
3447 IF V#(I)/V#(CC)<>INT(V#(I)/V#(CC)) THEN 3455
3448 NEXT I
3449 CO=CC
3450 P#(1)=V#(1)/-V#(CC)
3451 FOR I=2 TO ND+1
3452 P#(I)=V#(I)/V#(CC)
3453 NEXT I
3454 GOTO 3498
```

Finds the Gomory pivot column for the constraint in P#(ND+1).

```
3455 U(CO)=1
3456 CN=0
3457 FOR I=2 TO ND+1
3458 IF P#(I)>=0# THEN 3461
3459 CN=CN+1
3460 C(CN)=I
3461 NEXT I
3462 IF CN=1 THEN 3481
3463 L#=B#(RA,CO)
3464 FOR I=1 TO CN
3465 IF C(I)=CO THEN 3480
3466 MI#=BI#
3467 P#=B#(RA,C(I))
```

```
3468 IF RA=1 THEN 3472
3469 FOR J=1 TO RA-1
3470 IF B#(J,C(I))<>0# THEN 3479
3471 NEXT J
3472 MI#=INT(P#/L#)
3473 IF (P#/MI#)>L# THEN 3479
3474 FOR J=RA+1 TO ND+1
3475 IF (B#(J,C(I))/MI#)>B#(J,CO) THEN 3479
3476 IF (B#(J,C(I))/MI#)<B#(J,CO) THEN 3478
3477 NEXT J
3478 MI#=MI#-1#
3479 U(C(I))=MI#
3480 NEXT I
```

Finds the Gomory divisor L# to be used to "integerize" the cut.

```
3481 L#=0#
3482 FOR I=1 TO CN
3483 IF U(C(I))=BI# GOTO 3486
3484 IF L#>=(ABS(P#(C(I)))/U(C(I))) GOTO 3486
3485 L#=(ABS(P#(C(I)))/U(C(I)))
3486 NEXT I
```

Strengthens the Gomory cut using Wilson's method.

```
3487 P#=BI#
3488 IF INT(P#(1)/L#)<>-1 THEN P#=P#(1)/(1#+INT(P#(1)/L#)-SM#)
3489 FOR I=2 TO ND+1
3490 IF INT(P#(I)/L#)<=0 THEN 3492
3491 IF P#>P#(I)/(INT(P#(I)/L#)) THEN P#=P#(I)/(INT(P#(I)/L#))
3492 NEXT I
3493 IF L#<P# THEN L#=P#
3494 P#(1)=INT(P#(1)/L#)
3495 FOR I=2 TO ND+1
3496 P#(I)=-INT(P#(I)/L#)
3497 NEXT I
```

Transforms the simplex tableau using the dual simplex derived pivot row. If none is found that will result in an all integer tableau after the pivot, transforms the tableau using the Wilson cut.

```
3498 GOSUB 3700:REM TRAN-INV
3499 GOTO 3313
```

Constraint Transformation Subroutine -- File TRAN-CON

Same as for dual simplex method.

Inversion Subroutine -- File TRAN-INV

Same as for dual simplex method.

Nonlinear Convex Constraints for the Integer Methods

The same approach as used in the nonlinear modification of the dual simplex method can also be used with the Gomory method. Although a Gomory "integerized" pivot row can be generated from a supporting plane which is the tangent plane to a nonlinear convex constraint, in practice the method converges to slowly to be of practical use.

A stronger supporting plane or "deep cut" for the integer methods is one in which the plane is "moved" into the feasible region of the CP problem in such a manner that it cuts off the maximum amount of the feasible region without excluding any integer points of the region.

By extending the theory of the supporting plane or tangent plane which was derived from a single point on a constraint's surface, a deep cut supporting plane can be found using one point on a surface for each variable or dimension of a given constraint.

A Cut Through Three Points on a Hyperbolic of Two Sheets Surface

Leaving the derivation of the points for later, three (3) points on a hyperbolic of two sheets surface define a plane. Let one (1) point be (a%,b%,z%) with all three (3) coordinate values integer and satisfying the function X*(Y-Z)=c in which c>0 and integer. In the same Z=z% plane another point (g%,d%,z%) can be found on the surface where g% and d% are again integers coordinate values. A third point (g%,d%+1,z%+1) in the X=g% plane can also be found such that all the coordinates are integer

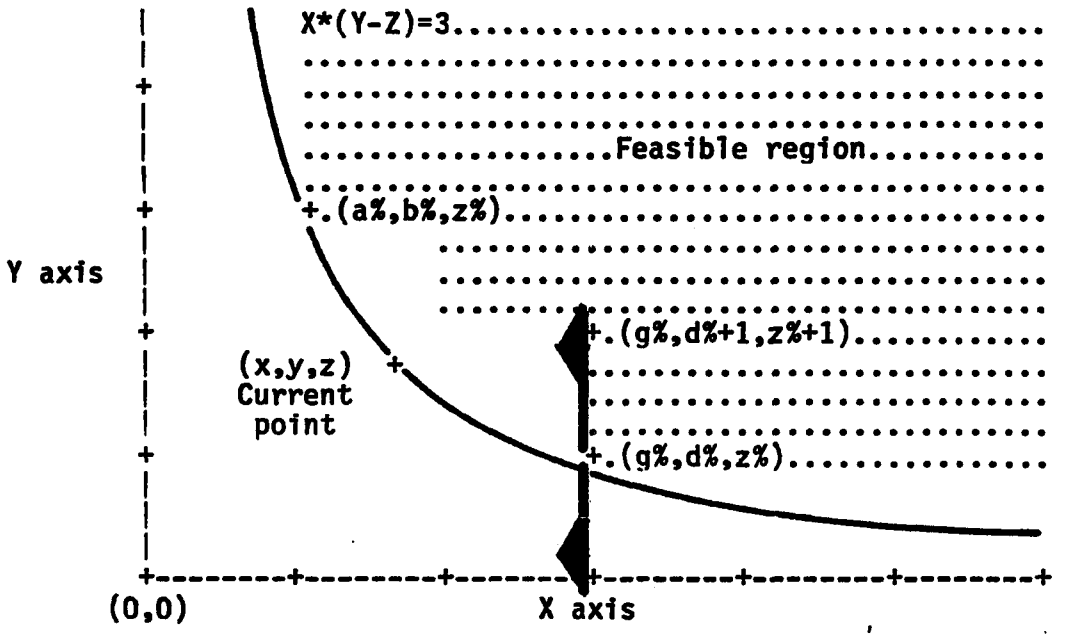


Figure 70. Hyperbolic sheet constraint deep cut X and Y axes view

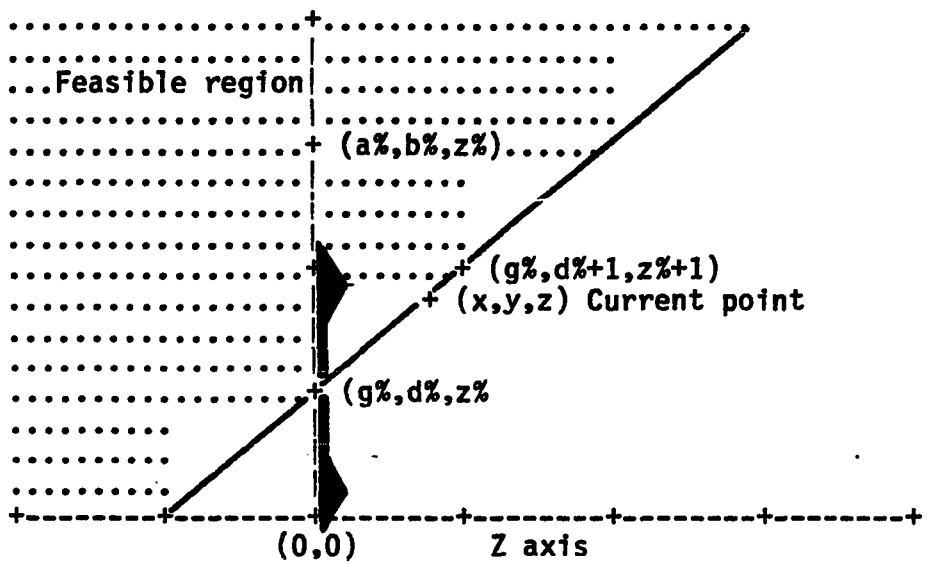


Figure 71. Hyperbolic sheet constraint deep cut Y and Z axes view

values and the point is again on the surface as shown in figures 70 and 71.

If a plane i*X+j*Y+k*Y=c is passed through the three (3) points:

$$\begin{aligned} i*(a\%)+j*(b\%) \quad +k*(z\%)&=c \\ i*(g\%)+j*(d\%) \quad +k*(z\%)&=c \\ i*(g\%)+j*(d\%+1)+k*(z\%+1)&=c \end{aligned}$$

the coefficients i, j, k, can be found as:

$$i=\frac{c*(b\%-d\%)}{a\%*(b\%-d\%)+(b\%-z\%)*(g\%-a\%)}$$

$$j=\frac{c*(g\%-a\%)}{a\%*(b\%-d\%)+(b\%-z\%)*(g\%-a\%)}$$

$$k=\frac{c*(a\%-g\%)}{a\%*(b\%-d\%)+(b\%-z\%)*(g\%-a\%)}$$

and the equation of the plane can be rewritten as:

$$(b\%-d\%)*X+(g\%-a\%)*Y+(a\%-g\%)*Z=g\%*b\%-a\%*d\%+z\%*(a\%-g\%)$$

A Cut Through Two Points on a Parabolic Curve The same procedure can be extended to the parabolic:

$$Y+a*(X-b)^2=c$$

for the two (2) integer points (a%,b%),(g%,d%) such that:

$$\begin{aligned} b\%+a*(a\%-b)^2&=c \\ d\%+a*(g\%-b)^2&=c \end{aligned}$$

for the equation of the line:

$$(b\%-d\%)*X+(g\%-a\%)*Y=g\%*b\%-a\%*d\%$$

A Cut Through Two Points on a Hyperbolic Curve The same procedure can be extended to the hyperbolic:

$$X*Y=c$$

for the two (2) integer points (a%,b%),(g%,b%) such that:

a%*b%=c
g%*d%=c

for the equation of the line:

(b%-d%)*X+(g%-a%)*Y=g%*b%-a%*d%

Parabolic Subroutines -- Files PAR-DEPA, PAR-DEPL, or PAR-DEPG

To find a deep cut with the line derived in the above equations, two points have to be found on the parabolic curve such that when the line is passed through the points, the line will not exclude any integer points from the feasible region of the integer CP problem.

Previously, in the nonlinear convex constraint subroutines three algorithms, the axial, line search, and Gordian, were described for locating one point on the function through which the tangent line or supporting plane could be passed. This single point can now be used to find the other two (2) points for the deep cut line.

If the parabolic function is graphed as shown in figure 72 then the

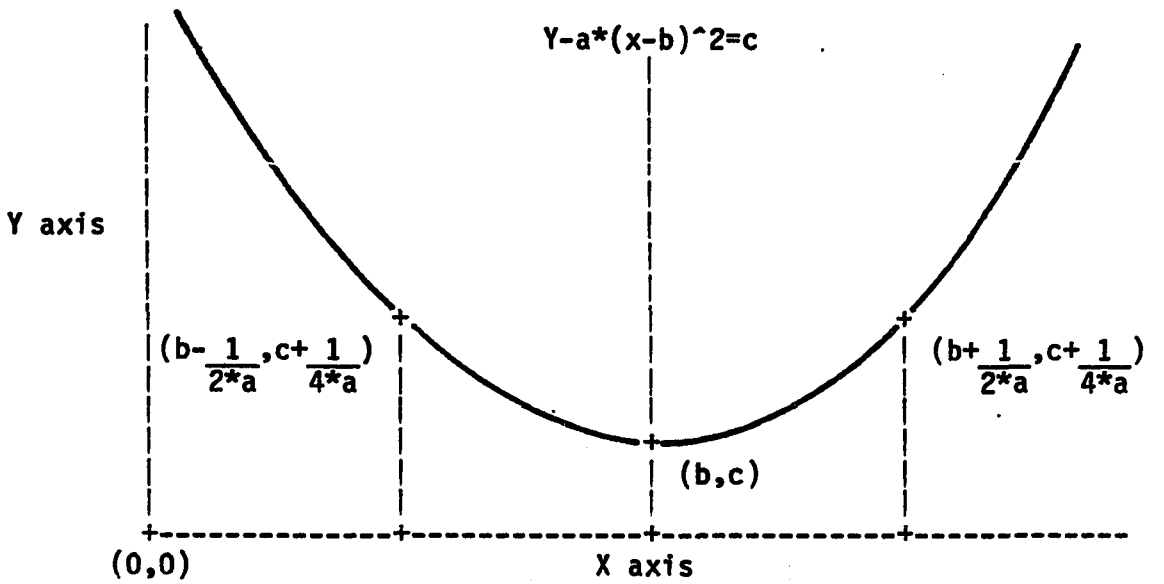


Figure 72. Parabolic constraint deep cut

curve can be divided into four (4) segments in which the slope of the curve is as graphed in figure 73.

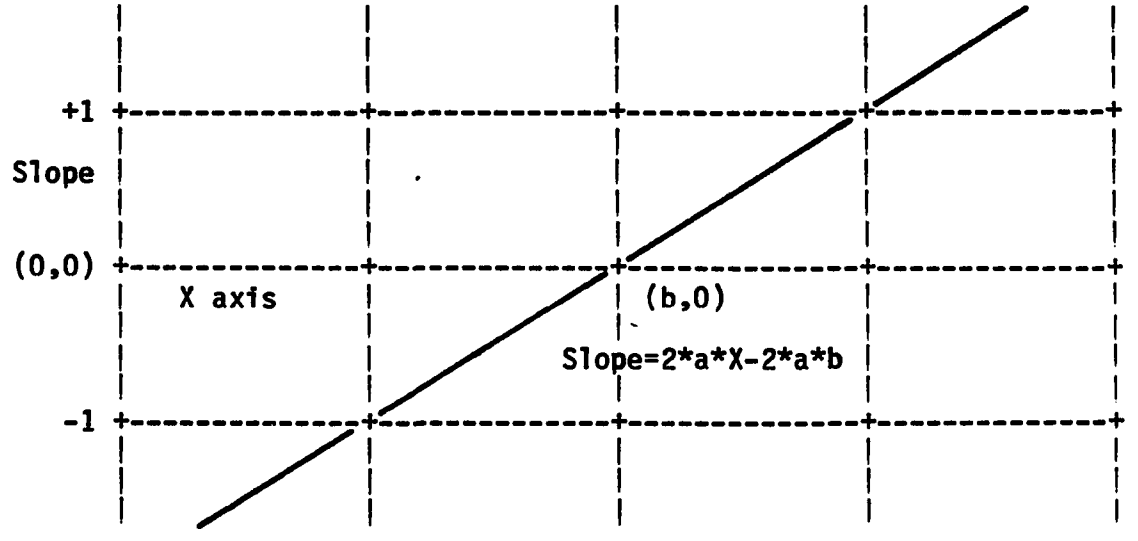


Figure 73. Parabolic constraint slope function

The slope in the first segment is negative and has an absolute value greater than one (1). The slope in the second segment is also negative but has an absolute value less or equal to one (1). In the third segment, the slope is positive and has a value less or equal to one (1). In the fourth segment, the slope is again positive but has a value greater than one (1).

By utilizing the characteristics of the slope, a step function can be superimposed on the parabolic curve as shown in figure 74. This step function, which "steps" at integer values and which does not exclude any integer points from the region above the parabolic function, can now be used to find two (2) points for a deep cut line.

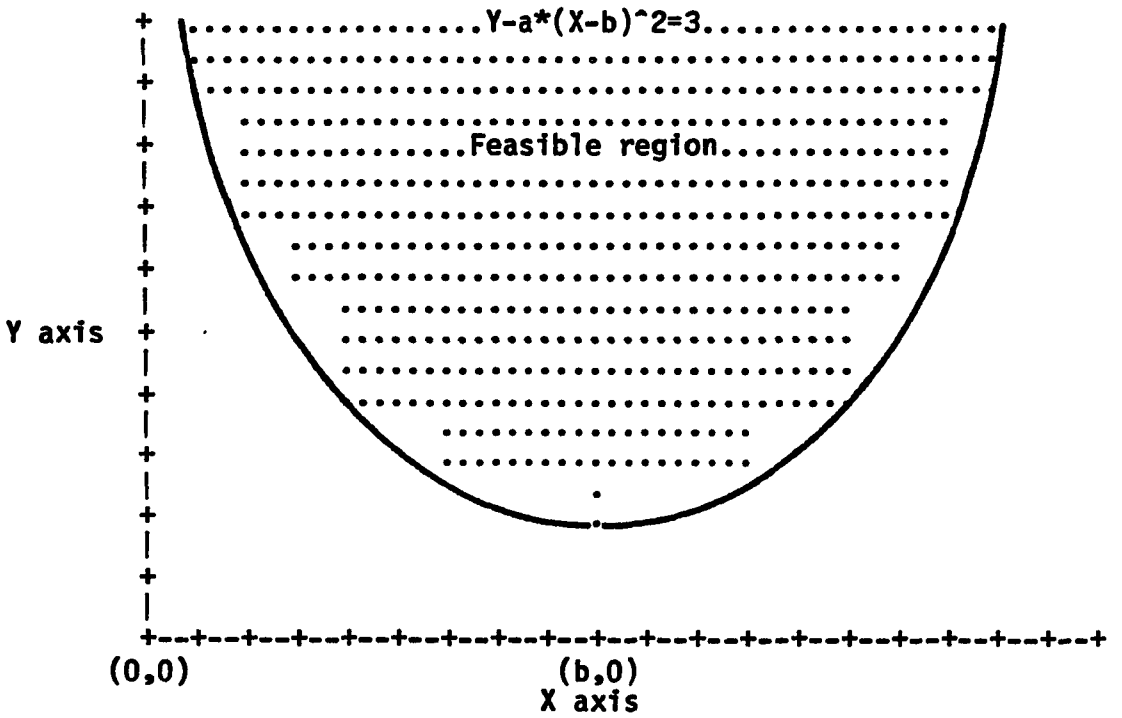


Figure 74. Parabolic constraint deep cut step function

Starting with the point used to find the tangent line in the non-integer case, an asterisk as shown in figure 75 can be placed on the

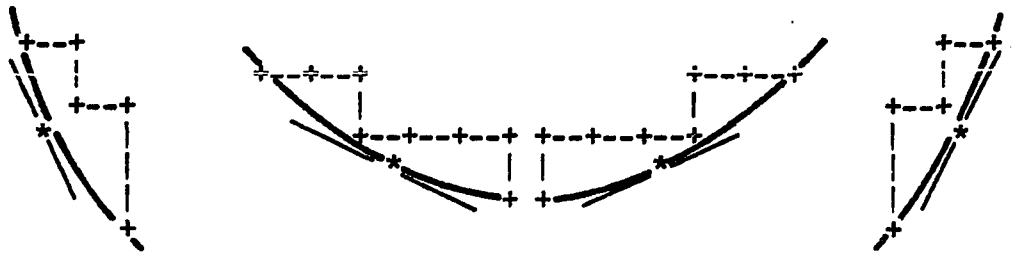


Figure 75. Parabolic constraint deep cut step function

curve to represent the tangent point in any one (1) of the four (4) segments of the curve as defined by the four (4) slope characteristics. If the asterisk is located on the segment of the curve in which the

slope is negative with an absolute value greater than one (1), then the X coordinate of the points on the step function which would provide for an deep cut are simply the round up and round down integer values of the X coordinate of the asterisk point. If the asterisk is located on the segment of the curve in which the slope is negative with an absolute value less than or equal to one (1), then the Y coordinates of the points on the step function are the round up and round down integer values of the Y coordinate of the asterisk point. Only one (1) coordinate of the remaining points can be found in a similar manner.

The other coordinate of the points on the step function are harder to find. Using as an example the segment of the function in which the slope is negative and the absolute value is less than or equal to one (1), the step function is shown figure 76. Above the asterisk point (0)

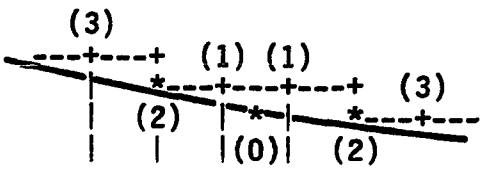


Figure 76. Deep cut step function

a set of integer points (1,1) on the step function can be found on each side of the asterisk, and a cut line (1->1) can be drawn through these two (2) points. Starting first on the left, a new integer point (2) on the step function can be found. If the point lies on or below the line (1->1) and the point is still in the same segment of the curve, then move the line so it passes through the point (2->1). If the next point (3) lies above the line (2->1) through the current set of

points, then let the point (2) be the left point through which the deep cut line passes. The right side is handled in a reversed fashion.

A deep cut line can be found for other segments of the curve by varying only the direction of the search along the step function.

The deep cut line derived above can be used as the source row to derive a Gomory or Wilson cut. Although the supporting line from the nonlinear modification of the dual simplex method can be used to derive a Gomory cut, the cut given above for which the code is given below, accelerates the conversion of the integer methods to follow.

Deep Cut for Parabolic Using The Axial Algorithm The deep cut for the parabolic constraint using the axial algorithm subroutine (PAR-DEPA) finds a point on the parabolic constraint, a constraint whose parameters are stored in the last seven (7) columns of the constraint matrix, by using the axial algorithm from which is constructed a deep cut line and then loads the line's coefficients into the linear portion of the constraint matrix.

```
4100 REM     * DEEP CUT FOR PARABOLIC USING AXIAL ALGORITHM *
4101 REM------------------------PAR-DEPA------------------------------

4102 ZX=A#(K,ND+5)+1
4103 ZY=A#(K,ND+6)+1
4104 A#=A#(K,ND+7)
4105 B#=A#(K,ND+2)
4106 IF B#<0# THEN B#=0#
4107 C#=A#(K,ND+8)
4108 Y#=B#(ZY,1)
4109 X#=B#(ZX,1)
4110 IF X#=H#(ZX) AND Y#=H#(ZY) THEN RETURN
4111 IF X#>=0# AND Y#>=0# AND Y#-A#*(X#-B#)^2#>=C# THEN RETURN
4112 IF X#<>B# THEN 4115
4113 Y#=C#
4114 GOTO 4127
```

```
4115 G#=A#*(B#-X#)*(B#-X#)
4116 D#=C#+1#/(2#*A#)
4117 IF D#<=Y# THEN D#=Y#+1#/(2#*A#)
4118 FOR W=1 TO 50
4119 K#=2#*D#*G#
4120 H#=(((Y#-D#)^2#+K#)-CDBL(SQR(((Y#-D#)^2#+K#)^2#-4#*G#*C#*(Y#-D#)^2#
     -K#^2#)))/(2#*G#)
4121 IF ABS(D#-H#-1#/(2#*A#))<.00001 THEN 4124
4122 D#=H#+1#/(2#*A#)
4123 NEXT W
4124 Y#=H#
4125 IF Y#<C# THEN Y#=C#
4126 X#=(B#*A#+SGN(X#-B#)*SQR(A#*(Y#-C#)))/A#
4127 IF Y#<=-INT(-C#) THEN 4183
4128 IF Y#>=C#+(1#/(4#*A#)) THEN 4152
4129 B%=-INT(-Y#)
4130 D%=B%-1
4131 A%=-INT(-SGN(X#-B#)*SQR((B%-C#)/A#)-B#)
4132 G%=-INT(-SGN(X#-B#)*SQR((D%-C#)/A#)-B#)
```

Finds the two (2) points on the curve through which the deep cut line is passed.

```
4133 P%=B%
4134 FOR I=1 TO BI#
4135 P%=P%+1
4136 IF P%>C#+(1#/(4#*A#)) THEN 4142
4137 J%=-INT(-SGN(X#-B#)*SQR((P%-C#)/A#)-B#)
4138 IF J%*(B%-D%)+P%*(G%-A%)>B%*G%-A%*D% THEN 4142
4139 B%=P%
4140 A%=J%
4141 NEXT I
4142 P%=D%
4143 FOR I=1 TO BI#
4144 P%=P%-1
4145 IF P%<C# THEN 4151
4146 J%=-INT(-SGN(X#-B#)*SQR((P%-C#)/A#)-B#)
4147 IF J%*(B%-D%)+P%*(G%-A%)>B%*G%-A%*D% THEN 4151
4148 D%=P%
4149 G%=J%
4150 NEXT I
4151 GOTO 4174
4152 G%=-INT(-X#)
4153 A%=G%-1
4154 D%=-INT(-C#-A#*(G%-B#)^2)
4155 B%=-INT(-C#-A#*(A%-B#)^2)
4156 P%=A%
4157 FOR I=1 TO BI#
```

```
4158 P%=P%-1
4159 J%=-INT(-C#-A#*(P%-B#)^2)
4160 IF J%<C#+(1#/(4#*A#)) THEN 4165
4161 IF P%*(B%-D%)+J%*(G%-A%)>B%*G%-A%*D% THEN 4165
4162 A%=P%
4163 B%=J%
4164 NEXT I
4165 P%=G%
4166 FOR I=1 TO BI#
4167 P%=P%+1
4168 J%=-INT(-C#-A#*(P%-B#)^2)
4169 IF J%<C#+(1#/(4#*A#)) THEN 4174
4170 IF P%*(B%-D%)+J%*(G%-A%)>B%*G%-A%*D% THEN 4174
4171 G%=P%
4172 D%=J%
4173 NEXT I
```

Finds the equation of the line through the two (2) points above.

```
4174 S#=1
4175 IF X#>B# AND Y#<C#+(1#/(4#*A#)) THEN S#=-1
4176 A#(K,1)=CDBL(B%*G%-A%*D%)*S#
4177 FOR I=2 TO ND+1
4178 A#(K,I)=0#
4179 IF I=ZX THEN A#(K,I)=CDBL(B%-D%)*S#
4180 IF I=ZY THEN A#(K,I)=CDBL(G%-A%)*S#
4181 NEXT I
4182 RETURN
4183 A#(K,1)=-INT(-C#)
4184 FOR I=2 TO ND+1
4185 A#(K,I)=0#
4186 IF I=ZY THEN A#(K,I)=1#
4187 IF I=ZX THEN A#(K,I)=0#
4188 NEXT I
4189 RETURN
```

Deep Cut for Parabolic Using Line Search Algorithm The deep cut for the parabolic constraint using the line search algorithm subroutine (PAR-DEPL) finds a point on the parabolic constraint, a constraint whose parameters are stored in the last seven (7) columns of the constraint matrix, by using the line search algorithm from which is constructed a deep cut line and then loads the line's coefficients into the linear

portion of the constraint matrix.

```
4100 REM        * DEEP CUT FOR PARABOLIC USING LINE SEARCH *
4101 REM-------------------------PAR-DEPL-------------------------------

4102 ZX=A#(K,ND+5)+1
4103 ZY=A#(K,ND+6)+1
4104 A#=A#(K,ND+7)
4105 B#=A#(K,ND+2)
4106 IF B#<0# THEN B#=0#
4107 C#=A#(K,ND+8)
4108 Y#=B#(ZY,1)
4109 X#=B#(ZX,1)
4110 IF X#=H#(ZX) AND Y#=H#(ZY) THEN RETURN
4111 IF X#>=0# AND Y#>=0# AND Y#-A#*(X#-B#)^2#>=C# THEN RETURN
4112 U#=X#
4113 IF X#=B# THEN 4126
4114 L#=B#
4115 S#=SGN(B#-X#)
4116 U#=U#+S#*1E-08
4117 IF Y#>C# THEN L#=B#+S#*CDBL(SQR((Y#-C#)/A#))
4118 FOR W=1 TO 100
4119 IF ABS(U#-L#)<1E-08 THEN 4125
4120 T#=U#
4121 S#=((X#-U#)-2#*A#*(B#-U#)*(Y#-A#*U#*U#+2#*A#*B#*U#-A#*B#*B#-C#))/(2
     #*A#*(B#-U#)*(X#-U#))
4122 U#=U#-((SGN(S#)*ABS(L#-U#))/2#)
4123 T#=E#
4124 NEXT W
4125 X#=U#
4126 Y#=A#*(U#-B#)*(U#-B#)+C#
4127 IF Y#<=-INT(-C#) THEN 4183
4128 IF Y#>=C#+(1#/(4#*A#)) THEN 4152
4129 B%=-INT(-Y#)
4130 D%=B%-1
4131 A%=-INT(-SGN(X#-B#)*CDBL(SQR((B%-C#)/A#))-B#)
4132 G%=-INT(-SGN(X#-B#)*CDBL(SQR((D%-C#)/A#))-B#)
```

Finds the two (2) points on the curve through which the deep cut line is passed.

```
4133 P%=B%
4134 FOR I=1 TO BI#
4135 P%=P%+1
4136 IF P%>C#+(1#/(4#*A#)) THEN 4142
4137 J%=-INT(-SGN(X#-B#)*SQR((P%-C#)/A#)-B#)
4138 IF J%*(B%-D%)+P%*(G%-A%)>B%*G%-A%*D% THEN 4142
4139 B%=P%
```

```
4140 A%=J%
4141 NEXT I
4142 P%=D%
4143 FOR I=1 TO BI#
4144 P%=P%-1
4145 IF P%<C# THEN 4151
4146 J%=-INT(-SGN(X#-B#)*SQR((P%-C#)/A#)-B#)
4147 IF J%*(B%-D%)+P%*(G%-A%)>B%*G%-A%*D% THEN 4151
4148 D%=P%
4149 G%=J%
4150 NEXT I
4151 GOTO 4174
4152 G%=-INT(-X#)
4153 A%=G%-1
4154 D%=-INT(-C#-A#*(G%-B#)^2)
4155 B%=-INT(-C#-A#*(A%-B#)^2)
4156 P%=A%
4157 FOR I=1 TO BI#
4158 P%=P%-1
4159 J%=-INT(-C#-A#*(P%-B#)^2)
4160 IF J%<C#+(1#/(4#*A#)) THEN 4165
4161 IF P%*(B%-D%)+J%*(G%-A%)>B%*G%-A%*D% THEN 4165
4162 A%=P%
4163 B%=J%
4164 NEXT I
4165 P%=G%
4166 FOR I=1 TO BI#
4167 P%=P%+1
4168 J%=-INT(-C#-A#*(P%-B#)^2)
4169 IF J%<C#+(1#/(4#*A#)) THEN 4174
4170 IF P%*(B%-D%)+J%*(G%-A%)>B%*G%-A%*D% THEN 4174
4171 G%=P%
4172 D%=J%
4173 NEXT
```

Finds the equation of the line through the two (2) points above.

```
4174 S#=1
4175 IF X#>B# AND Y#<C#+(1#/(4#*A#)) THEN S#=-1
4176 A#(K,1)=CDBL(B%*G%-A%*D%)*S#
4177 FOR I=2 TO ND+1
4178 A#(K,I)=0#
4179 IF I=ZX THEN A#(K,I)=CDBL(B%-D%)*S#
4180 IF I=ZY THEN A#(K,I)=CDBL(G%-A%)*S#
4181 NEXT I
4182 RETURN
4183 A#(K,1)=-INT(-C#)
4184 FOR I=2 TO ND+1
```

```
4185 A#(K,I)=0#
4186 IF I=ZY THEN A#(K,I)=1#
4187 IF I=ZX THEN A#(K,I)=0#
4188 NEXT I
4189 RETURN
```

Deep Cut for Parabolic Using Gordian Algorithm The deep cut for the parabolic constraint using the Gordian algorithm subroutine (PAR-DEPG) finds a point on the parabolic constraint, a constraint whose parameters are stored in the last seven (7) columns of the constraint matrix, by using the Gordian algorithm from which is constructed a deep cut line and then loads the line's coefficients into the linear portion of the constraint matrix.

```
4100 REM     * DEEP CUT FOR PARABOLIC USING GORDIAN ALGORITHM *
4101 REM-------------------------DEPG-PAR--------------------------------

4102 ZX=A#(K,ND+5)+1
4103 ZY=A#(K,ND+6)+1
4104 A#=A#(K,ND+7)
4105 B#=A#(K,ND+2)
4106 IF B#<0# THEN B#=0#
4107 C#=A#(K,ND+8)
4108 Y#=B#(ZY,1)
4109 X#=B#(ZX,1)
4110 IF X#=H#(ZX) AND Y#=H#(ZY) THEN RETURN
4111 IF X#>=0# AND Y#>=0# AND Y#-A#*(X#-B#)^2#>=C# THEN RETURN
4112 Y#=A#*(X#-B#)*(X#-B#)+C#
4113 IF Y#<=-INT(-C#) THEN 4169
4114 IF Y#>=C#+(1#/(4#*A#)) THEN 4138
4115 B%=-INT(-Y#)
4116 D%=B%-1
4117 A%=-INT(-SGN(X#-B#)*CDBL(SQR((B%-C#)/A#))-B#)
4118 G%=-INT(-SGN(X#-B#)*CDBL(SQR((D%-C#)/A#))-B#)
```

Finds the two (2) points on the curve through which the deep cut line is passed.

```
4119 P%=B%
4120 FOR I=1 TO BI#
4121 P%=P%+1
4122 IF P%>C#+(1#/(4#*A#)) THEN 4128
```

```
4123 J%=-INT(-SGN(X#-B#)*SQR((P%-C#)/A#)-B#)
4124 IF J%*(B%-D%)+P%*(G%-A%)>B%*G%-A%*D% THEN 4128
4125 B%=P%
4126 A%=J%
4127 NEXT I
4128 P%=D%
4129 FOR I=1 TO BI#
4130 P%=P%-1
4131 IF P%<C# THEN 4137
4132 J%=-INT(-SGN(X#-B#)*SQR((P%-C#)/A#)-B#)
4133 IF J%*(B%-D%)+P%*(G%-A%)>B%*G%-A%*D% THEN 4137
4134 D%=P%
4135 G%=J%
4136 NEXT I
4137 GOTO 4160
4138 G%=-INT(-X#)
4139 A%=G%-1
4140 D%=-INT(-C#-A#*(G%-B#)^2)
4141 B%=-INT(-C#-A#*(A%-B#)^2)
4142 P%=A%
4143 FOR I=1 TO BI#
4144 P%=P%-1
4145 J%=-INT(-C#-A#*(P%-B#)^2)
4146 IF J%<C#+(1#/(4#*A#)) THEN 4151
4147 IF P%*(B%-D%)+J%*(G%-A%)>B%*G%-A%*D% THEN 4151
4148 A%=P%
4149 B%=J%
4150 NEXT I
4151 P%=G%
4152 FOR I=1 TO BI#
4153 P%=P%+1
4154 J%=-INT(-C#-A#*(P%-B#)^2)
4155 IF J%<C#+(1#/(4#*A#)) THEN 4160
4156 IF P%*(B%-D%)+J%*(G%-A%)>B%*G%-A%*D% THEN 4160
4157 G%=P%
4158 D%=J%
4159 NEXT I
```

Finds the equation of the line through the two (2) points above.

```
4160 S#=1
4161 IF X#>B# AND Y#<C#+(1#/(4#*A#)) THEN S#=-1#
4162 A#(K,1)=CDBL(B%*G%-A%*D%)*S#
4163 FOR I=2 TO ND+1
4164 A#(K,I)=0#
4165 IF I=ZX THEN A#(K,I)=CDBL(B%-D%)*S#
4166 IF I=ZY THEN A#(K,I)=CDBL(G%-A%)*S#
4167 NEXT I
```

```
4168 RETURN
4169 A#(K,1)=-INT(-C#)
4170 FOR I=2 TO ND+1
4171 A#(K,I)=0#
4172 IF I=ZY THEN A#(K,I)=1#
4173 IF I=ZX THEN A#(K,I)=0#
4174 NEXT I
4175 RETURN
```

Hyperbolic Subroutine -- Files HYP-DEPA, HYP-DEPL, or HYP-DEPG

The hyperbolic function and parabolic functions are similar in that the segments of the curve with negative slope use the same procedure for finding a deep cut line as can be seen in figure 77 with its

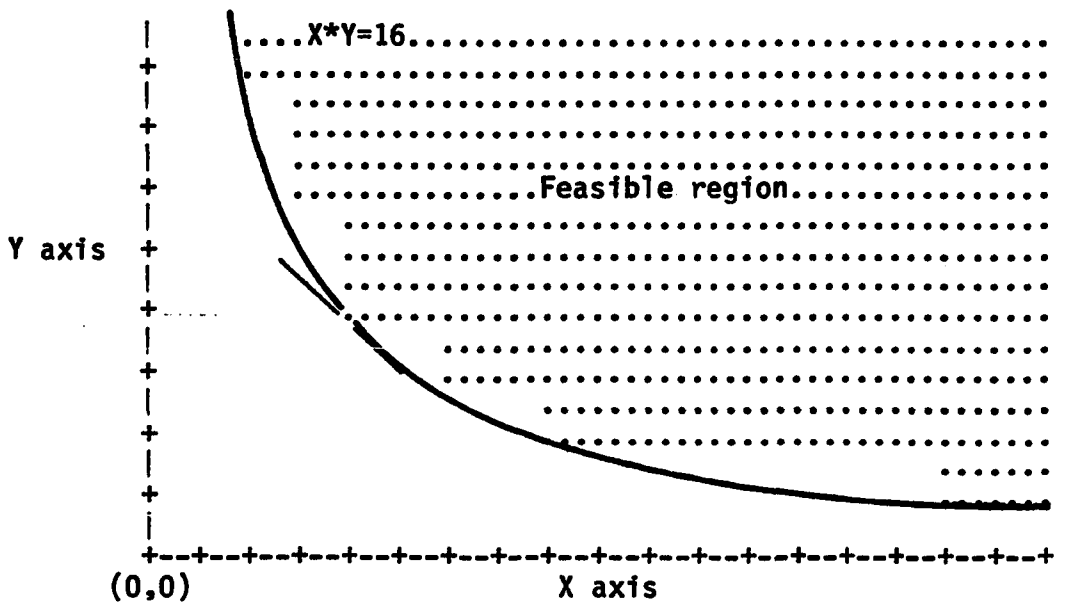


Figure 77. Hyperbolic constraint deep cut step function

superimposed step function.

Deep Cut for Hyperbolic Using Axial Algorithm The deep cut for the hyperbolic constraint using the axial algorithm subroutine (HYP-DEPA) finds a point on the hyperbolic constraint, a constraint whose

parameters are stored in the last seven (7) columns of the constraint matrix, by using the axial algorithm from which is constructed a deep cut line and then loads the line's coefficients into the linear portion of the constraint matrix.

```
4300 REM    * DEEP CUT FOR HYPERBOLIC USING AXIAL ALGORITHM *
4301 REM------------------------DEPA-HYP-------------------------------

4302 ZX=A#(K,ND+5)+1
4303 ZY=A#(K,ND+6)+1
4304 C#=A#(K,ND+3)
4305 Y#=B#(ZY,1)
4306 X#=B#(ZX,1)
4307 IF X#=H#(ZX) AND Y#=H#(ZY) THEN RETURN
4308 IF X#>0# AND Y#>0# AND X#*Y#>=C# THEN RETURN
4309 IF X#>=C# THEN 4374
4310 IF Y#>=C# THEN 4377
4311 D#=(C#+1#)
4312 IF X#>=D# THEN D#=(X#+1#)
4313 FOR W=1 TO 20
4314 G#=(X#*D#-Y#*D#+CDBL(SQR(((Y#*D#-X#*D#)^2#)+4#*(D#-X#)*(D#-Y#)*C#))
     )/(2#*(D#-Y#))
4315 D#=(G#*G#+C#)/G#
4316 NEXT W
4317 IF G#>=C# THEN 4374
4318 IF G#<=1# THEN 4377
4319 X#=G#
4320 Y#=C#/X#
4321 IF X#<CDBL(SQR(C#)) THEN 4345
4322 B%=-INT(-Y#)
4323 D%=B%-1
4324 A%=-INT(-C#/B%)
4325 G%=-INT(-C#/D%)
```

Finds the two (2) points on the curve through which the deep cut line is passed.

```
4326 P%=B%
4327 FOR I=1 TO BI#
4328 P%=P%+1
4329 J%=-INT(-C#/P%)
4330 IF J%<SQR(C#) THEN 4335
4331 IF J%*(B%-D%)+P%*(G%-A%)>B%*G%-A%*D% THEN 4335
4332 B%=P%
4333 A%=J%
```

```
4334 NEXT I
4335 P%=D%
4336 FOR I=1 TO BI#
4337 P%=P%-1
4338 IF P%<1 THEN 4344
4339 J%=-INT(-C#/P%)
4340 IF J%*(B%-D%)+P%*(G%-A%)>B%*G%-A%*D% THEN 4344
4341 D%=P%
4342 G%=J%
4343 NEXT I
4344 GOTO 4367
4345 G%=-INT(-X#)
4346 A%=G%-1
4347 D%=-INT(-C#/G%)
4348 B%=-INT(-C#/A%)
4349 P%=A%
4350 FOR I=1 TO BI#
4351 P%=P%-1
4352 IF P%<1 THEN 4358
4353 J%=-INT(-C#/P%)
4354 IF P%*(B%-D%)+J%*(G%-A%)>B%*G%-A%*D% THEN 4358
4355 A%=P%
4356 B%=J%
4357 NEXT I
4358 P%=G%
4359 FOR I=1 TO BI#
4360 P%=P%+1
4361 IF P%>SQR(C#) THEN 4367
4362 J%=-INT(-C#/P%)
4363 IF P%*(B%-D%)+J%*(G%-A%)>B%*G%-A%*D% THEN 4367
4364 G%=P%
4365 D%=J%
4366 NEXT I
```

Finds the equation of the line through the two (2) points above.

```
4367 A#(K,1)=CDBL(B%*G%-A%*D%)
4368 FOR I=2 TO ND+1
4369 A#(K,I)=0#
4370 IF I=ZX THEN A#(K,I)=CDBL(B%-D%)
4371 IF I=ZY THEN A#(K,I)=CDBL(G%-A%)
4372 NEXT I
4373 RETURN
4374 B#=1#
4375 A#=0#
4376 GOTO 4379
4377 A#=1#
4378 B#=0#
```

```
4379 A#(K,1)=1#
4380 FOR I=2 TO ND+1
4381 A#(K,I)=0#
4382 IF I=ZY THEN A#(K,I)=B#
4383 IF I=ZX THEN A#(K,I)=A#
4384 NEXT I
4385 RETURN
```

Deep Cut for Hyperbolic Using Line Search Algorithm The deep cut for the hyperbolic constraint using the line search algorithm subroutine (HYP-DEPL) finds a point on the hyperbolic constraint, a constraint whose parameters are stored in the last seven (7) columns of the constraint matrix, by using the line search algorithm from which is constructed a deep cut line and then loads the line's coefficients into the linear portion of the constraint matrix.

```
4300 REM  * DEEP CUT FOR HYPERBOLIC USING LINE SEARCH ALGORITHM *
4301 REM------------------------HYP-DEPL-------------------------------

4302 ZX=A#(K,ND+5)+1
4303 ZY=A#(K,ND+6)+1
4304 C#=A#(K,ND+3)
4305 Y#=B#(ZY,1)
4306 X#=B#(ZX,1)
4307 IF X#=H#(ZX) AND Y#=H#(ZY) THEN RETURN
4308 IF X#>0# AND Y#>0# AND X#*Y#>=C# THEN RETURN
4309 IF X#>=C# THEN 4374
4310 IF Y#>=C# THEN 4377
4311 G#=(C#*X#-C#*Y#+CDBL(SQR(((C#*Y#-C#*X#)^2#)+4#*C#*C#*C#)))/(2#*C#)
4312 FOR W=1 TO 100
4313 T#=(G#*G#*X#-C#*Y#+CDBL(SQR(((C#*Y#-G#*G#*X#)^2#)+4#*C#*C#*G#*G#)))
     B/(2#*G#*G#)
4314 IF ABS(G#-T#)<.0000001 THEN 4317
4315 G#=(G#+T#)/2#
4316 NEXT W
4317 IF G#>=C# THEN 4374
4318 IF G#<=1# THEN 4377
4319 X#=G#
4320 Y#=C#/X#
4321 IF X#<CDBL(SQR(C#)) THEN 4345
4322 B%=-INT(-Y#)
4323 D%=B%-1
```

```
4324 A%=-INT(-C#/B%)
4325 G%=-INT(-C#/D%)
```

Finds the two (2) points on the curve through which the deep cut line is passed.

```
4326 P%=B%
4327 FOR I=1 TO BI#
4328 P%=P%+1
4329 J%=-INT(-C#/P%)
4330 IF J%<SQR(C#) THEN 4335
4331 IF J%*(B%-D%)+P%*(G%-A%)>B%*G%-A%*D% THEN 4335
4332 B%=P%
4333 A%=J%
4334 NEXT I
4335 P%=D%
4336 FOR I=1 TO BI#
4337 P%=P%-1
4338 IF P%<1 THEN 4344
4339 J%=-INT(-C#/P%)
4340 IF J%*(B%-D%)+P%*(G%-A%)>B%*G%-A%*D% THEN 4344
4341 D%=P%
4342 G%=J%
4343 NEXT I
4344 GOTO 4367
4345 G%=-INT(-X#)
4346 A%=G%-1
4347 D%=-INT(-C#/G%)
4348 B%=-INT(-C#/A%)
4349 P%=A%
4350 FOR I=1 TO BI#
4351 P%=P%-1
4352 IF P%<1 THEN 4358
4353 J%=-INT(-C#/P%)
4354 IF P%*(B%-D%)+J%*(G%-A%)>B%*G%-A%*D% THEN 4358
4355 A%=P%
4356 B%=J%
4357 NEXT I
4358 P%=G%
4359 FOR I=1 TO BI#
4360 P%=P%+1
4361 IF P%>CDBL(SQR(C#)) THEN 4367
4362 J%=-INT(-C#/P%)
4363 IF P%*(B%-D%)+J%*(G%-A%)>B%*G%-A%*D% THEN 4367
4364 G%=P%
4365 D%=J%
4366 NEXT I
```

Finds the equation of the line through the two (2) points above.

```
4367 A#(K,1)=CDBL(B%*G%-A%*D%)
4368 FOR I=2 TO ND+1
4369 A#(K,I)=0#
4370 IF I=ZX THEN A#(K,I)=CDBL(B%-D%)
4371 IF I=ZY THEN A#(K,I)=CDBL(G%-A%)
4372 NEXT I
4373 RETURN
4374 B#=1#
4375 A#=0#
4376 GOTO 4379
4377 A#=1#
4378 B#=0#
4379 A#(K,1)=1#
4380 FOR I=2 TO ND+1
4381 A#(K,I)=0#
4382 IF I=ZY THEN A#(K,I)=B#
4383 IF I=ZX THEN A#(K,I)=A#
4384 NEXT I
4385 RETURN
```

Deep Cut for Hyperbolic Using Gordian Algorithm The deep cut for the hyperbolic constraint using the Gordian algorithm subroutine (HYP-DEPG) finds a point on the hyperbolic constraint, a constraint whose parameters are stored in the last seven (7) columns of the constraint matrix, by using the Gordian algorithm from which is constructed a deep cut line and then loads the line's coefficients into the linear portion of the constraint matrix.

```
4300 REM    *DEEP CUT FOR HYPERBOLIC USING GORDIAN ALGORITHM*
4301 REM--------------------HYP-DEPG-------------------------------

4302 ZX=A#(K,ND+5)+1
4303 ZY=A#(K,ND+6)+1
4304 C#=A#(K,ND+3)
4305 Y#=B#(ZY,1)
4306 X#=B#(ZX,1)
4307 IF X#=H#(ZX) AND Y#=H#(ZY) THEN RETURN
4308 IF X#>0# AND Y#>0# AND X#*Y#>=C# THEN RETURN
4309 G#=CDBL(SQR(C#))
4310 IF X#<G#/2# AND Y#<G#/2# THEN 4313
```

```
4311 IF X#>=Y# THEN 4316
4312 GOTO 4314
4313 Y#=G#
4314 X#=C#/Y#
4315 GOTO 4317
4316 Y#=C#/X#
4317 IF Y#<=1# THEN 4372
4318 IF X#<=1# THEN 4375
```

Finds the two (2) points on the curve through which the deep cut line is passed.

```
4319 IF X#<G# THEN 4343
4320 B%=-INT(-Y#)
4321 D%=B%-1
4322 A%=-INT(-C#/B%)
4323 G%=-INT(-C#/D%)
4324 P%=B%
4325 FOR I=1 TO BI#
4326 P%=P%+1
4327 J%=-INT(-C#/P%)
4328 IF J%<SQR(C#) THEN 4333
4329 IF J%*(B%-D%)+P%*(G%-A%)>B%*G%-A%*D% THEN 4333
4330 B%=P%
4331 A%=J%
4332 NEXT I
4333 P%=D%
4334 FOR I=1 TO BI#
4335 P%=P%-1
4336 IF P%<1 THEN 4342
4337 J%=-INT(-C#/P%)
4338 IF J%*(B%-D%)+P%*(G%-A%)>B%*G%-A%*D% THEN 4342
4339 D%=P%
4340 G%=J%
4341 NEXT I
4342 GOTO 4365
4343 G%=-INT(-X#)
4344 A%=G%-1
4345 D%=-INT(-C#/G%)
4346 B%=-INT(-C#/A%)
4347 P%=A%
4348 FOR I=1 TO BI#
4349 P%=P%-1
4350 IF P%<1 THEN 4356
4351 J%=-INT(-C#/P%)
4352 IF P%*(B%-D%)+J%*(G%-A%)>B%*G%-A%*D% THEN 4356
4353 A%=P%
4354 B%=J%
```

```
4355 NEXT I
4356 P%=G%
4357 FOR I=1 TO BI#
4358 P%=P%+1
4359 IF P%>SQR(C#) THEN 4365
4360 J%=-INT(-C#/P%)
4361 IF P%*(B%-D%)+J%*(G%-A%)>B%*G%-A%*D% THEN 4365
4362 G%=P%
4363 D%=J%
4364 NEXT I
```

Finds the equation of the line through the two (2) points above.

```
4365 A#(K,1)=CDBL(B%*G%-A%*D%)
4366 FOR I=2 TO ND+1
4367 A#(K,I)=0#
4368 IF I=ZX THEN A#(K,I)=CDBL(B%-D%)
4369 IF I=ZY THEN A#(K,I)=CDBL(G%-A%)
4370 NEXT I
4371 RETURN
4372 B#=1#
4373 A#=0#
4374 GOTO 4377
4375 A#=1#
4376 B#=0#
4377 A#(K,1)=1#
4378 FOR I=2 TO ND+1
4379 A#(K,I)=0#
4380 IF I=ZY THEN A#(K,I)=B#
4381 IF I=ZX THEN A#(K,I)=A#
4382 NEXT I
4383 RETURN
```

Hyperbolic Sheet Subroutine -- Files SHT-DEPA, SHT-DEPL, or SHT-DEPG

The hyperbolic of two sheets or X*(Y-Z)=c can be visualized as a cylinder in which sections in the X and Y axes plane are a series of hyperbolic like curves and slices in the Z and Y axes plane are a series of lines at forty-five (45) degrees as shown in figures 70 and 71. If the two (2) deep cut points on the hyperbolic like function that corresponds to the X and Y axis plane which passes through the [z] coordinate point can be found as say (a%,b%,[z]) and (g%,d%,[z]), then

the third point (g%,d%+1,[z]+1) can be found by assuming a forty-five (45) degree cut in the Z and Y axes plane.

Deep Cut for Sheet Using Axial Algorithm The deep cut for the sheet constraint using the axial algorithm subroutine (SHT-DEPA) finds a point on the sheet constraint, a constraint whose parameters are stored in the last seven (7) columns of the constraint matrix, by using the axial algorithm from which is constructed a deep cut plane and then loads the plane's coefficients into the linear portion of the constraint matrix.

```
4500 REM      * DEEP CUT FOR SHEET USING AXIAL ALGORITHM *
4501 REM-------------------------SHT-DEPA-------------------------------

4502 C#=A#(K,ND+4)
4503 ZX=1+A#(K,ND+5)
4504 ZY=1+A#(K,ND+6)
4505 ZZ=1+A#(K,ND+7)
4506 Y#=B#(ZY,1)
4507 X#=B#(ZX,1)
4508 Z#=B#(ZZ,1)
4509 IF X#=H#(ZX) AND Y#=H#(ZY) AND Z#=H#(ZZ) THEN RETURN
4510 G#=(Y#-Z#)
4511 IF X#>0# AND G#>0# AND X#*G#>=C# THEN RETURN
4512 A#=C#
4513 IF X#>A# THEN A#=(C#+1#)
4514 FOR W=1 TO 20
4515 B#=X#*(2#*A#-CDBL(SQR(C#))-Y#+Z#)+(A#-X#)*(Y#-Z#)
4516 K#=2#*(A#-X#)*(2#*A#-CDBL(SQR(C#))-Y#+Z#)
4517 T#=(-B#+CDBL(SQR((B#^2#)-2#*K#*((X#*(Y#-Z#))-C#))))/K#
4518 E#=X#+(A#-X#)*T#
4519 A#=(2#*(Y#+((2#*A#-Y#+Z#-CDBL(SQR(C#)))*T#/2#))*C#-2#*E#*E#*E#-C#*(
     Y#+Z#)+C#*SQR(C#))/(2#*C#-2#*E#*E#)
4520 NEXT W
4521 X#=E#
4522 Z#=Z#+((X#+Y#-2#*Z#-2#*A#+CDBL(SQR(C#)))*T#/2#)
4523 Z%=INT(Z#)
4524 Y#=(C#+Z%*X#)/X#
4525 IF Y#-Z%<=1# THEN 4581
4526 IF X#<=1# THEN 4588
4527 IF X#<CDBL(SQR(C#)) THEN 4551
```

```
4528 B%=-INT(-Y#)
4529 D%=B%-1
4530 A%=-INT(-C#/(B%-Z%))
4531 G%=-INT(-C#/(D%-Z%))
```

Finds the two (2) of three (3) points on the curve through which the deep cut plane is passed.

```
4532 P%=B%
4533 FOR I=1 TO BI#
4534 P%=P%+1
4535 J%=-INT(-C#/(P%-Z%))
4536 IF J%<CDBL(SQR(C#)) THEN 4541
4537 IF J%*(B%-D%)+P%*(G%-A%)>B%*G%-A%*D% THEN 4541
4538 B%=P%
4539 A%=J%
4540 NEXT I
4541 P%=D%
4542 FOR I=1 TO BI#
4543 P%=P%-1
4544 IF P%<Z%+1 THEN 4550
4545 J%=-INT(-C#/(P%-Z%))
4546 IF J%*(B%-D%)+P%*(G%-A%)>B%*G%-A%*D% THEN 4550
4547 D%=P%
4548 G%=J%
4549 NEXT I
4550 GOTO 4573
4551 G%=-INT(-X#)
4552 A%=G%-1
4553 D%=-INT(-(C#+Z%*G%)/G%)
4554 B%=-INT(-(C#+Z%*A%)/A%)
4555 P%=A%
4556 FOR I=1 TO BI#
4557 P%=P%-1
4558 IF P%<1 THEN 4564
4559 J%=-INT(-(C#+P%*Z%)/P%)
4560 IF P%*(B%-D%)+J%*(G%-A%)>B%*G%-A%*D% THEN 4564
4561 A%=P%
4562 B%=J%
4563 NEXT I
4564 P%=G%
4565 FOR I=1 TO BI#
4566 P%=P%+1
4567 IF P%>CDBL(SQR(C#)) THEN 4573
4568 J%=-INT(-(C#+P%*Z%)/P%)
4569 IF P%*(B%-D%)+J%*(G%-A%)>B%*G%-A%*D% THEN 4573
4570 G%=P%
4571 D%=J%
```

```
4572 NEXT I
```

Finds the equation of the plane through the two (2) points found above and the Z and Y axes plane point.

```
4573 A#(K,1)=CDBL((G%*B%-A%*D%)+Z%*(A%-G%))
4574 FOR J=2 TO ND+1
4575 A#(K,J)=0#
4576 IF ZX=J THEN A#(K,J)=CDBL(B%-D%)
4577 IF ZY=J THEN A#(K,J)=CDBL(G%-A%)
4578 IF ZZ=J THEN A#(K,J)=CDBL(A%-G%)
4579 NEXT J
4580 RETURN
4581 A#(K,1)=1#
4582 FOR J=2 TO ND+1
4583 A#(K,J)=0#
4584 IF J=ZY THEN A#(K,J)=1#
4585 IF J=ZZ THEN A#(K,J)=-1#
4586 NEXT J
4587 RETURN
4588 A#(K,1)=1#
4589 FOR I=2 TO ND+1
4590 A#(K,I)=0#
4591 IF I=ZX THEN A#(K,I)=1#
4592 NEXT I
4593 RETURN
```

Deep Cut for Sheet Using Line Search Algorithm The deep cut for the sheet constraint using the line search algorithm subroutine (SHT-DEPL) finds a point on the sheet constraint, a constraint whose parameters are stored in the last seven (7) columns of the constraint matrix, by using the line search algorithm from which is constructed a deep cut plane and then loads the plane's coefficients into the linear portion of the constraint matrix.

```
4500 REM    * DEEP CUT FOR SHEET USING LINE SEARCH ALGORITHM *
4501 REM-------------------------SHT-DEPL-------------------------------

4502 C#=A#(K,ND+4)
4503 ZX=1+A#(K,ND+5)
4504 ZY=1+A#(K,ND+6)
4505 ZZ=1+A#(K,ND+7)
```

```
4506 Y#=B#(ZY,1)
4507 X#=B#(ZX,1)
4508 Z#=B#(ZZ,1)
4509 IF X#=H#(ZX) AND Y#=H#(ZY) AND Z#=H#(ZZ) THEN RETURN
4510 G#=(Y#-Z#)
4511 IF X#>0# AND G#>0# AND X#*G#>=C# THEN RETURN
4512 IF Y#>=Z#+C# THEN 4587
4513 IF X#>=C# THEN 4580
4514 E#=((2#*C#*X#-Y#*C#+Z#*C#)+CDBL(SQR(((Y#*C#-Z#*C#-2#*C#*X#)^2#)+8#*
     C#*C#*C#)))/(4#*C#)
4515 FOR W=1 TO 100
4516 T#=((2#*E#*E#*X#-Y#*C#+Z#*C#)+CDBL(SQR(((Y#*C#-Z#*C#-2#*E#*E#*X#)^2
     #)+8#*E#*E#*C#*C#)))/(4#*E#*E#)
4517 IF ABS(T#-E#)<.000001 THEN 4520
4518 E#=(T#+E#)/2#
4519 NEXT W
4520 Z#=Z#-((E#*E#*E#-X#*E#*A#)/C#)
4521 X#=E#
4522 Z%=INT(Z#)
4523 Y#=(C#+Z%*X#)/X#
4524 IF Y#-Z%<=1# THEN 4580
4525 IF X#<=1# THEN 4587
4526 IF X#<CDBL(SQR(C#)) THEN 4550
4527 B%=-INT(-Y#)
4528 D%=B%-1
4529 A%=-INT(-C#/(B%-Z%))
4530 G%=-INT(-C#/(D%-Z%))
```

Finds the two (2) of three (3) points on the curve through which the deep cut plane is passed.

```
4531 P%=B%
4532 FOR I=1 TO BI#
4533 P%=P%+1
4534 J%=-INT(-C#/(P%-Z%))
4535 IF J%<CDBL(SQR(C#)) THEN 4540
4536 IF J%*(B%-D%)+P%*(G%-A%)>B%*G%-A%*D% THEN 4540
4537 B%=P%
4538 A%=J%
4539 NEXT I
4540 P%=D%
4541 FOR I=1 TO BI#
4542 P%=P%-1
4543 IF P%<Z%+1 THEN 4549
4544 J%=-INT(-C#/(P%-Z%))
4545 IF J%*(B%-D%)+P%*(G%-A%)>B%*G%-A%*D% THEN 4549
4546 D%=P%
4547 G%=J%
```

```
4548 NEXT I
4549 GOTO 4572
4550 G%=-INT(-X#)
4551 A%=G%-1
4552 D%=-INT(-(C#+Z%*G%)/G%)
4553 B%=-INT(-(C#+Z%*A%)/A%)
4554 P%=A%
4555 FOR I=1 TO BI#
4556 P%=P%-1
4557 IF P%<1 THEN 4563
4558 J%=-INT(-(C#+P%*Z%)/P%)
4559 IF P%*(B%-D%)+J%*(G%-A%)>B%*G%-A%*D% THEN 4563
4560 A%=P%
4561 B%=J%
4562 NEXT I
4563 P%=G%
4564 FOR I=1 TO BI#
4565 P%=P%+1
4566 IF P%>CDBL(SQR(C#)) THEN 4572
4567 J%=-INT(-(C#+P%*Z%)/P%)
4568 IF P%*(B%-D%)+J%*(G%-A%)>B%*G%-A%*D% THEN 4572
4569 G%=P%
4570 D%=J%
4571 NEXT I
```

Finds the equation of the plane through the two (2) points found above and the Z and Y axes plane point.

```
4572 A#(K,1)=CDBL((G%*B%-A%*D%)+Z%*(A%-G%))
4573 FOR J=2 TO ND+1
4574 A#(K,J)=0#
4575 IF ZX=J THEN A#(K,J)=CDBL(B%-D%)
4576 IF ZY=J THEN A#(K,J)=CDBL(G%-A%)
4577 IF ZZ=J THEN A#(K,J)=CDBL(A%-G%)
4578 NEXT J
4579 RETURN
4580 A#(K,1)=1#
4581 FOR J=2 TO ND+1
4582 A#(K,J)=0#
4583 IF J=ZY THEN A#(K,J)=1#
4584 IF J=ZZ THEN A#(K,J)=-1#
4585 NEXT J
4586 RETURN
4587 A#(K,1)=1#
4588 FOR I=2 TO ND+1
4589 A#(K,I)=0#
4590 IF I=ZX THEN A#(K,I)=1#
4591 NEXT I
```

```
4592 RETURN
```

Deep Cut for Sheet Using Gordian Algorithm The deep cut for the sheet constraint using the Gordian algorithm subroutine (SHT-DEPG) finds a point on the sheet constraint, a constraint whose parameters are stored in the last seven (7) columns of the constraint matrix, by using the Gordian algorithm from which is constructed a deep cut plane and then loads the plane's coefficients into the linear portion of the constraint matrix.

```
4500 REM   *DEEP CUT FOR SHEET USING GORDIAN ALGORITHM*
4501 REM-----------------------SHT-DEPG-------------------------

4502 ZX=A#(K,ND+5)+1
4503 ZY=A#(K,ND+6)+1
4504 ZZ=A#(K,ND+7)+1
4505 C#=A#(K,ND+4)
4506 X#=B#(ZX,1)
4507 Y#=B#(ZY,1)
4508 Z#=B#(ZZ,1)
4509 IF X#=H#(ZX) AND Y#=H#(ZY) AND Z#=H#(ZZ) THEN RETURN
4510 IF X#>=0# AND Y#>=0# AND Z#>=0# AND X#*(Y#-Z#)>=C# THEN RETURN
4511 G#=CDBL(SQR(C#))
4512 IF X#<G#/2# AND (Y#-Z#)<G#/2# THEN 4516
4513 IF X#>Y#-Z# THEN 4517
4514 X#=C#/(Y#-Z#)
4515 GOTO 4517
4516 X#=G#
4517 Z%=INT(Z#)
4518 Y#=(C#+Z%*X#)/X#
4519 IF Y#-Z%<=1# THEN 4575
4520 IF X#<=1# THEN 4582
```

Finds the two (2) of three (3) points on the curve through which the deep cut plane is passed.

```
4521 IF X#<CDBL(SQR(C#)) THEN 4545
4522 B%=-INT(-Y#)
4523 D%=B%-1
4524 A%=-INT(-C#/(B%-Z%))
4525 G%=-INT(-C#/(D%-Z%))
4526 P%=B%
```

```
4527 FOR I=1 TO BI#
4528 P%=P%+1
4529 J%=-INT(-C#/(P%-Z%))
4530 IF J%<SQR(C#) THEN 4535
4531 IF J%*(B%-D%)+P%*(G%-A%)>B%*G%-A%*D% THEN 4535
4532 B%=P%
4533 A%=J%
4534 NEXT I
4535 P%=D%
4536 FOR I=1 TO BI#
4537 P%=P%-1
4538 IF P%<Z%+1 THEN 4544
4539 J%=-INT(-C#/(P%-Z%))
4540 IF J%*(B%-D%)+P%*(G%-A%)>B%*G%-A%*D% THEN 4544
4541 D%=P%
4542 G%=J%
4543 NEXT I
4544 GOTO 4567
4545 G%=-INT(-X#)
4546 A%=G%-1
4547 D%=-INT(-(C#+Z%*G%)/G%)
4548 B%=-INT(-(C#+Z%*A%)/A%)
4549 P%=A%
4550 FOR I=1 TO BI#
4551 P%=P%-1
4552 IF P%<1 THEN 4558
4553 J%=-INT(-(C#+P%*Z%)/P%)
4554 IF P%*(B%-D%)+J%*(G%-A%)>B%*G%-A%*D% THEN 4558
4555 A%=P%
4556 B%=J%
4557 NEXT I
4558 P%=G%
4559 FOR I=1 TO BI#
4560 P%=P%+1
4561 IF P%>SQR(C#) THEN 4567
4562 J%=-INT(-(C#+P%*Z%)/P%)
4563 IF P%*(B%-D%)+J%*(G%-A%)>B%*G%-A%*D% THEN 4567
4564 G%=P%
4565 D%=J%
4566 NEXT I
```

Finds the equation of the plane through the two (2) points found above and the Z and Y axes plane point.

```
4567 A#(K,1)=CDBL((G%*B%-A%*D%)+Z%*(A%-G%))
4568 FOR J=2 TO ND+1
4569 A#(K,J)=0#
4570 IF ZX=J THEN A#(K,J)=CDBL(B%-D%)
```

```
4571 IF ZY=J THEN A#(K,J)=CDBL(G%-A%)
4572 IF ZZ=J THEN A#(K,J)=CDBL(A%-G%)
4573 NEXT J
4574 RETURN
4575 A#(K,1)=1#
4576 FOR J=2 TO ND+1
4577 A#(K,J)=0#
4578 IF J=ZY THEN A#(K,J)=1#
4579 IF J=ZZ THEN A#(K,J)=-1#
4580 NEXT J
4581 RETURN
4582 A#(K,1)=1#
4583 FOR I=2 TO ND+1
4584 A#(K,I)=0#
4585 IF I=ZX THEN A#(K,I)=1#
4586 NEXT I
4587 RETURN
```

Cubic Hyperbolic Subroutine -- File CBH-DEPG

The deep cut for the cubic hyperbolic function X*Y^2=c can be found

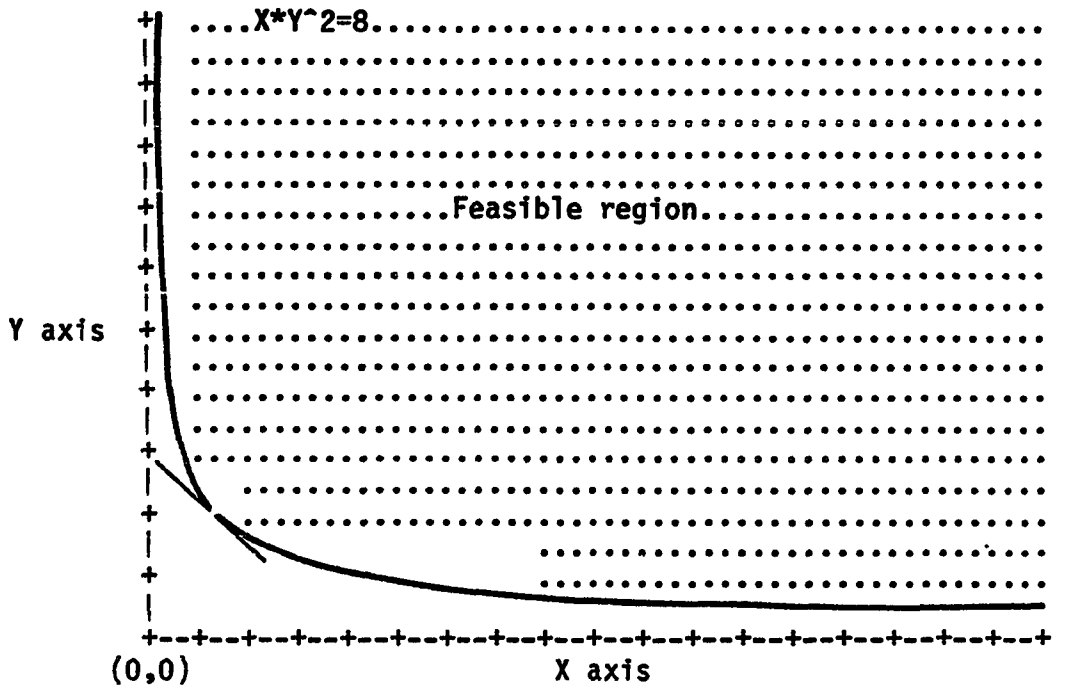


Figure 78. Cubic hyperbolic constraint step function

using the same approach as used for the hyperbolic function. The tangent point on the function can be found using the Gordian algorithm. From the single point, two (2) points on the underlying step function can be found using the same procedure as used for the hyperbolic function except that the slope characteristic segments of the function have been move as shown in figure 78.

Deep Cut for Cubic Hyperbolic Using Gordian Algorithm The deep cut for the cubic hyperbolic constraint using the Gordian algorithm subroutine (CBH-DEPG) finds a point on the cubic hyperbolic constraint, a constraint whose parameters are stored in the last seven (7) columns of the constraint matrix, by using the Gordian algorithm from which is constructed a deep cut line and then loads the line's coefficients into the linear portion of the constraint matrix.

```
4700 REM       *CUBIC HYPERBOLIC GORDIAN DEEP CUT*
4701 REM---------------------CBH-DEPG-------------------------------

4702 ZX=A#(K,ND+5)+1
4703 ZY=A#(K,ND+6)+1
4704 C#=A#(K,ND+3)
4705 Y#=B#(ZY,1)
4706 X#=B#(ZX,1)
4707 IF X#=H#(ZX) AND Y#=H#(ZY) THEN RETURN
4708 IF X#>0# AND Y#>0# AND X#*(Y#*Y#)>=C# THEN RETURN
4709 H#=(2#*C#)^(1#/3#)
4710 G#=C#/(H#*H#)
4711 IF X#<G# AND Y#<H# THEN 4716
4712 IF Y#>=H# THEN X#=C#/(Y#*Y#)
4713 IF X#>=C# THEN 4771
4714 IF X#<=1# THEN 4774
4715 GOTO 4717
4716 X#=G#
4717 Y#=CDBL(SQR(C#/X#))
```

Finds the two points on the curve through which the deep cut line is passed.

```
4718 IF X#<G# THEN 4742
4719 B%=-INT(-Y#)
4720 D%=B%-1
4721 A%=-INT(-C#/(B%*B%))
4722 G%=-INT(-C#/(D%*D%))
4723 P%=B%
4724 FOR L=1 TO BI#
4725 P%=P%+1
4726 J%=-INT(-C#/(P%*P%))
4727 IF J%<G# THEN 4732
4728 IF J%*(B%-D%)+P%*(G%-A%)>B%*G%-A%*D% THEN 4732
4729 B%=P%
4730 A%=J%
4731 NEXT L
4732 P%=D%
4733 FOR L=1 TO BI#
4734 P%=P%-1
4735 IF P%<1 THEN 4741
4736 J%=-INT(-C#/(P%*P%))
4737 IF J%*(B%-D%)+P%*(G%-A%)>B%*G%-A%*D% THEN 4741
4738 D%=P%
4739 G%=J%
4740 NEXT L
4741 GOTO 4764
4742 G%=-INT(-X#)
4743 A%=G%-1
4744 D%=-INT(-SQR(C#/G%))
4745 B%=-INT(-SQR(C#/A%))
4746 P%=A%
4747 FOR L=1 TO BI#
4748 P%=P%-1
4749 IF P%<1 THEN 4755
4750 J%=-INT(-SQR(C#/P%))
4751 IF P%*(B%-D%)+J%*(G%-A%)>B%*G%-A%*D% THEN 4755
4752 A%=P%
4753 B%=J%
4754 NEXT L
4755 P%=G%
4756 FOR L=1 TO BI#
4757 P%=P%+1
4758 IF P%>G# THEN 4764
4759 J%=-INT(-SQR(C#/P%))
4760 IF P%*(B%-D%)+J%*(G%-A%)>B%*G%-A%*D% THEN 4764
4761 G%=P%
4762 D%=J%
4763 NEXT L
```

Finds the equation of the line through the two (2) points above.

```
4764 A#(K,1)=CDBL(B%*G%-A%*D%)
4765 FOR L=2 TO ND+1
4766 A#(K,L)=0#
4767 IF L=ZX THEN A#(K,L)=CDBL(B%-D%)
4768 IF L=ZY THEN A#(K,L)=CDBL(G%-A%)
4769 NEXT L
4770 RETURN
4771 B#=1#
4772 A#=0#
4773 GOTO 4776
4774 A#=1#
4775 B#=0#
4776 A#(K,1)=1#
4777 FOR L=2 TO ND+1
4778 A#(K,L)=0#
4779 IF L=ZY THEN A#(K,L)=B#
4780 IF L=ZX THEN A#(K,L)=A#
4781 NEXT L
4782 RETURN
```

Cubic Hyperbolic Sheet Subroutine -- File CBS-DEPG

As with the sheet constraint, the cubic hyperbolic of two sheets function X*(Y-Z)^2=c can be visualized as a cylinder in which sections in the X and Y axes plane are a series of hyperbolic like functions and sections in the Z and Y axes plane are a series of lines at forty-five (45) degrees. When the two (2) deep cut points on the hyperbolic like function that corresponds to the X and Y axes plane which passes through the [z] coordinate point are found as say (a%,b%,[z]) and (g%,d%,[z]) by using the underlying step function, then the third point (g%,d%+1,[z]+1) can be found by assuming a forty-five (45) degree cut in the Z and Y axes plane.

Figure 79 shows the slope characteristic segments of the hyperbolic like curve in the X and Y axes plane.

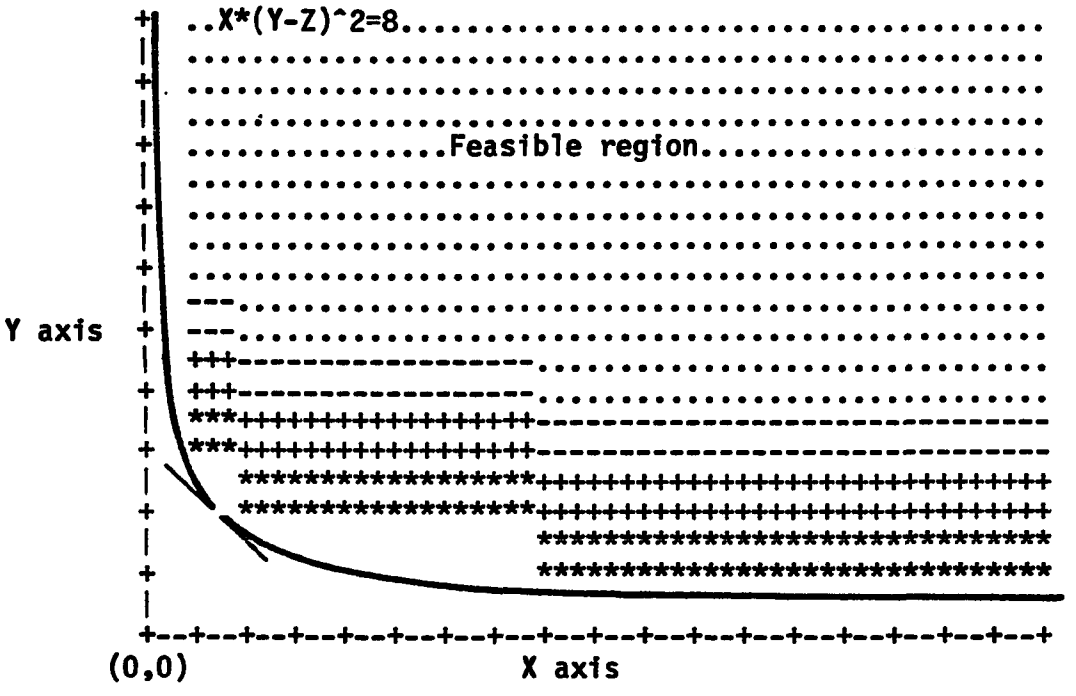


Figure 79. Cubic sheet constraint step function

Deep Cut for Cubic Sheet Using Gordian Algorithm The deep cut for cubic sheet constraint using the Gordian algorithm subroutine (CBS-DEPG) finds a point on the cubic sheet constraint, a constraint whose parameters are stored in the last seven (7) columns of the constraint matrix, by using the Gordian algorithm from which is constructed a deep cut plane and then loads the plane's coefficients into the linear portion of the constraint matrix.

```
4900 REM     *CUBIC SHEET GORDIAN DEEP CUT SUBROUTINE*
4901 REM--------------------CBS-DEPG--------------------------------

4902 C#=A#(K,ND+4)
4903 ZX=1+A#(K,ND+5)
4904 ZY=1+A#(K,ND+6)
4905 ZZ=1+A#(K,ND+7)
4906 Y#=B#(ZY,1)
```

```
4907 X#=B#(ZX,1)
4908 Z#=B#(ZZ,1)
4909 IF X#=H#(ZX) AND Y#=H#(ZY) AND Z#=H#(ZZ) THEN RETURN
4910 IF X#>0# AND Z#=0# AND Y#-Z#>0# AND X#*(Y#-Z#)*(Y#-Z#)>=C# THEN RET
     URN
4911 H#=(2#*C#)^(1#/3#)
4912 G#=C#/(H#*H#)
4913 IF X#<G# AND (Y#-Z#)<H# THEN 4916
4914 IF (Y#-Z#)>=H# THEN X#=C#/((Y#-Z#)*(Y#-Z#))
4915 GOTO 4917
4916 X#=G#
4917 Z%=INT(Z#)
4918 Y#=(SQR(C#)+Z%*SQR(X#))/SQR(X#)
4919 IF Y#-Z%<1# THEN 4975
4920 IF X#<=1# THEN 4982
```

Finds the two (2) of three (3) points on the function through which the deep cut plane is passed.

```
4921 IF X#<G# THEN 4945
4922 B%=-INT(-Y#)
4923 D%=B%-1
4924 A%=-INT(-C#/((B%-Z%)*(B%-Z%)))
4925 G%=-INT(-C#/((D%-Z%)*(D%-Z%)))
4926 P%=B%
4927 FOR I=1 TO BI#
4928 P%=P%+1
4929 J%=-INT(-C#/((P%-Z%)*(P%-Z%)))
4930 IF J%<G# THEN 4935
4931 IF J%*(B%-D%)+P%*(G%-A%)>B%*G%-A%*D% THEN 4935
4932 B%=P%
4933 A%=J%
4934 NEXT I
4935 P%=D%
4936 FOR I=1 TO BI#
4937 P%=P%-1
4938 IF (P%-Z%)<1 THEN 4944
4939 J%=-INT(-C#/((P%-Z%)*(P%-Z%)))
4940 IF J%*(B%-D%)+P%*(G%-A%)>B%*G%-A%*D% THEN 4944
4941 D%=P%
4942 G%=J%
4943 NEXT I
4944 GOTO 4967
4945 G%=-INT(-X#)
4946 A%=G%-1
4947 D%=-INT(-(SQR(C#)+Z%*SQR(G%))/SQR(G%))
4948 B%=-INT(-(SQR(C#)+Z%*SQR(A%))/SQR(A%))
4949 P%=A%
```

```
4950 FOR I=1 TO BI#
4951 P%=P%-1
4952 IF P%<1 THEN 4958
4953 J%=-INT(-(SQR(C#)+Z%*SQR(P%))/SQR(P%))
4954 IF P%*(B%-D%)+J%*(G%-A%)>B%*G%-A%*D% THEN 4958
4955 A%=P%
4956 B%=J%
4957 NEXT I
4958 P%=G%
4959 FOR I=1 TO BI#
4960 P%=P%+1
4961 IF P%>G# THEN 4967
4962 J%=-INT(-(SQR(C#)+Z%*SQR(P%))/SQR(P%))
4963 IF P%*(B%-D%)+J%*(G%-A%)>B%*G%-A%*D% THEN 4967
4964 G%=P%
4965 D%=J%
4966 NEXT I
```

Finds the equation of the plane through the two (2) points found above and the Y and Z axes plane point.

```
4967 A#(K,1)=CDBL((G%*B%-A%*D%)+Z%*(A%-G%))
4968 FOR J=2 TO ND+1
4969 A#(K,J)=0#
4970 IF ZX=J THEN A#(K,J)=CDBL(B%-D%)
4971 IF ZY=J THEN A#(K,J)=CDBL(G%-A%)
4972 IF ZZ=J THEN A#(K,J)=CDBL(A%-G%)
4973 NEXT J
4974 RETURN
4975 A#(K,1)=1#
4976 FOR J=2 TO ND+1
4977 A#(K,J)=0#
4978 IF J=ZY THEN A#(K,J)=1#
4979 IF J=ZZ THEN A#(K,J)=-1#
4980 NEXT J
4981 RETURN
4982 A#(K,1)=1#
4983 FOR I=2 TO ND+1
4984 A#(K,I)=0#
4985 IF I=ZX THEN A#(K,I)=1#
4986 NEXT I
4987 RETURN
```

Program Table of Contents

Table 19 can be used to reconstruct the above code from the computer disk and to organize subroutines from previous program listings. Since

Table 19. Gomory's method with nonlinear constraints BASIC program table of contents

File	Program lines	Page	Routines
MAIN-GOM	0001-0137	288	Gomory's method
INPT-OBJ	1200-1220	61	Objective coefficient input subroutine
INPT-CON	1300-1328	63	Constraint coefficient input subroutine
INPT-TYP	1400-1440	65	Constraint type input subroutine
INPT-BND	1500-1523	67	Variable bounds input subroutine
UTIL-OPT	1800-1803	22	Option line subroutine
UTIL-ERS	1850-1853	22	Erase option subroutine
UTIL-CON	1860-1866	22	Continue line subroutine
UTIL-CHX	1870-1882	23	Data check subroutine
REPT-SMP	2100-2125	69	Simplex report subroutine
ALGR-KEY	3000-3057	200	Constraint key subroutine
ALGR-GOM	3300-3499	294	Gomory's algorithm subroutine
TRAN-CON	3600-3633	74	Constraint transformation subroutine
TRAN-INV	3700-3743	76	Basis inversion subroutine
TRAN-RIV	3900-3960	78	Basis reinversion subroutine
PAR-DEPL	4100-4189	311	Deep cut for parabolic subroutine
HYP-DEPL	4300-4385	318	Deep cut for hyperbolic subroutine
SHT-DEPL	4500-4592	325	Deep cut for sheet subroutine

BASIC code is dependent on program line numbers for its subroutine branching, the statement numbers must be maintained as listed above.

Gomory's Method with Nonlinear Constraints: Integer Solutions to Example 11 Minimum Project Man Count Problem

Using the nonlinear modification of the dual simplex method, the ten activity minimum project man count problem was solved in real numbers for example 11. The same example can be solved in all integers using the nonlinear modification of Gomory's method. The results are displayed in table 20 which lists for each fixed project duration the number of iterations required to reach a solution, the seconds required

to reach the solution, the value of the objective function, and the value of the variables.

Table 20. Gomory's method, using deep cuts derived with the line search algorithm, integer solutions to the example 11 minimum project man count problem with hyperbolic man count functions

Dur.	100	90	80	70	60	50	40	30	20	10
Itr.	580	5298	1110	445	822	6737	140	83	110	107
Sec.	1540	13934	2718	1268	2189	17604	343	250	313	228
Obj.	16	19	20	22	24	29	34	46	65	132
T1	10	15	15	10	8	5	6	6	5	2
T2	50	35	35	30	28	27	20	14	10	5
T3	80	60	60	47	45	40	30	24	15	7
T4	100	90	80	67	60	50	40	30	20	10
M5	1	1	1	1	1	1	1	1	1	2
M6	1	1	1	1	1	1	1	2	2	4
M7	3	2	2	3	4	6	5	5	6	15
M8	1	2	2	2	2	2	3	5	8	14
M9	2	2	2	3	3	4	5	5	10	25
M10	3	2	3	3	4	6	6	10	12	20
M11	1	2	2	2	2	2	3	4	7	14
M12	2	3	3	3	3	3	4	6	8	16
M13	1	2	2	2	2	2	3	4	6	12
M14	1	2	2	2	2	2	3	4	5	10

BRANCH AND BOUND METHOD WITH HYPERBOLIC AND PARABOLIC CONSTRAINTS

Even when deep cut constraints are used to strengthen the Gomory or Wilson cut, Gomory's method does not consistently yield an integer solution within a reasonable number of iterations. A method which does consistently yield an integer solution is the branch and bound method.

Theory of Branch and Bound Method

The branch and bound method[52] is primarily a "bookkeeping" scheme, like the restart method, which creates and solves, or discards, a series of CP subproblems derived from the original CP problem. Each subproblem contains a partition of the original feasible region which has been systematically restricted to a smaller and smaller set of solutions by means of the addition of integer constraints to the original CP problem. The method continues to subdivide the feasible region until the minimum optimal solution to one of the subproblems is integer and is less than the optimal solution to all the other feasible subproblems that together form the original problem's feasible region.

The simplest means to demonstrate the branch and bound method is with a two (2) variable example 14.

minimize X+Y

subject to: X >=0
Y>=0
-X +Y>=-1
12*X+5*Y>=15

X,Y integer

Example 14 is shown graphically in figure 80. In addition to the

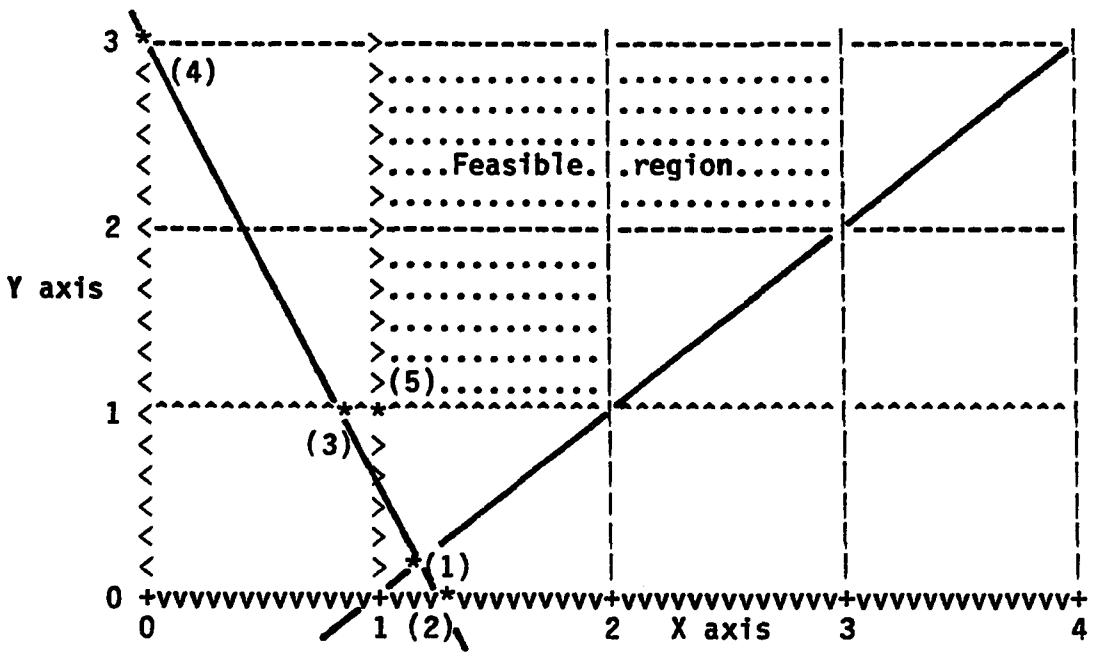


Figure 80. Branch and bound method example 14 feasible region

constraint lines and the X and Y axes, an integer lattice is added to the feasible region which defines the points within the feasible region that are allowable as integer solutions.

Branch and Bound Node Tree

Since the branch and bound method is a book keeping scheme which keeps track of a series of subproblems which are called "nodes", a method for storing the solutions of the subproblems must be defined.

Table 21. Branch and bound method node table entry

Line	Node value and bound var.	X	Y	New bounds
1	current objective value	upper bound	upper bound	upper bound
2	new bounded variable no.	lower bound	lower bound	lower bound

Node table 21 contains the information saved on each subproblem. An entry for each subproblem contains the value of the objective function when the subproblem is optimized, a set of upper and lower integer bounds for each variable in the subproblem, and a set of "tighter" bounds for one of the variables that is not integer in the subproblem solution.

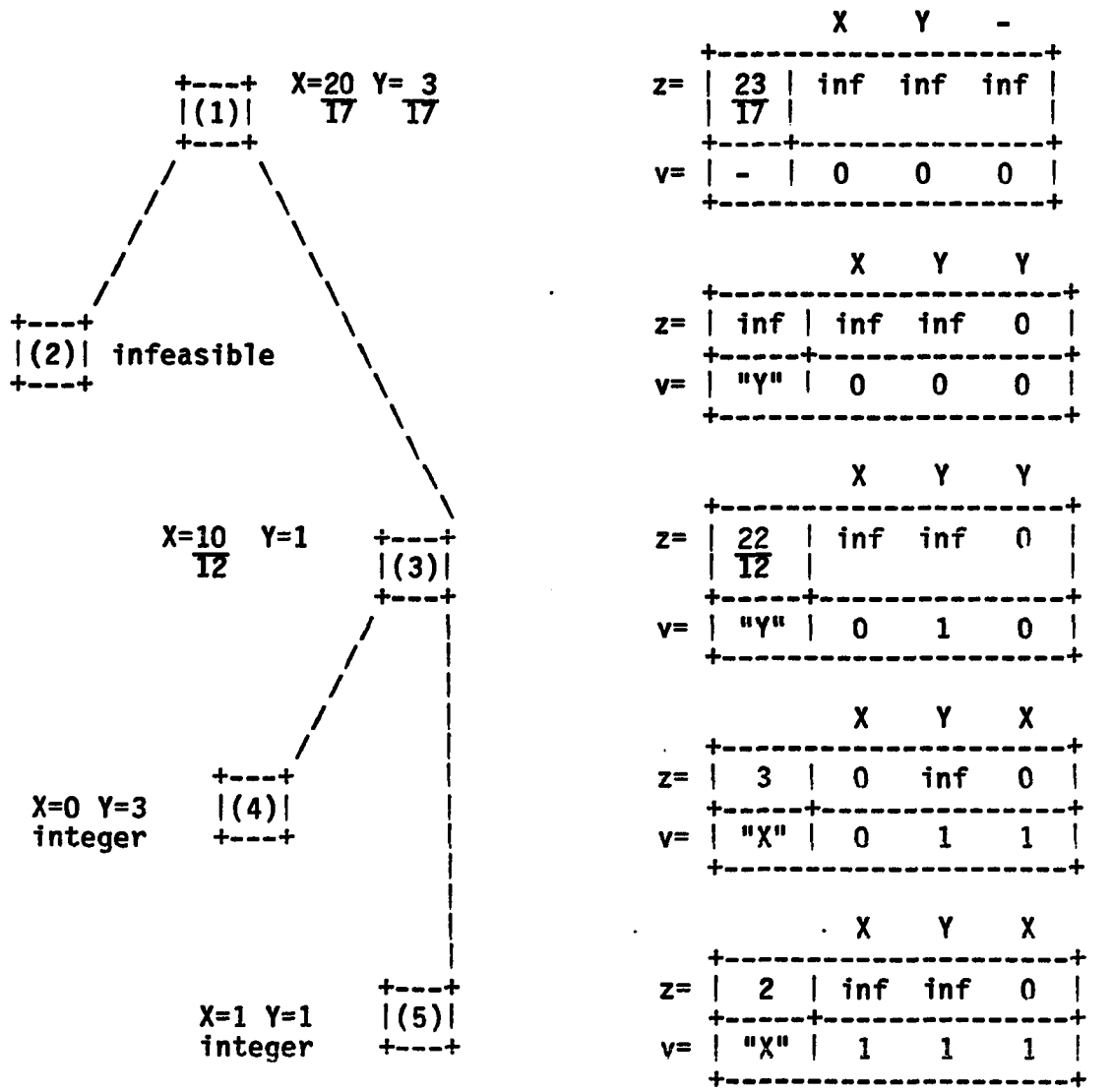


Figure 81. Branch and bound method example 14 branch and bound tree

To start the algorithm, the original example 14 integer LP problem is solved without any integer bounds other than the lower variable bounds of zero (0). This solution for example 14 corresponds to node (1) in a "branch and bound tree" as shown in figure 81.

Upper and Lower Integer Bounds

Without any integer restrictions on the variables, the optimal solution to example 14 is not integer. To find an integer solution, the feasible region must be restricted by adding an integer upper or lower bound to one of the variables. The general practice is to choose the variable, say Y, which is closest to being integer in the minimum subproblem or node solution and to set the integer bounds for two (2) new nodes at less than or equal [y] and greater than or equal [y]+1.

Again in figure 81, node (1) is "branched" into two (2) subproblems for nodes (2) and (3) and then discarded since it is not an integer solution. The variables X and Y are both as nearly integer, so Y is randomly selected for setting the bounds on two (2) new subproblem.

The simplex solutions for the two (2) new nodes' subproblem are shown in figures 80 or as solution points (2) and (3) on figure 80. Since the subproblem for node (2) is infeasible, the node is discarded and only node (3) is considered. If both nodes had solutions, then the node with the minimum optimal solution would be considered for branching. The solution for node (3) is integer in Y and noninteger in X, so the node is considered for the next branch with bounds set on X.

The next two (2) nodes are shown in figures 81 or as solution points

(4) and (5) in figure 80. The simplex solutions for the two (2) subproblems yield two (2) integer solutions. Any integer solution derived from further branching would have greater solution values; and since the integer solution to the subproblem at node (5) is the minimum solution of all the nodes not discarded, it is the optimal integer solution to the original problem.

Branch and Bound with Beale's Method BASIC Code

The branch and bound method consists of a scheme of selecting nodes and then branching from the selected nodes into other nodes. The actual processing of the subproblems associated with the nodes can be done with several simplex related algorithm and can be considered separate from the book keeping scheme of the branch and bound method.

The book keeping scheme is contained in the branch and bound algorithm (ALGR-BND). The processing of the subproblems in this program is done with Beale's algorithm with nonlinear constraints which allows for the use of a quadratic objective function and nonlinear convex constraints in the integer CP problem.

All programs listed in the text consist of a main calling routine and a series of input, output, and processing subroutine. These routines are listed in the text as well as on a computer disk compatible with IBM micro-computers.

Branch and Bound Main Routine -- File MAIN-BND

The branch and bound main routine (MAIN-BND) dimensions the thirteen (13) data arrays; writes the option menu to the screen as shown

```
                BRANCH AND BOUND METHOD

          NUMBER OF CONSTRAINTS          10
          NUMBER OF VARIABLES            14
          MAXIMUM ITERATIONS             1000
          BEST AVAILABLE SOLUTION        1000000000

                M-RETURN TO MENU

            O-OBJECTIVE COEFFICIENTS
            A-CONSTRAINT COEFFICIENTS
            C-CONSTRAINT TYPE
            B-BOUNDED VARIABLES
            Q-QUADRATIC COEFFICIENTS
            V-MAXIMUM VALUE OF OBJECTIVE
            U-EXECUTE ALGORITHM
            R-REPORT LISTING
            S-SAVE  F-FETCH
            N-NEW PROBLEM

        OPTION ?

                              O'X+X'QX=z
          1*X*2*Y>=c  1*X*(2*Y-3*X)>=c  1*Y+a*(X-b)^2>=c
```

Figure 82. Branch and bound main menu screen

in figure 82; calls the utility subroutines UTIL-OPT, UTIL-ERS, UTIL-CON, UTIL-CHX; calls the data input and output subroutines INPT-OBJ, INPT-CON, INPT-TYP, INPT-BND, INPT-QUD, and REPT-SMP; calls and times the processing algorithms ALGR-BND, ALGR-KEY, and ALGR-BEA; and saves and fetches the input data to and from disk.

```
1 REM                    *BRANCH AND BOUND METHOD*
2 REM----------------------------MAIN-BND-----------------------

3 REM      BE# - MAXIMUM VALUE OF OBJECTIVE ON STARTING
4 REM      BI# - MACHINE INFINITE
5 REM      CO  - PIVOT COLUMN FOR SIMPLEX TRANSFORMATION
6 REM      ER  - ERROR KEY
7 REM      IN  - NUMBER OF ITERATIONS BETWEEN REINVERSIONS
8 REM      IR  - MAXIMUM NUMBER OF ITERATIONS
9 REM      MD  - NUMBER OF CONSTRAINTS
```

```
10 REM     ND  - NUMBER OF VARIABLES
11 REM     PA  - NUMBER OF ATTEMPTS AT PIVOT
12 REM     PM  - SIGN KEY (+-)
13 REM     RO  - PIVOT ROW FOR SIMPLEX TRANSFORMATION
14 REM     RQ  - PIVOT ROW IN PARTIAL MATRIX
15 REM     SM# - MACHINE ZERO
16 REM     A#(MD+2+ND,ND+8) - ORIGINAL DATA AND KEYS
17 REM     B#(2*ND+1,ND+1) - PRIMAL-DUAL MATRIX
18 REM     H#(ND+1) - SOLUTIONS FORM LAST ITERATION
19 REM     M#(ND+1,2) - UPPER AND LOWER BOUNDS ON VARIABLES
20 REM     N%(2*MX,ND+2) - NODE MATRIX FOR BRANCH NETWORK
21 REM     O#(ND+1,ND+1+ND) - UNRESTRICTED BASIS SOLUTION
22 REM     P#(ND+1) - WORK VECTOR
23 REM     Q#(ND+1,ND+1) - QUADRATIC HESSIAN MATRIX
24 REM     R(MD+1+ND+1) - CONSTRAINT TYPE (1->=,0-=,-1-<=)
25 REM     S#(ND+1) - FREE VARIABLE COLUMN SWITCH
26 REM     T#(ND+1,ND+1) - INVERSION WORK MATRIX
27 REM     V#(MD+1+ND+1+ND+1+ND+1) - ROW AND COLUMN ARRAY
28 REM     X#(ND) - SOLUTION VECTOR
29 REM ------------------------------------------------------------
```

Sets MD to the default number of constraints in the integer CP problem and ND to the number of variables. Sets IN to the number of iterations before a reinversion of the augmented $\overline{B}$ matrix. Sets MX to the default maximum number of active nodes in the branch and bound node array N%(2*MX,ND+2). Sets IR to the default maximum number of iterations before the algorithm is terminated. Sets BI# to a number considered machine or computer infinite and SM# to a number considered machine zero.

```
30 MD=0
31 ND=0
32 IN=10
33 MX=100
34 IR=1000
35 BI#=1E+10
36 BI%=32000
37 SM#=1E-09
38 OB#=BI#
```

Prompts and reads the number of constraints MD in the integer CP problem, the number of variables ND, and the maximum number of iterations IR allowed before the algorithm is stopped.

```
39 CLS
40 LOCATE 1,10:PRINT "BRANCH AND BOUND METHOD"
41 LOCATE 3,1:PRINT "NUMBER OF CONSTRAINTS":LOCATE 3,31:INPUT "",L$
42 GOSUB 1870:REM UTIL-CHX
```

```
43 IF Z#<>BI# THEN MD=Z#
44 LOCATE 3,30:PRINT MD,"    ":LOCATE 4,1:PRINT "NUMBER OF VARIABLES"
   :LOCATE 4,31:INPUT "",L$
45 GOSUB 1870:REM UTIL-CHX
46 IF Z#<>BI# THEN ND=Z#
47 LOCATE 4,30:PRINT ND,"    ":LOCATE 5,1:PRINT "MAXIMUM ITERATIONS"
   :LOCATE 5,31:INPUT "",L$
48 GOSUB 1870:REM UTIL-CHX
49 IF Z#<>BI# THEN IR=Z#:LOCATE 5,30:PRINT IR,"   "
50 LOCATE 5,30:PRINT IR,"   "
```

Dimensions the A#(MD+2+ND+1,ND+8) array which contains the linear portions of the quadratic objective function, the constraint coefficients, and the first step of the transformation of the quadratic partials. Dimensions the augmented $\overline{B}$ inverse matrix and the augmented $\overline{B}$ matrix array B#(2*ND+1,ND+1). Dimensions the holding array H#(ND+1), the upper and lower bounds array M#(ND+1), the pivot row P#(ND+1), the augmented $\overline{Q}$ matrix array Q#(ND+1,ND+1), the enlarged constraint type array R(MD+1+ND+1), the switch array S#(ND+1), the reinversion working space array T#(ND+1,ND+1), the pivot row selection array V#(MD+1+ND+1+ND+1+ND+1), and the solution vector X#(ND). Dimensions the augmented $\overline{B}$ inverse and augmented $\overline{B}$ matrix of the noninteger optimal solution array O#(ND+1+ND,ND+1). Dimensions the branch and bound node array N%(2*MX,ND+2).

```
51 DIM A#(MD+2+ND+1,ND+8)
52 DIM B#(2*ND+1,ND+1)
53 DIM H#(ND+1)
54 DIM M#(ND+1,2)
55 DIM O#(ND+1+ND,ND+1)
56 DIM P#(ND+1)
57 DIM N%(2*MX,ND+2)
58 DIM Q#(ND+1,ND+1)
59 DIM R(MD+1+ND+1)
60 DIM S#(ND+1)
61 DIM T#(ND+1,ND+1)
62 DIM V#(MD+1+ND+1+ND+1+ND+1)
63 DIM X#(ND)
```

Initializes the constraint type array to all greater than or equals.

```
64 FOR I=2 TO MD+1+ND+1
65 R(I)=1
66 NEXT I
```

Prints the option menu to the screen; calls the option line routine UTIL-OPT; and pauses for the entry of "M", "O", "A", "C", "B", "Q", "U", "R", "S", "F", "N" for the option variable L$.

```
67 LOCATE 8,15:PRINT              "M-RETURN TO MENU"
68 LOCATE 10,10:PRINT         "O-OBJECTIVE COEFFICIENTS"
69 LOCATE 11,10:PRINT         "A-CONSTRAINT COEFFICIENTS"
70 LOCATE 12,10:PRINT         "C-CONSTRAINT TYPES"
71 LOCATE 13,10:PRINT         "B-BOUNDED VARIABLES"
72 LOCATE 14,10:PRINT         "Q-QUADRATIC COEFFICIENTS"
73 LOCATE 15,10:PRINT         "V-MAXIMUM VALUE OF OBJECTIVE"
74 LOCATE 16,10:PRINT         "U-EXECUTE ALGORITHM"
75 LOCATE 17,10:PRINT         "R-REPORT LISTING"
76 LOCATE 18,10:PRINT         "S-SAVE F-FETCH"
77 LOCATE 19,10:PRINT         "N-NEW PROBLEM"
78 LOCATE 22,1:PRINT "                    OX'+XQX'=z"
79 LOCATE 23,1:PRINT "1*X*2*Y>=c 1*X*(2*Y-3*Z)>=c  1*Y+a*(X-b)^2>=c"
80 GOSUB 1800:REM UTIL-OPT
```

Prompts and reads the maximum value of the objective function OB# at the start of the branch and bound method.

By having a good guess at the solution at the start of the branching algorithm, many of the useless nodes can be eliminated early.

```
81 LOCATE 21,8:INPUT "",L$
82 IF L$<>"V" THEN 87
83 LOCATE 6,1:PRINT "MAXIMUM VALUE OF OBJECTIVE":LOCATE 6,31:INPUT "",L$
84 GOSUB 1870:REM UTIL-CHX
85 IF Z#<>BI# THEN OB#=Z#
86 LOCATE 6,30:PRINT OB#,"    "
```

Calls either the objective function input subroutine INPT-OBJ, the constraint input subroutine INPT-CON, the constraint type input subroutine INPT-TYP, the upper and lower bounds input subroutine INPT-BND, the quadratic $\bar{Q}$ matrix input subroutine INPT-QUD, the processing subroutine ALGR-BND, or the report routine REPT-SMP based on the option variable L$.

```
87 CLS
88 H=0
89 G=2
90 IF L$<>"O" THEN 93
91 GOSUB 1200:REM INPT-OBJ
92 GOTO 82
93 IF L$<>"A" THEN 96
94 GOSUB 1300:REM INPT-CON
95 GOTO 82
96 IF L$<>"C" THEN 99
97 GOSUB 1400:REM INPT-TYP
98 GOTO 82
```

```
99 IF L$<>"B" THEN 102
100 GOSUB 1500:REM INPT-BND
101 GOTO 82
102 IF L$<>"Q" THEN 105
103 GOSUB 1600:REM INPT-QUD
104 GOTO 82
105 IF L$<>"U" THEN 110
106 TM=3600*VAL(MID$(TIME$,1,2))+60*VAL(MID$(TIME$,4,2))+VAL(MID$(TIME$,
    7,2))
107 GOSUB 3000:REM ALGR-KEY
108 TM=3600*VAL(MID$(TIME$,1,2))+60*VAL(MID$(TIME$,4,2))+VAL(MID$(TIME$,
    7,2))-TM
109 GOTO 67
110 IF L$<>"R" THEN 113
111 GOSUB 2200:REM REPT-SMP
112 GOTO 82
```

Saves the content of MD, ND, M#(ND+1,2), A#(MD+1+ND,ND+8), Q#(ND+1,ND+1), and R(MD+1+ND+1) to disk file "DATA" as an ASCI file if option "S" is selected.

```
113 IF L$<>"S" THEN 131
114 OPEN "O",#1,"DATA"
115 PRINT #1,STR$(MD)
116 PRINT #1,STR$(ND)
117 FOR I=1 TO ND+1
118 FOR J=1 TO ND+1
119 PRINT #1,STR$(Q#(I,J))
120 NEXT J
121 PRINT #1,STR$(M#(I,1))
122 PRINT #1,STR$(M#(I,2))
123 NEXT I
124 FOR I=1 TO MD+1
125 FOR J=1 TO ND+8
126 PRINT #1,STR$(A#(I,J))
127 NEXT J
128 PRINT #1,STR$(R(I))
129 NEXT I
130 CLOSE #1
```

Load to MD, ND, M#(ND+1,2), A#(MD+1+ND,ND+8), Q#(ND+1,ND+1), and R(MD+1+ND+1) the disk file "DATA" if option "F" is selected.

```
131 IF L$<>"F" THEN 157
132 OPEN "I",#1,"DATA"
133 INPUT #1,X$
134 MD=VAL(X$)
135 INPUT #1,X$
```

```
136 ND=VAL(X$)
137 FOR I=1 TO ND+1
138 FOR J=1 TO ND+1
139 INPUT #1,X$
140 Q#(I,J)=VAL(X$)
141 NEXT J
142 INPUT #1,X$
143 M#(I,1)=VAL(X$)
144 INPUT #1,X$
145 M#(I,2)=VAL(X$)
146 NEXT I
147 FOR I=1 TO MD+1
148 FOR J=1 TO ND+8
149 INPUT #1,X$
150 A#(I,J)=VAL(X$)
151 NEXT J
152 INPUT #1,X$
153 R(I)=VAL(X$)
154 NEXT I
155 CLOSE #1
156 GOTO 67
```

Restarts program for new run if option "N" is selected.

```
157 IF L$="N" THEN RUN
158 GOTO 67
```

Utility Subroutines -- Files UTIL-OPT, UTIL-ERS, UTIL-CON and UTIL-CHX

Same as for critical path method.

Input Subroutines -- Files INPT-OBJ, INPT-CON, INPT-TYP, and INPT-BND

Same as for dual simplex method.

Input Subroutine -- File INPT-QUD

Same as for Beale's method.

Report Subroutine -- Files REPT-SMP

Same as for dual simplex method.

Constraint Keys Subroutine -- ALGR-KEY

Same as for dual simplex method with nonlinear constraints except for the following lines which must be changed to:

```
3024 REM
3026 GOSUB 3100:REM ALGR-BND
3027 FOR I=2 TO ND+1
3028 M#(I,1)=N%(1,I)
3029 M#(I,2)=N%(2,I)
3030 NEXT I
```

Branch and Bound Algorithm Subroutine -- File ALGR-BND

The branch and bound algorithm subroutine (ALGR-BND) is the book-keeping scheme of the branch and bound method. The subroutine follows the logic of the branch and bound tree by executing the following steps.

(1) Solves the integer CP problem using the primal-dual version of Beale's algorithm. The results of the first solution are stored in the first two (2) rows of array N%(2*MX,ND+2) as shown in table 20 as node one (1). The augmented $\overline{B}$ inverse matrix and the augmented $\overline{B}$ matrix at the optimal solution are stored in the array O#(ND+1+ND,ND+1). If the solution is all integer, then set ER=1 and return to the main routine. If the solution is infeasible then set ER=2 and return to the main routine. Otherwise, go to step (2).

(2) Of all the solutions or nodes stored in the array N%(2*MX,ND+2), select the node that has the solution with the minimum value of the objective function as the node to branch on. After the branching is completed, the node is no longer considered for branching again or as a potential solution. Go to step (3).

(3) From the values of the variables in the minimum solution selected above, select the variable, say x, that is closest to an integer value either by rounding up or rounding down. Define two (2) new CP subproblems called nodes in which one has the added constraint

X<=[x] and the other has the constraint X>=[x]+1. Go to step (4).

(4) Using the restart method and the augmented $\overline{B}$ inverse matrix and augmented $\overline{B}$ matrix stored in the array O#(ND+1+ND,ND+1) solve for the optimal solution of the two (2) new nodes and save the results in the node array N%(2*MX,ND+2) by adding the nodes to the bottom of the current list of nodes. If the node array is full then set ER=4 and go to the main routine. Otherwise, go to step (5).

(5) Check each of the two (2) solutions or nodes. If a solution for the node is infeasible then discard the node from further consideration by deleting it from the node array. Go to step (6).

(6) If the values of the variables of the solution are all integer, then compare the value solution with all the other solution values in the node array. If any solution value in the node array is greater than or equal to the all integer solution value, then delete the corresponding node from the node array and also from further consideration. Go to step (7).

(7) If in step (6) all but one (1) solution was deleted from the node array, then set ER=1 and return to the main routine. If the total number of iterations used in the solution of the subproblems to this point is greater than the maximum allowed, then set ER=0 and return to the main routine. Otherwise, go to step (2)

```
3100 REM              * BRANCH AND BOUND ALGORITHM *
3101 REM-----------------------ALGR-BND-------------------------------
```

Sets the deleted node DL count to zero (0), the total iteration

count IT to zero (0), the subproblem iteration count ITR to zero (0), the branch and bound iteration or cycle count CYC to zero (0), and the node line number in the node array to one (1). Sets LL to the number of nodes used for branching at the start of the algorithm. Clears the node array N%(2*MX,ND+2) and sets the first node's upper and lower bounds to those of the starting integer CP problem.

```
3102 DL=0
3103 ITR=0
3104 IT=0
3105 SP=2
3106 CYC=1
3107 LL=1
3108 FOR I=1 TO 2*MX
3109 FOR J=1 TO ND+1
3110 N%(I,J)=0%
3111 NEXT J
3112 NEXT I
3113 FOR I=2 TO ND+1
3114 N%(1,I)=INT(M#(I,1))
3115 N%(2,I)=-INT(-M#(I,2))
3116 NEXT I
```

Checks the iteration count to see if it exceeds the maximum limit IR; and if the limit is exceeded, sets ER=0 and returns to main routine.

```
3117 ER=0
3118 IF IT>IR THEN RETURN
```

Sets LN to the line number in the node array N%(2*MX,ND+1) which holds the bounds for the current subproblem node.

```
3119 LN=2*(CYC-DL)-1
```

Sets up the switch, holding, and upper and lower bound arrays for the Beale's algorithm so that the subproblem can be processed.

```
3120 FOR I=2 TO ND+1
3121 S#(I)=0#
3122 H#(I)=BI#
3123 M#(I,1)=N%(LN,I)
3124 IF N%(LN,I)=BI% THEN M#(I,1)=BI#
3125 M#(I,2)=N%(LN+1,I)
3126 NEXT I
```

Runs Beale's algorithm from the point at the origin if the cycle count is one (1). Otherwise, uses the restart method.

```
3127 IF CYC>1 THEN 3140
3128 GOSUB 3300:REM ALGR-BEA
```

Reinverts the augmented $\overline{B}$ matrix and the objective function and saves the switch array and the augmented $\overline{B}$ inverse matrix and the augmented $\overline{B}$ matrix for the optimal noninteger solution in the array O#(ND+1+ND,ND+2).

The reinversion is done as a precaution since all succeeding subproblems start from this simplex tableau.

```
3129 GOSUB 3900:REM TRAN-RIV
3130 GOSUB 3550:REM TRAN-OBJ
3131 FOR I=1 TO ND+1
3132 O#(1,I)=S#(I)
3133 NEXT I
3134 FOR I=2 TO ND+1+ND
3135 FOR J=1 TO ND+1
3136 O#(I,J)=B#(I,J)
3137 NEXT J
3138 NEXT I
3139 GOTO 3149
```

Processes the subproblem using the restart method.

The restart method starts the subproblem from the optimal point reached in cycle one. If more computer array space where available the restart could be started form the optimal point reached at the most recent node on the branch and bound tree by saving the augmented $\overline{B}$ inverse matrix and the augmented $\overline{B}$ matrix of that node's subproblem.

```
3140 FOR I=1 TO ND+1
3141 S#(I)=O#(1,I)
3142 NEXT I
3143 FOR I=2 TO ND+1+ND
3144 FOR J=1 TO ND+1
3145 B#(I,J)=O#(I,J)
3146 NEXT J
3147 NEXT I
```

The initialization lines of the Beale's algorithm are skipped since the starting point for the algorithm is not the origin but the restart point.

```
3148 GOSUB 3318:REM ALGR-BEA+18
```

Totals the number of iterations for all the subproblems.

```
3149 ITR=ITR+IT
3150 IT=ITR
```

Returns to the main routine if the error code on the first cycle indicates the problem is infeasible or has exceeded the maximum iterations.

```
3151 IF ER<>1 AND CYC=1 THEN RETURN
```

Sets the value of the objective function in the node array to infinity.

```
3152 N%(LN,1)=BI%
```

Lets the objective function value in the node array stay at infinity if the error code indicates the subproblem is infeasible or the best current integer solution is less or equal to the solution of the subproblem.

```
3153 IF ER<>1 OR B#(1,1)>=OB# THEN 3179
```

Checks the values of the variables of the solution to the subproblem to see if they are integer.

```
3154 X=0
3155 FOR I=2 TO ND+1
3156 I%=B#(I,1)
3157 IF ABS(B#(I,1)-I%)>SM# THEN X=1
3158 NEXT I
```

Goes to the solution search if the solution is integer.

```
3159 IF X=0 THEN 3174
```

Finds the variable value that is closest to integer and sets the new upper and lower bounds for the node in the node array as well as rounds up the solution value to an integer value.

```
3160 MM#=BI#
3161 FOR I=2 TO ND+1
3162 I%=CINT(B#(I,1))
3163 A#=ABS(B#(I,1)-I%)
3164 IF A#<=SM# THEN 3170
3165 IF MM#<A# THEN 3170
3166 MM#=A#
3167 N%(LN+1,1)=I
3168 N%(LN,ND+2)=INT(B#(I,1))
3169 N%(LN+1,ND+2)=-INT(-B#(I,1))
```

```
3170 NEXT I
3171 IF -INT(-B#(1,1))>=OB# THEN 3179
3172 N%(LN,1)=-INT(-B#(1,1))
3173 GOTO 3189
```

Sets the best integer solution to the new integer solution.

```
3174 OB#=B#(1,1)
3175 FOR I=2 TO ND+1
3176 X#(I-1)=B#(I,1)
3177 NEXT I
```

Sets ER=1 and returns to the main routine if there is only one (1) node in the branch and bound tree and the solution is integer.

```
3178 IF LL=1 THEN RETURN
3179 IF OB#=BI# THEN 3189
```

Checks the node array N%(2*MX,ND+2) for any nodes with objective function values greater than or equal to the current node's integer value. If the current nodes value is less than or equal, then set the node's solution value to infinity.

```
3180 X=-1
3181 FOR I=1 TO LL STEP 2
3182 IF N%(I,1)=BI% THEN 3185
3183 IF N%(I,1)<OB# THEN 3186
3184 N%(I,1)=BI%
3185 X=X+2
3186 NEXT I
```

Sets ER=1 and returns to the main routine if only one (1) integer solution is left in the node array.

```
3187 ER=1
3188 IF X=LL THEN RETURN
```

Returns to process the other node of the pair on the branch.

Since at each branch two nodes are created, both nodes must be processed before another branch is taken.

```
3189 IF CYC/2=INT(CYC/2) THEN 3214
```

Executes the node compression subroutine if the node array is full.

```
3190 IF CYC-DL+1>=MX THEN GOSUB 3250:REM ALGR-CPR
```

Sets ER=4 and returns to the main routine if the node array cannot be cleared for more node entries

```
3191 ER=4
3192 IF CYC-DL+1>=MX THEN RETURN
```

Finds the node NODE with the minimum value of the objective in the node array.

```
3193 NODE=0
3194 MI#=BI#
3195 FOR I=1 TO LN STEP 2
3196 IF MI#<=N%(I,1) THEN 3199
3197 MI#=N%(I,1)
3198 NODE=I
3199 NEXT I
```

Sets ER=2 and returns to the main routine if no minimum node objective value can be found.

```
3200 ER=2
3201 IF NODE=0 OR DL=CYC THEN RETURN
```

Creates two (2) new nodes for the node tree and transfers the current upper and lower bounds to the new nodes.

```
3202 FOR I=1 TO ND+2
3203 N%(LN+2,I)=N%(NODE,I)
3204 N%(LN+3,I)=N%(NODE+1,I)
3205 N%(LN+4,I)=N%(NODE,I)
3206 N%(LN+5,I)=N%(NODE+1,I)
3207 NEXT I
```

Adds the new upper and lower bound restrictions to the new nodes and returns to process the two (2) new nodes.

```
3208 LL=LN+4
3209 N%(NODE,1)=BI%
3210 NW=N%(NODE+1,1)
3211 N%(LN+2,NW)=N%(NODE,ND+2)
3212 N%(LN+5,NW)=N%(NODE+1,ND+2)
3213 IF N%(LN+2,NW)=0% THEN N%(LN+2,NW)=BI%
3214 CYC=CYC+1
3215 GOTO 3117
```

Node Compressing Subroutine -- File ALGR-CPR

The node array N%(2*MX,ND+2) contains all the nodes from the branch

and bound tree. As nodes are discarded from the tree, the array must be cleared so that the space can be reused. Since the clearing process, or actually a reorganization of the file which compresses the valid nodes to the start of the array, takes time, the array is only compressed when the array is nearly full.

The node compressing subroutine (ALGR-CPR) compresses the discarded nodes from the node array.

```
3250 REM            * NODE COMPRESSING SUBROUTINE *
3251 REM-------------------------ALGR-CPR-------------------------------

3252 X=SP+1
3253 FOR I=SP+1 TO LL STEP SP
3254 IF N%(I,1)<>BI% THEN 3258
3255 REM
3256 DL=DL+1
3257 GOTO 3266
3258 IF X=I THEN 3265
3259 FOR J=1 TO ND+2
3260 N%(X,J)=N%(I,J)
3261 REM
3262 N%(X+1,J)=N%(I+1,J)
3263 REM
3264 NEXT J
3265 X=X+2
3266 NEXT I
3267 FOR I=X TO 2*MX
3268 FOR J=1 TO ND+2
3269 N%(I,J)=0%
3270 REM
3271 NEXT J
3272 NEXT I
3273 LN=2*(CYC-DL)-1
3274 RETURN
```

Beale's Algorithm Subroutine -- File ALGR-BEA

Same as for Beale's method except for the following lines:

```
3342 IF A#(K,ND+2)<>0# THEN GOSUB 4100:REM PAR-DEPL
3344 IF A#(K,ND+3)<>0# THEN GOSUB 4300:REM HYP-DEPL
```

```
3346 IF A#(K,ND+4)<>0# THEN GOSUB 4500:REM SHT-DEPL
```

which must be added so that the convex constraints can be used.

Quadratic Tableau Transformation Subroutine -- File TRAN-QUD

Same as for Beale's method.

Constraint Transformation Subroutine -- File TRAN-CON

Same as for dual simplex method.

Inversion Subroutine -- File TRAN-INV

Same as for dual simplex method.

Reinversion Subroutine -- File TRAN-RIV

Same as for dual simplex method.

Parabolic Subroutines -- Files PAR-DEPA, PAR-DEPL, or PAR-DEPG

Same as for Gomory's method with nonlinear constraints.

Hyperbolic Subroutines -- Files HYP-DEPA, HYP-DEPL, or HYP-DEPG

Same as for Gomory's method with nonlinear constraints.

Hyperbolic Sheet Subroutines -- Files SHT-DEPA, SHT-DEPL, or SHT-DEPG

Same as for Gomory's method with nonlinear constraints.

Program Table of Contents

Table 22 can be used to reconstruct the above computer code from the computer disk and to organize subroutines from previous listings. Since BASIC code is dependent on program line numbers for subroutine branching, the statement numbers must be maintained as listed below.

Table 22. Branch and Bound BASIC program table of contents

File	Program lines	Page	Routines
MAIN-BND	0001-0156	344	Branch and bound method
INPT-OBJ	1200-1220	61	Objective coefficient input subroutine
INPT-CON	1300-1328	63	Constraint coefficient input subroutine
INPT-TYP	1400-1440	65	Constraint type input subroutine
INPT-BND	1500-1523	67	Variable bounds input subroutine
INPT-QUD	1600-1628	166	Quadratic input subroutine
UTIL-OPT	1800-1803	22	Option line subroutine
UTIL-ERS	1850-1853	22	Erase option subroutine
UTIL-CON	1860-1866	22	Continue line subroutine
UTIL-CHX	1870-1882	23	Data check subroutine
REPT-SMP	2200-2224	69	Simplex report subroutine
ALGR-KEY	3000-3057	200	Constraint key subroutine
ALGR-BND	3100-3215	352	Branch and bound algorithm subroutine
ALGR-CPR	3200-3274	357	Node compressing subroutine
ALGR-BEA	3300-3481	172	Beale's algorithm subroutine
TRAN-QUD	3550-3584	180	Quadratic tableau transformation subroutine
TRAN-CON	3600-3633	74	Constraint transformation subroutine
TRAN-INV	3700-3743	76	Basis inversion subroutine
TRAN-RIV	3900-3960	78	Basis reinversion subroutine
PAR-DEPL	4100-4189	311	Deep cut for parabolic subroutine
HYP-DEPL	4300-4385	318	Deep cut for hyperbolic subroutine
SHT-DEPL	4500-4592	325	Deep cut for sheet subroutine

The branch and bound method requires that several lines of the constraint keys subroutine ALGR-KEY be changed to:

```
3024 REM
3026 GOSUB 3100:REM ALGR-BND
3027 FOR I=2 TO ND+1
3028 M#(I,1)=N%(1,I)
3029 M#(I,2)=N%(2,I)
3030 NEXT I
```

and that the following lines be added to Beale's algorithm ALGR-BEA:

```
3342 IF A#(K,ND+2)<>0 THEN GOSUB 4100:REM DEPL-PAR
3344 IF A#(K,ND+3)<>0 THEN GOSUB 4300:REM DEPL-HYP
3346 IF A#(K,ND+4)<>0 THEN GOSUB 4500:REM DEPL-SHT
```

so that the nonlinear constraints can be used.

Branch and Bound Method:
Integer Solutions to Example 11 Minimum Project Man Count Problem

Example 11 was solved using the nonlinear modification of the dual simplex method. The same problem can be solved in all integers using the branch and bound method with Beale's algorithm with nonlinear convex constraints as listed in table 23.

Table 23. Branch and bound method, using deep cuts derived with the line search algorithm, integer solutions to the example 11 minimum project man count problem with parabolic man count functions

Dur.	100	90	80	70	60	50	40	30	20	10
Itr.	75	145	135	263	204	422	49	49	35	46
Sec.	240	456	440	797	649	1249	174	172	147	176
Obj.	16	19	20	22	24	29	34	46	65	132
T1	10	15	15	15	10	10	10	6	5	2
T2	50	35	35	33	32	28	20	14	10	5
T3	80	60	60	50	45	38	30	24	15	8
T4	100	90	80	70	60	50	40	30	20	10
M5	1	1	1	1	1	1	1	1	1	2
M6	1	1	1	1	1	1	1	2	2	4
M7	3	2	2	2	3	3	3	5	6	15
M8	1	2	2	3	2	3	3.9+	5	8	14
M9	2	2	2	3	4	5	5	5	10	17
M10	3	2	3	3	4	5	6	20	12	30
M11	1	2	2	2	2	3	4	4	7	12
M12	2	3	3	3	3	3	4	6	8	16
M13	1	2	2	2	2	3	3	4	6	12
M14	1	2	2	2	2	2	3	4	5	10

Branch and Bound Method:
Integer Solutions to Example 12 Minimum Project Supervision Cost Problem

Another problem that can now be solved in all integers is the minimum project supervision cost problem which was orginally stated and solved as example 12.

It might appear that the problem used in example 13 and solved with

the primal-dual simplex method would be a better candidate for the branch and bound method. But in example 13, the variables representing man counts, which must be held to an integer value, were eliminated from the problem when the quadratic objective function was reduced to nonlinear constraints. This would result in a solution with integer dollars and integer node times, but not integer man counts.

Table 24 is the branch and bound integer solution to example 12.

Table 24. Branch and bound method, using deep cuts derived with the line search algorithm, integer solutions to the example 12 minimum project supervision cost problem with hyperbolic man count and parabolic cost functions

Dur.	100	90	80	70	60	50	40	30	20	10
Itr.	151	153	152	411	812	726	71	155	88	403
Sec.	634	644	621	1488	2784	2433	304	586	410	1597
Obj.	595	595	595	610	653	733	872	1443	2687	10940
T1	15	15	15	15	9.9+	10	10	6	5.+	2
T2	35	35	35	35	28	20	20	14	10	4
T3	52	52	52	52	45	34	30	22	15	7
T4	72	72	72	67	60	50	40	30	20	10
M5	1	1	1	1	1	1	1	1	1	2
M6	1	1	1	1	1	1	1	2	2	4
M7	2	2	2	2	3	3	3	5	5.9+	15
M8	2	2	2	2	3	4	4	5	8	20
M9	3	3	3	3	3	4	5	7	10	17
M10	3	3	3	4	4	4	6	8	12	20
M11	2	2	2	2	2	3	4	5	7	14
M12	3	3	3	3	3	4	4	6	7.9+	20
M13	2	2	2	2	2	3	3	4	6	12
M14	2	2	2	2	2	2	3	4	5	10

Theory of Driebeek's Penalty

In order to gain the greatest flexibility for the branch and bound method, Beale's algorithm was used as the subproblem solution algorithm.

But Beale's algorithm is a primal-dual method in which the value of the objective function can either increase or decrease at each iteration. If only the dual or only the primal algorithm is used, as in the dual simplex method, then the value of the objective function only increases, or only decreases, from one iteration to the next. Driebeek's penalty[53] takes advantage of this fact.

If any variable value is not integer in an optimal solution at a node in the branch and bound tree, then to bring the variable, say X, to an integer value from its optimal value, say x, either an upper or lower bound constraint must be added to the problem as either:

$$x-\underline{/\backslash}x>=X \quad \text{where} \quad \underline{/\backslash}x=x-[x]$$

or:

$$x+\underline{/\backslash}x<=X \quad \text{where} \quad \underline{/\backslash}x=[x]+1-x$$

where the last constraint can be rewritten in the form:

$$-x-\underline{/\backslash}x>=-X$$

Using the matrix operation for the simplex transformation of both of the above constraints:

$$|-(x-\underline{/\backslash}x),\ 0,\ .,\ 1,\ 0\ | * \left|\begin{array}{c|c} 1 & \overline{0} \\ \hline \overline{B}^{-1}\overline{b} & \overline{B}^{-1} \end{array}\right| = |\ -\underline{/\backslash}x,\ \overline{b}'^{-1}\ |$$

and:

$$|-(-x-\underline{/\backslash}x),\ 0,\ .,-1,\ 0\ | * \left|\begin{array}{c|c} 1 & \overline{0} \\ \hline \overline{B}^{-1}\overline{b} & \overline{B}^{-1} \end{array}\right| = |\ \underline{/\backslash}x,\ -\overline{b}'^{-1}\ |$$

The result of the transformation is simply the row of the augmented $\overline{B}$ inverse matrix which corresponds to the noninteger variable with the first column set equal to the change in value required to make the variable integer.

Let tableau 31 represent the augmented B inverse portion of the simplex tableau at the optimal solution. Then, for each noninteger variable value in the tableau, an equation can be written directly from

	1	s1	.	.	.	sn
z=	b_{00}	b_{01}	.	.	.	b_{0n}
x1=	b_{10}	b_{11}	.	.	.	b_{1n}
.	.	.	.	.	.	.
xm=	b_{m0}	b_{m1}	.	.	.	b_{mn}

Tableau 31. Driebeek's method dual simplex tableau

the augmented $\overline{B}$ matrix for the upper bound:

$$b_{10}-[b_{10}]>= b_{11}*s1+.....b_{1n}*sn$$

The minimum "impact" on the objective function that the constraint would have after at least one more iteration of the dual simplex algorithm can be found by selecting the minimum b_{m0} value after pivoting on all the positive coefficients in the corresponding row of the augmented $\overline{B}$ inverse matrix or:

$$z_{1+}=\min\left(b_{00}+\frac{b_{0p}}{b_{1p}}*(b_{10}-[b_{10}])\right) \text{ all positive } b_{11},...b_{1n}$$

Every pivot will result in an increase in the value of the objective function in the dual simplex algorithm.

For the lower bound equation:

$$-([b_{10}]+1-b_{10})>= -(b_{11}*s1+.....b_{1n}*sn)$$

a minimum impact to the objective function can be found by pivoting on the negative elements of the row or:

$$z_{1-}=\min\left(b_{00}+\frac{b_{0p}}{b_{1p}}*(b_{10}-[b_{10}]-1)\right) \text{ all negative } b_{11},\ldots.b_{1n}$$

All the variables must be forced to integer values for the solution value to be integer, so select the variable with the greatest impact on the objective function or:

$$\max\ (\ z_{1+},z_{1-},\ldots\ldots\ldots z_{n+},z_{n-})$$

and set the minimum solution value for the node to this larger value.

With this larger solution value or penalty, the chance that the node will be eliminated sooner is greater allowing the method to converge to an integer solution at a faster rate.

Branch and Bound with Driebeek's Penalty BASIC Code

Driebeek's penalty is almost the same, from the standpoint of code, as the branch and bound method. The only difference between the two (2) methods is in the branch and bound subroutines and the dual simplex subroutine which are called by Driebeek's penalty main routine.

Branch and Bound Main Routine -- File MAIN-BND

Same as for the branch and bound method except the quadratic part of the objective function must be eliminated by deleting lines:

```
102 REM
103 REM
104 REM
```

Utility Subroutines -- Files UTIL-OPT, UTIL-ERS, UTIL-CON and UTIL-CHX

Same as for critical path method method.

Input Subroutines -- Files INPT-OBJ, INPT-CON, INPT-TYP, and INPT-BND

Same as for the dual simplex method.

Report Subroutine -- Files REPT-SMP

Same as for dual simplex method.

Constraint Keys Subroutine -- File ALGR-KEY

Same as for dual simplex method with nonlinear constraints except for the following lines:

```
3024 REM
3026 GOSUB 3100:REM ALGR-DBK
3027 FOR I=2 TO ND+1
3028 M#(I,1)=N%(1,I)
3029 M#(I,2)=N%(2,I)
3030 NEXT I
```

Driebeeks's Penalty Algorithm Subroutine -- File ALGR-DBK

The Driebeek's penalty algorithm subroutine (ALGR-DBK) is the book-keeping scheme of the branch and bound method. The subroutine follows the logic of the branch and bound tree by executing the following steps.

(1) Solve the integer CP problem using the dual simplex algorithm. Store the results of the first solution in the first two (2) rows of array N%(2*MX,ND+2) as shown in table 21 as node one (1). Store the augmented $\overline{B}$ inverse matrix and the augmented $\overline{B}$ matrix at the optimal solution in the array O#(ND+1+ND,ND+1). If the solution is all integer, then set ER=1 and return to the main routine. If the solution is infeasible, then set ER=2 and return to the main routine. Otherwise, go to step (2).

(2) The current optimal solution to the node just calculated can be increased by using Driebeek's penalty as described above. The penalized solution is saved as the solution for the node. Go to step (3).

(3) Of all the solutions or nodes stored in the array N%(2*MX,ND+2), select the node that has the solution with the minimum value of the objective function as the node to branch on. After the branching is completed, the node is no longer considered for branching again. Go to step (4).

(4) From the variables in the minimum solution selected above, select the variable, say X, that is closest to an integer value either by rounding up or rounding down. Define two (2) new CP subproblems, called nodes, in which one has the added constraint X<=[x] and the other has the added constraint X>=[x]+1. Go to step (5).

(5) Using the dual simplex restart method and the augmented $\overline{B}$ inverse matrix and augmented $\overline{B}$ matrix stored in the array O#(ND+1+ND,ND+1) solve for the optimal solution of the two (2) new nodes and save the results in the node array N%(2*MX,ND+2) by adding the node to the bottom of the list of nodes. If the node array is full, then set ER=4 and go to the main routine. Otherwise, go to step (6).

(6) If at any time during the execution of the dual simplex algorithm the value of the objective function exceeds the best current all integer solution, then stop the dual simplex subproblem as if it were infeasible. Go to step (7).

(7) Check each of the two (2) solutions or nodes. If a solution for the node is infeasible, then discard the node from further consideration by deleting it from the node array. Go to step (8).

(8) If the values of the variables of the solution are all integer,

then compare the value of the solution with all the other node solutions remaining in the node array. If any solution in the node array is greater than or equal to the the all integer solution, then delete the node from the node array and also from further consideration. Go to step (9).

(9) If in step (8) all but one (1) node was deleted from the node array, then set ER=1 and return to the main routine. If the total number of iterations used in the solution of the subproblems to this point is greater than the maximum allowed then, set ER=0 and return to the main routine. Otherwise, go to step (2)

```
3100 REM              * DRIEBEEK'S PENALTY ALGORITHM *
3101 REM-------------------------ALGR-DBK-------------------------------
```

Sets the deleted node DL count to zero (0), the total iteration count IT to zero (0), the subproblem iteration count ITR to zero (0), the branch and bound iteration or cycle count CYC to zero (0), and the node line number in the node array to one (1). Sets LL to the number of nodes used for branching at the start of the algorithm. Clears the node array N%(2*MX,ND+2) and sets the first node's upper and lower bounds to those imposed in the starting integer CP problem.

```
3102 DL=0
3103 ITR=0
3104 IT=0
3105 CP=2
3106 CYC=1
3107 LL=1
3108 FOR I=1 TO 2*MX
3109 FOR J=1 TO ND+1
3110 N%(I,J)=0%
3111 NEXT J
3112 NEXT I
3113 FOR I=2 TO ND+1
3114 N%(1,I)=INT(M#(I,1))
3115 N%(2,I)=-INT(-M#(I,2))
3116 NEXT I
```

Checks the iteration count to see if it exceeds the maximum limit

IR; and if the limit is exceeded, sets ER=0 and returns to main routine.

```
3117 ER=0
3118 IF IT>IR THEN RETURN
```

Sets LN to the line number in the node array N%(2*MX,ND+1) which holds the bounds for the current subproblem or node.

```
3119 LN=2*(CYC-DL)-1
```

Sets up the holding, upper and lower bound arrays for the dual simplex algorithm so that the subproblem can be processed.

```
3120 FOR I=2 TO ND+1
3121 H#(I)=BI#
3122 M#(I,1)=N%(LN,I)
3123 IF N%(LN,I)=BI% THEN M#(I,1)=BI#
3124 M#(I,2)=N%(LN+1,I)
3125 NEXT I
```

Runs the dual simplex algorithm from the point at the origin if the cycle count is one (1). Otherwise, uses the restart method.

```
3126 IF CYC>1 THEN 3136
3127 GOSUB 3300:REM ALGR-SMP
3128 IF ER<>1 THEN RETURN
```

Reinverts the augmented $\overline{B}$ inverse matrix and the augmented $\overline{B}$ matrix for the optimal noninteger solution in the array O#(ND+1+ND,ND+2).

The reinversion is done as a precaution since all succeeding subproblems start from this matrix.

```
3129 GOSUB 3900:REM TRAN-RIV
3130 FOR I=1 TO ND+1+ND
3131 FOR J=1 TO ND+1
3132 O#(I,J)=B#(I,J)
3133 NEXT J
3134 NEXT I
3135 GOTO 3142
```

Processes the subproblem using the restart method.

The restart method starts the subproblem from the optimal point reached in cycle one (1). If more computer array space were available, the restart could be started from the optimal point reached at the most recent node on the branch and bound tree by saving the augmented $\overline{B}$ inverse matrix and the augmented $\overline{B}$ matrix of that node's subproblem.

```
3136 FOR I=1 TO ND+1+ND
3137 FOR J=1 TO ND+1
3138 B#(I,J)=O#(I,J)
3139 NEXT J
3140 NEXT I
```

The intialization lines of the dual simplex algorithm are skipped since the starting point for the algorithm is not the origin but the restart point.

```
3141 GOSUB 3313:REM ALGR-SMP+13
```

Totals the number of iterations for all the subproblems.

```
3142 ITR=ITR+IT
3143 IT=ITR
```

Returns to the main routine if the error code on the first cycle indicates the subproblem is infeasible or has exceeded the maximum iterations.

```
3144 IF ER<>1 AND CYC=1 THEN RETURN
```

Sets the value of the objective function in the node array to infinity.

```
3145 N%(LN,1)=BI%
```

Lets the objective function value in the node array stay at infinity if the error code indicates the subproblem is infeasible or the best current integer solution is less or equal to the solution of the subproblem.

```
3146 IF ER<>1 OR B#(1,1)>=OB# THEN 3183
```

Checks the values of the variables of the solution to the subproblem to see if they are integer.

```
3147 X=0
3148 FOR I=2 TO ND+1
3149 I%=CINT(B#(I,1))
3150 IF ABS(B#(I,1)-I%)>SM# THEN X=1
3151 NEXT I
```

Goes to the solution search if the solution is integer.

```
3152 IF X=0 THEN 3178
```

Increases the value of the objective function using Driebeek's penalty. Finds the variable value that is closest to integer and sets the new upper and lower bounds for the node in the node array as well as the solution value rounded up to an integer value.

```
3153 MA#=0#
3154 NW=0
3155 MM#=BI#
3156 FOR I=2 TO ND+1
3157 I%=CINT(B#(I,1))
3158 A#=ABS(B#(I,1)-I%)
3159 IF A#<=SM# THEN 3174
3160 IF MM#<A# THEN 3165
3161 MM#=A#
3162 N%(LN+1,1)=I
3163 N%(LN,ND+2)=INT(B#(I,1))
3164 N%(LN+1,ND+2)=-INT(-B#(I,1))
3165 MI#=BI#
3166 FOR J=2 TO ND+1
3167 IF ABS(B#(I,J))<SM# THEN 3172
3168 IF B#(I,J)>0# THEN A#=B#(I,1)+INT(-B#(I,1))
3169 IF B#(I,J)<0# THEN A#=-B#(I,1)+INT(B#(I,1))
3170 A#=(-A#*B#(1,J))/ABS(B#(I,J))
3171 IF MI#>A# THEN MI#=A#
3172 NEXT J
3173 IF MA#<MI# THEN MA#=MI#
3174 NEXT I
3175 IF -INT(-B#(1,1)-MA#)>=OB# THEN 3183
3176 N%(LN,1)=-INT(-B#(1,1)-MA#)
3177 GOTO 3193
```

Sets the best integer solution to the new integer solution.

```
3178 OB#=B#(1,1)
3179 FOR I=2 TO ND+1
3180 X#(I-1)=B#(I,1)
3181 NEXT I
```

If there is only one (1) node in the branch and bound tree and the solution is integer, returns to the main routine.

```
3182 IF LL=1 THEN RETURN
3183 IF OB#=BI# THEN 3193
```

Checks the node array N%(2*MX,ND+2) for any nodes with objective function values greater than or equal to the current node's integer value. If the current node's value is less than or equal, then set the other node's objective function value to infinity.

```
3184 X=-1
3185 FOR I=1 TO LL STEP 2
3186 IF N%(I,1)=BI% THEN 3189
3187 IF N%(I,1)<OB# THEN 3190
3188 N%(I,1)=BI%
3189 X=X+2
3190 NEXT I
```

Sets ER=1 and returns to the main routine if only one (1) integer solution is left in the node array.

```
3191 ER=1
3192 IF X=LL THEN RETURN
```

Returns to process the other node of the pair on the branch.

Since at each branch two (2) nodes are created, both nodes must be processed before another branch is taken.

```
3193 IF CYC/2=INT(CYC/2) THEN 3218
```

Executes the node compression subroutine if the node array is full.

```
3194 IF CYC-DL+1>=MX THEN GOSUB 3250:REM ALGR-CPR
```

Sets ER=4 and returns to the main routine if the node array cannot be cleared for more active nodes.

```
3195 ER=4
3196 IF CYC-DL+1>=MX THEN RETURN
```

Finds the node NODE with the minimum solution in the node array.

```
3197 NODE=0
3198 MI#=BI#
3199 FOR I=1 TO LN STEP 2
3200 IF MI#<=N%(I,1) THEN 3203
3201 MI#=N%(I,1)
3202 NODE=I
3203 NEXT I
```

Sets ER=2 and returns to the main routine if no minimum node solution can be found.

```
3204 ER=2
3205 IF NODE=0 OR DL=CYC THEN RETURN
```

Creates two (2) new nodes for the branch and bound tree and

transfers the current upper and lower bounds to the new nodes.

```
3206 FOR I=1 TO ND+2
3207 N%(LN+2,I)=N%(NODE,I)
3208 N%(LN+3,I)=N%(NODE+1,I)
3209 N%(LN+4,I)=N%(NODE,I)
3210 N%(LN+5,I)=N%(NODE+1,I)
3211 NEXT I
```

Adds the new upper and lower bound restrictions to the new nodes and returns to process the two (2) new nodes.

```
3212 LL=LN+4
3213 N%(NODE,1)=BI%
3214 NW=N%(NODE+1,1)
3215 N%(LN+2,NW)=N%(NODE,ND+2)
3216 N%(LN+5,NW)=N%(NODE+1,ND+2)
3217 IF N%(LN+2,NW)=0 THEN N%(LN+2,NW)=BI%
3218 CYC=CYC+1
3219 GOTO 3117
```

Node Compressing Subroutine -- File ALGR-CPR

Same as for branch and bound method.

Dual Simplex Algorithm Subroutine -- File ALGR-SMP

Same as for dual simplex method except the following code must be modified for nonlinear convex constraints by adding:

```
3333 IF A#(K,ND+2)<>0# THEN GOSUB 4100:REM PAR-DEPL
3335 IF A#(K,ND+3)<>0# THEN GOSUB 4300:REM HYP-DEPL
3337 IF A#(K,ND+4)<>0# THEN GOSUB 4500:REM SHT-DEPL
3369 IF B#(1,1)>=0B# THEN RETURN
```

Constraint Transformation Subroutine -- File TRAN-CON

Same as for dual simplex method.

Inversion Subroutine -- File TRAN-INV

Same as for dual simplex method.

Reinversion Subroutine -- File TRAN-RIV

Same as for dual simplex method.

Parabolic Subroutines -- Files PAR-DEPA, PAR-DEPL, or PAR-DEPG

Same as for Gomory's method with nonlinear constraints.

Hyperbolic Subroutines -- Files HYP-DEPA, HYP-DEPL, or HYP-DEPG

Same as for Gomory's method with nonlinear constraints.

Hyperbolic Sheet Subroutines -- Files SHT-DEPA, SHT-DEPL, or SHT-DEPG

Same as for Gomory's method with nonlinear constraints.

Program Table of Contents

Table 25 can be used to reconstruct the above computer code from

Table 25. Branch and Bound (with Driebeek's penalty) BASIC program table of contents

File	Program lines	Page	Routines
MAIN-BND	0001-0136	344	Branch and bound method
INPT-OBJ	1200-1220	61	Objective coefficient input subroutine
INPT-CON	1300-1328	63	Constraint coefficient input subroutine
INPT-TYP	1400-1440	65	Constraint type input subroutine
INPT-BND	1500-1523	67	Variable bounds input subroutine
UTIL-OPT	1800-1803	22	Option line subroutine
UTIL-ERS	1850-1853	22	Erase option subroutine
UTIL-CON	1860-1866	22	Continue line subroutine
UTIL-CHX	1870-1882	23	Data check subroutine
REPT-SMP	2100-2125	69	Simplex report subroutine
ALGR-KEY	3000-3058	200	Constraint key subroutine
ALGR-DBK	3100-3219	367	Driebeek's penalty algorithm subroutine
ALGR-CPR	3200-3274	357	Node compressing subroutine
ALGR-SMP	3300-3499	70	Dual simplex algorithm subroutine
TRAN-CON	3600-3633	74	Constraint transformation subroutine
TRAN-INV	3700-3745	76	Basis inversion subroutine
TRAN-RIV	3900-3964	78	Basis reinversion subroutine
DEPL-PAR	4100-4199	311	Deep cut for parabolic subroutine
DEPL-HYP	4300-4399	318	Deep cut for hyperbolic subroutine
DEPL-SHT	4500-4599	325	Deep cut for sheet subroutine

the computer disk and to organize subroutines from previous listings. Since BASIC code is dependent on program line numbers for subroutine branching, the statement numbers must be maintained as listed below.

The main routine for the branch and bound method MAIN-BND can no longer accept a quadratic objective function if the dual simplex method is used to solve the subproblems. To eliminate this option, delete:

```
102 REM
103 REM
104 REM
```

from MAIN-BND. The constraint key subroutine must be changed as before in the branch and bound method with the following lines:

```
3024 REM
3026 GOSUB 3100:REM ALGR-DBK
3027 FOR I=2 TO ND+1
3028 M#(I,1)=N%(1,I)
3029 M#(I,2)=N%(2,I)
3030 NEXT I
```

and the dual simplex algorithm ALGR-SMP must be modified for nonlinear convex constraints by adding:

```
3333 IF A#(K,ND+2)<>0# THEN GOSUB 4100:REM PAR-DEPL
3335 IF A#(K,ND+3)<>0# THEN GOSUB 4300:REM HYP-DEPL
3337 IF A#(K,ND+4)<>0# THEN GOSUB 4500:REM SHT-DEPL
3369 IF B#(1,1)>=0B# THEN RETURN
```

Branch and Bound Method with Driebeek's Penalty: Integer Solutions to Example 11 Minimum Project Man Count Problem

Using the nonlinear modification of the dual simplex method, the ten (10) activity network minimum project man count problem was solved in example 11. The same problem can be solved in all integers using the branch and bound method with Driebeek's penalty as listed in table 26.

Table 26. Branch and bound method with Driebeek's penalty, using deep cuts derived with the line search algorithm, integer solutions to the example 11 minimum project man count problem with hyperbolic man count functions

Dur.	100	90	80	70	60	50	40	30	20	10
Itr.	75	156	100	235	168	385	39	45	28	40
Sec.	129	240	174	332	250	484	95	103	94	115
Obj.	16	19	20	22	24	29	34	46	65	132
T1	10	15	15	15	9.9+	10	10	6	5	2
T2	50	35	35	33	32	20	20	14	10	5
T3	80	60	60	50	45	37	30	24	15	8
T4	100	90	80	70	60	50	40	30	20	10
M5	1	1	1	1	1	1	1	1	1	2
M6	1	1	1	1	1	1	1	2	2	4
M7	3	2	2	2	3	3	3	5	6	15
M8	1	2	2	3	2	4	3.9+	5	8	14
M9	2	2	2	3	4	3	5	5	10	17
M10	3	2	3	3	4	5	6	10	12	30
M11	1	2	2	2	2	3	4	4	7	12
M12	2	3	3	3	3	4	4	6	8	16
M13	1	2	2	2	2	3	3	4	6	12
M14	1	2	2	2	2	2	3	4	5	10

SECTION III. CURVE FITTING PROBLEM AND REAL NUMBER SOLUTIONS

All the methods presented so far have solved for a project network schedule with either a minimum total costs, a minimum total man count, or a minimum total supervision cost. With each of these minimum solutions there is also a cumulative cost or mandays versus time curve.

In many cases, the cumulative curve has been determined by outside factors, and the objective is to find a project schedule which minimizes the deviation from the predetermined curve. The most common of these predetermined cumulative curves is the cumulative mandays curve which reflects the availability of labor at a project site and the cumulative cash flow curve which reflects limits project funding.

Section III presents a model to determine the deviations of a cumulative project mandays curve from a predetermined curve and a method to solve the minimum deviation problem.

MINIMUM DEVIATION FROM CUMULATIVE MANDAYS CURVE

The manday curve fitting problem can be can be expressed as a a CP problem in which the product of pairs of variables are restricted to zero (0).

First define the following variables as:

S_i	- Start time of activity i
dur.	- Project duration
m_i	- Man count on activity i
d_i	- Duration of activity i
sp_i	- End of span i
$mandays_i$	- Mandays in span i

Then, approximate a cumulative mandays curve by a step function as shown in figure 83.

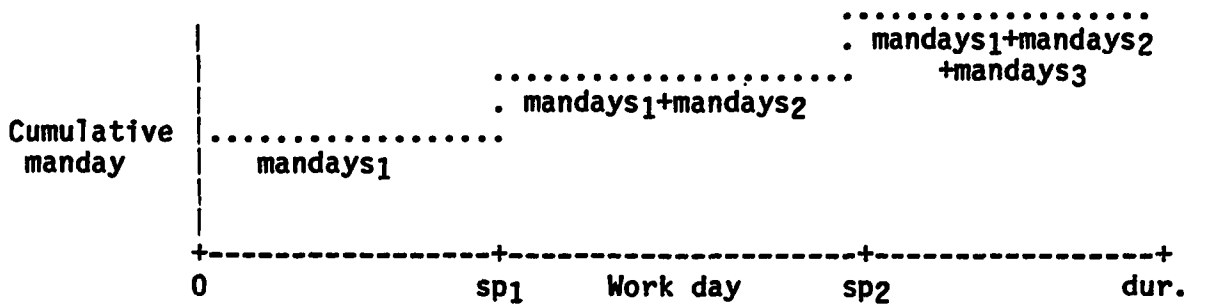


Figure 83. Cumulative mandays curve step function

An activity, as in the CPM method, can be represented as an arrow between two (2) nodes where S_1 is the start time for the activity #1 and S2 is start time of the following activity. If activity #1 is placed on

Activity #1
S_1 ----------> S_2
m_1 - Man count
d_1 - Duration
S_1 - Early start

on the time scale of the cumulative mandays step function which has been

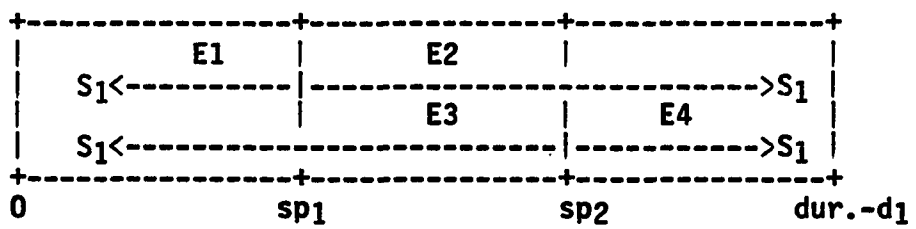
divided into the time spans (0->sp_1->sp_2->dur.), then the number of time increments that the early start of activity #1 is either precedes (E1,E3) or succeeds (E2,E4) each spans ending (sp_1,sp_2) point is given by:

$$S_1+E1-E2=sp_1$$
$$S_1+E3-E4=sp_2$$

with the restriction that:

$$E1*E2=0$$
$$E3*E4=0$$

Or graphically:



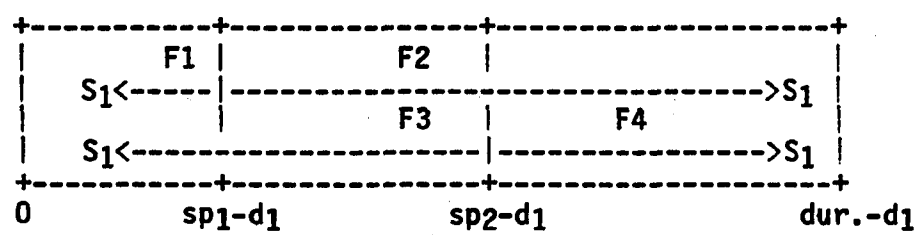
The same formulation can be used to find the number of time increments the early finish of activity #1 either precedes or succeeds each span ending point:

$$S_1+d_1+F1-F2=sp_1$$
$$S_1+d_2+F3-F4=sp_2$$

with the restriction that:

$$F1*F2=0$$
$$F3*F4=0$$

Or graphically:



Adding the schedule restriction of:

$$S_1<Dur.-d_1$$

the deviations V1,V2,V3,V4 from the mandays step function are then:

m_1*(E1-F1)+V1-V2=mandays$_1$
m_1*(E3-F3)+V3-V4=mandays$_1$+mandays$_2$

With these components, a CP problem can be constructed to minimize the deviation of the activity mandays from the cumulative mandays curve.

minimize V1+V2+V3+V4

subject to:
S_1+E1-E2=sp_1
S_1+E3-E4=sp_2
S_1+d_1+F1-F2=sp_1
S_1+d_1+F3-F4=sp_2

S_1<Dur.-d_1

m_1*(E1-F1)+V1-V2=mandays$_1$
m_1*(E3-F3)+V3-V4=mandays$_1$+mandays$_2$

E1*E2=0
E3*E4=0
F1*F2=0
F3*F4=0

S_1,E1,E2,E3,E4,F1,F2,F3,F4,V1,V2,V3,V4>=0

Theory of Restricted Pairs Method

In the problem described above, pairs of variables are restricted so that in any feasible solution at least one (1) member of the pair has the value of zero (0). In the branch and bound method the values of the variables were restricted in a similar manner to the integer values. By utilizing the same book keeping scheme of the branch and bound method, the restricted pairs method solves a series of CP subproblems with the dual simplex method in which tighter and tighter restrictions in the form of zero (0) upper bounded variables are added to the subproblem constraint sets until a minimum optimal solution is reached for which all the restricted pairs have at least one zero (0) member.

Restricted Pairs Method BASIC Code

Restricted Pairs Main Routine -- MAIN-RST

The restricted pairs main calling routine (MAIN-RST) dimensions the twelve (12) data arrays; writes the option menu to the screen as shown in figure 84; calls the utility subroutines UTIL-OPT, UTIL-ERS,

```
                RESTRICTED PAIRS METHOD

NUMBER OF CONSTRAINTS              34
NUMBER OF VARIABLES                60
MAXIMUM ITERATIONS                 1000
NUMBER OF RESTRICTED PAIRS         20

                M-RETURN TO MENU

          O-OBJECTIVE COEFFICIENTS
          A-CONSTRAINT COEFFICIENTS
          C-CONSTRAINT TYPES
          B-BOUNDED VARIABLES
          R-RESTRICTED PAIRS
          U-EXECUTE ALGORITHM
          R-REPORT LISTING
          S-SAVE  F-FETCH
          N-NEW PROBLEM

     OPTION ?
```

Figure 84. Restricted pairs main menu screen

UTIL-CONS, UTIL-CHX; calls the data input and output subroutines INPT-OBJ, INPT-CON, INPT-TYP, INPT-BND, and REPT-SMP; calls and times the processing algorithms ALGR-KEY, ALGR-RST, and ALGR-SMP; and saves and fetches the input data to the disk file "DATA".

```
1 REM                  *RESTRICTED PAIRS METHOD*
2 REM-----------------------------MAIN-PRS-------------------------
3 REM     BI# - MACHINE INFINITE
4 REM     CO  - PIVOT COLUMN FOR SIMPLEX TRANSFORMATION
5 REM     ER  - ERROR KEY
6 REM     IR  - MAXIMUM NUMBER OF ITERATIONS
```

```
7 REM      MD  - NUMBER OF CONSTRAINTS
8 REM      ND  - NUMBER OF VARIABLES
9 REM      MX  - NUMBER OF NODES IN NODES MATRIX
10 REM     OB# - VALUE OF THE OBJECTIVE FUNCTION
11 REM     RO  - PIVOT ROW FOR SIMPLEX TRANSFORMATION
12 REM     SM# - MACHINE ZERO
13 REM     A#(MD+1,ND+8) - ORIGINAL DATA AND KEYS
14 REM     B#(2*ND+1,ND+1) - SIMPLEX MATRIX
15 REM     E(BD) - RESTRICTED PAIRS MATRIX
16 REM     M#(ND+1,2) - LOWER AND UPPER BOUNDS OF VARIABLES
17 REM     N#(2*MX,BD+2) - NODES MATRIX
18 REM     O#(2*ND+1,ND+1) - UNRESTRICTED SOLUTION BASIS AND INVERSE
19 REM     P#(ND+1) - CURRENT PIVOT ROW
20 REM     R(MD+1) - CONSTRAINT TYPES (1-">=",0-"=",-1-"<=")
21 REM     S#(ND+1) - WORK SPACE
22 REM     T#(ND+1,ND+1) - REINVERSION WORK SPACE
23 REM     X#(ND) - SOLUTION VECTOR
24 REM ------------------------------------------------------------
```

Sets MD to the default number of constraints in the restricted pairs problem and ND to the number of variables. Sets IN to the number of iterations before a reinversion of the augmented $\overline{B}$ matrix. Sets the default limit IR on the maximum number of iterations to one thousand. Sets BI# to a number considered machine infinity and SM# to a number considered machine zero. Sets MX to the maximum number of active nodes allowed on the branch and bound tree.

```
25 MD=0
26 ND=0
27 IN=20
28 IR=1000
29 BI#=1E+10
30 SM#=1E-10
31 MX=40
```

Prompts and reads from the keyboard MD the number of constraints in the CP problem, ND the number of variables, IR the maximum number of iterations allowed before the restricted pairs method stopped, and BD to the number of restricted variables in the restricted pairs.

```
32 CLS
33 LOCATE 1,5:PRINT "RESTRICTED PAIRS METHOD"
34 LOCATE 3,1:PRINT "NUMBER OF CONSTRAINTS":LOCATE 3,31:INPUT "",L$
35 GOSUB 1870:REM UTIL-CHX
36 IF Z#<>BI# THEN MD=Z#
37 LOCATE 3,30:PRINT MD,"    ":LOCATE 4,1:PRINT "NUMBER OF VARIABLES"
   :LOCATE 4,31:INPUT "",L$
38 GOSUB 1870:REM UTIL-CHX
```

```
39 IF Z#<>BI# THEN ND=Z#
40 LOCATE 4,30:PRINT ND,"    ":LOCATE 5,1:PRINT "MAXIMUM ITERATIONS"
   :LOCATE 5,31:INPUT "",L$
41 GOSUB 1870:REM UTIL-CHX
42 IF Z#<>BI# THEN IR=Z#
43 LOCATE 5,30:PRINT IR,"    ":LOCATE 6,1:PRINT "NUMBER OF RESTRICTED PA
   IRS":LOCATE 6,31:INPUT "",L$
44 GOSUB 1870:REM UTIL-CHX
45 IF Z#<>BI# THEN BD=2*Z#
46 LOCATE 6,30:PRINT BD/2,"    "
```

Dimensions the A#(MD+1,ND+8) array which contains the constraint coefficients. Dimensions the augmented $\overline{B}$ matrix and augmented $\overline{B}$ matrix array B#(2*ND+1,ND+1). Dimensions the restricted pairs array E(BD). Dimensions the holding array H#(ND+1), the branch and bound node array N#(MX,BD+2), the augmented $\overline{B}$ inverse and augmented $\overline{B}$ matrix of the optimal solution of the unrestricted CP problem array O#(2*ND+1,ND+1), the the upper and lower bound array M#(ND+1), the pivot row P#(ND+2), the constraint type array R(MD+1), the reinversion working array T#(ND+1,ND+1), and the solution array X#(ND).

The node array for the restricted pairs method is different from the node array in the branch and bound method. The array N#(MX,BD+2) is noninteger. Each nodes information is contained on a single row of the array in which the first entry in the row is the value of the objective function followed by a pair of entries for each restricted pair containing a set of switches (set at either zero (0) for a variable not restricted, one (1) for a variable restricted to zero (0), or minus one (-1) for a variable allowed to be free) and the number of the variable to be restricted for the next set of subproblems.

Table 27. Restricted pairs node table

Line	Node value	Restricted pairs		Variable number
1	Current objective value	Switch	Switch	New bound variable

```
47 DIM A#(MD+1,ND+8)
48 DIM B#(2*ND+1,ND+1)
49 DIM E(BD)
50 DIM M#(ND+1,2)
51 DIM N#(MX,BD+2)
52 DIM O#(2*ND+1,ND+1)
53 DIM P#(ND+1)
```

```
54 DIM H#(ND+1)
55 DIM R(MD+1)
56 DIM S#(ND+1)
57 DIM T#(ND+1,ND+1)
58 DIM X#(ND)
```

Initializes the constraint type array to all greater than or equals.

```
59 FOR I=2 TO MD+1
60 R(I)=1
61 NEXT I
```

Prints the option menu to the screen; calls the option line routine UTIL-OPT; pauses for the entry of "M", "O", "A", "C", "B", "U", "R", "S", "F", "N" for the option variable L$.

```
62 LOCATE 8,12:PRINT       "M-RETURN TO MENU"
63 LOCATE 10,7:PRINT    "O-OBJECTIVE COEFFICIENTS"
64 LOCATE 11,7:PRINT    "C-CONSTRAINT VALUES"
65 LOCATE 12,7:PRINT    "A-A MATRIX COEFFICIENTS"
66 LOCATE 13,7:PRINT    "B-UPPER BOUNDED VARIABLES"
67 LOCATE 14,7:PRINT    "P-RESTRICTED PAIRS"
68 LOCATE 15,7:PRINT    "U-EXECUTE ALGORITHM"
69 LOCATE 16,7:PRINT    "R-REPORT LISTING"
70 LOCATE 17,7:PRINT    "N-NEW PROBLEM"
71 LOCATE 18,7:PRINT    "S-SAVE  F-FETCH"
72 GOSUB 1800:REM UTIL-OPT
73 LOCATE 21,8:INPUT "",L$
74 CLS
```

Calls either the objective function input subroutine INPT-OBJ, the constraint input subroutine INPT-CON, the constraint type input subroutine INPT-TYP, the upper and lower bounds input subroutine INPT-BND, the processing subroutine ALGR-KEY, the report subroutine REPT-SMP, or the restricted pairs input subroutine INPT-PRS based on the option variable L$.

```
75 IF L$<>"O" THEN 78
76 GOSUB 1200:REM INPT-OBJ
77 GOTO 74
78 IF L$<>"A" THEN 81
79 GOSUB 1300:REM INPT-CON
80 GOTO 74
81 IF L$<>"C" THEN 84
82 GOSUB 1400:REM INPT-TYP
83 GOTO 74
84 IF L$<>"B" THEN 87
85 GOSUB 1500:REM INPT-BND
```

```
86 GOTO 74
87 IF L$<>"U" THEN 92
88 TM=3600*VAL(MID$(TIME$,1,2))+60*VAL(MID$(TIME$,4,2))+VAL(MID$(TIME$,7
   ,2))
89 GOSUB 3000:REM ALGR-KEY
90 TM=3600*VAL(MID$(TIME$,1,2))+60*VAL(MID$(TIME$,4,2))+VAL(MID$(TIME$,7
   ,2))-TM
91 GOTO 62
92 IF L$<>"R" THEN 95
93 GOSUB 2200:REM REPT-SMP
94 GOTO 74
95 IF L$<>"P" THEN 98
96 GOSUB 1700:REM INPT-PRS
97 GOTO 74
```

Saves the content of MD, ND, M#(ND+1,2), A#(MD+1,ND+8), and R(ND+1) to disk file "DATA" as an ASCI file if option "S" is selected.

```
98 IF L$<>"S" THEN 116
99 OPEN "O",#1,"DATA"
100 PRINT #1,STR$(MD)
101 PRINT #1,STR$(ND)
102 FOR I=1 TO ND+1
103 FOR J=1 TO ND+1
104 PRINT #1,""
105 NEXT J
106 PRINT #1,STR$(M#(I,1))
107 PRINT #1,STR$(M#(I,2))
108 NEXT I
109 FOR I=1 TO MD+1
110 FOR J=1 TO ND+8
111 PRINT #1,STR$(A#(I,J))
112 NEXT J
113 PRINT #1,STR$(R(I))
114 NEXT I
115 CLOSE #1
```

Loads to MD, ND, M#(ND+1,2), A#(MD+1,ND+8), and R(ND+1) the disk file "DATA" if option "F" is selected.

```
116 IF L$<>"F" THEN 140
117 OPEN "I",#1,"DATA"
118 INPUT #1,X$
119 MD=VAL(X$)
120 INPUT #1,X$
121 ND=VAL(X$)
122 FOR I=1 TO ND+1
123 FOR J=1 TO ND+1
```

```
124 INPUT #1,X$
125 NEXT J
126 INPUT #1,X$
127 M#(I,1)=VAL(X$)
128 INPUT #1,X$
129 M#(I,2)=VAL(X$)
130 NEXT I
131 FOR I=1 TO MD+1
132 FOR J=1 TO ND+8
133 INPUT #1,X$
134 A#(I,J)=VAL(X$)
135 NEXT J
136 INPUT #1,X$
137 R(I)=VAL(X$)
138 NEXT I
139 CLOSE #1
```

Restarts program for new run if option "N" is selected.

```
140 IF L$="N" THEN RUN
141 GOTO 62
```

Utility Subroutines -- Files UTIL-OPT, UTIL-ERS, UTIL-CON, and UTIL-CHX

Same as for critical path method.

Input Subroutines -- Files INPT-OBJ, INPT-CON, INPT-TYP, and INPT-BND

Same as for dual simplex method.

Restricted Pairs Input Subroutine -- File INPT-PRS.

The restricted pairs subroutine (INPT-PRS) is the interactive input of the restricted pairs array E(BD). To reach the input screen from the main menu, type "R" as the OPTION ?. The screen, as shown in figure 85, consists of a pair NO. column, a VARIABLE column, and a second VARIABLE column. When the screen is first entered the cursor "_" will be located in the NO. column. When a restricted pair identification number or a blank is typed and entered, the cursor will move to the first variable column so that the variable number of the first variable of the

```
        RESTRICTED PAIRS
NO.     VARIABLE    VARIABLE
1       3           4
2       5           6
3       7           8
4       9           10
5       11          12
6       13          14
7       15          16
8       17          18

OPTION ?
```

Figure 85. Restricted pairs input screen

restricted pair can be typed and entered. The cursor will then move to the second variable column so that the second variable number can be typed and entered.

If an error is made when entering a restricted pair, then the pair identification number is retyped and the data reentered. To select other options and leave the current screen, the option letter is typed in the NO. column and entered.

```
1700 REM                * RESTRICTED PAIRS INPUT *
1701 REM-------------------------INPT-PRS-------------------------------

1702 H=0
1703 G=2
1704 LOCATE 1,11:PRINT "RESTRICTED PAIRS":LOCATE 2,7:PRINT     "NO.
     VARIABLE  VARIABLE"
1705 G=G+1
1706 H=H+1
1707 GOSUB 1800:REM UTIL-OPT
1708 LOCATE G,7:INPUT "",L$
1709 GOSUB 1870:REM UTIL-CHX
1710 IF L$<>"" AND Z#=BI# THEN RETURN
1711 IF Z#<>BI# THEN H=Z#
1712 IF 2#*H>BD OR H<=0 THEN 1707
1713 GOSUB 1850:REM UTIL-ERS
1714 LOCATE G,6:PRINT H,"  ":LOCATE G,12:PRINT E((H*2)-1):LOCATE G,13
     :INPUT "",L$
```

```
1715 GOSUB 1870:REM UTIL-CHX
1716 IF Z#<>BI# AND E(H*2)<>Z# THEN E((H*2)-1)=Z#
1717 LOCATE G,12:PRINT E((H*2)-1),"   ":LOCATE G,22:PRINT E(H*2)
     :LOCATE G,23:INPUT "",L$
1718 GOSUB 1870:REM UTIL-CHX
1719 IF Z#<>BI# AND E((H*2)-1)<>Z# THEN E(H*2)=Z#
1720 LOCATE G,22:PRINT E(H*2),"   "
1721 IF G<18 THEN 1705
1722 GOSUB 1860:REM UTIL-CON
1723 GOTO 1706
```

Constraint Keys Subroutine -- File ALGR-KEY

Same as for dual simplex method with nonlinear constraints except for the following lines:

```
3024 REM
3026 GOSUB 3100:REM ALGR-PRS
3027 FOR I=2 TO ND+1
3028 M#(I,1)=N%(1,I)
3029 M#(I,2)=N%(2,I)
3030 NEXT I
```

Restricted Pairs Algorithm Subroutine -- File ALGR-PRS

The restricted pairs algorithm subroutine (ALGR-PRS) is the bookkeeping scheme of the restricted pairs method. The subroutine follows the logic of a branch and bound tree by executing the following steps.

(1) Solve the CP problem of the restricted pairs method in which no restriction is placed on the restricted pairs using the dual simplex algorithm. Store the results of the first solution in the first row of array N#(MX,ND+2) as shown in table 27 as node one (1). Store the augmented $\overline{B}$ inverse matrix and the augmented $\overline{B}$ matrix of the optimal solution in the array O#(ND+1+ND,ND+1). If the solution satisfies the restricted pairs restrictions, then set ER=1 and return to the main routine. If the solution is infeasible, then

set ER=2 and return to the main routine. Otherwise, go to step (2).

(2) Using Dreibeek's penalty, increase the value of the optimal value of the objective function for the current node and store the penalized solution in the node array. Go to step (3).

(3) Of all the solutions or nodes stored in the array N#(MX,ND+2), select the node that has the solution with the minimum value of the objective function as the node to branch on. After the branching is completed, the node is no longer considered for branching in following iterations. Go to step (4).

(4) From the values of the variables in the minimum solution selected above, select the restricted pair, say x and y, which has a product is not zero (0) and which has a member closest to zero (0). Define two (2) new CP subproblems called nodes in which one has the added constraint X=0 and the other has the added constraint Y=0. Go to step (5).

(5) Using the restart method and the augmented $\overline{B}$ inverse matrix and augmented $\overline{B}$ matrix stored in the array O#(ND+1+ND,ND+1), solve for the optimal solution of the two (2) new nodes and save the results in the node array N#(MX,ND+2) by adding the nodes to the bottom of the current set of nodes. If the node array is full, set ER=4 and go to the main routine. Otherwise, go to step (6).

(6) Check each of the two (2) solutions or nodes. If a solution for the node is infeasible then discard the node from further consideration by deleting it from the node array. Go to step (7).

(7) If the restricted pairs of the solution are all satisfied, then compare the value of the objective function with all the other values remaining in the node array. If any objective function value in the node array is greater than or equal to the current objective function value, then delete the node from the node array and from further consideration. Go to step (8).

(8) If in step (7) all but one (1) solution was deleted from the node array, then set ER=1 and return to the main routine. If the total number of iterations used in the solution of the subproblems to this point is greater than the maximum allowed, then set ER=0 and return to the main routine. Otherwise, go to step (2).

```
3100 REM              * RESTRICTED PAIRS ALGORITHM *
3101 REM-------------------------ALGR-PRS-------------------------------
```

Sets the deleted node count DL to zero (0), the total iteration count IT to zero (0), the subproblem iteration count ITR to zero (0), the branch and bound iteration or cycle count CYC to zero (0), and the node line number in the node array to one (1). Sets LL to the number of nodes used for branching at the start of the algorithm. Clears the node array N#(MX,ND+2) and sets the first node's upper and lower bounds to those imposed in the starting restricted pairs CP problem.

```
3102 DL=0
3103 ITR=0
3104 IT=0
3105 SP=1
3106 OB#=BI#
3107 CYC=1
3108 LL=1
3109 FOR I=1 TO MX
3110 FOR J=1 TO BD+1
3111 N#(I,J)=0#
3112 NEXT J
3113 NEXT I
```

Checks the iteration count to see if it exceeds the maximum limit IR; and if the limit is exceeded, sets ER=0 and returns to main routine.

```
3114 ER=0
3115 IF IT>IR THEN RETURN
```

Sets LN to the line number in the node array N%(2*MX,ND+1) which holds the bounds for the current subproblem or node.

```
3116 LN=CYC-DL
```

Sets up the upper and lower bound arrays for the dual simplex algorithm so that the subproblem can be processed.

```
3117 FOR I=1 TO BD
3118 M#(E(I)+1,1)=0#
3119 IF N#(LN,I)=1 THEN M#(E(I)+1,1)=BI#
3120 NEXT I
```

Runs the dual simplex algorithm from the point at the origin if the cycle count is one (1). Otherwise, uses the restart method.

```
3121 IF CYC>1 THEN 3130
3122 GOSUB 3300:REM ALGR-SMP
```

Reinverts the augmented $\overline{B}$ matrix and saves the augmented $\overline{B}$ inverse matrix and the augmented $\overline{B}$ matrix for the optimal unrestricted solution in the array O#(ND+1+ND,ND+2).

The reinversion is done as a precaution since all succeeding subproblems start from this matrix.

```
3123 GOSUB 3900:REM TRAN-RIV
3124 FOR I=1 TO 2*ND+1
3125 FOR J=1 TO ND+1
3126 O#(I,J)=B#(I,J)
3127 NEXT J
3128 NEXT I
3129 GOTO 3136
```

Processes the subproblem using the restart method.

The restart method starts the subproblem from the optimal point reached in cycle one (1). If more computer array space where available the restart could be started from the optimal point reached at the most recent node on the branch and bound tree by saving the augmented $\overline{B}$ inverse matrix and the augmented $\overline{B}$ matrix of that node's subproblem.

```
3130 FOR I=1 TO 2*ND+1
3131 FOR J=1 TO ND+1
3132 B#(I,J)=O#(I,J)
```

```
3133 NEXT J
3134 NEXT I
```

The intialization lines of the dual simplex algorithm are skipped since the starting point for the algorithm is not the origin but the restart point.

```
3135 GOSUB 3313:REM ALGR-SMP+13
```

Totals the number of iterations for all the subproblems.

```
3136 ITR=ITR+IT
3137 IT=ITR
```

Returns to the main routine if the error code on the first cycle indicates the model is infeasible or has exceeded the maximum iterations.

```
3138 IF ER<>1 AND CYC=1 THEN RETURN
3139 IF BD=0 THEN 3170
```

Sets the value of the objective function in the node array to infinity.

```
3140 N#(LN,BD+1)=BI#
```

Lets the objective function value in the node array stay at infinity if the error code indicates the subproblem is infeasible or the best current restricted pair solution is less or equal to the solution of the subproblem.

```
3141 IF ER<>1 OR B#(1,1)>=0B# THEN 3176
```

Checks the restricted pairs of the solution to the subproblem to see if they are all satisfied.

```
3142 FOR I=1 TO BD STEP 2
3143 IF B#(E(I)+1,1)*B#(E(I+1)+1,1)>0# THEN 3146
3144 NEXT I
3145 GOTO 3170
```

Finds the restricted pair which is closest to being satisfied, and sets NM equal to one of the variables of the pair.

```
3146 MM#=BI#
3147 MA#=0#
3148 NW=0
3149 FOR I=1 TO BD STEP 2
```

```
3150 IF B#(E(I)+1,1)*B#(E(I+1)+1,1)=0# THEN 3157
3151 IF B#(E(I)+1,1)>=MM# THEN 3154
3152 MM#=B#(E(I)+1,1)
3153 NW=I
3154 IF B#(E(I+1)+1,1)>=MM# THEN 3157
3155 MM#=B#(E(I+1)+1,1)
3156 NW=I+1
```

Finds the amount of the Driebeek's penalty for the current subproblem solution.

```
3157 MI#=BI#
3158 FOR J=0 TO 1
3159 FOR K=2 TO ND+1
3160 IF B#(E(I+J)+1,K)<=SM# THEN 3162
3161 IF MI#>(B#(E(I+J)+1,1)*B#(1,K))/B#(E(I+J)+1,K) THEN MI#=(B#(E(I+J)+
     1,1)*B#(1,K))/B#(E(I+J)+1,K)
3162 NEXT K
3163 NEXT J
3164 IF MI#>MA# THEN MA#=MI#
3165 NEXT I
3166 IF B#(1,1)+MA#>OB# THEN 3176
3167 N#(LN,BD+1)=B#(1,1)+MA#
3168 N#(LN,BD+2)=NW
3169 GOTO 3186
```

Sets the best restricted pair solution to the new restricted pair solution.

```
3170 OB#=B#(1,1)
3171 FOR I=2 TO ND+1
3172 X#(I-1)=B#(I,1)
3173 NEXT I
```

Returns to the main routine and sets ER=1 if there is only one (1) node in the branch and bound tree and the solution satisfies the restricted pairs.

```
3174 N#(LN,BD+1)=BI#
3175 IF BD=0 OR LL=1 THEN RETURN
3176 IF OB#=BI# THEN 3186
```

Checks the node array N#(MX,ND+2) for any nodes with solutions greater than or equal to the current node's restricted pairs solution. If the current nodes solution is less than or equal, then set the node's solution value to infinity.

```
3177 X=0
```

```
3178 FOR I=1 TO LL
3179 IF N#(I,BD+1)=BI# THEN 3182
3180 IF N#(I,BD+1)<OB# THEN 3183
3181 N#(I,BD+1)=BI#
3182 X=X+1
3183 NEXT I
```

Sets ER=1 and returns to the main routine if only one (1) restricted pairs solution is left in the node arrays.

```
3184 ER=1
3185 IF X=LL THEN RETURN
```

Returns to process the other node of the pair on the branch.

Since at each branch two (2) nodes are created, both nodes must be processed before another branch is taken.

```
3186 IF CYC/2=INT(CYC/2) THEN 3214
```

Executes the node compression subroutine if the node array is full.

```
3187 IF CYC-DL+1>=MX THEN GOSUB 3250:REM ALGR-CPR
```

Sets ER=4 and returns to the main routine if the node array cannot be cleared for new nodes.

```
3188 ER=4
3189 IF CYC-DL+1>=MX THEN RETURN
```

Finds the node NODE with the minimum solution in the node array.

```
3190 NODE=0
3191 MI#=BI#
3192 FOR I=1 TO LN
3193 IF MI#<=N#(I,BD+1) THEN 3196
3194 MI#=N#(I,BD+1)
3195 NODE=I
3196 NEXT I
```

Sets ER=2 and returns to the main routine if no minimum node solution can be found.

```
3197 ER=2
3198 IF NODE=0 OR DL=CYC THEN RETURN
```

Creates two (2) new nodes for the branch and bound tree and transfers the current upper and lower bounds to the new nodes.

```
3199 FOR I=1 TO BD+2
3200 N#(LN+1,I)=N#(NODE,I)
3201 N#(LN+2,I)=N#(NODE,I)
3202 NEXT I
```

Adds the new upper and lower bound restrictions to the new nodes and returns to process the two (2) new nodes.

```
3203 LL=LN+2
3204 NW=N#(NODE,BD+2)
3205 N#(NODE,BD+1)=BI#
3206 N#(LN+1,NW)=1#
3207 N#(LN+2,NW)=-1#
3208 IF NW/2=INT(NW/2) THEN 3212
3209 N#(LN+1,NW+1)=-1#
3210 N#(LN+2,NW+1)=1#
3211 GOTO 3214
3212 N#(LN+1,NW-1)=-1#
3213 N#(LN+2,NW-1)=1#
3214 CYC=CYC+1
3215 GOTO 3114
```

Node Compressing Subroutine -- File ALGR-CPR

Same as for the branch and bound method except for lines:

```
3254 REM N%(I,1)<>BI% THEN 3258
3255 N#(I,1)<>BI# THEN 3258
3260 REM N%(X,J)=N%(I,J)
3261 N#(X,J)=N#(I,J)
3262 REM N%(X+1,J)=N%(I+1,J)
3263 N#(X+1,J)=N#(I+1,J)
3269 REM N%(I,J)=0%
3270 N#(I,J)=0#
```

Dual Simplex Algorithm Subroutine -- File ALGR-SMP

Same as for dual simplex method except for lines:

```
3332 IF A#(K,ND+2)<>0# THEN GOSUB 4000:REM PAR-TANL
3334 IF A#(K,ND+3)<>0# THEN GOSUB 4200:REM HYP-TANL
3336 IF A#(K,ND+4)<>0# THEN GOSUB 4400:REM SHT-TANL
3369 IF B#(1,1)>=OB# THEN RETURN
```

Constraint Transformation Subroutine -- File TRAN-CON

Same as for dual simplex method.

Inversion Subroutine -- File TRAN-INV

Same as for dual simplex method.

Reinversion Subroutine -- File TRAN-RIV

Same as for dual simplex method.

Parabolic Subroutine -- File PAR-TANA, PAR-TANL, or PAR-TANG

Same as for dual simplex method with nonlinear constraints.

Hyperbolic Subroutine -- File HYP-TANA, HYP-TANL, or HYP-TANG

Same as for dual simplex method with nonlinear constraints.

Hyperbolic Sheet Subroutine -- File SHT-TANA, SHT-TANL, or SHT-TANG

Same as for dual simplex method with nonlinear constraints.

Program Table of Contents

Table 28 can be used to reconstruct the above computer code from the computer disk and to organize subroutines from previous program listings. Since BASIC code is dependent on program line numbers for its branching, the statement numbers must be maintained as listed below.

The constraint key subroutine ALGR-KEY must be changed as before in the branch and bound method with the following lines:

```
3024 REM
3026 GOSUB 3100:REM ALGR-PRS
3027 FOR I=2 TO ND+1
3028 M#(I,1)=N%(1,I)
3029 M#(I,2)=N%(2,I)
3030 NEXT I
```

and the dual simplex algorithm ALGR-SMP must be modified by adding:

```
3332 IF A#(K,ND+2)<>0# THEN GOSUB 4000:REM PAR-TANL
3334 IF A#(K,ND+3)<>0# THEN GOSUB 4200:REM HYP-TANL
3336 IF A#(K,ND+4)<>0# THEN GOSUB 4400:REM SHT-TANL
3369 IF B#(1,1)>=0B# THEN RETURN
```

Since the node array has been changed to a REAL array rather than an INTEGER array, the node compressing subroutine ALGR-CPR must be changed with the following lines of code:

```
3254 REM N%(I,1)<>BI% THEN 3258
3255 N#(I,1)<>BI# THEN 3258
3260 REM N%(X,J)=N%(I,J)
3261 N#(X,J)=N#(I,J)
3262 REM N%(X+1,J)=N%(I+1,J)
3263 N#(X+1,J)=N#(I+1,J)
3269 REM N%(I,J)=0%
3270 N#(I,J)=0#
```

Table 28. Restricted pairs BASIC program table of contents

File	Program lines	Page	Routines
MAIN-PRS	0001-0141	382	Restricted pairs method
INPT-OBJ	1200-1220	61	Objective coefficient input subroutine
INPT-CON	1300-1328	63	Constraint coefficient input subroutine
INPT-TYP	1400-1440	65	Constraint type input subroutine
INPT-BND	1500-1523	67	Variable bounds input subroutine
INPT-PRS	1700-1723	388	Restricted pairs input subroutine
UTIL-OPT	1800-1803	22	Option line subroutine
UTIL-ERS	1850-1853	22	Erase option subroutine
UTIL-CON	1860-1866	22	Continue line subroutine
UTIL-CHX	1870-1882	23	Data check subroutine
REPT-SMP	2200-2224	69	Simplex report subroutine
ALGR-KEY	3000-3057	200	Constraint keys subroutine
ALGR-PRS	3100-3270	391	Restricted pairs algorithm subroutine
ALGR-CPR	3200-3274	357	Node compressing subroutine
ALGR-SMP	3300-3398	70	Dual simplex algorithm subroutine
TRAN-CON	3600-3633	74	Constraint transformation subroutine
TRAN-INV	3700-3743	76	Basis inversion subroutine
TRAN-RIV	3900-3960	78	Basis reinversion subroutine
PAR-TANL	4000-4034	311	Supporting plane for parabolic subr.
HYP-TANL	4200-4221	318	Supporting plane for hyperbolic subr.
SHT-TANL	4400-4425	325	Supporting plane for sheet subroutine

Restricted Pairs Method: Solutions to Example 15 Cumulative Mandays Curve Fitting Problem

The solution to the cumulative mandays curve fitting problem can be found by the restricted pairs method. If the ten (10) activity example network that was used in all previous examples is used as a restricted pairs problem, even if restricted to only two (2) spans of time on the mandays step function, the number of variables in the CP problem would be thirty-four (34). Since the number of variables, or the array space, in the restricted pairs program code would exceed the 64K capacity of the IBM BASIC compiler without the use of overlays, the problem has been reduced to a two (2) activity network and a three (3) span step function.

As before, the following variables are defined for the example 15 restricted pairs problem:

S_i	- Start time of activity i
dur.	- Project duration
m_i	- Man count on activity i
d_i	- Duration of activity i
sp_i	- End of span i
$mandays_i$	- Mandays allowed in span i

The two (2) activity network is shown in figure 86.

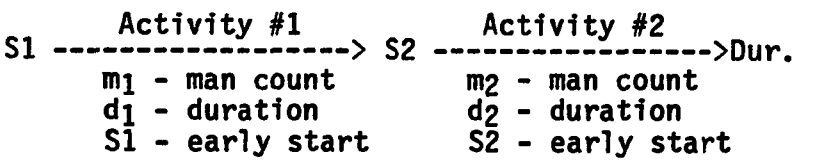


Figure 86. Restricted pairs model example 15

If the cumulative mandays curve is approximated by a three (3) segment step function in which each segment has $mandays_1$, $mandays_2$, $mandays_3$

total mandays, then a set of variables representing the number of days that each start time and each finish time precedes and succeeds the end of span points (sp_1,sp_2) of the cumulative mandays step function can be used to construct a restricted pairs CP problem as shown in figure 87

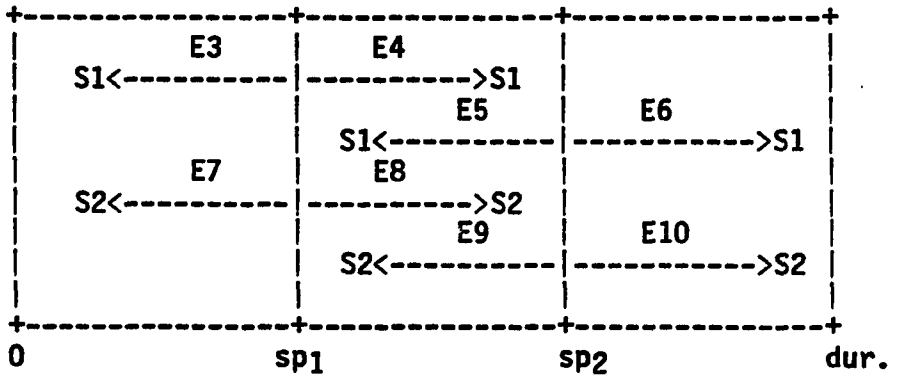


Figure 87. Restricted pairs model example 15 early start barchart

and figure 88 which minimizes the deviation of the scheduled activities' mandays from the predetermined cumulative mandays curve.

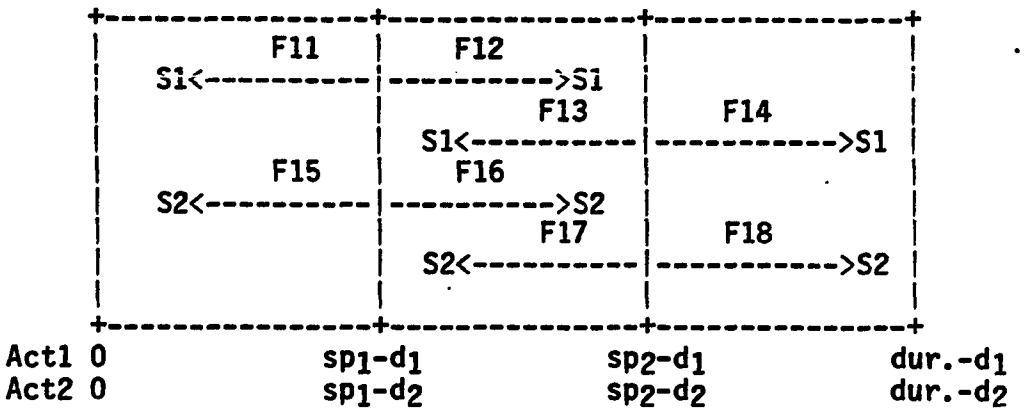


Figure 88. Restricted pairs model example 15 early finish barchart

The resulting restricted pairs CP problem is:

minimize $V19+V20+V21$

Subject to:
$S2-d_1-S1>=0$
$S2<=dur.-d_2$
$S1+E3-E4=sp_1$
$S1+F11-F12=sp_1-d_1$
$S1+E5-E6=sp_2$
$S1+F13-F14=sp_2-d_1$
$S2+E7-E8=sp_1$
$S2+F15-F16=sp_1$
$S2+E9-E10=sp_2$
$S2+F17-F18=sp_2-d_2$
$m_1*(E3-F11)+m_2*(E7-F15)+V19-V20=mandays_1$
$m_1*(E5-F13)+m_2*(E9-F17)+V21-V20=mandays_1+mandays_2$
$E3*E4=0$
$E4*E6=0$
$E7*E8=0$
$E9*E10=0$
$F11*F12=0$
$F13*F14=0$
$F15*F16=0$
$F17*F18=0$

$S1,S2,E3,E4,...V21>=0$

The value of the constants can now be defined as:

$sp_1=5$
$sp_2=10$
$d_1=5$
$d_2=5$
$m_1=2$
$m_2=2$
$dur.=20$
$mandays_1=10$
$mandays_2=0$
$mandays_3=10$

The solution of the example problem using the restricted pairs method is listed in figures 89 and 90 as it would appear on the computer screen. Since the number of variables exceeds the capacity of a single screen, twenty-four (24) lines, the listing requires two (2) screens.

```
SOLUTION FOUND IN 17 ITR        262 SEC
VALUE OF OBJECTIVE              0

        VARIABLE NO.    VALUES
        1               0
        2               10
        3               5
        4               0
        5               10
        6               0
        7               0
        8               5
        9               0
        10              0
        11              0
        12              0
        13              5
        14              0
        15              0

        (PRESS RETURN TO CONTINUE)
```

Figure 89. Restricted pairs problem example 15 solution, screen one

```
        16              10
        17              0
        18              5
        19              0
        20              0
        21              0

        OPTION ?
```

Figure 90. Restricted pairs problem example 15 solution, screen two

The solution was reached in seventeen (17) iterations in two hundred sixty-two (262) seconds. The restricted pairs cumulative mandays curve and early start times that best fit the predetermined cumulative mandays curve are shown in figure 91.

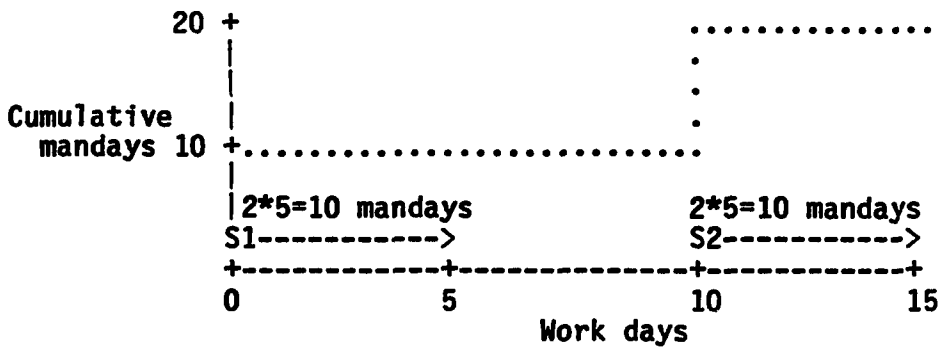


Figure 91. Restricted pairs problem example 15 cumulative mandays curve

SUMMARY OF METHODS

Seventeen (17) different mathematical programming methods were coded into BASIC computer code, compiled using an IBM BASIC compiler, and run on a Panasonic Sr. Partner micro-computer for the purpose of solving eight (8) mathematical models and problems derived from a common ten (10) activity project network diagram. The eight (8) problems were:

(1) The minimum project duration with fixed activity durations and a fixed activity sequence.

(2) The minimum project cost with costs as linear functions of activity duration, variable activity durations, a fixed activity sequence, and a fixed project duration.

(3) The minimum project cost curve with costs as linear functions of activity duration, variable activity durations, a fixed activity sequence, and a variable project duration.

(4) The minimum project cost with costs as quadratic functions of activity duration, variable activity durations, a fixed activity sequence, and a fixed project duration.

(5) The minimum project man count with man counts as hyperbolic function of activity duration, variable activity durations, a fixed activity sequence, and a fixed project duration.

(6) The minimum project supervision cost with costs as parabolic function of man count, man counts as hyperbolic function of activity duration, variable activity durations, a fixed activity sequence, and a fixed project duration.

(7) The minimum project supervision cost curve with costs as parabolic function of man count, man counts as hyperbolic function of activity duration, variable activity durations, a fixed activity sequence, and a variable project duration.

(8) The minimum deviation of the cumulative project mandays curve from a predetermined cumulative curve given a fixed activity sequence.

Summary of Solutions

The results of the computer runs which solved the eight (8) different problems are summarized in the following tables for the purpose of establishing benchmarks for the different methods.

The CPM method finds the minimum project duration by utilizing a network model with fixed activity durations. The solution for the ten (10) activity network schedule is summarized in table 29.

Table 29. Critical path method minimum project duration solution to the example 2 project network model

Dur.	4
Itr.	noniterative method
Sec.	4 seconds
Obj.	nonoptimizing method

By changing the network model to a linear programming problem with costs which are linear functions of activity duration and variable activity durations, the dual simplex method can be used to solve the ten (10) activity LP cost problem for a minimum project cost at ten (10) different predetermined project durations as summarized in table 30.

Table 30. Dual simplex method solutions to the example 4 minimum project cost problem with linear cost functions

Dur.	160	120	101	100	91	90	72	43	11	4
Itr.	7	8	10	9	11	12	13	13	14	15
Sec.	5	6	7	6	8	8	9	9	10	10
Obj.	750	1950	2710	2755	3700	3835	6265	11195	17275	18640
CyT.	.71	.75	.70	.66	.72	.66	.69	.69	.71	.66

The dual simplex method solves the linear cost problem for only one predetermined project duration at a time. If the LP problem is expanded to include a variable project duration, then out-of-kilter method can be used to solve for the same ten (10) simplex solutions of table 30, or the project cost curve, in the time listed in table 31.

Table 31. Out-of-kilter method solution to the example 6 minimum project cost problem with linear cost functions

Dur. range between 160-4
Itr. 11
Sec. 3
Obj. project cost curve

Expanding the problem to include a quadratic cost function, Beale's

Table 32. Beale's method solutions to the example 10 minimum project cost problem with quadratic cost functions.

Dur.	100	90	80	70	60	50	40	30	20	10
Itr.	41	28	31	31	31	49	37	32	28	44
Sec.	133	87	103	103	103	164	122	100	83	138
Obj	14780.	19121.	24378.	30551.	37640.	45673.	55545.	68007.	83069.	100740.
CyT.	3.24	3.10	3.32	3.32	3.32	3.34	3.29	3.12	2.96	3.13

method can be used to solve the QP cost problem for minimum project cost at each of ten (10) predetermined project durations as summarized in table 32.

Using only the quadratic or linear cost function problems is over simplistic since variable cost are more directly related to activity man counts than to the activity durations.

In order to model activity man counts, the LP problem is expanded to include man counts as hyperbolic functions of activity duration. The CP problem is then solved for ten (10) different predetermined project durations using the dual simplex method with nonlinear constraints approximated with supporting planes derived with a line search algorithm as summarized in table 33.

Table 33. Dual simplex method, using supporting planes derived with the line search algorithm, solutions to the example 11 minimum project man count problem with hyperbolic man count functions

Dur.	100	90	80	70	60	50	40	30	20	10
Itr.	115	114	107	115	116	109	112	114	107	112
Sec.	205	198	186	206	216	190	194	209	193	209
Obj.	12.	14.	15.	18.	21.	25.	31.	42.	63.	127.
CyT.	1.78	1.78	1.73	1.79	1.86	1.74	1.73	1.83	1.80	1.86

A simpler method for solving the man count CP problem is to use the same dual simplex method; but instead of using supporting planes derived with the usual line search algorithm, use supporting planes derived with a projection technique or the Gordian algorithm. The results of this simpler alogrithm are summarized in table 34.

Table 34. Dual simplex method, using supporting planes derived with the Gordian algorithm, solutions to the example 11 minimum project man count problem with hyperbolic man count functions

Dur.	100	90	80	70	60	50	40	30	20	10
Itr.	117	129	117	119	111	124	112	113	115	121
Sec.	141	156	137	138	130	149	133	135	139	153
Obj.	12.	14.	15.	18.	21.	25.	31.	42.	63.	127.
CyT.	1.20	1.20	1.17	1.15	1.17	1.20	1.18	1.19	1.20	1.26

A variation of the dual simplex method which results in greater precision is the corner cut method. Solving the same man count CP problem using the corner cut method with supporting planes derived with the line search method results in table 35.

Table 35. Corner cut method, using supporting planes derived with the line search algorithm, solutions to the example 11 minimum project man count problem with hyperbolic man count functions

Dur.	100	90	80	70	60	50	40	30	20	10
Itr.	152	136	150	147	133	152	189	139	214	195
Sec.	393	342	367	384	361	364	480	375	597	534
Obj.	12.	14.	15.	18.	21.	25.	31.	42.	63.	127.
CyT.	2.58	2.51	2.44	2.61	2.71	2.39	2.53	2.69	2.78	2.73

As with the dual simplex method, a simpler variation of the corner cut method for solving the same man count CP problem is the corner cut method using supporting planes derived with projection technique or the Gordian algorithm as summarized in table 36.

Table 36. Corner cut method, using supporting planes derived with the Gordian algorithms, solutions to the example 11 minimum project man count problem with hyperbolic man count functions

Dur.	100	90	80	70	60	50	40	30	20	10
Itr.	192	215	192	218	160	179	188	277	318	260
Sec.	412	439	391	435	335	366	381	590	652	572
Obj.	12.	14.	15.	18.	21.	25.	31.	42.	63.	127.
CyT.	2.14	2.04	2.03	1.99	2.09	2.04	2.02	2.12	2.05	2.20

A comparison of the four (4) methods is summarized in table 37.

Table 37. Comparison between simplex and corner cut methods

Parameter	Dual Simplex Method		Corner Cut Method	
	Line Search	Gordian	Line Search	Gordian
Mean itr.	112.1	117.8	160.7	219.9
Standard deviation	3.34	5.65	28.12	49.83
Mean sec.	200.6	141.1	419.7	457.3
Standard deviation	9.77	8.71	86.59	108.00
Mean cyc. time	1.790	1.192	2.597	2.072
Standard deviation	0.048	0.029	0.130	0.064

By expanding the man count CP problem, a supervision cost CP problem can be derived from the man count problem by assuming that supervision cost is a parabolic function of activity man count. Beale's method with supporting planes derived with the line search method provides a means of solving the supervision cost CP problem for ten (10) predetermined project durations as summarized in table 38.

Table 38. Beale's method, using supporting planes derived with the line search algorithm, solutions to the example 12 minimum project supervision cost problem with hyperbolic man count and parabolic cost functions

Dur.	100	90	80	70	60	50	40	30	20	10
Itr.	230	230	215	202	184	199	162	164	174	177
Sec.	1333	1333	1200	1049	974	1031	820	846	872	931
Obj.	554.	554.	554.	556.	581.	654.	831.	1270.	2619.	10124.
CyT.	5.55	5.55	5.58	5.19	5.29	5.18	5.06	5.15	5.01	5.25

A simpler method for solving the same supervision cost CP problem is Beale's method with supporting planes derived with a projection technique or the Gordian algorithm as summarized in table 39.

Table 39. Beale's method, using supporting planes derived with the Gordian algorithm, solutions to the example 12 minimum project supervision cost problem with hyperbolic man count and parabolic cost functions

Dur.	100	90	80	70	60	50	40	30	20	10
Itr.	203	203	213	206	189	181	169	157	163	172
Sec.	842	842	908	854	766	743	706	645	694	733
Obj.	554.	554.	554.	556.	581.	654.	831.	1270.	2619.	10124.
CyT.	4.14	4.14	4.26	4.14	4.05	4.10	4.17	4.10	4.25	4.26

The quadratic objective function and hyperbolic constraints of the supervision cost CP problem can be algebraically combined into a set of cubic hyperbolic constraints. By incorporating these constraints into the primal-dual method, the supervison cost problem can be solved for the ten (10) predetermined project durations as summarized in table 40.

Table 40. Primal-dual method, using supporting planes derived with the Gordian algorithm, solutions to the minimum project supervision cost model with cubic hyperbolic cost functions

Dur.	100	90	80	70	60	50	40	30	20	10
Itr.	119	123	119	101	108	111	109	118	126	140
Sec.	210	225	209	172	190	203	202	226	230	264
Obj.	554.	554.	554.	556.	581.	654.	831.	1270.	2619.	10124.
CyT.	1.76	1.82	1.75	1.70	1.75	1.82	1.85	1.91	1.82	1.88

Another method which has been used to solve nonlinear problems is the ellipsoidal method. As a basis of comparison with established methods, the same supervision cost CP problem as solved above is solved using the ellipsoidal method. The results of using the ellipsoidal method for the solution of the CP problem are summarized in table 41.

Table 41. Ellipsoidal method, using supporting planes derived with the line search algorithm, solutions to the minimum project supervision cost model 12 with hyperbolic man count and parabolic cost function

Dur.	100	90	80	70	60	50	40	30	20	10
Itr.	5439	5352	5295	5308	5538	5496	5475	5680	5745	6730
Sec.	16386	17147	16079	15582	16186	16077	16043	16461	16722	19250
Obj.	554.	554.	554.	556.	581.	654.	831.	1270.	2619.	10124.
CyT.	3.01	3.20	3.03	2.93	2.92	2.92	2.93	2.89	2.91	2.86

The results of Beale's method using supporting planes derived with both the line search algorithm and the Gordian algorithm, the primal-dual method with cubic hyperbolic constraints, and the ellipsoidal method in solving the supervision cost CP problem are summarized in table 42.

Table 42. Comparison of Beale's method, Cubic constraint method and Goffin's method

Parameter	Beale's Method Line Search	Beale's Method Gordian	Cubic Constraints	Ellipsoidal Method
Mean iterations	295.7	185.6	117.4	5605.8
Standard deviation	28.76	19.98	11.00	421.63
Mean seconds	1038.9	773.3	213.1	16593.3
Standard deviation	190.88	84.30	25.07	1025.46
Mean cycle time	5.284	4.166	1.812	2.963
Standard deviation	0.209	0.073	0.065	0.099

The above methods solved the minimum project supervison cost problem for only one predetermined project duration at a time. If the problem is expanded to include a variable project duration for a cost versus duration curve, the restart method can be used to find the same ten (10) solutions of table 38 in the time and iterations shown in table 43.

Table 43. Restart method, using supporting planes derived from the line search algorithm, solution to the example 12 minimum project supervision cost problem cost curve with hyperbolic man count and parabolic cost functions

Dur. range between 74.39-10
Itr. 1321
Sec. 4577
Obj. project cost curve

In the solution of the above problems, integer solutions were not considered essential. In actual practice, the noninteger solution can not be used. This is obvious, considering that an integer man works a

minimum of an integer day on most construction sites.

In the case of the minimum duration model and the minimum project cost problem with linear activity costs, the optimal solutions are always integer due to the unimodular structure of the problem. In the case of the nonlinear problems, the methods do not guarantee integer solutions.

To find integer solutions for the nonlinear problems, Gomory's method was modified with deep cut supporting planes derived with the line search algorithm. Table 44 summarizes Gomory's method integer solutions to the man count CP problem for the same ten (10) predetermined project duration.

Table 44. Gomory's method, using deep cuts derived with the line search algorithm, integer solutions to the example 11 minimum project man count problem with hyperbolic man count functions

Dur.	100	90	80	70	60	50	40	30	20	10
Itr.	580	5298	1110	445	822	6737	140	83	110	107
Sec.	1540	13934	2718	1268	2189	17604	343	250	313	228
Obj.	16	19	20	22	24	29	34	46	65	132
CyT	2.65	2.63	2.44	2.84	2.66	2.61	2.45	3.01	2.84	2.13

Another method for solving the integer man count CP problem is the branch and bound method with deep cut supporting planes derived with the line search algorithm. This method is a hybrid of either the dual simplex or Beale's method in which a series of subproblems are solved with tighter and tighter restrictions on the feasible region. A summary of the results from the branch and bound method with deep cut supporting planes derived with the line search algorithm for the integer man count CP problem for ten predetermined durations is shown in table 45.

Table 45. Branch and bound method, using deep cuts derived with the line search algorithm, integer solutions to the example 11 minimum project man count problem with hyperbolic man count functions

Dur.	100	90	80	70	60	50	40	30	20	10
Itr.	75	145	135	263	204	422	49	49	35	46
Sec.	240	456	440	797	649	1249	174	172	147	176
Obj.	16	19	20	22	24	29	34	46	65	132
CyT.	3.20	3.14	3.25	3.03	3.18	2.95	3.55	3.51	4.20	3.82

The convergence rate of the branch and bound method can be enhanced by using Driebeek's penalty. The summary of the results are shown in table 46 for the man count CP problem.

Table 46. Branch and bound method with Driebeek's penalty, using deep cuts derived with the line search algorithm, integer solutions to the minimum project man count problem with hyperbolic man count functions

Dur.	100	90	80	70	60	50	40	30	20	10
Itr.	75	156	100	235	168	385	39	45	28	40
Sec.	129	240	174	332	250	484	95	103	94	115
Obj.	16	19	20	22	24	29	34	46	65	132
CyT.	1.75	1.53	1.74	1.41	1.48	1.25	2.43	2.28	3.35	2.87

Table 47. Comparison of Gomory's method, branch and bound method, and Driebeek's penalty for solution of minimum man count problem

Parameter	Gomory's method	Branch and bound	Driebeek's penalty
Mean iterations	1543.2	142.3	127.1
Standard deviation	2406.78	124.45	113.53
Mean seconds	4038.7	450.0	201.6
Standard deviation	6301.98	359.47	127.70
Mean cycle time	2.626	3.383	2.000
Standard deviation	0.247	0.389	0.697

A comparison of the three (3) methods is summarized in table 47.

The supervision cost problem does not require integer dollars solutions; but in the final solution, the days and men which determine the supervision cost, must still be integer. To find this integer solution, the branch and bound method with Beale's method with nonlinear constraints can be used. The result of the branch and bound method when applied to the project supervision cost problem is summarized in tables 48 and 49.

Table 48. Branch and bound method, using deep cuts derived with the line search algorithm, integer solutions to the example 12 minimum project supervision cost problem with hyperbolic man count and parabolic cost functions

Dur.	100	90	80	70	60	50	40	30	20	10
Itr.	151	153	152	411	812	726	71	155	88	403
Sec.	634	644	621	1488	2784	2443	304	586	410	1597
Obj.	595	595	595	610	653	733	872	1443	2687	10940
CyT.	4.19	4.20	4.08	3.62	3.42	3.36	4.28	3.78	4.65	3.96

Table 49. Summary of branch and bound method solving the example 12 project supervision cost problem

Mean iterations	312.2
Standard deviation	268.57
Mean seconds	1151.1
Standard deviation	884.39
Mean cycle time	3.954
Standard deviation	0.408

By finding the minimum project cost or the minimum project man count, a cumulative mandays curve is determined. In many cases the

cumulative mandays curve is predetermined. To find the project schedule which best fits the predetermined curve, a curve fitting CP problem can be derived and solved using the restricted pairs method. The results of the restricted pairs method for the example 15 three (3) activity network are summarized in table 50.

Table 50. Restricted pairs solution to the example 15 cumulative mandays curve fitting problem

Dur. 20
Itr. 17
Sec. 262
Obj. 0

CONCLUSION

The critical path method, or CPM, is a proven method for modeling, in the form of a network diagram, the essential information of construction project schedules. Using this method, most construction projects can be modeled by networks with between fifty (50) to three hundred (300) activities. Even for larger projects, networks exceeding twelve thousand (12000) activities have been successfully analyzed with the aid of the computer.

The simplex method is also a proven method for optimizing linear programming problems with up to twenty thousand (20000) variables[54]. By constructing linear programming, or LP, cost problems derived from CPM networks models in which project costs are represented by linear functions of activity durations, networks with up to thirteen thousand (13000) activity (assuming 1.5 simplex variables per network node) can be cost optimized using the simplex method.

The LP cost problem, as solved by the simplex method, is a practical means of finding the minimum project cost for a given project durations if the costs are linear functions of activity durations.

In practice, most project cost and manpower functions are parabolic or hyperbolic in relation to activity durations. This results in a nonlinear problem, rather than a linear problem, in which costs and manpower are parabolic or hyperbolic functions of activity durations.

To find the optimal cost or manpower solution to the nonlinear problem, a modification of the simplex method was developed in this

dissertation in which the nonlinear convex functions are incorporated into the simplex method by means of supporting planes used as linear approximations of the nonlinear constraints.

As a demomstration of the computer requirements and the performance specifications of the nonlinear simplex methods, nine (9) linear and nine (9) nonlinear schedule problems in which the the number of network activities are increased from five (5) to fifty (50) were solved on a Panasonic Sr. Partner computer with a simple overlay version of the programs presented above. The central processing unit (CPU) requirements

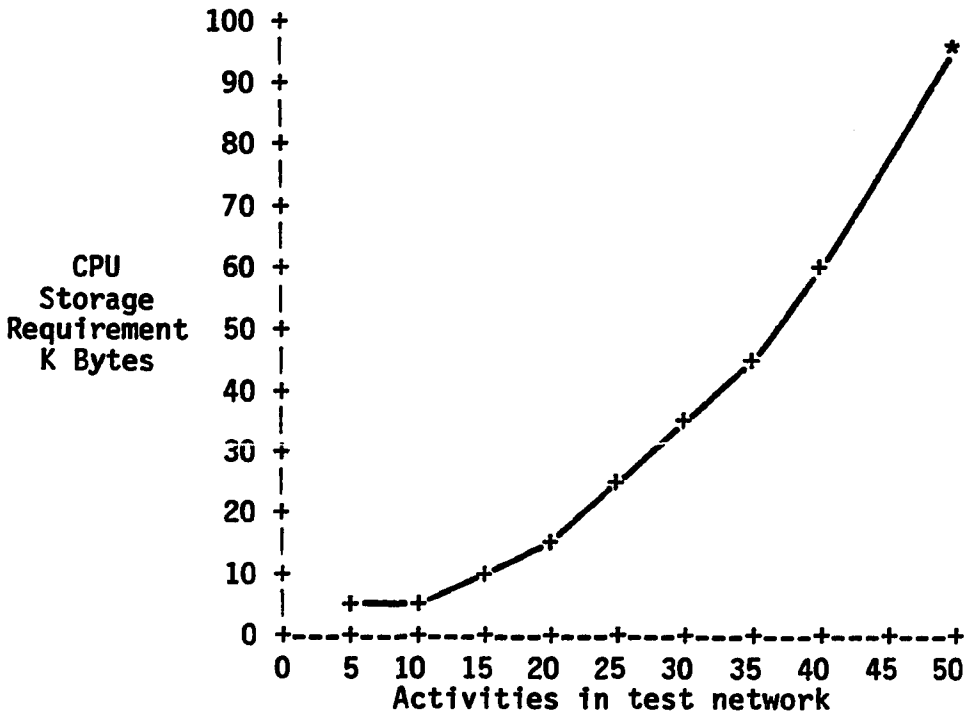


Figure 92. CPU requirements per number of network activities

for each problem are shown in figure 92, and the iterations and seconds per iteration required for each computer solution are shown in table 51.

Table 51. Number of iterations and seconds per iteration required to solve minimum cost and manpower problems for networks with 5 to 50 activities using the primal-dual simplex method with linear and nonlinear constraints with precision set at 1E-08.

Activities in network	Constraint Types: Linear Simplex Itr.	Linear Simplex Sec./Itr.	Nonlinear Cubic Supporting plane Itr.	Nonlinear Cubic Supporting plane Sec./Itr.	Nonlinear Deep cut Itr.	Nonlinear Deep cut Sec./Itr.
5	10	1.700	61	1.420	16	1.810
10	18	2.300	120	2.530	28	2.714
15	27	3.185	174	4.017	43	3.744
21	38	5.157	527	7.530	64	6.562
26	49	7.224	335	9.414	80	7.975
31	67	10.641	457	12.927	117	11.478
35	71	11.676	383	13.349	122	12.991
40	81	14.395	655	17.499	162	17.277
50	105	52.476	560	86.820	186	86.660

Table 52. Ratios of the number of iteration and seconds per iteration required to solve minimum cost and manpower problems with cubic cost and hyperbolic man count functions constraint to the number of iteration and seconds per iteration required to solve the minimum cost problems with linear cost functions for networks with 5 to 50 activities with precision set at 1E-08.

Activities in network	Cubic supporting plane Itr./Itr.	Cubic supporting plane Sec./Itr./Sec./Itr.	Hyperbolic deep cut Itr./Itr.	Hyperbolic deep cut Sec./Itr./Sec./Itr.
5	6.10	0.83	1.60	1.06
10	6.66	1.10	1.55	1.18
15	6.44	1.26	1.59	1.17
21	13.86	1.46	1.68	1.27
26	6.83	1.30	1.63	1.10
31	6.82	1.21	1.74	1.07
35	5.39	1.14	1.71	1.11
40	8.08	1.21	2.00	1.20
50	5.33	1.65	1.77	1.65
Mean	7.27	1.24	1.69	1.20
Std. Dev.	2.60	0.22	0.13	0.18

Based on these computer runs as summarized in table 52, an increase in iterations of approximately 7.27 times and an increase in processing time of approximately 7.27*1.24=9.01 times the linear problem can be expected for the solution of nonlinear supervision cost or manpower problems.

Even with this increase in time and iterations, networks of fifty (50) activities can be reasonably optimized on micro-computers with 64K bytes CPU memories; and networks of three hundred (300) activities can be optimized on main-frame computers with 4M bytes CPU memories using the nonlinear modification of the simplex method.

Most schedule problems require integer solutions since schedule dates are integer by contract and crew sizes are integer for efficiency.

In the case of the linear cost problems as optimized by the simplex method, the optimal solution is also an integer solution. This does not hold true for the nonlinear cost and manpower problems. The modified simplex solutions to the nonlinear cost and manpower problems with parabolic or hyperbolic functions are usually not integer.

For the nonlinear manpower problem, a good, if not guaranteed optimal solution, can be found by utilizing the partial unimodularity of the problem and "rounding up" the solution found with primal-dual simplex method with deep cut constraints as demonstrated in Appendix B. If this suboptimal solution is used, based on the nine (9) computer solutions above as summarized in table 51, an increase in iterations of approximately 1.69 times and an increase in processing time of

approximately 1.20*1.69=2.02 times the linear problem can be expected.

To guarantee an optimal integer solution, a branch and bound algorithm is used which incorporates a dual simplex algorithm with deep cut constraints and a Driebeek's penalty function.

Based again on the nine (9) computer solutions above, an increase in iterations of approximately 1.69 times the linear cost problem can be attributed to the deep cuts and an increase of two (2) to five (5) time the linear problem can be attributed to the branch and bound algorithm[55] for a total of 1.69*5=8.45 times the iterations of the linear cost problem. The increase in processing time is then approximately 8.45*1.20=10.14 times the linear problem.

To solve the integer supervision cost problem requires the branch and bound algorithm combined with Beale's algorithm with deep cut constraints. Although the program code for the method can be executed within the CPU requirements shown in figure 92, the transformation time of the tableau increases rapidly from 8.45 seconds per iteration for a five (5) activity network to 315.5 seconds for a fifty (50) activity network. This increase in iteration time, combined with the less predictable convergence rate of the branch and bound method without Driebeek's penalty, limits the method. Needless to say, the nine (9) networks were not solved in a reasonable amount of time (less than one day) with a micro-computer.

Man count integer problem networks of fifty (50) activities can be integer optimized on micro-computers with 64K bytes CPU memories; but

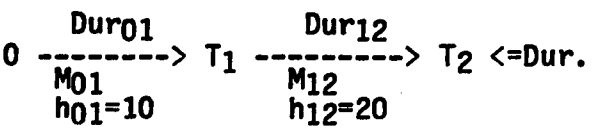
for the supervision cost integer problem, the time of execution of the programs might limit the applicability of the methods to larger and faster machines.

FURTHER RESEARCH

From the program listings and example solutions, it might appear that the simplex and simplex related algorithms consistently converge to a unique solution in a fixed number of iterations depending on the formulation of the problem. In actual practice, the convergence rate is a matter of precision depended on the choice of the program variable SM#.

Using the minimum man count problem as an example, a small two (2) activity network can be formulated as below:

T_i - Node time for node i
Dur_{ij} - Duration of activity ij
M_{ij} - Men assigned to complete activity ij
h_{ij} - Mandays to complete activity ij
Dur. - Project Duration.



and then rewritten as a convex programming problem:

$$\text{Minimize} \quad M_{01}+M_{12}+T_2$$

$$\text{Subject to:} \quad M_{01}*(T_1-0)>=h_{01}=10$$
$$M_{12}*(T_2-T_1)>=h_{12}=20$$

$$M_{01},M_{12},T_1,T_2>=0$$

in which the man count is balanced against the duration of the project.

If the dual simplex method with nonlinear constraints is used to solve the problem, the precision, or the value at which the algorithm's code considers variables zero (0), can be varied from 1E-06 to 1E-16 and a series of different "solutions" can be found for the same problem which are shown in table 53.

Table 53. Solutions to a man count problem for six levels of precision or values of the variable SM# in the dual simplex method with convex constraints derived with Gordian cut.

	Precision		
	SM#=1E-06	SM#=1E-08	SM#=1E-10
Itr.	23	31	37
Sec.	29	35	40
Obj.	15.26882723033593	15.26882723033592	15.26882723033592
T1	3.160411643603943	3.162161034172499	3.162263081914992
T2	7.629908652670576	7.634132055108864	7.634378420148291
M01	3.164143676732816	3.16239428616426	3.162292238421766
M12	4.474774900932526	4.472300889062794	4.47215657176586

Table 53. Continued

	Precision		
	SM#=1E-12	SM#=1E-14	SM#=1E-16
Itr.	43	78	82
Sec.	44	86	89
Obj.	15.26882723033592	15.26882723106905	15.26882723106905
T1	3.162275837853207	3.16227767030145	3.162277651051399
T2	7.634409215676056	7.634470885385542	7.634470866128593
M01	3.162279482483552	3.162277650035304	3.16227766928536
M12	4.472138532176309	4.4720786956482	4.472078695655098

If the precision is lowered too far the algorithm will stop at a nonoptimal solution. If the precision is increased, the number of iterations also increased, until a precision is reached for which the algorithm will never terminate or the problem will be found infeasible.

In the small example the iteration count was doubled as the precision was increased from 1E-06 to 1E-16. Unfortunately, in larger

problems, the iteration count may become excessive without reaching a satisfactory solution.

In the case of the man count problem, a closed form solution is also available by using the cost function:

$$\min.\ F(Dur_{01},Dur_{12})=\frac{h_{01}}{Dur_{01}}+\frac{h_{12}}{Dur_{12}}+(Dur_{01}+Dur_{12})=\frac{10}{Dur_{01}}+\frac{20}{Dur_{12}}+(Dur_{01}+Dur_{12})$$

and setting the partials equal to zero (0):

$$\frac{\&F(Dur_{01},Dur_{12})}{\&Dur_{01}}=\frac{10}{Dur_{01}\hat{}2}+Dur_{01}=0$$

$$\frac{\&F(Dur_{01},Dur_{12})}{\&Dur_{12}}=\frac{20}{Dur_{12}\hat{}2}+Dur_{12}=0$$

The duration of the activities and the value of the objective can be found as:

$$Dur_{01}=\sqrt{10}\qquad Dur_{12}=\sqrt{20}\qquad F(Dur_{01},Dur_{12})=2*\sqrt{10}+2*\sqrt{20}$$

which translates to the following solution for the convex programming problem:

Obj.	15.2688273835449
T_1	3.162277698516846
T_2	7.634413719177246
M_{01}	3.162277698516846
M_{12}	4.4721360206604

Using the closed form solution for comparison, the effect of precision can be observed in table 53 as the number of significant digits in the convex programming solution increase from three (3) to nine (9).

Further study is needed in the area of precision or the computer application of the closed form methods before the programs presented here can be effectively expanded to larger problems.

BIBLIOGRAPHY

1. Richter, I.; Mitchell, R. S. "Hand Book of Construction Law and Claims"; Reston Publishing Company: Reston, VA, 1982.

2. O'Brien, J. J. "CPM in Construction Management"; McGraw-Hill Book Company: New York, NY, 1971.

3. Federal Construction Management Task Force "President's Private Sector Survey on Cost Control: Report on Federal Construction Management"; U.S. Government Printing Office: Washington, DC, 1983.

4. Gantt, H. L. "Organization for Work"; Harcourt, Brace and Company: New York, NY, 1919.

5. Walker, M. R.; Sayer, J. S. "Project Planning and Scheduling"; E. I. du Pont de Nemoirs and Company: Wilmington, DE, March 1959, Report 6959.

6. "PERT Summary Report, Phase 1"; Bureau of Naval Weapons, Special Projects Office, Department of the Navy: Washington, DC, July 1958.

7. Elmaghraby, S. E. "Activity Network: Project Planning: Project Planning and Control by Network Models"; John Wiley and Sons: New York, NY, 1977.

8. Hillier, S. E.; Lieberman, G. J. "Introduction to Operation Research"; Holden-Day Inc.: San Francisco, CA, 1980.

9. Kelley, J. E. "Critical-Path Planning and Scheduling: Mathematical Basis"; Operation Research 1961, 9, 296-320.

10. Charnes, A.; Cooper, W. W. "A Network Interpretation and a Directed Subdual Algorithm for Critical Path Scheduling"; Journal of Industrial Engineering 1962, 13, 213-218.

11. Fulkerson, D. R. "A Network Flow Computation for Project Cost Curves"; Management Science 1961, 7, 167-178.

12. Kapur, K. C. "An Algorithm for Project Cost-Duration Analysis Problem with Quadratic and Convex Cost functions"; AIIE Transactions 1973, 5, 314-322.

13. Wagner, H. M. "An Integer Linear-Programming Model for Machine Scheduling"; Naval Research Logistic Quarterly 1959, 6, 131-139.

14. Bowman, E. H. "The Schedule-Sequencing Problem"; Operations Research 1959, 7, 621-624.

15. Manne, A. S. "On the Job-shop Scheduling Problem"; Operations Research 1960, 8, 219-223.

16. Pritsker, A. A. B.;Watters, L. J.; Wolfe, P. M. "Multiproject Scheduling with Limited Resources: A Zero-one Programming Approach"; Management Science 1974, 16, 93-108.

17. Patterson, J. H.; Huber, W. D. "A Horizon-varying, Zero-one Approach to Project Scheduling"; Management Science 1974, 20, 998-990.

18. Petrovic, R. "Optimization of Resources Allocation in Project Planning"; Operations Research 1968, 16, 559-568.

19. "Project Management Systems IV Resource Allocation Processor Program Descriptions and Operations Manual"; International Business Machines Corporation: White Plains, NY, 1975, IBM Technical Manual NO. 5734-XP4.

20. "Management Scheduling and Control System Application Manual"; McDonald Douglas Automation Company: St. Louis, MO, 1980, MacAuto Technical Manual No. M5979078.

21. "Basic Manual Project/2"; Project Software and Development: Cambridge, MA, 1979, PSDI Technical Manual No. 2089.

22. "PREMIS Technical Manual"; SUN Information Services Company: Valley Forge, PA, 1980.

23. Pritsker, A. A. B.; Happ, W. W. "GERT: Graphical Evaluation and Review Technique, Part I. Fundamentals"; Journal of Industrial Engineering 1966, 17, 267-274.

24. Pritsker, A. A. B.; Pegden, C. D. "Introduction to Simulation and Slam"; John Wiley and Sons: New York, NY, 1979.

25 Wolfe, C. S. "Cutting Plane and Branch and Bound for Solving a Class of Scheduling Problems"; IIE Transactions 1984, 16, 50-58.

26 "Operations/DOS Reference Guide to Sr. Partner Portable Computer"; Panasonic Industrial Company: Secaucus, NJ, 1983.

27 "Personal Computer, Computer Language Series, BASIC Compiler"; International Business Machine Corporation: White Plains, NY, 1982.

28 Radcliffe, B. M.; Kawal, D. E.; Stephenson, R. J. "Critical Path Method"; Cahners Publishing Company, Inc.: Chicago, IL, 1967.

29 Smith, D. K. "Network Optimization Practice, A Computational Guide"; Ellis Horwood Limited: West Sussex, England, 1982.

30 Rich, R. P. "Internal Sorting Methods Illustrated with PL/1 Programs"; Prentice-Hall, Inc.: Englewood Cliffs, NJ, 1972.

31 Moder, J. J.; Phillips C. R. "Project Management with CPM and PERT"; Van Nostrand Reinhold Company: New York, NY, 1970.

32 Reynolds, L. G. "Economics, a General Introduction"; Irwin-Dorsey Limited: Georgetown, Canada, 1969.

33 Sposito, V. A. "Linear and Nonlinear Programming"; Iowa State University Press: Ames, IA, 1975.

34 Hu, T. C. "Integer Programming and Network Flows"; Addison-Wesley Publishing Company: Reading, MA, 1969.

35 Grossman, S. I. "Elementary Linear Algebra"; Wadsworth Publishing Company: Belmont, CA, 1980.

36 Charnes, A.; Cooper, W. W. "Management Models and Industrial Applications of Linear Programming"; John Wiley and Sons, Inc.: New York, NY, 1961.

37 Salkin, H. M. "Integer Programming"; Addison-Wesley Publishing Company: Reading, MA, 1975.

38 Luenberger, D. G. "Introduction to Linear and Nonlinear Programming"; Addison-Wesley Publishing Company: Reading, MA, 1973.

39 Shoup, T. E. "Numerical Methods for the Personal Computer"; Prentice-Hall, Inc.: Englewood Cliffs, NJ, 1983.

40 Kuhn, H. W.; Tucker, A. W. "Nonlinear Programming"; Proceedings of the Second Berkley Symposium on Mathematics, Statistics, and Probability; University of California Press: Berkeley, CA, 1950, 481-492.

41 Elmaghraby, S. E. "The Design of Production Systems"; Reinhold Publishing Corporation: New York, NY, 1966.

42 Kunzi, H. P.; Krelle, W. "Nonlinear Programming"; Blaisdell Publishing Company: Waltham, MA, 1966.

43 Owen, G. "Game Theory"; W. B. Saunders Company: Philadelphia, PA, 1968.

44 Ignizio, J. P. "Linear Programming in Single and Multiple Objective Systems"; Prentice-Hall, Inc.: Englewood Cliffs, NJ, 1982.

45 Land, A. H.; Morton, G. "An Inverse-Basis Method for Beale's Quadratic Programming Algorithm"; Management Science 1973, 19, 510-516.

46 Witzgall, C. "An All-integer Programming Algorithm with Parabolic Constraints"; Journal of the Society of Industrial and Applied Mathematics 1963, 11, 855-871.

47 Leithold, L. "The Calculus with Analytic Geometry"; Harper and Row, Publishers: New York, NY, 1968.

48 Akgul, M. "Topics in Relaxation and Ellipsoidal Methods"; Pitman Publishing, Inc.: Marshfield, MA, 1984.

49 Muth, J. F.; Thompson, G. L. "Industrial Scheduling"; Prentice-Hall, Inc.: Englewood Cliffs, NJ, 1963.

50 Wilson, R. "Stronger Cuts in Gomory's All-integer Programming Algorithm"; Operations Research 1967, 15, 155-157.

51 Zoints, S. "Linear and Integer Programming"; Prentice-Hall, Inc.: Englewood Cliffs, NJ, 1974.

52 Land, A. H.; Doig, A. G. "An Automatic Method of Solving Discrete Programming Problems"; Econometrica 1960, 28, 497-520.

53 Driebeek, N. J. "An Algorithm For the Solution of Mixed Integer Programming Problems"; Management Science 1966, 12, 576-587.

54 Angier, N. "Folding the Perfect Corner"; Time 1984, 23, 63.

55 Geoffrion, A. M.; Marsten, R. E. "Integer Programming Algorithms: A Framework and State-of-the-art Survey"; Management Science 1972, 18, 465-401.

56 "Autonet, An Automatic Network Display Program"; California Computer Products, Inc.: Anaheim, CA, 1973.

57 "Application Manual, MAPS/TMAPS"; McDonnell Douglas Automation Company: Saint Louis, MO, 1979.

58 Goffin, J. "Variable Metric Relaxation Methods, Part II: The Ellipsoid Method"; Mathematical Programming 1984, 30, 147-162.

59 Barankin, E.; Dorfman, R. "A Method for Quadratic Programming", Econometrica 1956, 24, 340.

APPENDIX A: PRECEDENCE DIAGRAMMING METHOD

The CPM arrow diagramming technique has proved cumbersome to use in actual applications. This is primarily due to the fact that the CPM in arrow diagram, the activities can only be identified by a completed diagram's nodes numbers and that the relationships between activities are limited to only the simple linkage at the nodes.

Theory of Precedence Diagramming Method

To enhance CPM, the precedence diagramming method[28] (PDM) was developed which utilized a "precedence diagram" rather that an arrow diagram in which the activities are assigned to the nodes and the relationships between activities are represented by the arrows.

Relationships and Lags

In PDM, a precedence diagram is constructed which represents the relationships between scheduled activities. Almost all scheduled relationships can be expressed as one of four (4) types -- "finish to start", "start to start", "finish to finish", and "start to finish".

In the finish to start relationship (N), or what can be considered the "normal" relationship in the arrow diagramming technique, the preceding activity must be finished before the succeeding activity starts. In the start to start relationship (S), the preceding activity must be started before the succeeding activity starts. In the finish to finish relationship (F), the preceding activity must be finished before the succeeding activity may finish; and in the start to finish

relationship (B), which is rarely used and is "backward" to the normal relationship, the preceding activity must be started before the succeeding activity finishes.

In addition to the relationship type, there is "lag". Often a preceding activity must be finished a number of work increments or a lag time before the succeeding activity can start. This is expressed as a duration which is assigned to each relationship. When the lag is combined with the relationship type, almost any schedule can be modeled using a PDM diagram.

Identification Numbers

Using the ten activity network from the CPM example, it is possible to construct the relationship table A-1. In addition to activity

Table A-1. Table of relationships

#	Preceding Activity	ID	Succeeding activity	ID	Relationship Type	Lag
1	Activity #3	1	Activity #1	2	start to start	0
2	Activity #3	1	Activity #4	3	end to start	0
3	Activity #3	1	Activity #7	8	end to start	0
4	Activity #3	1	Activity #9	9	end to start	0
5	Activity #3	1	Activity #8	4	start to start	0
6	Activity #3	1	Activity #10	5	start to start	0
7	Activity #1	2	Activity #6	10	end to start	0
8	Activity #4	3	Activity #2	6	end to start	0
9	Activity #8	4	Activity #2	6	end to start	0
10	Activity #4	3	Activity #5	7	end to start	0
11	Activity #8	4	Activity #5	7	end to start	0
12	Activity #10	5	Activity #6	10	finish to finish	0
13	Activity #5	7	Activity #6	10	end to start	0
14	Activity #7	8	Activity #6	10	end to start	0
15	Activity #2	6	Activity #6	10	finish to finish	0
16	Activity #9	9	Activity #6	10	finish to finish	0

decriptions such as "Activity #3", a set of identification numbers (ID) is assigned to each activity which for the following program must be ascending order within each pair of activities forming a relationship.

This ascending order prevents logical loops in the PDM network.

Precedence Diagram

To convert the table to a precedence diagram as shown in figure A-1,

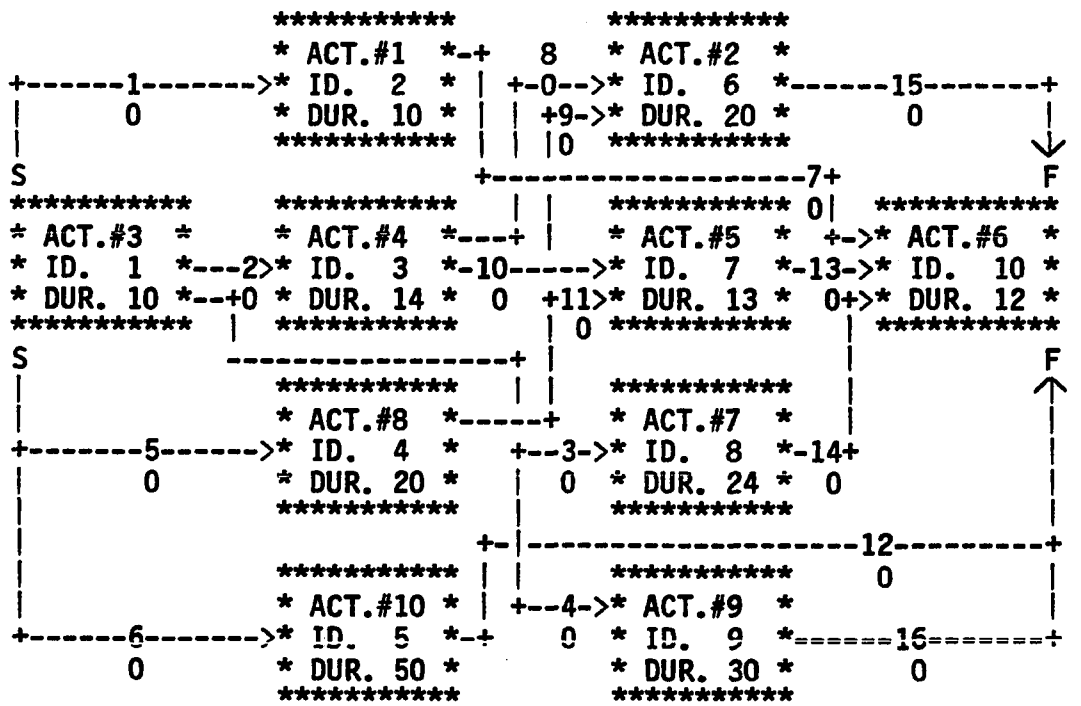


Figure A-1. Precedence diagram network

each activity is drawn as a square or node. Then, between each node is drawn the relationship arrows. To simplify the diagram, each relationship arrow is numbered with a relationship number, a lag, and a head and tail representing the relationship direction or type. This format, often called "activity on node" as compared to the "activity on

arrow", is the graphic basis of PDM.

In large computer packages such as IBM's PMSIV[19] or PROJACS this diagram is printed by the computer. Software plotting packages such as Calcomp's Autonet[56] or MacAuto's T-Map[57] can also draw the networks on mechanical plotters from the precedence table.

Work Day Calendars

In the original CPM, all activity durations, lags, and schedule times were expressed as work increment defined by points on the CPM time scale. In actual practice, schedules are expressed in work days and "calendar dates" in which the CPM time scale points are calendar dates.

In the PDM method, calendar dates (01JUN83, e.g.) are assigned to each CPM time scale point. The new time scale with calendar date points is then called the "base calendar".

To add another dimension to the time scale, not all activities can be scheduled to work on every day of the base calendar. To convert activity durations, which might be restricted to any one of a number of different "worked day calendars", to a base calendar, a set of worked day calendars must be defined.

These calendars, numbering one (1) through three (3) in the computer program to follow, are used to assign a "worked day" flag ("Y","N") to each day worked in the base calendar, and a work day calendar number to each activity duration and relationship lag.

Schedule dates

Once a set of worked day calendars has been defined, schedule dates

can be used to force activities to start after (01JUN83S, e.g.) or finish before (01JUN83F, e.g.) particular dates. The most often used of these is the date after which the project can start and the date before which the project must end.

PDM Reports

The processing and output for the PDM method is essentially the same as the CPM program. The only difference in the PDM program is that the CPM time scale is converted to calendar dates so that early start and late finishes are also listed as dates.

Precedence Diagramming Method BASIC Code Inputs and Reports

As a demonstration of the PDM, the ten (10) activity network will be used which was optimized by the branch and bound method at fifty (50) days for the example 11 minimum project man count problem. Table A-2 optimal integer man counts and optimal integer activity durations for

Table A-2. Optimal integer man count for 50 day project

Activity #	Men scheduled	Days scheduled	Mandays required
1	1	10	10
2	1	20	20
3	3	10	30
4	3	14	40
5	4	13	50
6	5	12	60
7	3	24	70
8	4	20	80
9	3	30	90
10	2	50	100

each of the ten (10) activities of the precedence diagram in figure A-1.

Main Menu Screen

On initial execution of the code, the menu screen will appear as in figure A-2. The screen lists the system options and pauses the program

```
PRECEDENCE DIAGRAM NETWORKING MENU

M-RETURN TO MENU

* DATA FILES *
C-CALENDAR  A-ACTIVITY  R-RELATION
U-UPDATE

* REPORTS *
RE-BY STARTS  RA-BY ACTIVITY
RS-RESOURCES  BC-BARCHART

* FILE MAINTENANCE *
E-EDIT  L-LIST  I-INSERT  D-DELETE
S-SAVE  F-FETCH

OPTION ?
```

Figure A-2. Precedence diagramming method main menu screen

execution for one of eleven OPTION ? entries ("M","C","A","R","U","RE", "RA","RS","BC","S","F").

The "M" option returns the program to the menu screen from any other screen in the program. The "C" option lists and edits through the computer screen the file containing the work day calendars. The "A" option lists and edits the activity file. The "R" option lists and edits the relationships file. The "U" option process the PDM model. The "S" and "F" save and fetch the inputed data to and from the computer disk as the ASCI file "DATA". The "RE", "RA", "RS", and "BC" options list to the computer screen the critcal path listing in early

start order, the critcal path listing in input file order, the resource report, and the Gantt or bar chart, respectively.

Calendar Input File

Option "C" calls the calendar file input screen as shown in figure A-3. When the screen is first displayed, the calendar file may be

```
                    CALENDAR
          START DAY DATE  01JUN85
         DAY  DATE      NO.1 NO.2 NO.3
         1    01JUN85   N    Y    Y
         2    02JUN85   N    Y    Y
         3    03JUN85   Y    Y    Y
         4    04JUN85   Y    Y    Y
         6    06JUN85   Y    Y    Y
         7    07JUN85   Y    Y    Y
         8    08JUN85   N    Y    Y
         9    09JUN85   N    Y    Y
         10   10JUN85   Y    Y    Y
         11   11JUN85   Y    Y    Y
         12   12JUN85   Y    Y    Y
         13   13JUN85   Y    Y    Y
         14   14JUN85   Y    Y    Y
         15   15JUN85   N    Y    Y
        I5    05JUN85   Y    Y    Y
        D6
         6    06JUN85   Y    Y    Y
         _

   OPTION ?
```

Figure A-3. Calendar input screen

either listed "L" or edited "E". If "L" is entered for the OPTION ?, then LINE ? prompts for the file line number from which the screen is to start listing the calendars entries. The file is listed one screen at a time followed by a prompt to continue to the next screen or return to main menu.

The list option does not allow changes to be made in the calendar

file and only lists the file. If changes are to be made to the calendar file, then "E" or blank is entered at the initial OPTION ?.

The calendar screen assigns a calendar date to each time scale point, sets the project start date, and defines three work day calendars. When the screen is first entered using the edit option, the cursor will be on the START DAY DATE. If the proper value is displayed press enter, otherwise type the start date in day/month/year format (01JUN85, e.g.) and enter. The cursor will then move under the DAY column where a time scale point number can be typed and enter. The cursor will then move to the CALENDAR DATE column where the calendar date can be typed and enter. At this point, the cursor will be under the NO.1 column so that a "N", if the day is not a work day, or a "Y", if the day is a work day, can be typed and enter. The worked day calendars can be entered until the cursor returns to the DAY column.

If an error is made, the line can be retyped. For deletion of a date in the sequence, type a "D" in the DAY column and then the day number. To add a date in sequence, type "I" in the DAY column and then type in a new line as shown in figure A-3.

Activity Input File

Option "A" calls the activity input screen. When the option is first entered, the activity file can be either listed "L" or edited "E" as in the calendar option. If "E" or blank is entered, the screen is set for editing as in figure A-4.

```
                              ACTIVITIES
                        MAXIMUM ID NUMBER   10
 ID  DESCRIPTION                      SCH.DATE DUR CAL RESOURCES TYPE
 1   ACTIVITY 7                       01JUN85S 10  1   30        1
 2   ACTIVITY 5                                10  1   10        1
 3   ACTIVITY 8                                14  1   42        1
 4   ACTIVITY 12                               20  1   80        1
 5   ACTIVITY 14                               50  1   100       1
 6   ACTIVITY 6                                20  1   20        1
 7   ACTIVITY 9                                13  1   52        1
 8   ACTIVITY 11                               24  1   72        1
 9   ACTIVITY 13                               30  1   90        1
 10  ACTIVITY 10                               12  1   60        1
 11  ERROR
D11
_

OPTION ?
```

Figure A-4. Activity input screen

On entering the screen, the cursor will be at MAXIMUM ID NUMBER _. This number limits the number of activity IDs that the system needs to process and so greatly reduces the processing time when lower numbers are selected. The number can be changed at any time and is initially set at the maximum capacity of the system.

After typing the maximum identification number and entering, the cursor will move to the ID column so that the activity data; identification number; description; schedule date followed by start "S" or finish "F"; activity duration in work days; activity calendar number as defined on the calendar screen; total resources assigned to the activity; and the type identification of the resource (1-5) can be entered.

If an error is made, the line is reentered. If a line needs to be

deleted, enter "D" in the ID column and the activity identification number. Since activities are uniquely defined by ID numbers, the "I" option is not available on the "A" screen.

Relationship Input File

Option "R" calls the relationships screen. When the screen is first entered, the file can be either listed "L" or edited "E" as in the calendar option. If "E" or blank is entered, the screen is set for editing as in figure A-5.

```
                    RELATIONSHIPS
NUMBER OF RELATIONSHIPS                16
REL.  PRED.   SUCC.  TYPE  LAG  CAL.
1     1       2      S     0    0
2     1       3      N     0    0
3     1       8      N     0    0
4     1       9      N     0    0
5     1       4      S     0    0
6     1       5      S     0    0
7     2       10     N     0    0
8     3       6      N     0    0
9     4       6      N     0    0
10    3       7      N     0    0
11    4       7      N     0    0
12    5       10     F     0    0
13    7       10     N     0    0
14    8       10     N     0    0
15    6       10     F     0    0
16    9       10     F     0    0
_

OPTION ?
```

Figure A-5. Relationships input screen

On entering the screen, the cursor will be at NUMBER OF RELATIONSHIPS. This entry limits the number of relationships the system will process and can be changed at any time. When first entering the

screen, the number will be set at the maximum number of relationships the system can hold.

After typing the current number of relationships and entering, the cursor will move to the REL column. Now the relationships data, the relationship identification number, the predecessor identification number, the successor identification number, the relationship type ("S"-start to start, "N"-finish to start, "F"-finish to finish, and "B"-finish to start), the lag in working days, and the calendar of the lag can be entered.

Deletions and insertions are the same as for the calendar screen.

Processing the PDM Network

The calendar, activity, and relationships files complete the input of the PDM network data. To process the network, select "U" from the master menu and enter. The results are listed on five output screens, two (2) of which have the same format but a different listing sequence.

Critical Path Listing in Early Start Sequence

Option "RE" calls the critical path listing in early start sequence. When the screen is first entered, a second option of LINE ? pauses the program so that the listing can be started on any report line number entered.

The report consists of the project duration in the CPM time scale increments and two (2) output lines for each activity. Line one (1) is the activity identification; the description of the activity; the schedule date, start or finish; the duration in worked days; the

activity calendar number; the total resources for the activity; the type of resource; the total float in work days; the free float in work days; and an asterisk for critical activities. The second line is the CPM time scale early start time and the early start date; the early finish time and date; late start time and date; and late finish time and date.

The report is listed one screen at a time as shown in figure A-6,

```
                          CRITICAL PATH TABLE
                        PROJECT DURATION   70
ID  DESCRIPTION                SCH.DATE DUR CAL RESOURCES TYPE TF  FF  CR
        EARLY START    EARLY FINISH    LATE  START    LATE FINISH
1   ACTIVITY 7                  01JUN83S 10  1   30       1    0   0   *
        3 -03JUN85     14-14JUN85     3 -03JUN85     14-14JUN85
2   ACTIVITY 5                         10  1   10       1    28  28
        3 -03JUN85     14-14JUN85     41-11JUL85     54-24JUL85
4   ACTIVITY 12                        20  1   80       1    5   5
        3 -03JUN85     28-28JUN85     10-10JUN85     35-05JUL85
5   ACTIVITY 14                        50  1   100      1    0   0   *
        3 -03JUN85     70-09AUG85     3 -03JUN85     70-09AUG85
3   ACTIVITY 8                         14  1   42       1    1   0
        17-17JUN85     34-04JUL85     18-18JUN85     35-05JUL85
8   ACTIVITY 11                        24  1   72       1    4   4
        17-17JUN85     48-18JUL85     21-21JUN85     54-24JUL85
9   ACTIVITY 13                        30  1   90       1    10  10
        17-17JUN85     56-26JUL85     31-01JUL85     70-09AUG85
6   ACTIVITY 6                         20  1   20       1    6   6
        35-05JUL85     62-01AUG85     45-15JUL85     70-09AUG85
7   ACTIVITY 9                         13  1   52       1    1   1
        35-0RJUL85     53-23JUL85     38-08JUL85     54-24JUL85
10  ACTIVITY 10                        12  1   60       1    0   0   *
        55-25JUL85     70-09AUG85     55-25JUL85     70-09AUG85

OPTION ?
```

Figure A-6. Critical path listing screen

followed by a prompt to continue to the next screen by entering blank or to return to main menu by entering any option.

Critical Path Table Listing in Input Sequence

The "RA" option calls the same report and screen format as the "RE" option above with the exception that the activities are listed in ID number sequence. The report is used primarily to check the validity of the data input.

Resource Listing

The "RS" option calls for a resource listing by CPM time scale point and date. When the screen is first entered, a second option of LINE ? pauses the program so that the report can be started on any line number entered.

Figure A-7 is the resource screen for the example problem. The

RESOURCES

START DAY DATE 01JUN85

DAY	DATE	TYPE 1	TYPE 2	TYPE 3	TYPE 4	TYPE 5
1	01JUN85	0	0	0	0	0
2	02JUN85	0	0	0	0	0
3	03JUN85	10	0	0	0	0
4	04JUN85	10	0	0	0	0
5	05JUN85	10	0	0	0	0
6	06JUN85	10	0	0	0	0
7	07JUN85	10	0	0	0	0
8	08JUN85	0	0	0	0	0
9	09JUN85	0	0	0	0	0
10	10JUN85	10	0	0	0	0
11	11JUN85	10	0	0	0	0
12	12JUN85	10	0	0	0	0
13	13JUN85	10	0	0	0	0
14	14JUN85	10	0	0	0	0
15	15JUN85	0	0	0	0	0

OPTION ?

Figure A-7. Resource report screen

report consists of the start date of the project and a listing of the daily usage of each of the five (5) types of resources.

The report is listed one (1) screen at a time followed by a prompt to continue to the next screen by entering blank or to return to main menu by entering any option.

If all the screens of the resource listing are displayed and summarized, the resources usage, which in this case would be men per day, would be as in table A-3. Although this is not a report of the PDM

Table A-3. Resource usage report summary

Date	Men	Date	Men	Date	Men	Date	Men
16JUN85	0	01JUL85	11	16JUL85	13	31JUL85	8
17JUN85	15	02JUL85	11	17JUL85	13	01AUG85	8
18JUN85	15	03JUL85	11	18JUL85	13	02AUG85	7
19JUN85	15	04JUL85	11	19JUL85	10	03AUG85	0
20JUN85	15	05JUL85	13	20JUL85	0	04AUG85	0
21JUN85	15	06JUL85	0	21JUL85	0	05AUG85	7
22JUN85	0	07JUL85	0	22JUL85	10	06AUG85	7
23JUN85	0	08JUL85	13	23JUL85	10	07AUG85	7
24JUN85	15	09JUL85	13	24JUL85	6	08AUG85	7
25JUN85	15	10JUL85	13	25JUL85	11	09AUG85	7
26JUN85	15	11JUL85	13	26JUL85	11	10AUG85	0
27JUN85	15	12JUL85	13	27JUL85	0	11AUG85	0
28JUN85	15	13JUL85	0	28JUL85	0	12AUG85	0
29JUN85	0	14JUL85	0	29JUL85	8	13AUG85	0
30JUN85	0	15JUL85	13	30JUL85	8	14AUG85	0

program listed here, the table and following graph, figure A-8, are the mandays curve for the minimum activity man counts needed to complete the example ten (10) activity network in fifty (50) days.

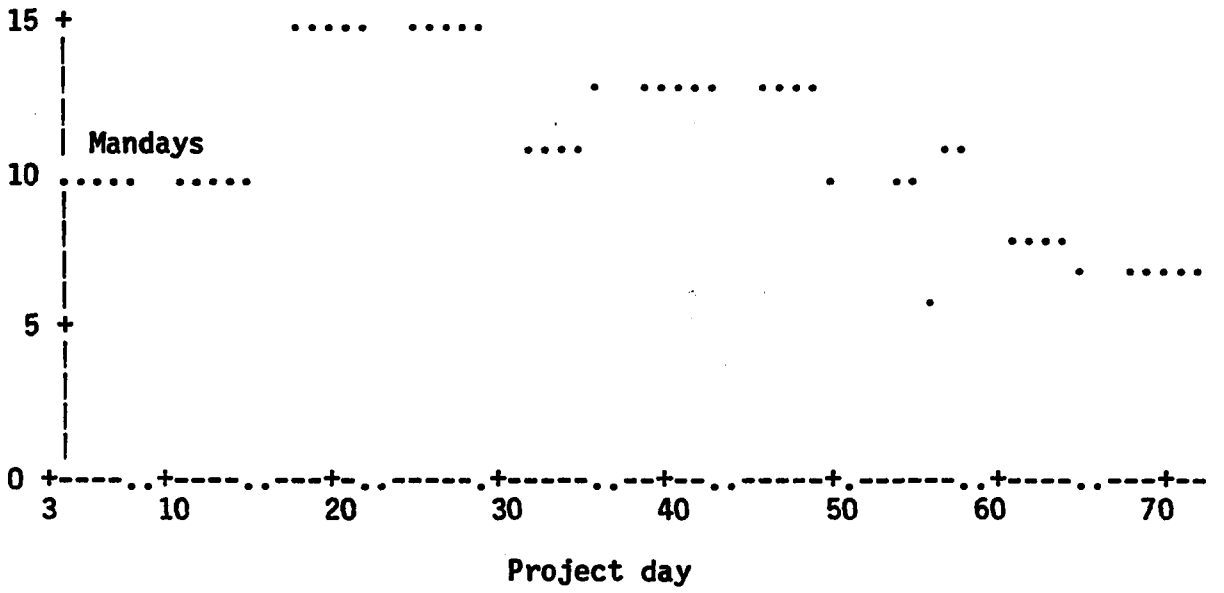


Figure A-8. Mandays curve

Bachart or Gantt Chart

The "BC" option calls the Gantt or bar chart. Since most barcharts would exceed not only the height, but also the width of the computer display screen, the intermediate options of START ? and LINE ? allow the screen display of the chart to be started at any selected time scale point and at any activity line. The Barchart for the example precedence diagram is shown in figure A-9.

The report consists of a time scale, a list of activity identification numbers, activity descriptions, and a set of bars representing the scheduled times for the listed activity with a "C" for critical, "X" for noncritical, and "-" for free float.

```
                          BARCHART
                        01JUN85        15JUN85        29JUN85
ID  DESCRIPTION         1      08JUN85        22JUN85        06JUL85
1   ACTIVITY 7            CCCCC  CCCCC
                        |      |      |      |      |      |      |
2   ACTIVITY 5            XXXXX  XXXXX  -----  -----  -----  -----
                        |      |      |      |      |      |      |
4   ACTIVITY 12           XXXXX  XXXXX  XXXXX  XXXXX  -----
                        |      |      |      |      |      |      |
5   ACTIVITY 14           CCCCC  CCCCC  CCCCC  CCCCC  CCCCC  CCCCC
                        |      |      |      |      |      |      |
3   ACTIVITY 8                           XXXXX  XXXXX  XXXX-
                        |      |      |      |      |      |      |
8   ACTIVITY 11                          XXXXX  XXXXX  XXXXX  XXXXX
                        |      |      |      |      |      |      |
9   ACTIVITY 13                          XXXXX  XXXXX  XXXXX  XXXXX
                        |      |      |      |      |      |      |
6   ACTIVITY 6                                                X  XXXXX
                        |      |      |      |      |      |      |
7   ACTIVITY 9                                                X  XXXXX
                        |      |      |      |      |      |      |
10  ACTIVITY 10
                        |      |      |      |      |      |      |

OPTION ?
```

Figure A-9. Barchart screen

Precedence Diagramming Method BASIC Code

The BASIC program code for the PDM method is rather complex, depending on multiple branching, multiple subroutines, and extensive reuse of report headings and output lines. Because of these complexities, the code will only be listed rather than documented in detail.

Precedence Diagramming Method -- MAIN-PDM

```
1 REM                 * PRECEDENCE DIAGRAMMING METHOD *
2 REM-----------------------------MAIN-PDM-------------------------------

3 ACT=100
```

```
4 CAL=1464
5 REL=300
6 DIM A(ACT,10)
7 DIM A$(ACT)
8 DIM H(5)
9 DIM R(REL,5)
10 DIM S$(ACT)
11 DIM S(REL)
12 DIM Z$(CAL)
13 EN=0
14 NACT=100
15 NREL=0
16 U=0
17 CLS
18 PRINT ""
19 LOCATE 2,20:PRINT  "PRECEDENCE DIAGRAM NETWORKING MENU"
20 LOCATE 4,28:PRINT        "M-RETURN TO MENU"
21 LOCATE 6,29:PRINT         "* DATA FILES *"
22 LOCATE 7,19:PRINT  "C-CALENDAR  A-ACTIVITY  R-RELATION"
23 LOCATE 8,31:PRINT           "U-UPDATE"
24 LOCATE 10,30:PRINT         "* REPORTS *"
25 LOCATE 11,22:PRINT "RE-BY STARTS  RA-BY ACTIVITY"
26 LOCATE 12,22:PRINT   "RS-RESOURCES  BC-BARCHART"
27 LOCATE 14,26:PRINT     "* FILE MAINTENANCE *"
28 LOCATE 15,19:PRINT "E-EDIT  L-LIST  I-INSERT  D-DELETE"
29 LOCATE 16,27:PRINT       "S-SAVE  F-FETCH"
30 GOSUB 623
31 GOSUB 623
32 LOCATE 21,9:INPUT "",P$
33 U=U+1
34 I=1
35 V=2
36 X=1
37 Y=CAL
38 Z=1
39 IF P$="C" THEN GOSUB 126
40 V=3
41 X=2
42 Y=NACT
43 Z=2
44 IF P$="A" THEN GOSUB 126
45 V=4
46 X=3
47 Y=NREL
48 Z=3
49 IF P$="R" THEN GOSUB 126
50 V=0
51 X=4
```

```
52 Y=NACT
53 Z=4
54 I=2
55 IF P$="RE" THEN GOSUB 126
56 V=1
57 IF P$="RA" THEN GOSUB 126
58 X=5
59 Y=9999
60 Z=5
61 IF P$="RS" THEN GOSUB 126
62 X=6
63 Y=NACT
64 Z=6
65 IF P$="BC" THEN GOSUB 397
66 IF U=1 THEN 33
67 IF P$="U" THEN GOSUB 156
68 IF P$<>"F" THEN 100
69 OPEN "I",#1,"DATA"
70 FOR C=1 TO CAL
71 INPUT #1,Z$(C)
72 NEXT C
73 FOR R=1 TO REL
74 FOR I=1 TO 5
75 INPUT #1,B$
76 R(R,I)=VAL(B$)
77 NEXT I
78 NEXT R
79 FOR A=1 TO ACT
80 INPUT #1,A$(A)
81 INPUT #1,S$(A)
82 IF S$(A)="X" THEN S$(A)="        "
83 INPUT #1,B$
84 S(A)=VAL(B$)
85 FOR I=1 TO 10
86 INPUT #1,B$
87 A(A,I)=VAL(B$)
88 NEXT I
89 NEXT A
90 INPUT #1,B$
91 NACT=VAL(B$)
92 INPUT #1,B$
93 NREL=VAL(B$)
94 INPUT #1,B$
95 BEGIN=VAL(B$)
96 INPUT #1,B$
97 EN=VAL(B$)
98 INPUT #1,B$
99 CLOSE #1
```

```
100 IF P$<>"S" THEN 125
101 OPEN "O",#1,"DATA"
102 FOR C=1 TO CAL
103 PRINT #1,Z$(C)
104 NEXT C
105 FOR R=1 TO REL
106 FOR I=1 TO 5
107 PRINT #1,STR$(R(R,I))
108 NEXT I
109 NEXT R
110 FOR A=1 TO ACT
111 PRINT #1,A$(A)
112 IF S$(A)="         " THEN PRINT #1,"X"
113 IF S$(A)<>"         " THEN PRINT #1,S$(A)
114 PRINT #1,STR$(S(A))
115 FOR I=1 TO 10
116 PRINT #1,STR$(A(A,I))
117 NEXT I
118 NEXT A
119 PRINT #1,STR$(NACT)
120 PRINT #1,STR$(NREL)
121 PRINT #1,STR$(BEGIN)
122 PRINT #1,STR$(EN)
123 PRINT #1,B$
124 CLOSE #1
125 GOTO 16
126 IF P$<>"L" AND I=1 THEN GOSUB 147
127 IF P$<>"L" AND I=2 THEN GOSUB 131
128 IF P$="L" THEN GOSUB 131
129 IF P$="L" OR P$="E" THEN 126
130 RETURN
131 U=1
132 ON X GOSUB 596,600,604,608,613,616
133 GOSUB 627
134 IF P$<>"" AND G=99999! THEN RETURN
135 IF G<=0 OR G>Y THEN G=1
136 FOR W=G TO Y
137 ON Z GOSUB 345,349,353,359,377,405
138 IF L<19 THEN 143
139 GOSUB 623
140 LOCATE 21,9:INPUT "",P$
141 IF P$<>"" THEN RETURN
142 ON X GOSUB 596,600,604,608,613,616
143 NEXT W
144 GOSUB 623
145 LOCATE 21,9:INPUT "",P$
146 RETURN
147 U=1
```

```
148 W=0
149 ON X GOSUB 596,600,604,608,613,616
150 GOSUB 623
151 ON V-1 GOSUB 429,479,531
152 IF G=99999! AND P$<>"" THEN RETURN
153 IF W<Y THEN W=W+1
154 IF L<19 THEN 150
155 GOTO 149
156 CLS
157 FOR D=1 TO CAL
158 IF B$=MID$(Z$(D),4,7) THEN 161
159 NEXT D
160 D=1
161 BEGIN=D
162 FOR A=1 TO NACT
163 IF ""=MID$(S$(A),1,1) THEN 174
164 A(A,3)=BEGIN
165 A(A,4)=CAL
166 IF "       "=MID$(S$(A),2,7) THEN 174
167 FOR G=BEGIN TO CAL
168 D=G
169 H$=MID$(S$(A),2,7)
170 IF H$=MID$(Z$(G),4,7) THEN 172
171 NEXT G
172 IF "S"=MID$(S$(A),1,1) THEN A(A,3)=D
173 IF "F"=MID$(S$(A),1,1) THEN A(A,4)=D+1
174 NEXT A
175 FOR R=1 TO NREL
176 S(R)=R
177 NEXT R
178 IF NREL<=1 THEN 195
179 Y=NREL
180 Y=INT(Y/2)
181 FOR Z=1 TO NREL-Y
182 IF R(Z+Y,1)=0 THEN 193
183 H(1)=R(S(Z+Y),1)
184 H(2)=R(S(Z+Y),2)
185 H(3)=S(Z+Y)
186 FOR W=Z TO 1 STEP -Y
187 IF R(W,1)=0 THEN 193
188 IF R(S(W),1)<H(1) THEN 192
189 IF R(S(W),1)=H(1) AND R(S(W),2)<H(2) THEN 192
190 S(W+Y)=S(W)
191 NEXT W
192 S(W+Y)=H(3)
193 NEXT Z
194 IF Y>1 THEN 180
195 G=1
```

```
196 FOR I=1 TO NACT
197 IF ""=MID$(S$(I),1,1) THEN 241
198 DUR=A(I,5)
199 CL=A(I,6)
200 NOW=A(I,3)
201 GOSUB 665
202 A(I,3)=NOW+CDUR
203 IF NREL=0 THEN 241
204 FOR F=G TO NREL
205 R=S(F)
206 IF R(R,1)<I OR R(R,1)=0 THEN 239
207 IF R(R,1)>I THEN 240
208 J=R(R,2)
209 IF J>NACT THEN 240
210 NOW=A(I,3)
211 X=R(R,3)
212 IF X<>1 AND X<>3 THEN 217
213 CL=A(I,6)
214 DUR=A(I,5)
215 GOSUB 676
216 NOW=NOW-CDUR
217 CL=R(R,5)
218 DUR=R(R,4)
219 IF DUR>=0 THEN 224
220 DUR=-DUR
221 GOSUB 676
222 NOW=NOW-CDUR
223 GOTO 226
224 GOSUB 665
225 NOW=NOW+CDUR
226 IF X<>2 AND X<>3 THEN 238
227 CL=A(J,6)
228 DUR=A(J,5)
229 GOSUB 676
230 X=NOW
231 NOW=NOW-CDUR
232 GOSUB 665
233 IF X=NOW+CDUR THEN 238
234 DUR=DUR-1
235 NOW=X
236 GOSUB 676
237 NOW=NOW-CDUR
238 IF A(J,3)<NOW THEN A(J,3)=NOW
239 NEXT F
240 G=F
241 NEXT I
242 EN=0
243 FOR A=1 TO NACT
```

```
244 IF ""=MID$(S$(A),1,1) THEN 249
245 F=A(A,3)
246 G=A(A,4)
247 IF "F"=MID$(S$(A),1,1) AND F>G THEN F=G
248 IF EN<F THEN EN=F
249 NEXT A
250 FOR A=1 TO NACT
251 IF ""=MID$(S$(A),1,1) THEN 253
252 IF "F"<>MID$(S$(A),1,1) OR EN<A(A,4) THEN A(A,4)=EN
253 NEXT A
254 IF NREL=0 THEN 274
255 G=1
256 FOR X=1 TO NACT
257 J=NACT-X+1
258 IF ""=MID$(S$(J),1,1) THEN 273
259 IF G=NREL THEN 264
260 FOR Y=G TO NREL
261 IF J>=R(S(NREL-Y+1),1) THEN 263
262 NEXT Y
263 G=Y
264 FOR F=1 TO NREL-G+1
265 R=S(F)
266 IF J<>R(R,2) THEN 272
267 I=R(R,1)
268 IF I>NACT THEN 272
269 NOW=A(J,4)
270 GOSUB 630
271 IF NOW<A(I,4) THEN A(I,4)=NOW
272 NEXT F
273 NEXT X
274 G=1
275 FOR I=1 TO NACT
276 IF ""=MID$(S$(I),1,1) THEN 323
277 A(I,8)=CAL
278 IF NREL=0 THEN 292
279 FOR F=G TO NREL
280 R=S(F)
281 IF R(R,1)<I THEN 291
282 IF R(R,1)=I THEN 285
283 G=F
284 GOTO 292
285 J=R(R,2)
286 IF J>NACT THEN 291
287 NOW=A(J,3)
288 GOSUB 630
289 H=NOW-A(I,3)
290 IF H>=0 AND H<A(I,8) THEN A(I,8)=H
291 NEXT F
```

```
292 CL=A(I,6)
293 X=A(I,3)
294 Y=A(I,4)
295 F=Y-X
296 IF CL<=0 OR X<=0 OR Y<=0 THEN 308
297 F=0
298 IF X=Y THEN 308
299 IF X<Y THEN 302
300 X=Y
301 Y=A(I,3)
302 FOR Z=X TO Y-1
303 IF Z>CAL THEN 305
304 IF "N"=MID$(Z$(Z),CL,1) THEN 306
305 F=F+1
306 NEXT Z
307 IF A(I,3)>A(I,4) THEN F=-F
308 A(I,7)=F
309 IF A(I,7)<A(I,8) THEN A(I,8)=A(I,7)
310 H=1
311 DUR=A(I,5)
312 IF DUR=0 THEN H=0
313 NOW=A(I,3)
314 GOSUB 676
315 A(I,3)=NOW-H
316 A(I,2)=NOW-CDUR
317 NOW=A(I,4)
318 GOSUB 676
319 NOW=NOW-CDUR
320 GOSUB 665
321 A(I,4)=NOW
322 A(I,1)=NOW+CDUR-H
323 NEXT I
324 FOR A=1 TO NACT
325 S(A)=A
326 NEXT A
327 IF NACT<=1 THEN 343
328 Y=NACT
329 Y=INT(Y/2)
330 FOR Z=1 TO NACT-Y
331 IF ""=MID$(S$(Z+Y),1,1) THEN 341
332 H(1)=A(S(Z+Y),2)
333 H(2)=S(Z+Y)
334 FOR W=Z TO 1 STEP -Y
335 IF ""=MID$(S$(W),1,1) THEN 341
336 IF A(S(W),2)<H(1) THEN 340
337 IF A(S(W),2)=H(1) AND S(W)<H(2) THEN 340
338 S(W+Y)=S(W)
339 NEXT W
```

```
340 S(W+Y)=H(2)
341 NEXT Z
342 IF Y>1 THEN 329
343 EN=EN-BEGIN
344 RETURN
345 IF "          "=MID$(Z$(W),1,10) OR LEN(Z$(W))=0 THEN RETURN
346 LOCATE L,21:PRINT W:LOCATE L,27:PRINT MID$(Z$(W),4,7):LOCATE L,36
    :PRINT MID$(Z$(W),1,1):LOCATE L,41:PRINT MID$(Z    $(W),2,1):LOCATE
    L,46:PRINT MID$(Z$(W),3,1)
347 L=L+1
348 RETURN
349 IF ""=MID$(S$(W),1,1) THEN RETURN
350 LOCATE L,6:PRINT W:LOCATE L,11:PRINT A$(W):LOCATE L,44:PRINT
    MID$(S$(W),2,7);MID$(S$(W),1,1):LOCATE L,52:PRINT A(W,5):LOCATE
    L,56:PRINT A(W,6):LOCATE L,60:PRINT A(W,9):LOCATE L,70:PRINT A(W,10)
351 L=L+1
352 RETURN
353 IF R(W,1)=0 THEN RETURN
354 T=R(W,3)
355 GOSUB 692
356 LOCATE L,21:PRINT W:LOCATE L,26:PRINT R(W,1):LOCATE L,33:PRINT
    R(W,2):LOCATE L,40:PRINT P$:LOCATE L,44:PRINT R(W,4):LOCATE L,48
    :PRINT R(W,5)
357 L=L+1
358 RETURN
359 A=W
360 IF V=0 THEN A=S(W)
361 IF A=0 THEN RETURN
362 IF ""=MID$(S$(A),1,1) THEN RETURN
363 LOCATE L,1:PRINT A:LOCATE L,6:PRINT A$(A):LOCATE L,39:PRINT
    MID$(S$(A),2,7);MID$(S$(A),1,1):LOCATE L,47:PRINT A(A,5):LOCATE
    L,51:PRINT A(A,6):LOCATE L,55:PRINT A(A,9):LOCATE L,64:PRINT
    A(A,10):LOCATE L,69:PRINT A(A,7):LOCATE L,73:PRINT A(A,8)
364 IF A(A,7)<>0 THEN 366
365 LOCATE L,78:PRINT "*"
366 L=L+1
367 H=A(A,2)
368 FOR G=11 TO 56 STEP 15
369 IF G=26 THEN H=A(A,3)
370 IF G=41 THEN H=A(A,4)
371 IF G=56 THEN H=A(A,1)
372 IF H<=0 OR H>CAL THEN 374
373 LOCATE L,G:PRINT H:LOCATE L,G+4:PRINT "-";MID$(Z$(H),4,7)
374 NEXT G
375 L=L+1
376 RETURN
377 FOR I=1 TO 5
378 H(I)=0
```

```
379 NEXT I
380 FOR A=1 TO NACT
381 IF ""=MID$(S$(A),1,1) OR A(A,9)<=0 OR A(A,2)>W THEN 391
382 CL=A(A,6)
383 IF W>CAL OR CL=0 THEN 385
384 IF "N"=MID$(Z$(W),CL,1) THEN 391
385 DUR=A(A,5)
386 IF DUR=0 THEN 391
387 IF A(A,3)<W THEN 391
388 I=1
389 IF A(A,10)>0 AND A(A,10)<6 THEN I=A(A,10)
390 H(I)=H(I)+(A(A,9)/DUR)
391 NEXT A
392 LOCATE L,3:PRINT W:LOCATE L,15:PRINT H(1):LOCATE L,27:PRINT H(2)
    :LOCATE L,39:PRINT H(3):LOCATE L,51:PRINT H(4):LOCATE L,63:PRINT
    H(5)
393 IF W>CAL THEN 395
394 LOCATE L,8:PRINT MID$(Z$(W),4,7)
395 L=L+1
396 RETURN
397 CLS
398 LOCATE 1,25:PRINT "BARCHART":LOCATE 21,1:PRINT "START ?":LOCATE 21,8
    :INPUT "",P$
399 GOSUB 655
400 IF P$<>"" AND G=99999! THEN RETURN
401 V=G
402 IF G=99999! THEN V=BEGIN
403 GOSUB 131
404 RETURN
405 A=S(W)
406 IF A=0 THEN RETURN
407 IF ""=MID$(S$(A),1,1) THEN RETURN
408 LOCATE L,1:PRINT A:LOCATE L,5:PRINT A$(A):LOCATE L+1,37:PRINT "|"
    :LOCATE L+1,44:PRINT "|":LOCATE L+1,51:PRINT "|":LOCATE L+1,58
    :PRINT "|":LOCATE L+1,65:PRINT "|":LOCATE    L+1,72:PRINT "|"
409 IF A(A,2)>V+34 OR A(A,1)<V THEN 427
410 CL=A(A,6)
411 DUR=A(A,5)
412 F=A(A,7)
413 I=0
414 IF A(A,2)>V THEN I=A(A,2)-V
415 FOR H=I TO 41
416 IF A(A,1)<V+H THEN 427
417 IF H+V>CAL OR CL=0 THEN 419
418 IF "N"=MID$(Z$(V+H),CL,1) THEN 426
419 IF F<0 OR DUR=0 OR A(A,3)<V+H THEN 425
420 IF F=0 THEN 423
421 LOCATE L,H+37:PRINT "X"
```

```
422 GOTO 426
423 LOCATE L,H+37:PRINT "C"
424 GOTO 426
425 LOCATE L,H+37:PRINT "-"
426 NEXT H
427 L=L+2
428 RETURN
429 IF W<>0 THEN 431
430 LOCATE 2,41:INPUT "",P$
431 IF W<=0 THEN 433
432 LOCATE L,22:INPUT "",P$
433 GOSUB 655
434 IF P$="" THEN 438
435 IF P$="I" THEN 452
436 IF P$="D" THEN 443
437 IF (W>0 AND G=99999!) OR P$="L" OR P$="M" OR P$="A" OR P$="R" OR
    P$="U" OR P$="RE" OR P$="RA" OR P$="RS" OR P$="BC" THEN RETURN
438 IF W>0 THEN 460
439 G=0
440 IF P$<>"" THEN B$=P$
441 LOCATE 2,41:PRINT B$
442 RETURN
443 LOCATE L,21:PRINT "D ":LOCATE L,22:INPUT "",P$
444 GOSUB 655
445 IF G=99999! THEN 443
446 FOR H=G TO CAL-1
447 Z$(H)=Z$(H+1)
448 NEXT H
449 Z$(CAL)="           "
450 LOCATE L,22:PRINT P$
451 GOTO 477
452 LOCATE L,21:PRINT "I ":LOCATE L,22:INPUT "",P$
453 GOSUB 655
454 IF G=99999! THEN 453
455 FOR H=G TO CAL-1
456 H$=Z$(H+1)
457 Z$(H+1)=Z$(G)
458 Z$(G)=H$
459 NEXT H
460 IF G<=CAL THEN W=G
461 IF W<=0 THEN 429
462 LOCATE L,21:PRINT W:LOCATE L,27:INPUT "",P$
463 GOSUB 625
464 IF LEN(Z$(W))=0 THEN Z$(W)="YYY       "
465 IF P$<>"" THEN Z$(W)=MID$(Z$(W),1,3)+P$
466 LOCATE L,27:PRINT MID$(Z$(W),4,7)
467 G=36
468 FOR H=1 TO 3
```

```
469 LOCATE L,G:INPUT "",P$
470 IF P$<>"Y" AND P$<>"N" AND P$<>"" THEN 469
471 IF P$="" THEN 474
472 IF H=1 THEN Z$(W)=P$+MID$(Z$(W),2,9)
473 IF H>1 THEN Z$(W)=LEFT$(Z$(W),H-1)+P$+MID$(Z$(W),H+1,10-H)
474 LOCATE L,G:PRINT MID$(Z$(W),H,1)
475 G=G+5
476 NEXT H
477 L=L+1
478 RETURN
479 IF W<>0 THEN 481
480 LOCATE 2,48:INPUT "",P$
481 IF W<=0 THEN 483
482 LOCATE L,7:INPUT "",P$
483 GOSUB 655
484 IF P$="" THEN 491
485 IF P$="I" THEN 482
486 IF P$="D" THEN 494
487 IF G=99999! THEN RETURN
488 IF W>0 THEN 504
489 IF G<=ACT THEN NACT=G
490 Y=NACT
491 IF W>0 THEN 504
492 LOCATE 2,47:PRINT NACT;"    "
493 RETURN
494 LOCATE L,6:PRINT "D ":LOCATE L,7:INPUT "",P$
495 GOSUB 655
496 IF G=99999! THEN 495
497 A$(G)=""
498 S$(G)=""
499 FOR H=1 TO 10
500 A(G,H)=0
501 NEXT H
502 LOCATE L,7:PRINT P$
503 GOTO 528
504 IF G<=NACT THEN W=G
505 IF W<=0 THEN 479
506 IF LEN(S$(W))=0 THEN S$(W)="          "
507 LOCATE L,6:PRINT W:LOCATE L,11:INPUT "",P$
508 GOSUB 625
509 IF P$<>"" THEN A$(W)=P$
510 LOCATE L,11:PRINT A$(W):LOCATE L,44:INPUT "",P$
511 IF P$="" THEN 514
512 IF RIGHT$(P$,1)<>" " AND RIGHT$(P$,1)<>"S" AND RIGHT$(P$,1)<>"F"
    THEN 510
513 S$(W)=RIGHT$(P$,1)+LEFT$(P$,7)
514 LOCATE L,44:PRINT MID$(S$(W),2,7);LEFT$(S$(W),1)
515 LOCATE L,53:INPUT "",P$
```

```
516 GOSUB 655
517 IF G<>99999! THEN A(W,5)=G
518 LOCATE L,52:PRINT A(W,5):LOCATE L,57:INPUT "",P$
519 GOSUB 655
520 IF G<=3 THEN A(W,6)=G
521 LOCATE L,56:PRINT A(W,6):LOCATE L,61:INPUT "",P$
522 GOSUB 655
523 IF G<>99999! THEN A(W,9)=G
524 LOCATE L,60:PRINT A(W,9):LOCATE L,71:INPUT "",P$
525 GOSUB 655
526 IF G<>99999! AND G>0 AND G<6 THEN A(W,10)=G
527 LOCATE L,70:PRINT A(W,10)
528 L=L+1
529 G=0
530 RETURN
531 IF W<>0 THEN 533
532 LOCATE 2,50:INPUT "",P$
533 IF W<=0 THEN 535
534 LOCATE L,22:INPUT "",P$
535 GOSUB 655
536 IF P$="" THEN 543
537 IF P$="I" THEN 559
538 IF P$="D" THEN 546
539 IF G=99999! THEN RETURN
540 IF W>0 THEN 569
541 IF G<=REL THEN NREL=G
542 Y=NREL
543 IF W>0 THEN 569
544 LOCATE 2,49:PRINT NREL
545 RETURN
546 LOCATE L,21:PRINT "D ":LOCATE L,22:INPUT "",P$
547 GOSUB 655
548 IF G=99999! THEN 547
549 FOR F=G TO NREL-1
550 FOR H=1 TO 5
551 R(F,H)=R(F+1,H)
552 NEXT H
553 NEXT F
554 FOR H=1 TO 5
555 R(NREL,H)=0
556 NEXT H
557 LOCATE L,22:PRINT P$
558 GOTO 593
559 LOCATE L,21:PRINT "I ":LOCATE L,22:INPUT "",P$
560 GOSUB 655
561 IF G=99999! THEN 560
562 FOR F=G TO NREL
563 FOR H=1 TO 5
```

```
564 H(H)=R(F+1,H)
565 R(F+1,H)=R(G,H)
566 R(G,H)=H(H)
567 NEXT H
568 NEXT F
569 IF G<=NREL THEN W=G
570 IF W<=0 THEN 531
571 LOCATE L,21:PRINT W:LOCATE L,27:INPUT "",P$
572 GOSUB 625
573 GOSUB 655
574 IF G<>99999! THEN R(W,1)=G
575 IF R(W,1)<1 OR R(W,1)>NACT THEN 571
576 LOCATE L,26:PRINT R(W,1):LOCATE L,34:INPUT "",P$
577 GOSUB 655
578 IF G<>99999! THEN R(W,2)=G
579 IF R(W,1)>=R(W,2) OR R(W,2)<=0 THEN 576
580 LOCATE L,33:PRINT R(W,2):LOCATE L,40:INPUT "",P$
581 T=R(W,3)
582 IF P$<>"" THEN 584
583 GOSUB 692
584 GOSUB 687
585 R(W,3)=T
586 LOCATE L,40:PRINT P$:LOCATE L,45:INPUT "",P$
587 GOSUB 655
588 IF G<>99999! THEN R(W,4)=G
589 LOCATE L,44:PRINT R(W,4):LOCATE L,49:INPUT "",P$
590 GOSUB 655
591 IF G<=3 THEN R(W,5)=G
592 LOCATE L,48:PRINT R(W,5)
593 L=L+1
594 G=0
595 RETURN
596 CLS
597 LOCATE 1,32:PRINT "CALENDAR":LOCATE 2,25:PRINT "START DAY DATE"
    :LOCATE 2,41:PRINT B$:LOCATE 3,20:PRINT "  DAY  DATE     NO.1 NO.2
    NO.3"
598 L=4
599 RETURN
600 CLS
601 LOCATE 1,32:PRINT "ACTIVITIES":LOCATE 2,28:PRINT "MAXIMUM ID NUMBER"
    :LOCATE 2,47:PRINT NACT:LOCATE 3,7:PRINT "ID  DESCRIPTION":LOCATE
    3,44:PRINT "SCH.DATE DUR CAL RESOURCES TYPE"
602 L=4
603 RETURN
604 CLS
605 LOCATE 1,29:PRINT "RELATIONSHIPS":LOCATE 2,22:PRINT "NUMBER OF RELAT
    IONSHIPS":LOCATE 2,49:PRINT NREL:LOCATE 3,22:PRINT "REL. PRED.  SUCC.
    TYPE LAG CAL"
```

```
606 L=4
607 RETURN
608 CLS
609 LOCATE 1,30:PRINT "CRITICAL PATH TABLE":LOCATE 2,30:PRINT "PROJECT D
    URATION ";EN:LOCATE 3,2:PRINT "ID  DESCRIPTION":LOCATE 3,39:PRINT
    "SCH.DATE DUR CAL RESOURCE TYPE TF  FF  CR"
610 LOCATE 4,12:PRINT "EARLY START    EARLY FINISH   LATE START    LATE
    FINISH"
611 L=5
612 RETURN
613 GOSUB 596
614 LOCATE 1,32:PRINT "RESOURCES":LOCATE 3,4:PRINT "DAY DATE        TYPE
    1      TYPE 2      TYPE 3      TYPE 4      TYPE 5"
615 RETURN
616 CLS
617 LOCATE 1,35:PRINT "BARCHART":LOCATE 3,1:PRINT "ID  DESCRIPTION"
    :LOCATE 3,36:PRINT V:LOCATE 3,43:PRINT V+7:LOCATE 3,50:PRINT V+14
    :LOCATE 3,57:PRINT V+21:LOCATE 3,64:PRINT  V+28:LOCATE 3,71:PRINT
    V+35
618 IF V+21>CAL THEN 621
619 IF "       "=MID$(Z$(V+35),4,7) OR LEN(Z$(V+35))=0 THEN 621
620 LOCATE 2,37:PRINT MID$(Z$(V),4,7):LOCATE 2,51:PRINT MID$(Z$(V+14),4,
    7):LOCATE 2,65:PRINT MID$(Z$(V+28),4,7):LOCATE 3,44:PRINT MID$(Z$(V+
    7),4,7):LOCATE 3,58:PRINT MID$(Z$(V+21),4,7):LOCATE 3,72:PRINT
    MID$(Z$(V+35),4,7)
621 L=4
622 RETURN
623 LOCATE 21,1:PRINT "OPTION  ?"
624 RETURN
625 LOCATE 21,1:PRINT "          "
626 RETURN
627 LOCATE 21,1:PRINT "LINE   ?":LOCATE 21,8:INPUT "",P$
628 GOSUB 655
629 RETURN
630 H=R(R,3)
631 IF H=2 OR H=3 THEN 636
632 DUR=A(J,5)
633 CL=A(J,6)
634 GOSUB 676
635 NOW=NOW-CDUR
636 DUR=R(R,4)
637 CL=R(R,5)
638 IF DUR>=0 THEN 643
639 DUR=1-DUR
640 GOSUB 665
641 NOW=NOW+CDUR-1
642 GOTO 645
643 GOSUB 676
```

```
644 NOW=NOW-CDUR
645 IF H<>1 AND H<>3 THEN RETURN
646 CL=A(I,6)
647 DUR=A(I,5).
648 GOSUB 665
649 H=NOW
650 NOW=NOW+CDUR
651 GOSUB 676
652 IF H=NOW-CDUR THEN RETURN
653 NOW=NOW-1
654 RETURN
655 IF P$="" THEN 660
656 G=LEN(P$)
657 FOR H=1 TO G
658 G$=MID$(P$,H,1)
659 IF G$="." OR G$="0" OR G$="1" OR G$="2" OR G$="3" OR G$="4" OR
    G$="5" OR G$="6" OR G$="7" OR G$="8" OR G$="9" OR (H=1 AND G$="-")
    THEN 662
660 G=99999!
661 RETURN
662 NEXT H
663 G=VAL(P$)
664 RETURN
665 CDUR=DUR
666 IF CL=0 THEN RETURN
667 CDUR=0
668 IF DUR=0 THEN RETURN
669 FOR T=1 TO DUR
670 CDUR=CDUR+1
671 D=NOW+CDUR-1
672 IF D>CAL OR D<=0 THEN 674
673 IF "N"=MID$(Z$(D),CL,1) THEN 670
674 NEXT T
675 RETURN
676 CDUR=DUR
677 IF CL=0 THEN RETURN
678 CDUR=0
679 IF DUR=0 THEN RETURN
680 FOR T=1 TO DUR
681 CDUR=CDUR+1
682 D=NOW-CDUR
683 IF D>CAL OR D<=0 THEN 685
684 IF "N"=MID$(Z$(D),CL,1) THEN 681
685 NEXT T
686 RETURN
687 T=0
688 IF P$="S" THEN T=1
689 IF P$="F" THEN T=2
```

```
690 IF P$="B" THEN T=3
691 RETURN
692 P$="N"
693 IF T=1 THEN P$="S"
694 IF T=2 THEN P$="F"
695 IF T=3 THEN P$="B"
696 RETURN
```

APPENDIX B: CORNER CUT METHOD

The dual simplex method has been proven through actual applications to be a consistent method for solving LP problems. Although the method has not been improved upon in the general application, in specific applications, variations of the method can be faster or more precise.

The corner cut method is a variation of the dual simplex method developed to increase the precision maintained in solving a problem with nonlinear constraints approximated with supporting planes. Specifically for reducing the error inherent in finding the most geometric distant constraint from the simplex current point.

There is a penalty for maintaining greater precision in that the number of iterations to reach an optimal solution is increased over the dual simplex method. But if the application requires the precision, as in the nonlinear problem were very small errors can greatly effect the optimal solution, the "corner cut" method is a valuable tool.

Theory of Corner Cut Method

The corner cut method, like the simplex method, starts with the solution point (0,0) at the origin defined by the zero axes. If this current point is not a feasible point, then it violates a set of constraints. From this set of constraints, the method evaluates all of the intersections that each constraint makes with the constraints defining the current point. From the intersection points of each constraint, the point with the minimum value of the objective function

is selected. If a tie is found for a constraint, then the point associated with the smallest column number in the tableau is selected.

Of all the violated constraints and their minimum objective function value points, the constraint with the point that has maximum value of the objective is selected as the pivot row for a simplex transformation. If a tie is found, then the point associated with the constraint farthest from the current point is used.

The current point is then set equal to the point selected above, and the tableau is transformed. This procedure is repeated until no violated constraints can be found, the current point is a minimum feasible solution.

The Two Dimensional Case of the Corner Cut Method

To visualize the method and to provide a geometry for the algorithm, the two (2) dimensional example 16 LP problem:

$$\text{minimize} \quad a*X+b*Y$$

$$\begin{aligned} \text{subject to:} \quad & X \geq 0 && \text{(B-1)} \\ & Y \geq 0 && \text{(B-2)} \\ & c*X+d*Y \geq e && \text{(B-3)} \\ & g*X+h*Y \geq f && \text{(B-4)} \end{aligned}$$

$$a,b,c,d,e,f,g,h>0$$

is defined where all the lower case letters are positive coefficients, and all constraints are in the greater than or equal form. The positive restrictions on the X and Y variables, as well as the nonnegative coefficients of the objective function, are requirements of the corner cut method (the dual version) as they are for the dual simplex method.

Figure B-1 is the example 16 problem in graphic form.

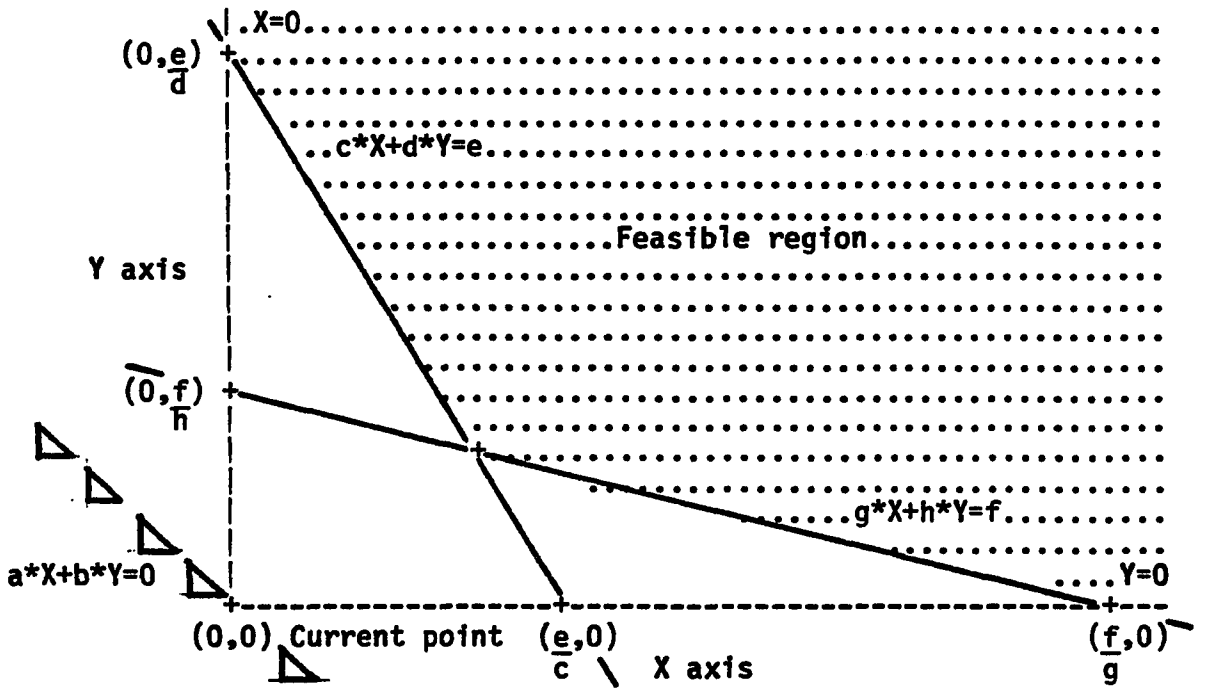


Figure B-1. Corner cut method example 16

The similarities of the corner cut method and the simplex method continues into the tableau where example 16 is shown in tableau B-1 with the pivot row still to be determined.

	1	X	Y
z=	0	a	b
x=	0	1	0
y=	0	0	1
s=		$\bar{p}'$	

Tableau B-1. Corner cut method example 16 tableau

The transformation of any constraint, as with the dual simplex

method, can be found with the augmented $\overline{B}$ inverse matrix by:

$$\left| -a \;,\; \overline{a}' \right| * \begin{vmatrix} 1 & \overline{0}' \\ \overline{B}^{-1}\overline{b} & \overline{B}^{-1} \end{vmatrix} = \left| -a+\overline{a}'\overline{B}^{-1}\overline{b} \;,\; \overline{a}'\overline{B}^{-1} \right|$$

The first constraint (B-1), X>=0, is transformed as follows:

$$\left| -0 \;,\; 1 \;,\; 0 \right| * \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix} = \left| 0 \;,\; 1 \;,\; 0 \right|$$

where the slack x is nonnegative and the constraint is not violated. Constraint (B-2) is transformed as:

$$\left| -0 \;,\; 0 \;,\; 1 \right| \text{----->} \left| 0 \;,\; 0 \;,\; 1 \right|$$

Again, the slack y is nonnegative so the constraint is not violated. Constraint (B-3) is transformed as:

$$\left| -e \;,\; c \;,\; d \right| \text{----->} \left| -e \;,\; c \;,\; d \right|$$

In this case, the slack s1 is negative so it is one of the set of violated constraints. The violated constraint (B-3) intersects the current point's basis constraints X>=0 and Y>=0 at the two (2) point as shown in figure B-2. These points can be found directly from the tableau by a "partial" simplex pivot on the columns of the tableau which are associated with the intersections and are identified by the positive

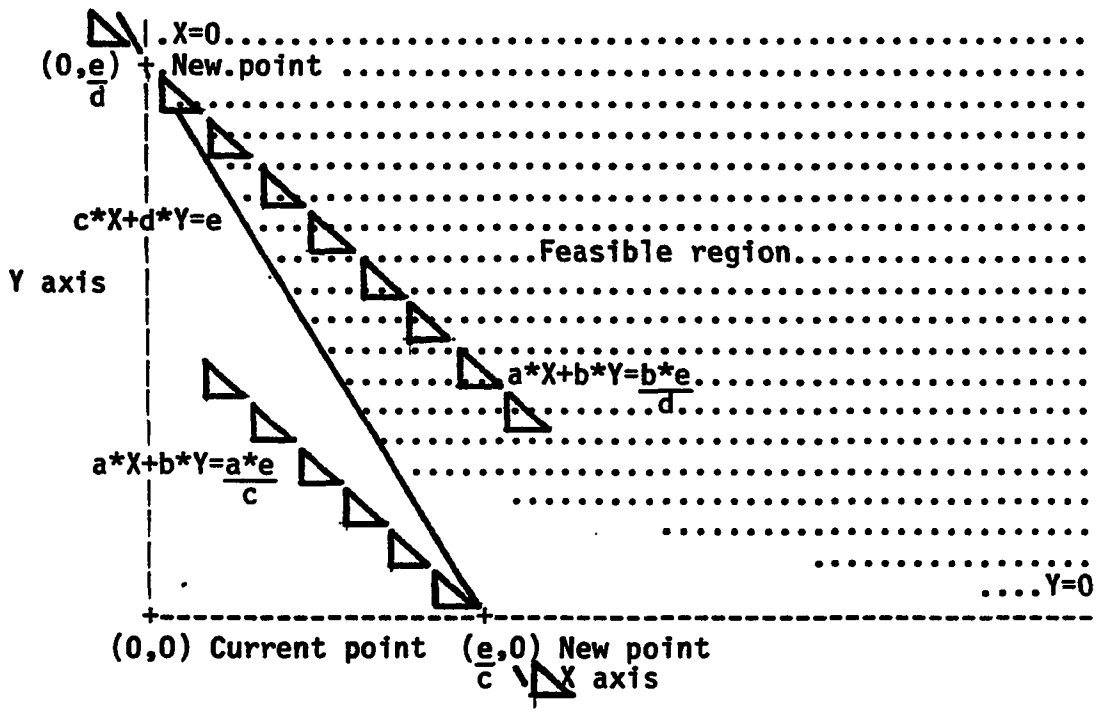


Figure B-2. Corner cut method example 16, iteration one

elements of the transformed constraint. (This step is the same as the dual simplex search for a pivot element. If no positive elements can be found, the constraint and the problem are infeasible and the procedure stops.)

z=	0	a		a*e/c	a/c
x=	0	1	----->	e/c	1/c
y=	0	0		0	0
s=	-e	c		0	1

This partial transformation yields the value of the objective function and the coordinates for one (1) of two (2) intersection points.

$$z=\frac{a*e}{c} \quad x=\frac{e}{c} \quad y=0$$

The second point of intersection can be found in a similar manner with a partial pivot on the third column of the tableau:

```
        +----+---+                +-----+---+
z=      |  0 | b |                | b*e | _ |
        +----+---+                |  d  |   |
x=      |  0 | 0 |                +-----+---+
        |    |   | ----->         |  0  | - |
y=      |  0 | 1 |                |     |   |
        +----+---+                | e   | _ |
s=      | -e | d |                | d   |   |
        +----+---+                +-----+---+
                                   |  -  | - |
                                   +-----+---+
```

resulting in:

$$z=\frac{b*e}{d} \quad x=0 \quad y=\frac{e}{d}$$

Of the two (2) possible points associated with constraint (B-3) only the first point needs to be considered assuming the objective function values:

$$\text{minimum} \quad \left(\frac{a*e}{c},\frac{b*e}{d}\right)=\frac{a*e}{c}$$

Transforming the fourth constraint (B-4):

$$| -f , g , h | \text{----} > | -f , g , h |$$

results in a negative slack s2, so constraint (B-4) is also part of the violated constraints. This constraint again has two (2) intersection points as shown in figure B-3.

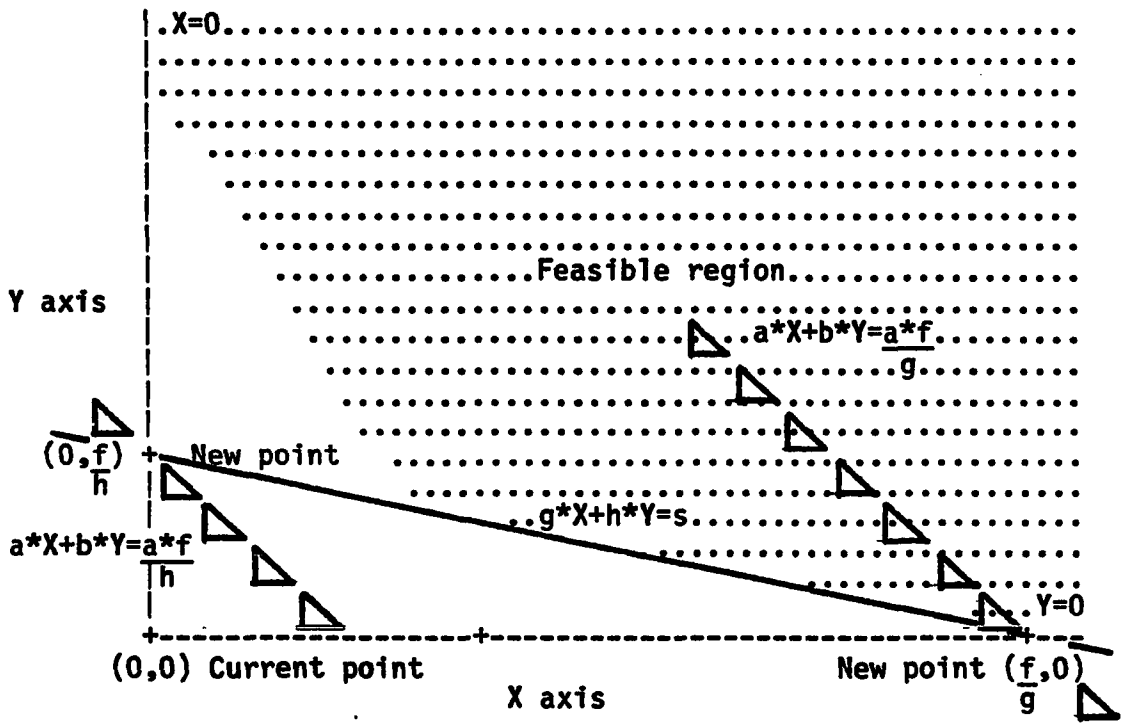


Figure B-3. Corner cut method example 16, iteration one

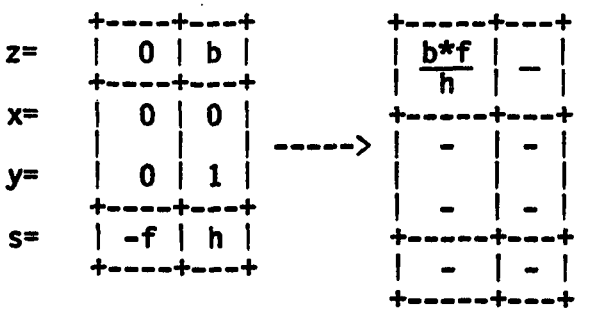
To find the value of the objective function for the intersection points of the new constraints, only the objective function of the partial pivot need to be performed:

z=	0	a		a*f/g	_
x=	0	1	----->	-	-
y=	0	0			
s=	-f	g		-	-
				-	-

resulting in:

$$z=\frac{a*f}{g}$$

and:



Of the two (2) possible points associated with constraint (B-4) only the second point needed to be considered assuming the objective function values:

$$\text{minimum } (\frac{a*f}{g},\frac{a*f}{h})=\frac{a*f}{h}$$

Of the two violated constraints and their minimum intersection points, select the maximum value of the objective function. Assuming:

$$\text{maximum } (\frac{a*e}{c},\frac{a*f}{h})=\frac{a*e}{c}$$

If there is a tie, then the tie is broken by the greatest distance to the constraint or:

$$\text{maximum } (\frac{(-e)\hat{}2}{c\hat{}2+d\hat{}2},\frac{(-f)\hat{}2}{g\hat{}2+h\hat{}2})$$

Using the selected transformed constraint as the pivot row and with

$$\bar{p}' = | \ -e \ , \ c \ , \ d \ |$$

a Gauss-Jordan elimination pivoting on the first column, the tableau is transformed to represent the new point.

The result of the transformation is shown in tableau B-2 and figure B-4.

z=	$\frac{a*e}{c}$	a	$b-\frac{a*d}{c}$
x=	$\frac{e}{c}$	1	$-\frac{d}{c}$
y=	0	0	1
s1=	0	1	0

Tableau B-2. Corner cut method example 16 tableau, iteration one

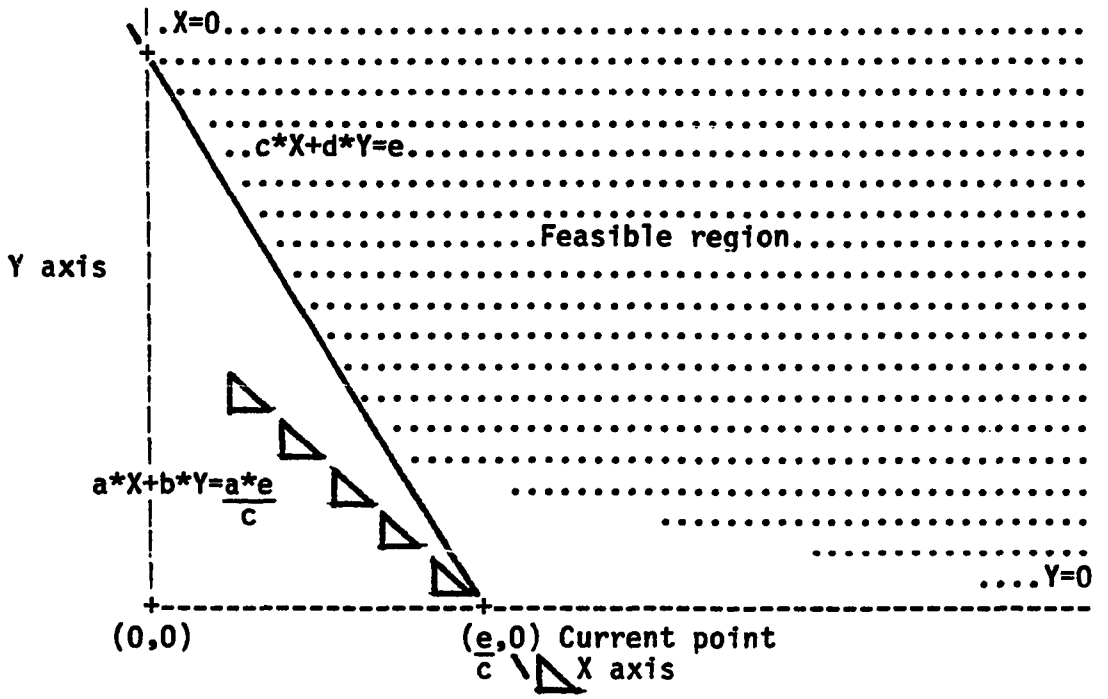


Figure B-4. Corner cut method example 16, iteration one

With the new simplex tableau the constraints can again be transformed. In this set of constraint transformation, the transformation is not completed if the first element is positive and constraint is not a violated constraint.

For constraint (B-1) the transformation is:

$$| -0 , 1 , 0 | \text{ ---> } | \frac{e}{c} , - , - |$$

For constraint (B-2):

$$| -0 , 0 , 1 | \text{ ---> } | 0 , - , - |$$

For constraint (B-3):

$$| -e , c , d | \text{ ---> } | 0 , - , - |$$

For constraint (B-4):

$$| -f , g , h | \text{ ---> } | \frac{e*g-f}{c} , g , h-\frac{d*g}{c} |$$

Constraint (B-4) as shown in figure B-5 is violated. Assuming that

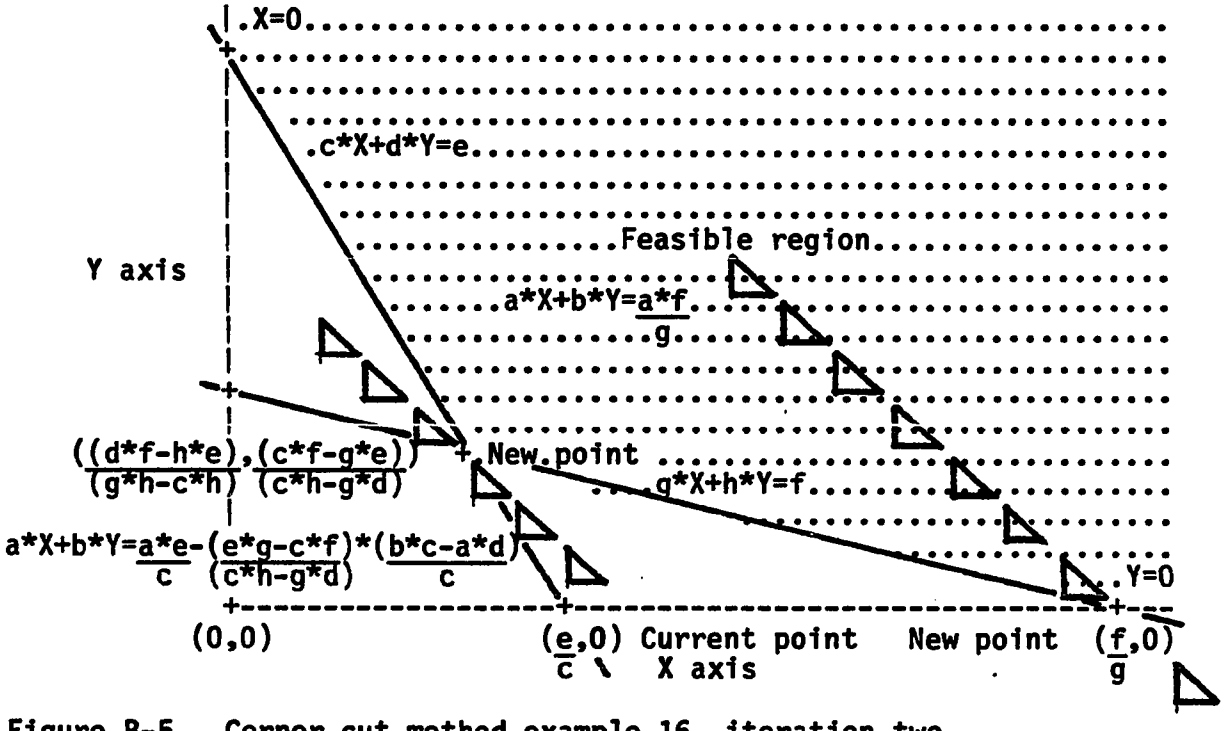


Figure B-5. Corner cut method example 16, iteration two

the transformed constraint elements are:

$$\frac{e*g-f}{c}<0 \quad g>0 \quad h-\frac{d*g}{c}>0$$

then the minimum value of the objective function is with a pivot on the second column.

$$\text{minimum } \left(\frac{a*e-(e*g-c*f)*a}{c \quad c \quad g}, \frac{a*e-(e*g-c*f)*(b*c-a*d)}{c \quad (c*h-g*d) \quad c}\right)=\frac{a*e-(e*g-c*f)*(b*c-a*d)}{c \quad (c*h-g*d) \quad c}$$

Again using the transformation of constraint (B-4) as the pivot row:

$$\overline{p}' = \left| \frac{(e*g-c*f)}{(h*c-g*d)}, \frac{c*g}{(h*c-g*d)}, 1 \right|$$

results in tableau B-3 as the new simplex tableau.

z=	$\frac{a*e-(e*g-c*f)*(b*c-a*d)}{c \quad (c*h-g*d) \quad c}$	$a-\frac{c*g}{(c*h-g*d)}*\frac{(b*c-a*d)}{c}$	$\frac{c*b-a*d}{c}$
x=	$\frac{e}{c}+\frac{(e*g-c*f)*d}{(c*h-g*d) \quad c}$	$1+\frac{c*g}{(c*h-g*d)}*\frac{d}{c}$	$\frac{-d}{c}$
y=	$\frac{-(e*g-c*f)}{(c*h-g*d)}$	$\frac{-c*g}{(c*h-g*d)}$	1
s2=	0	0	1

Tableau B-3. Corner cut method example 16 tableau, solution

The new point can now be evaluated by the transformation of the constraints. Again, the transformation is stopped if the constraint is not violated.

For constraint (B-1) the transformation is:

$$\left| -0 , 1 , 0 \right| \text{ ----> } \left| \frac{e}{c}+\frac{(g*e-c*f)*d}{(c*h-g*d) \quad c}, -, - \right|$$

For constraint (B-2):

$$\left| -0 , 0 , 1 \right| \text{ ----> } \left| -\frac{(g*e-c*f)}{(c*h-g*d)}, -, - \right|$$

For constraint (B-3):

| -e , c , d | -----> | 0 ,-,- |

For constraint (B-4):

| -f , g , h | ----> | 0 ,-,- |

All the constraints are satisfied so the current point is a feasible solution. To determine if the solution is optimal, the objective function is set equal to or less than the current point's value of the objective function minus an amount determined by the desired accuracy and transformed into a pivot row as if it were a constraint.

$$a*X+b*Y<=\frac{a*e}{c}-\frac{(c*f-g*e)}{(c*h-g*d)}*\frac{(b*c-a*d)}{c}-\text{small amount}$$

If the current solution is optimal, the objective function pivot row will be infeasible.

Corner Cut Method BASIC Code

The corner cut method was derived from the dual simplex method by modifying the simplex pivoting rules to improve the computational accuracy in solving CP problems. The BASIC computer code for the method is based on the simplex tableau and uses the same array formats and variable definitions as the dual simplex program.

All programms listed in the text consist of a main calling routine and a series of input, output, and processing subroutines. These routines are listed in the text as well as on a computer disk compatible with IBM micro-computers.

Corner Cut Main routine -- File MAIN-CRN

The corner cut main routine (MAIN-CRN) dimensions nine (9) arrays; writes the options menu to the screen as shown in figure B-6; calls the

```
                    CORNER CUT METHOD

          NUMBER OF CONSTRAINTS              10
          NUMBER OF VARIABLES                14
          MAXIMUM ITERATIONS                 1000
          LOWER BOUND OF OBJECTIVE           0

                    M-RETURN TO MENU

               O-OBJECTIVE COEFFICIENTS
               A-CONSTRAINT COEFFICIENTS
               C-CONSTRAINT TYPES
               B-BOUNDED VARIABLES
               U-EXECUTE ALGORITHM
               R-REPORT LISTING
               N-NEW PROBLEM
               S-SAVE F-FETCH

          OPTION ?

            1*X*2*Y>=c    1*X*(2*Y-3*Z)>=c    1*Y+a*(X-b)^2>=c
```

Figure B-6. Corner cut method main menu screen

utility subroutines UTIL-OPT, UTIL-ERS, UTIL-CON, UTIL-CHX; calls the data input and output subroutines INPT-OBJ, INPT-CON, INPT-TYP, INPT-BND, and REPT-SMP; times and calls the processing algorithm ALGR-CRN; and saves and fetches the input data to the ASCI disk file "DATA".

```
1 REM                  * CORNER CUT METHOD *
2 REM-----------------------MAIN-CRN-----------------------------

3 REM      BI# - MACHINE INFINITE
4 REM      CO  - PIVOT COLUMN FOR SIMPLEX TRANSFORMATION
5 REM      ER  - ERROR KEY
6 REM
7 REM      IR  - MAXIMUM NUMBER OF ITERATIONS
8 REM      MD  - NUMBER OF CONSTRAINTS
```

```
9 REM     ND  - NUMBER OF VARIABLES
10 REM    RO  - PIVOT ROW FOR SIMPLEX TRANSFORMATION
11 REM    SM# - MACHINE ZERO
12 REM    A#(MD+2,ND+8) - ORIGINAL DATA AND KEYS
13 REM    B#(2*ND+1,ND+1) - PRIMAL-DUAL MATRIX
14 REM    H#(ND+1) -  SOLUTION FROM LAST ITERATION
15 REM    M#(ND+1,2) - UPPER AND LOWER BOUNDS ON VARIABLES
16 REM    P#(ND+1) - WORK VECTOR
17 REM    R(MD+2) - CONSTRAINT TYPE (1-">=",0-"=",-1-"<=")
18 REM    S#(ND+1) - COLUMN SWITCHES
19 REM    T#(ND+1,ND+1) - REINVERSION WORK SPACE
20 REM    X#(ND) - SOLUTION VECTOR
21 REM ------------------------------------------------------------
```

Sets MD to the default number of constraints and ND to the number of variables in the CP problem to be optimized. Sets IN to the number of iterations before reinversion of the augmented B̄ matrix. Sets the default limit on number of iterations to 1000. Sets BI# to a number considered machine infinite and SM# to a number considered machine zero.

```
22 MD=0
23 ND=0
24 IN=20
25 IR=1000
26 BI#=1E+10
27 SM#=1E-10
```

Prompts and reads from the keyboard the number of constraints MD; the number of variables ND; the maximum number of iterations IT; and the lower bound on the objective function DS.

```
28 CLS
29 LOCATE 1,1:PRINT "CORNER CUT METHOD"
30 LOCATE 3,1:PRINT "NUMBER OF CONSTRAINTS":LOCATE 3,31:INPUT "",L$
31 GOSUB 1870:REM UTIL-CHX
32 IF Z#<>BI# THEN MD=Z#
33 LOCATE 3,30:PRINT MD,"    ":LOCATE 4,1:PRINT "NUMBER OF VARIABLES":LO
   CATE 4,31:INPUT "",L$
34 GOSUB 1870:REM UTIL-CHX
35 IF Z#<>BI# THEN ND=Z#
36 LOCATE 4,30:PRINT ND,"    ":LOCATE 5,1:PRINT "MAXIMUM ITERATIONS":LOC
   ATE 5,31:INPUT "",L$
37 GOSUB 1870:REM UTIL-CHX
38 IF Z#<>BI# THEN IR=Z#
39 LOCATE 5,30:PRINT IR,"    ":LOCATE 6,1:PRINT "LOWER BOUND ON OBJECTIV
   E":LOCATE 6,31:INPUT "",L$
40 GOSUB 1870:REM UTIL-CHX
41 IF Z#<>BI# THEN DS=Z#
```

```
42 LOCATE 6,30:PRINT DS,"     "
```

Dimensions the objective function and constraint coefficients array A#(MD+2,ND+8).

The array structure of the corner cut method is the same as for the dual simplex method with only one (1) exception. The last row of the A#(MD+2,ND+8) array, or row MD+2, is the negative of the coefficients of the objective function. This row is used to define the constraint used to prove optimality.

Dimensions the augmented $\overline{B}$ inverse and augmented $\overline{B}$ matrix array B#(2*ND+1,ND+1), the holding array H#(ND+1) for the last current point values; the upper and lower bounds array M#(ND+1,2); the pivot row array P#(ND+2); the constraint type array R(MD+2); the reinversion work array T#(ND+1,ND+1); and the solution array X#(ND).

```
43 DIM A#(MD+2,ND+8)
44 DIM B#(2*ND+1,ND+1)
45 DIM H#(ND+1)
46 DIM M#(ND+1,2)
47 DIM P#(ND+2)
48 DIM R(MD+2)
49 DIM S#(ND+1)
50 DIM T#(ND+1,ND+1)
51 DIM X#(ND)
```

Initializes the constraint type array to greater than or equals.

```
52 FOR I=2 TO MD+2
53 R(I)=1
54 NEXT I
```

Prints the option menu to the screen; calls the option line subroutine UTIL-OPT; and pauses for the entry of either "M", "O", "A", "C", "B", "U", "R", "N", "S", "F" for the option variable L$.

```
55 LOCATE 8,15: PRINT          "M-RETURN TO MENU"
56 LOCATE 10,10:PRINT       "O-OBJECTIVE COEFFICIENTS"
57 LOCATE 11,10:PRINT       "C-CONSTRAINT VALUES"
58 LOCATE 12,10:PRINT       "A-A MATRIX COEFFICIENTS"
59 LOCATE 13,10:PRINT       "B-UPPER BOUNDED VARIABLES"
60 LOCATE 14,10:PRINT       "U-EXECUTE ALGORITHM"
61 LOCATE 15,10:PRINT       "R-REPORT LISTING"
62 LOCATE 17,10:PRINT       "S-SAVE  F-FETCH"
63 LOCATE 18,10:PRINT       "N-NEW PROBLEM"
64 LOCATE 23,1:PRINT "1*X*2*Y>=C  1*X*(2*Y-3*Z)>=C  1*Y+A*(X-B)^2>=C"
65 GOSUB 1800:REM UTIL-OPT
```

```
66 LOCATE 21,8:INPUT "",L$
```

Calls either the objective function input subroutine INP-OBJ, the constraint input subroutine INPT-CON, the constraint type input subroutine INPT-TYP, the upper and lower bounds input subroutine INPT-BND, the processing subroutine ALGR-KEY, or the report subroutine REPT-SMP based on the option variable L$.

```
67 CLS
68 H=0
69 G=2
70 IF L$<>"O" THEN 73
71 GOSUB 1200:REM INPT-OBJ
72 GOTO 67
73 IF L$<>"A" THEN 76
74 GOSUB 1300:REM INPT-CON
75 GOTO 67
76 IF L$<>"C" THEN 79
77 GOSUB 1400:REM INPT-TYP
78 GOTO 67
79 IF L$<>"B" THEN 82
80 GOSUB 1500:REM INPT-BND
81 GOTO 67
82 IF L$<>"U" THEN 90
83 TM=3600*VAL(MID$(TIME$,1,2))+60*VAL(MID$(TIME$,4,2))+VAL(MID$(TIME$,7
   ,2))
84 GOSUB 3000:REM ALGR-KEY
```

Sets the last current point of the corner cut method to the optimal solution.

```
85 FOR I=1 TO ND
86 X#(I)=B#(I+1,1)
87 NEXT I
88 TM=3600*VAL(MID$(TIME$,1,2))+60*VAL(MID$(TIME$,4,2))+VAL(MID$(TIME$,7
   ,2))-TM
89 GOTO 55
90 IF L$<>"R" THEN 93
91 GOSUB 2200:REM REPT-SMP
92 GOTO 67
```

Saves the contents of MD, ND, M#(ND+1,2), A#(MD+2,ND+8), and R(ND+1) to disk file "DATA" as an ASCI file if option "S" is selected.

```
93 IF L$<>"S" THEN 112
94 OPEN "O",#1,"DATA"
95 PRINT #1,STR$(MD)
96 PRINT #1,STR$(ND)
```

```
97 FOR I=1 TO ND+1
98 FOR J=1 TO ND+1
99 PRINT #1,""
100 NEXT J
101 PRINT #1,STR$(M#(I,1))
102 PRINT #1,STR$(M#(I,2))
103 NEXT I
104 FOR I=1 TO MD+1
105 FOR J=1 TO ND+8
106 PRINT #1,STR$(A#(I,J))
107 NEXT J
108 PRINT #1,STR$(R(I))
109 NEXT I
110 CLOSE #1
111 GOTO 55
```

Loads to MD, ND, M#(ND+1,2), A#(MD+2,ND+8), and R(ND+1) from the disk file "DATA" if option "F" is selected.

```
112 IF L$<>"F" THEN 137
113 OPEN "I",#1,"DATA"
114 INPUT #1,X$
115 MD=VAL(X$)
116 INPUT #1,X$
117 ND=VAL(X$)
118 FOR I=1 TO ND+1
119 FOR J=1 TO ND+1
120 INPUT #1,X$
121 NEXT J
122 INPUT #1,X$
123 M#(I,1)=VAL(X$)
124 INPUT #1,X$
125 M#(I,2)=VAL(X$)
126 NEXT I
127 FOR I=1 TO MD+1
128 FOR J=1 TO ND+8
129 INPUT #1,X$
130 A#(I,J)=VAL(X$)
131 NEXT J
132 INPUT #1,X$
133 R(I)=VAL(X$)
134 NEXT I
135 CLOSE #1
136 GOTO 55
137 IF L$="N" THEN RUN
138 GOTO 55
```

Utility Subroutines -- Files UTIL-OPT, UTIL-ERS, UTIL-CON, and UTIL-CHX

Same as for critical path method.

Input Subroutines -- Files INPT-OBJ, INPT-CON, INPT-TYP, and INPT-BND

Same as for dual simplex method.

Output Subroutine -- File REPT-SMP

Same as for dual simplex method.

Constraint Keys Subroutine -- File ALGR-KEY

Same as for dual simplex method with nonlinear constraints.

Corner Cut Algorithm Subroutine -- File ALGR-CRN

The corner cut algorithm first initializes a dual simplex tableau to represent a point at the origin and the set of constraints that form the zero axes. Then, in an iterative manner, proceeds through the following steps.

(1) Evaluate all the constraints of the CP problem to find the set of violated constraints. If no constraint is violated, then set ER=1 and return to main routine. Otherwise, go to step (2).

(2) For each of the supporting planes of the violated constraints, find the point from among the intersection points of the plane and the constraints defining the current point which minimizes the value of the objective function. Set ER=2 if any violated constraint's supporting plane does not have an intersection point and return to the main routine. Otherwise go to step (3).

(3) From the set of minimum points, one (1) for each violated constraints, let the point which maximizes the objective function be

the next point. If there is a tie, then select as the new point the point defined by the supporting plane farthest from the current point. Go to step (4).

(4) Using the transformed supporting plane that defined the selected new point as a pivot row, transform the simplex tableau using the Gauss-Jordan elimination so that the tableau represents the new point and its constraints. If the transformation fails, set ER=3 and return to main routine. Otherwise, go to step (5).

(5) Set the current point equal to the new point and return to step (1). If the cycle or iteration is repeated more than a preset limit, set ER=0 and return to the main routine.

```
3300 REM                    *CORNER CUT ALGORITHM*
3301 REM-------------------------ALGR-CRN-------------------------------
```

Initializes the simplex tableau to represent the origin by setting the $\overline{B}$ matrix equal to the identity matrix. Sets the iteration count variable IT to zero (0), the current value of the objective OB# to zero (0), and the value of the last current solution to infinity.

To set a lower limit to the value of the objective function, the last constraint in the A#(MD+2,ND+8) is set equal to the negative of the objective function. The function is set greater than or equal to the lower bound unless no bound is input, in which case it is set greater than or equal to negative infinity.

```
3302 FOR I=2 TO 2*ND+1
3303 FOR J=1 TO ND+1
3304 B#(I,J)=0#
3305 NEXT J
3306 NEXT I
3307 IT=0
3308 OB#=0#
3309 BB#=0#
3310 A#(1,1)=0#
3311 IF DS<>0 THEN B#(1,1)=DS
3312 A#(MD+2,1)=-BI#
3313 FOR I=2 TO ND+1
```

```
3314 B#(1,I)=A#(1,I)
3315 B#(I,I)=1#:B#(I,1)=M#(I,2)
3316 B#(ND+I,I)=1#:B#(ND+I,1)=M#(I,2)
3317 A#(MD+2,I)=-A#(1,I)
3318 H#(I)=BI#
3319 NEXT I
```

Sets BB# equal to the current value of the objective and increments the iteration count by one (1). If the maximum count is exceeded, then set ER=0 and return to the main routine.

```
3320 BB#=OB#
3321 IT=IT+1
3322 ER=0
3323 IF IT>IR THEN RETURN
3324 ER=2
```

Intializes, for the current iteration, the distance variable MA# and the maximum value of the current value of the objective function to negative infinity.

```
3325 MA#=-BI#
3326 OB#=-BI#
```

Checks the upper bound for a violated constraint. If the constraint is violated, then for each positive coefficient in the transformed constraint find through a partial pivot the value of the objective function.

To accommodate upper and lower bounds, the number of constraints is automatically increased to ND+1+ND+1+MD+1. The extra constraint on each group of contraints is to allow for the offset of the objective function in the simplex tableau. This simplifies the addressing at the expense of phantom lines in the tableau.

```
3327 FOR I=2 TO ND+1
3328 IF B#(I,1)-M#(I,2)>=0# THEN 3352
3329 K#=BI#
3330 FOR J=2 TO ND+1
3331 IF M#(J,1)=BI# OR B#(I,J)<=SM# THEN 3343
3332 L#=(B#(I,1)-M#(I,2))/B#(I,J)
```

As an alternate to using a Gauss-Jordan elimination on the first row of the tableau, the solution for the new point can be found for each positive element of the transformed constraint and the values substituted into the objective function directly. This eliminates the error in the transformation.

```
3333 REM-----------------------------------------------------------------
3334 A#=0#
3335 FOR K=2 TO ND+1
3336 IF A#(1,K)<>0# THEN A#=A#+A#(1,K)*(B#(K,1)-B#(K,J)*L#)
3337 NEXT K
3338 REM-----------------------------------------------------------------
3339 REM A#=BB#-B#(1,J)*L#
3340 IF A#>=K# THEN 3343
3341 K#=A#
3342 CC=J
3343 NEXT J
```

Sets M# equal to distance from the current point to the constraint.

```
3344 M#=(B#(I,1)-M#(I,2))*(B#(I,1)-M#(I,2))
```

If the minimum point is less than the maximum value of the objective found so far on this iteration, then the constraint is not considered further.

```
3345 IF K#<OB# THEN 3352
```

Returns to the main routine with ER=2 if no intersection is found.

```
3346 IF K#=BI# THEN RETURN
```

Uses the distance variables M# and MA# to break the tie if the minimum point has an objective function value equal to the maximum value found so far on this iteration. Sets the OB# equal to the value of the objective at the new point, MA# equal to the constraints distance from the current point, RO equal to the row in the tableau for the upper bound, and CO to the column for the pivot.

```
3347 IF K#=OB# AND MA#>=M# THEN 3352
3348 OB#=K#
3349 MA#=M#
3350 RO=I
3351 CO=CC
3352 NEXT I
```

As with the upper bound, finds the maximum minimum value of the objective for the lower bounds.

```
3353 FOR J=2 TO ND+1
3354 IF M#(J,1)=0# OR (M#(J,1)-B#(J,1))>=0# THEN 3378
3355 K#=BI#
3356 FOR I=2 TO ND+1
3357 IF M#(J,1)=BI# OR -B#(J,I)<=SM# THEN 3369
```

```
3358 L#=(M#(J,1)-B#(J,1))/(-B#(J,I))
3359 REM-------------------------------------------------------------
3360 A#=0#
3361 FOR K=2 TO ND+1
3362 IF A#(1,K)<>0# THEN A#=A#+A#(1,K)*(B#(K,1)-B#(K,I)*L#)
3363 NEXT K
3364 REM-------------------------------------------------------------
3365 REM A#=BB#-B#(1,I)*L#
3366 IF A#>=K# THEN 3369
3367 K#=A#
3368 CC=I
3369 NEXT I
3370 M#=(M#(J,1)-B#(J,1))*(M#(J,1)-B#(J,1))
3371 IF K#<OB# THEN 3378
3372 IF K#=BI# THEN RETURN
3373 IF K#=OB# AND MA#>=M# THEN 3378
3374 OB#=K#
3375 MA#=M#
3376 RO=J+ND+1
3377 CO=CC
3378 NEXT J
```

As with the lower bounds, finds the maximum minimum value of the objective from among the A#(MD+2,ND+8) array constraints.

```
3379 FOR K=2 TO MD+2
3380 REM-------------------------------------------------------------
3381 IF A#(K,ND+2)<>0# THEN GOSUB 4000:REM PAR-TANL
3382 IF A#(K,ND+3)<>0# THEN GOSUB 4200:REM HYP-TANL
3384 IF A#(K,ND+4)<>0# THEN GOSUB 4400:REM SHT-TANL
3385 REM-------------------------------------------------------------
3386 P#(1)=-A#(K,1)
3387 FOR I=2 TO ND+1
3388 P#(I)=0#
3389 P#(1)=P#(1)+A#(K,I)*B#(I,1)
3390 NEXT I
3391 SN=R(K)
3392 IF SN=0 THEN SN=-SGN(P#(1))
3394 P#(1)=P#(1)*CDBL(SN)
3395 IF P#(1)>=-SM# THEN 3428
3396 Z#=0#
3397 FOR I=2 TO ND+1
3398 Z#=Z#+A#(K,I)*A#(K,I)
3399 IF A#(K,I)=0# THEN 3403
3400 FOR J=2 TO ND+1
3401 P#(J)=P#(J)+A#(K,I)*B#(I,J)*CDBL(SN)
3402 NEXT J
3403 NEXT I
```

```
3404 K#=BI#
3405 FOR I=2 TO ND+1
3406 IF M#(I,1)=BI# OR P#(I)<=SM#  THEN 3418
3407 L#=P#(1)/P#(I)
3408 REM-----------------------------------------------------------------
3409 A#=0#
3410 FOR J=2 TO ND+1
3411 IF A#(1,J)<>0# THEN A#=A#+A#(1,J)*(B#(J,1)-B#(J,I)*L#)
3412 NEXT J
3413 REM-----------------------------------------------------------------
3414 REM A#=BB#-B#(1,I)*L#
3415 IF A#>=K# THEN 3418
3416 K#=A#
3417 CC=I
3418 NEXT I
3419 M#=BI#
3420 IF Z#>SM# THEN M#=(P#(1)*P#(1))/Z#
3421 IF K#<OB# THEN 3428
3422 IF K#=BI# THEN RETURN
3423 IF K#=OB# AND MA#>=M# THEN 3428
3424 OB#=K#
3425 MA#=M#
3426 RO=K+ND+1+ND+1
3427 CO=CC
3428 NEXT K
```

Transforms the selected constraint and stores it in the pivot row array.

```
3429 GOSUB 3600:REM TRAN-CON
```

Sets the objective function less than or equal to the current value of the objective less .1 if no constraints were violated. If this constraint is infeasible, then the current solution is optimal.

```
3430 IF OB#>-BI# THEN 3434
3431 A#(MD+2,1)=-BB#+.1#
3432 ER=1
3433 GOTO 3325
```

Transforms the tableau using a Gauss-Jordan elimination on the pivot row and column CO.

```
3434 GOSUB 3700:REM TRAN-INV
```

Returns to the main routine with ER=3 if the tableau transformation fails.

```
3435 IF ER=3 THEN RETURN
3436 GOTO 3320
```

Constraint Transformation Subroutine -- File TRAN-CON

Same as for dual simplex method.

Inversion Subroutine -- File TRAN-INV

Same as for dual simplex method.

Reinversion Subroutine -- File TRAN-RIV

Same as for dual simplex method.

Parabolic Subroutines -- Files PAR-TANA, PAR-TANL, or PAR-TANG

Same as for dual simplex method with nonlinear constraints.

Hyperbolic Subroutines -- Files HYP-TANA, HYP-TANL, or HYP-TANG

Same as for dual simplex method with nonlinear constraints.

Hyperbolic Sheet Subroutines -- Files SHT-TANA, SHT-TANL, or SHT-TANG

Same as for dual simplex method with nonlinear constraints.

Corner Cut Method with Nonlinear Constraints: Solutions to Example 11 Minimum Project Man Count Problem

The corner cut method is used to solve the same problems as used in the text for the dual simplex method and its variations.

Table B-1 can be used to reconstruct the corner cut method with nonlinear constraints from the computer disk and to organize subroutines from previous code listings. Since BASIC code is dependent on program line numbers for subroutine branching, the line numbers must be maintained as listed below.

Table B-1. Corner cut method with nonlinear constraints BASIC program table of contents

File	Program lines	Page	Routine
MAIN-CRN	0001-0138	479	Corner cut method
INPT-OBJ	1200-1220	61	Objective coefficient input subroutine
INPT-CON	1300-1328	63	Constraint coefficient input subroutine
INPT-CON	1400-1440	65	Constraint type input subroutine
INPT-BND	1500-1523	67	Variable bounds input subroutine
UTIL-OPT	1800-1803	22	Option line subroutine
UTIL-ERS	1850-1853	22	Erase option subroutine
UTIL-CON	1860-1866	22	Continue line subroutine
UTIL-CHX	1870-1882	23	Data check subroutine
REPT-SMP	2200-2224	69	Simplex report subroutine
ALGR-KEY	3000-3057	200	Constraint key subroutine
ALGR-CRN	3300-3434	495	Corner cut algorithm subroutine
TRAN-CON	3600-3633	74	Constraint transformation subroutine
TRAN-INV	3700-3743	76	Basis inversion subroutine
TRAN-RIV	3900-3960	78	Basis reinversion subroutine
PAR-TANL	4000-4034	209	Supporting line for parabolic subroutine
HYP-TANL	4200-4221	217	Supporting line for hyperbolic subroutine
SHT-TANL	4400-4425	228	Supporting plane for sheet subroutine

The results of using the corner cut method with supporting planes derived with the line search alogrithm to solve the example 11 man count problem are summarized in table B-2. The data are arranged and displayed in the same format as in table 10 for the dual simplex method using supporting planes derived with the line search algorithm.

The primary difference between the solutions found with the dual simplex method and the corner cut method is the increases in the number of iterations. The difference in the values of the objective function (simplex method 12.75378215430363, corner cut 12.75378215451283) are not evident when only eight (8) digits are listed for the solutions at the level of precision were 1E-10 or less is assumed to be zero (0).

Table B-2. Corner cut method, using supporting planes derived with the line search algorithm, solutions to the example 11 minimum project man count problem with hyperbolic man count functions

Dur.	100	90	80	70	60
Itr.	152	136	150	147	133
Sec.	393	342	367	384	361
Obj.	12.75378	14.17086	15.94222	18.21968	21.25630
T1	19.12823	17.21543	15.30260	13.38977	11.47694
T2	48.67250	43.80527	38.93801	34.07073	29.20350
T3	75.02407	67.52169	60.01927	52.51685	45.01447
T4	100	90	80	70	60
M5	0.13329	0.14810	0.16661	0.19041	0.22215
M6	0.38965	0.43294	0.48706	0.55664	0.64942
M7	1.56836	1.74262	1.96045	2.24051	2.61393
M8	1.35390	1.50433	1.69237	1.93414	2.25650
M9	1.89742	2.10824	2.37177	2.71059	3.16236
M10	2.40231	2.66924	3.00289	3.43187	4.00386
M11	1.25232	1.39147	1.56541	1.78904	2.08721
M12	1.64363	1.82626	2.05454	2.34805	2.73939
M13	1.11287	1.23652	1.39109	1.58981	1.85478
M14	1	1.11111	1.25	1.42857	1.66666

Table B-2. Continued

Dur.	50	40	30	20	10
Itr.	152	189	139	214	195
Sec.	364	480	375	597	534
Obj.	25.50756	31.88445	42.51260	63.76891	127.53782
T1	9.56412	7.65130	5.73847	3.82564	1.91282
T2	24.33624	19.46901	14.60175	9.73449	4.86725
T3	37.51203	30.00963	22.50722	15.00482	7.50241
T4	50	40	30	20	10
M5	0.26658	0.33322	0.44430	0.66645	1.33290
M6	0.77930	0.97413	1.29884	1.94827	3.89654
M7	3.13672	3.92090	5.22787	7.84181	15.68360
M8	2.70780	3.38474	4.51300	6.76950	13.53901
M9	3.79483	4.74355	6.32473	9.48708	18.97416
M10	4.80462	6.00578	8.00771	12.01158	24.02318
M11	2.50465	3.13082	4.17443	6.26164	12.52328
M12	3.28727	4.10909	5.47879	8.21819	16.43638
M13	2.22574	2.78218	3.70957	5.56436	11.12873
M14	2	2.5	3.33333	5	10

Table B-3 can be used to reconstruct the corner cut method with supporting planes derived with the Gordian algorithm from the computer disk and to organize subroutines from previous code listings. The line numbers must be maintained as listed below.

Table B-3. Corner cut method with nonlinear constraints BASIC program table of contents

File	Program lines	Page	Routine
MAIN-CRN	0001-0138	479	Corner cut method
INPT-OBJ	1200-1220	61	Objective coefficient input subroutine
INPT-CON	1300-1328	63	Constraint coefficient input subroutine
INPT-CON	1400-1440	65	Constraint type input subroutine
INPT-BND	1500-1523	67	Variable bounds input subroutine
UTIL-OPT	1800-1803	22	Option line subroutine
UTIL-ERS	1850-1853	22	Erase option subroutine
UTIL-CON	1860-1866	22	Continue line subroutine
UTIL-CHX	1870-1882	23	Data check subroutine
REPT-SMP	2200-2224	69	Simplex report subroutine
ALGR-KEY	3000-3057	200	Constraint key subroutine
ALGR-CRN	3300-3434	495	Corner cut algorithm subroutine
TRAN-CON	3600-3633	74	Constraint transformation subroutine
TRAN-INV	3700-3743	76	Basis inversion subroutine
TRAN-RIV	3900-3960	78	Basis reinversion subroutine
PAR-TANG	4000-4018	211	Supporting line for parabolic subroutine
HYP-TANG	4200-4226	219	Supporting line for hyperbolic subroutine
SHT-TANG	4400-4430	231	Supporting plane for sheet subroutine

To run the corner cut method with another version of the nonlinear constraint subroutines requires changing the calling subroutine. The following lines must be changed in the corner cut algorithm ALGR-CRN to call the parabolic, hyperbolic, and sheet subroutines.

```
3380 IF A#(K,ND+2)<>0# THEN GOSUB 4000:REM PAR-TANG
3382 IF A#(K,ND+3)<>0# THEN GOSUB 4200:REM HYP-TANG
3384 IF A#(K,ND+4)<>0# THEN GOSUB 4400:REM SHT-TANG
```

Table B-4. Corner cut method, using supporting planes derived with the Gordian algorithm, solutions to the example 11 minimum project man count problem with hyperbolic man count functions

Dur.	100	90	80	70	60
Itr.	192	215	192	218	160
Sec.	412	439	391	435	335
Obj.	12.75378	14.17086	15.94222	18.21968	21.25630
T1	19.12826	17.21542	15.30258	13.38978	11.47696
T2	48.67252	43.80526	38.93799	34.07073	29.20351
T3	75.02414	67.52169	60.01926	52.51684	45.01445
T4	100	90	80	70	60
M5	0.13329	0.14810	0.16661	0.19041	0.22215
M6	0.38965	0.43294	0.48706	0.55664	0.64942
M7	1.56835	1.74262	1.96045	2.24051	2.61392
M8	1.35390	1.50433	1.69237	1.93414	2.25650
M9	1.89741	2.10824	2.37177	2.71059	3.16236
M10	2.40232	2.66924	3.00289	3.43187	4.00385
M11	1.25232	1.39147	1.56541	1.78904	2.08721
M12	1.64363	1.82626	2.05454	2.34805	2.73939
M13	1.11287	1.23652	1.39109	1.58981	1.85478
M14	1	1.11111	1.25	1.42857	1.66666

Table B-4. Continued

Dur.	50	40	30	20	10
Itr.	179	188	277	318	260
Sec.	366	381	590	652	572
Obj.	25.50756	31.88445	42.51260	63.76891	127.53782
X1	9.56412	7.65129	5.73846	3.82564	1.91282
X2	24.33625	19.46900	14.60174	9.73450	4.86725
X3	37.51204	30.00963	22.50722	15.00481	7.50240
X4	50	40	30	20	10
X5	0.26658	0.33322	0.44430	0.66645	1.33290
X6	0.77930	0.97413	1.29884	1.94827	3.89654
X7	3.13672	3.92090	5.22788	7.84180	15.68361
X8	2.70780	3.38475	4.51300	6.76950	13.53901
X9	3.79483	4.74354	6.32472	9.48710	18.97418
X10	4.80462	6.00578	8.00771	12.01156	24.02314
X11	2.50465	3.13082	4.17442	6.26164	12.52329
X12	3.28727	4.10909	5.47879	8.21819	16.43638
X13	2.22574	2.78218	3.70957	5.56434	11.12871
X14	2	2.5	3.33333	5.00000	10

The results of using the corner cut method with supporting planes derived with the Gordian algorithm to solve example 11 are summarized in table B-4. The data are arranged and displayed in the same format as in table 10 for the dual simplex method.

Table B-5 can be used to reconstruct the corner cut method with deep cuts derived with the line search algorithm from the computer disk.

Table B-5. Corner cut method BASIC program table of contents

File	Program lines	Page	Routine
MAIN-CRN	0001-0138	479	Corner cut method
INPT-OBJ	1200-1220	61	Objective coefficient input subroutine
INPT-CON	1300-1328	63	Constraint coefficient input subroutine
INPT-TYP	1400-1440	65	Constraint type input subroutine
INPT-BND	1500-1523	67	Variable bounds input subroutine
UTIL-OPT	1800-1803	22	Option line subroutine
UTIL-ERS	1850-1853	22	Erase option subroutine
UTIL-CON	1860-1866	22	Continue line subroutine
UTIL-CHX	1870-1882	23	Data check subroutine
REPT-SMP	2200-2224	69	Simplex report subroutine
ALGR-KEY	3000-3057	200	Constraint key subroutine
ALGR-CRN	3300-3434	495	Corner cut algorithm subroutine
TRAN-CON	3600-3633	74	Constraint transformation subroutine
TRAN-INV	3700-3743	76	Basis inversion subroutine
TRAN-RIV	3900-3960	78	Basis reinversion subroutine
PAR-DEPL	4100-4189	311	Deep cut for parabolic subroutine
HYP-DEPL	4300-4285	318	Deep cut for hyperbolic subroutine
SHT-DEPL	4500-4592	325	Deep cut for sheet subroutine

To run the corner cut method with the deep cut subroutines, the following lines must be changed in the corner cut algorithm ALGR-CRN.

```
3381 IF A#(K,ND+2)<>0# THEN GOSUB 4100:REM PAR-DEPL
3383 IF A#(K,ND+3)<>0# THEN GOSUB 4300:REM HYP-DEPL
3385 IF A#(K,ND+4)<>0# THEN GOSUB 4500:REM SHT-DEPL
```

The results of using the corner cut method are summarized in table

B-6. The data are arranged and displayed in the same format as for the solutions found using Gomory's algorithm in table 19.

Table B-6. Corner cut method, using deep cuts derived with the line search algorithm, solutions to the example 11 minimum project man count problem with hyperbolic man count functions

Dur.	100	90	80	70	60	50	40	30	20	10
Itr.	23	22	24	23	25	25	26	26	29	30
Sec.	51	51	53	53	61	54	54	60	79	83
Obj.	14.8	16.3	18.0	20.1	22.94	27.0	33.2	44.7	65	130.9
T1	15	15	15	15	10	10	10	6	5	2
T2	55	45	40	35	28	25	20	14	10	5
T3	80	70	60	52	45	38	30	24	15	8
T4	100	90	80	70	60	50	40	30	20	10
M5	1	1	1	1	1	1	1	1	1	1.4
M6	1	1	1	1	1	1	1	1.4	2	4
M7	2	2	2	2	3	3	3	5	6	15
M8	1	1.5	1.7	2	2.3	2.8	4	5	8	14
M9	2	2	2.6	3	3	4.0	5	5	10	17
M10	3	3	3	3.4	4	5	6	10	12	30
M11	1.1	1.4	1.7	1.9	2	2.6	3.6	4	7	12
M12	1.6	1.8	2	2.3	2.9	3.2	4	6	8	16
M13	1.1	1.3	1.5	1.7	1.8	2.3	3	3.8	6	11.5
M14	1	1.2	1.4	1.6	1.8	2	2.6	3.4	5	10

Round Up Integer Solution to Minimum Project Man Count Problem

Gomory's method is an all integer method so all the values of the solution in table 19 are integer. An interesting observation is that the round up solution to the corner cut method using deep cuts is also the integer solution.

Without proof, a good starting solution to the branch and bound method would be the round up solution. Not only has this been a good solution in all the cases tried to date, but the round up solution has also been the optimal solution for the minimum project man count problem.

APPENDIX C: ELLIPSOIDAL METHOD

The variations of the simplex method are not the only means of solving the nonlinear problem. Another approach is the ellipsoidal method. Even though the method is still no match for the simplex method, its potential for solving problems with almost any differentiable convex constraints makes it worth presenting as a alternate to the simplex methods.

Theory of Ellipsoidal Deep Cut Method (Perfect Arithmetic)

The objective function and feasible region of the LP problem can be written in matrix notation as:

$$\overline{c}'\overline{x} <= z$$

$$\overline{Ax} >= \overline{a}$$

where the $\overline{c}$ vector is the coefficients of the objective function which in this example is linear, $\overline{A}$ is the coefficients of the constraint equations, and $\overline{a}$ is a vector of the constants of the constraint equations.

As an initial solution for the algorithm, z is set equal to a very large number and $\overline{x}$ is set equal to the zero vector ($\overline{0}$). This starting solution can be geometrically visualized as a point defining the center of a sphere constructed so that its radius r is large enough so that the circle contains all of the feasible region defined by the LP constraints. The objective function $\overline{c}'\overline{x}$ is also included in the constraint set, but for the first iteration it is set less than or equal to z or infinity

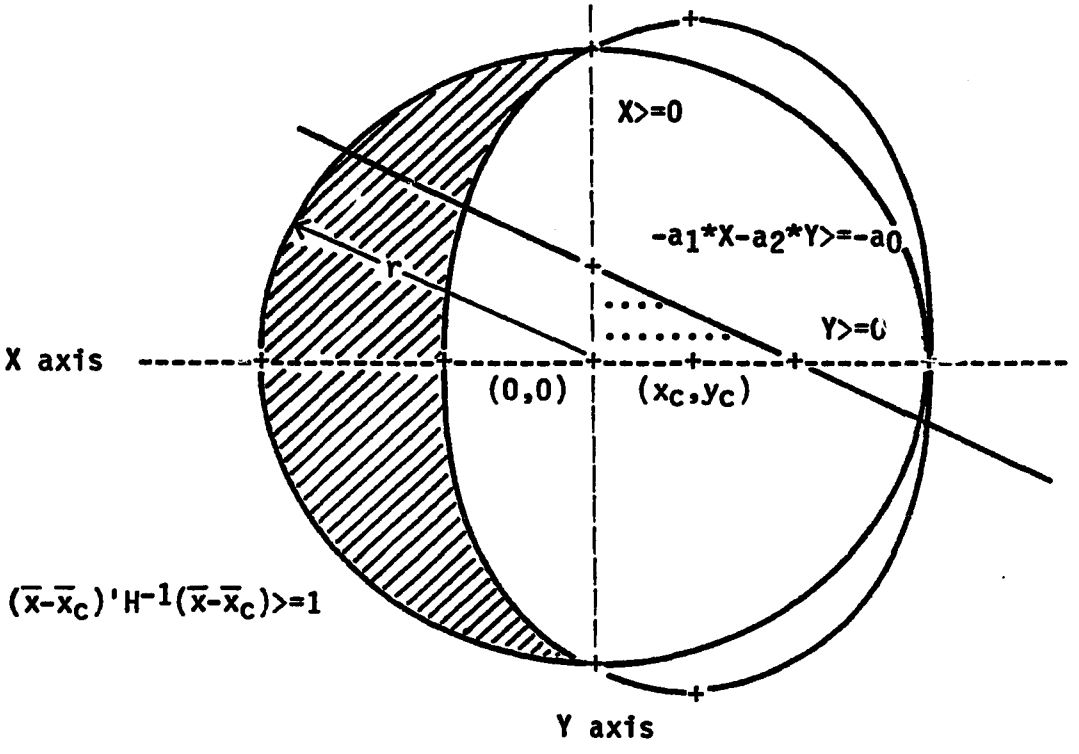


Figure C-1. Ellipsoidal method starting cirle in two dimensions

so it is not a boundary of the feasible region.

This circle, or ellipsoid in later iterations, can be defined in matrix notation as a n dimension sphere with radius r centered at:

$$\bar{x}_c=\bar{0}$$

by the matrix:

$$\bar{H}=r\bar{r}\bar{I}$$

and the set of $\bar{x}$ points:

$$\text{Ellipse}=\{\ \bar{x}\ e\ \text{Real}\ :\ (\bar{x}-\bar{x}_c)'H^{-1}(\bar{x}-\bar{x}_c)<=1\ \}$$

As in the simplex method, a slack, in this case w_i, can be calculated for each of the i equations indicating the percentage of the ellipsoid's radius that is "cut off" or violated by the constraint.

$$w_i=\frac{(a-\bar{a}_i'\bar{x}_c)}{\sqrt{\bar{a}_i'\overline{Ha}_i}}$$

If the violated constraint, say p, with the greatest slack wp is found:

w_p= maximum (w_i) for all i constraints

p=i of maximum constraint

then the starting circle can be reduced in volume by "cutting" off that portion that lies outside the violated constraint and a new ellipse drawn around the remaining portion as shown in figure C-1.

In matrix notation this results in:

$$\bar{x}_{cnew}=\bar{x}_c+\frac{d\overline{Ha}_p}{\sqrt{\bar{a}_p'\overline{Ha}_p}}$$

and:

$$\bar{H}_{new}=e*\left(\bar{H}-f*\frac{\overline{Ha}_p\bar{a}_p'\bar{H}}{\bar{a}_p'\overline{Ha}_p}\right)$$

where:

$$d=\frac{(1+n*w_p)}{(n+1)}$$

$$e=\frac{n\hat{}2*(1-w_p\hat{}2)}{(n\hat{}2-1)}$$

$$f=\frac{1-(n-1)*(1-w)}{(n+1)*(1+w_p)}$$

This new point can be checked with the constraints of the LP problem; and if a constraint is violated, a new point and smaller ellipsoid found.

If the LP is not infeasible, an assumption of the method, then a point $\overline{xc}$ can be found that satisfies all the constraints by continually reducing the size of the ellipse.

Once the constraints are satisfied, the objective function constant z is set equal to $\overline{c}'\overline{xc}$, and the objective function used as a cut through the center of the current ellipse. (Geometrically, passing a cut in the form of the objective function through the center of the ellipse reduces the volume of the ellipse even though the slack wi for the function is zero (0).)

After the new point and ellipse resulting from the objective function cut is found, the algorithm continues to reduce the volume of the ellipse with cuts derived from the constraints of the LP problem if they are violated or the objective function. This process continues until the ellipse is reduced to a radius of the desired precision and center of the ellipse is considered the optimal solution in the feasible region.

Theory of Goffin's Method (Finite Arithmetic)

The precision needed to execute the ellipsoidal algorithm can not be maintained on present day computers. Often the $\overline{H}$ matrix or the matrix defining the ellipsoid becomes singular when one of the axes of the ellipsoid becomes small relative to the other axes. To circumvent this problem, the volume reduction formulas for the ellipsoid can be modified to tolerate the limit of sixteen (16) significant digits available on a sixteen (16) bit word computer.

A modification of the ellipsoidal algorithm by Goffin[58] is used in the following computer code. Goffin factors the $\overline{H}$ matrix into a two $\overline{B}$ matrices:

$$\overline{BB}'=\overline{H}$$

and then redefines the ellipsoid:

$$\overline{x}_c=\overline{0}$$

$$\overline{B}=r\overline{I}$$

and the slacks in terms of the $\overline{B}$ matrix:

$$q_i= \overline{a}_i\overline{BB}'\overline{a}_i'$$

$$w_i= \frac{a-a_i'x_c}{\sqrt{q_i}}$$

w_p=maximum (w_i) for all i constraints

p=i of maximum constraint

The transformation of the ellipsoid parameters are then:

$$\overline{x}_{cnew}=\overline{x}_c+d*\left(\frac{\overline{BB}'a_p'}{\sqrt{q_p}}\right)$$

$$\overline{B}_{new}=e*\sqrt{d}*\left(\overline{B}-\left(1-\sqrt{f}\right)*\frac{(\overline{BB}'\overline{a}_p'\overline{a}_p\overline{B})}{\left(\sqrt{q_p}\right)^2}\right)$$

where:

$$d=\frac{(1+n^2w_p)}{(n+1)}$$

$$e=\frac{1+(1+n*w_p)^2}{24*n^2}$$

$$f=\frac{(n-1)*(1-w_p)}{(n+1)*(1+w_p)}$$

The following tableau is used to store the input data and $\bar{B}$ matrix of the ellipsoidal method. The similarities with the simplex tableau allows many of the input routines and reports to be shared by both methods. The corresponding program arrays used in the BASIC program code are given on the right hand column.

			program variable
	$\bar{0}$	$\bar{0}'$	Q#(1,ND+1)
	$\bar{0}$	$\bar{Q}$	Q#(2 to ND+1,ND+1)
	0	$\bar{c}'$	A#(1,ND+1)
	$-\bar{a}$	$\bar{A}$	A#(2 to MD+1,ND+8) (column one negative)
z=	$\bar{c}'\bar{x}_c+\bar{x}_c\bar{Q}\bar{x}_c'$	$\bar{a}_p\bar{B}$	B#(1,ND+1)
$\bar{x}$=	$\bar{x}_c$	$\bar{B}$	B#(2 to ND+1,ND+1)
	$-a_p$	$\bar{a}_p'$	P#(ND+1)
	-	-	

Ellipsoidal Method BASIC Code

The following code is a version of the ellipsoidal method proposed by Goffin. Several different versions of the method were tried, and the following method had the best convergence rate and numerical stability.

Considering the current interest in the ellipsoidal method, future research will most likely improve the convergence rates demonstrated by Goffin's algorithm.

All the BASIC programs listed in the text consist of a main calling routine and a series of input, output, and processing subroutines. The routines listed in the text are also on a computer disk which is compatible with IBM mirco-computers.

Elipsoidal Method Main Routine -- File MAIN-GOF

The ellipsoidal method main calling routine (MAIN-GOF) dimensions ten (10) arrays; writes the option menu to the screen as shown in figure

```
                    ELLIPSOID METHOD

NUMBER OF CONSTRAINTS          10
NUMBER OF VARIABLES            14
MAXIMUM ITERATIONS             1000

                    M-RETURN TO MENU

          O-OBJECTIVE COEFFICIENTS
          A-CONSTRAINT COEFFICIENTS
          C-CONSTRAINT TYPES
          B-BOUNDED VARIABLES
          Q=QUADRATIC COEFFICIENTS
          U-EXECUTE ALGORITHM
          R-REPORT LISTING
          S-SAVE  F-FETCH
          N-NEW PROBLEM

OPTION ?
                          O'X+X'QX=z
   1*X*2*Y>=c    1*X*2*Y>=c    1*Y*a*(X-b)^2>=c
```

Figure C-2. Ellipsoid method main menu screen

C-2; calls the utility subroutines UTIL-OPT, UTIL-ERS, UTIL-CON, UTIL-CHX; calls the data input and output subroutines INPT-OBJ, INPT-CON,

INPT-TYP, INPT-BND, INPT-QUD, and REPT-SMP; calls and times the processing algorithms ALGR-KEY and ALGR-GOF; and saves and fetches the input data to disk.

```
1 REM                    *ELLIPSOIDAL METHOD*
2 REM-----------------------MAIN-ELP-------------------------------

3 REM      BI# - MACHINE INFINITE
4 REM      ER  - ERROR KEY
5 REM      IR  - MAXIMUM NUMBER OF ITERATIONS
6 REM      MD  - NUMBER OF CONSTRAINTS
7 REM      ND  - NUMBER OF VARIABLES
8 REM      RO  - PIVOT ROW FOR SIMPLEX TRANSFORMATION
9 REM      SM# - MACHINE ZERO
10 REM     SN  - CONSTRAINT TYPE SIGN
11 REM     W#  - DEPTH OF CUT INTO ELLIPSE IN PERCENTAGE OF RADIUS
12 REM     A#(MD+1,ND+8) - ORIGINAL DATA AND KEYS
13 REM     B#(ND+1,ND+1) - ELLIPSOID PARAMETRIC MATRIX
14 REM     M#(ND+1,2) - LOWER AND UPPER BOUNDS OF VARIABLES
15 REM     P#(ND+1) - PIVOT ROW VECTOR
16 REM     Q#(ND+1,ND+1) - QUADRATIC MATRIX
17 REM     R(MD+1) - TYPE OF CONSTRAINT (1-">=",0-"=",-1-"<=")
18 REM     V#(ND+1) - WORK AREA
19 REM     X#(ND) - SOLUTION VECTOR
20 REM -------------------------------------------------------------
```

Sets MD to the default number of constraints and ND to the number of variables in the CP problem. Sets IR to the default maximum number of iterations. Sets BI# to a number considered machine infinite and SM# to a number considered machine zero.

```
21 MD=0
22 ND=0
23 IR=1000
24 BI#=1E+10
25 SM#=1E-10
```

Prompts and reads from the keyboard the number of constraints ND in the CP problem; the number of variables MD; and the maximum number of iterations.

```
26 CLS
27 LOCATE 1,10:PRINT "ELLIPSOID METHOD"
28 LOCATE 3,1:PRINT "NUMBER OF CONSTRAINTS":LOCATE 3,31:INPUT "",L$
29 GOSUB 1870:REM UTIL-CHX
30 IF Z#<>BI# THEN MD=Z#
```

```
31 LOCATE 3,30:PRINT MD,"    ":LOCATE 4,1:PRINT "NUMBER OF VARIABLES"
   :LOCATE 4,31:INPUT "",L$
32 GOSUB 1870:REM UTIL-CHX
33 IF Z#<>BI# THEN ND=Z#
34 LOCATE 4,30:PRINT ND,"    ":LOCATE 5,1:PRINT "MAXIMUM ITERATIONS"
   :LOCATE 5,31:INPUT "",L$
35 GOSUB 1870:REM UTIL-CHX
36 IF Z#<>BI# THEN IR=Z#
37 LOCATE 5,30:PRINT IR,"    "
```

Dimensions the objective function and constraint coefficients array A#(MD+1,ND+8); the ellipsoid matrix array B#(ND+1,ND+1); the holding array H#(ND+1); the upper and lower bounds array M#(ND+1,2), the cut constraint P#(ND+1); the augmented $\overline{Q}$ matix array Q#(ND+1,ND+1) for the quadratic objective function; the constraint type array R(MD+1); two working space arrays S#(ND+1) and V#(ND+1); and the solution array X#(ND).

These arrays are the same as used in Beale's method with the exception of the B#(ND+1,ND+1) which is the ellipsoid matrix $\overline{B}$ rather than the augmented $\overline{B}$ and augmented $\overline{B}$ inverse.

```
38 DIM A#(MD+1,ND+8)
39 DIM B#(ND+1,ND+1)
40 DIM H#(ND+1)
41 DIM M#(ND+1,2)
42 DIM P#(ND+1)
43 DIM Q#(ND+1,ND+1)
44 DIM R(MD+1)
45 DIM S#(ND+1)
46 DIM V#(ND+1)
47 DIM X#(ND)
48 FOR I=2 TO MD+1
49 R(I)=1
50 NEXT I
```

Prints the option menu to the screen; calls the option line routine UTIL-OPT; and pauses for the entry of "M", "O", "A", "C", "B", "Q", "U", "R", "S", "F", or "N" for the option variable L$.

```
51 LOCATE 8,15: PRINT          "M-RETURN TO MENU"
52 LOCATE 10,10:PRINT        "O-OBJECTIVE COEFFICIENTS"
53 LOCATE 11,10:PRINT        "A-CONSTRAINT COEFFICIENTS"
54 LOCATE 12,10:PRINT        "C-CONSTRAINT TYPES"
55 LOCATE 13,10:PRINT        "B-BOUNDED VARIABLES"
56 LOCATE 14,10:PRINT        "Q-QUADRATIC COEFFICIENTS"
57 LOCATE 15,10:PRINT        "U-EXECUTE ALGORITHM"
58 LOCATE 16,10:PRINT        "R-REPORT LISTING"
```

```
59 LOCATE 17,10:PRINT       "S-SAVE F-FETCH"
60 LOCATE 18,10:PRINT       "N-NEW PROBLEM"
61 LOCATE 22,1:PRINT "                  OX'+XQX'=z"
62 LOCATE 23,1:PRINT "1*X*2*Y>=c  1*X*(2*Y-3*Z)>=c  1*Y*a*(X-b)^2>=c"
63 GOSUB 1800:REM UTIL-OPT
64 LOCATE 21,8:INPUT "",L$
```

Calls either the objective function input subroutine INPT-OBJ, the constraint input subroutine INPT-CON, the constraint type subroutine INPT-CON, the upper and lower bounds input subroutine INPT-BND, the $\overline{Q}$ matrix input subroutine INPT-QUD, the processing subroutine ALGR-KEY, or the report subroutine REPT-SMP based on the option variable L$.

```
65 CLS
66 H=0
67 G=2
68 IF L$<>"O" THEN 71
69 GOSUB 1200:REM INPT-OBJ
70 GOTO 65
71 IF L$<>"A" THEN 74
72 GOSUB 1300:REM INPT-CON
73 GOTO 65
74 IF L$<>"C" THEN 77
75 GOSUB 1400:REM INPT-TYP
76 GOTO 65
77 IF L$<>"B" THEN 80
78 GOSUB 1500:REM INPT-BND
79 GOTO 65
80 IF L$<>"Q" THEN 83
81 GOSUB 1600:REM INPT-QUD
82 GOTO 65
83 IF L$<>"U" THEN 92
84 TM=3600*VAL(MID$(TIME$,1,2))+60*VAL(MID$(TIME$,4,2))+VAL(MID$(TIME$,7
   ,2))
85 GOSUB 3000:REM ALGR-KEY
```

Sets the solution equal to the last ellipsoid center found in the ellipsoidal algorithm.

```
86 OB#=B#(1,1)
87 FOR I=1 TO ND
88 X#(I)=B#(I+1,1)
89 NEXT I
90 TM=3600*VAL(MID$(TIME$,1,2))+60*VAL(MID$(TIME$,4,2))+VAL(MID$(TIME$,7
   ,2))-TM
91 GOTO 51
92 IF L$<>"R" THEN 95
93 GOSUB 2200:REM REPT-SMP
```

```
94 GOTO 65
```

Saves the content of MD, ND, Q#(ND+1,ND+1), M#(ND+1,2), A#(MD+1,ND+8), and R(MD+1) to the disk file "DATA" as an ASCI file if option "S" is selected.

```
95 IF L$<>"S" THEN 113
96 OPEN "O",#1,"DATA"
97 PRINT #1,STR$(MD)
98 PRINT #1,STR$(ND)
99 FOR I=1 TO ND+1
100 FOR J=1 TO ND+1
101 PRINT #1,STR$(Q#(I,J))
102 NEXT J
103 PRINT #1,STR$(M#(I,1))
104 PRINT #1,STR$(M#(I,2))
105 NEXT I
106 FOR I=1 TO MD+1
107 FOR J=1 TO ND+8
108 PRINT #1,STR$(A#(I,J))
109 NEXT J
110 PRINT #1,STR$(R(I))
111 NEXT I
112 CLOSE #1
```

Loads to MD, ND, Q#(ND+1,ND+1), M#(ND+1,2), A#(MD+1,ND+8), and R(MD+1) the disk file "DATA" if option "F" is selected.

```
113 IF L$<>"F" THEN 139
114 OPEN "I",#1,"DATA"
115 INPUT #1,X$
116 MD=VAL(X$)
117 INPUT #1,X$
118 ND=VAL(X$)
119 FOR I=1 TO ND+1
120 FOR J=1 TO ND+1
121 INPUT #1,X$
122 Q#(I,J)=VAL(X$)
123 NEXT J
124 INPUT #I,X$
125 M#(I,1)=VAL(X$)
126 INPUT #1,X$
127 M#(I,2)=VAL(X$)
128 NEXT I
129 FOR I=1 TO MD+1
130 FOR J=1 TO ND+8
131 INPUT #1,X$
132 A#(I,J)=VAL(X$)
```

```
133 NEXT J
134 INPUT #1,X$
135 R(I)=VAL(X$)
136 NEXT I
137 CLOSE #1
138 GOTO 51
```

Restarts the execution of the program if option "N" is selected.

```
139 IF L$="N" THEN RUN
140 GOTO 51
```

Utility Subroutines -- Files UTIL-OPT, UTIL-ERS, UTIL-CON, and UTIL-CHX

Same as for critical path method.

Input Subroutines -- Files INPT-OBJ, INPT-CON, INPT-TYP, and INPT-BND

Same as for dual simplex method.

Input Subroutine -- File INPT-QUD

Same as for Beale's method.

Report Subroutine -- File REPT-SMP

Same as dual simplex method.

Constraint Key Algorithm -- File ALGR-KEY

Same as for dual simplex method with nonlinear constraints.

Goffin's Algorithm Subroutine -- File ALGR-GOF

Goffin's algorithm is a modification of the ellipsoidal method designed to compensate for the lack of computer precision. For the mathematical formulation, see the section on the theory of the ellipsoidal method.

```
3300 REM                    *GOFFIN'S ALGORITHM*
3301 REM-------------------------ALGR-GOF-------------------------------
```

Prompts for the radius of the starting sphere as shown in figure C-3 and intializes the $\overline{B}$ matrix.

```
                ELLIPSOID PARAMETERS

NUMBER OF CONSTRAINTS              10
NUMBER OF VARIABLES                14
UPPER BOUND ON VARIABLES          100


              (PRESS RETURN TO CONTINUE)
```

Figure C-3. Ellipsoid parameter input screen

```
3302 CLS
3303 LOCATE 1,12:PRINT "ELLIPSOID PARAMETERS"
3304 LOCATE 3,1:PRINT "NUMBER OF CONSTRAINTS":LOCATE 3,30:PRINT MD
3305 LOCATE 4,1:PRINT "NUMBER OF VARIABLES":LOCATE 4,30:PRINT ND
3306 LOCATE 5,1:PRINT "UPPER BOUND ON VARIABLES":LOCATE 5,31:INPUT "",L$
3307 GOSUB 1870:REM DATA UTIL-CHX
3308 IF Z#<>BI# THEN R#=Z#*CDBL(SQR(ND))
3309 LOCATE 5,30:PRINT R#/CDBL(SQR(ND))
3310 LOCATE 20,7:PRINT "(PRESS RETURN TO CONTINUE)"
3311 LOCATE 5,31:INPUT "",L$
3312 IF L$<>"" THEN 3307
3313 CLS
3314 FOR I=1 TO ND+1
3315 H#(I)=BI#
3316 FOR J=1 TO ND+1
3317 B#(I,J)=0#
3318 NEXT J
3319 B#(I,I)=R#
3320 NEXT I
```

Sets the objective function equal to machine infinite. Sets the iteration count IT and the error code ER=0.

```
3321 B#(1,1)=BI#
3322 IT=0
3323 ER=0
```

Increments the iteration count by one (1), and if the count has exceeded the maximum count, sets ER=0 and returns to the main routine.

```
3324 IT=IT+1
3325 IF IT>IR THEN RETURN
```

Searches for the greatest slack from among the lower bounds.

```
3326 RO=0
```

```
3327 W#=SM#
3328 FOR I=2 TO ND+1
3329 IF B#(I,1)-M#(I,2)>=SM#/10# THEN 3338
3330 C#=0#
3331 FOR J=2 TO ND+1
3332 C#=C#+B#(I,J)*B#(I,J)
3333 NEXT J
3334 IF W#>=(M#(I,2)-B#(I,1))/CDBL(SQR(C#)) THEN 3338
3335 W#=(M#(I,2)-B#(I,1))/CDBL(SQR(C#))
3336 RO=I
3337 Q#=C#
3338 NEXT I
```

Searches for the greatest slack from among the upper bounds.

```
3339 FOR J=2 TO ND+1
3340 IF M#(J,1)=0# THEN 3352
3341 A#=M#(J,1)
3342 IF A#=BI# THEN A#=0#
3343 IF A#-B#(J,1)>=SM#/10# THEN 3352
3344 C#=0#
3345 FOR K=2 TO ND+1
3346 C#=C#+B#(J,K)*B#(J,K)
3347 NEXT K
3348 IF W#>=(B#(J,1)-M#(J,1))/CDBL(SQR(C#)) THEN 3352
3349 W#=(B#(J,1)-M#(J,1))/CDBL(SQR(C#))
3350 RO=J+ND+1
3351 Q#=C#
3352 NEXT J
```

Searches for the greatest slack from among the CP problem constraints.

```
3353 FOR K=2 TO MD+1
3354 REM-----------------------------------------------------------
3355 IF A#(K,ND+2)<>0# THEN GOSUB 4000:REM PAR-TANL
3356 IF A#(K,ND+3)<>0# THEN GOSUB 4200:REM HYP-TANL
3357 IF A#(K,ND+4)<>0# THEN GOSUB 4400:REM SHT-TANL
3359 REM-----------------------------------------------------------
3360 A#=0#
3361 FOR I=2 TO ND+1
3362 A#=A#+A#(K,I)*B#(I,1)
3363 NEXT I
3364 A#=A#(K,1)-A#
3365 IF A#=0# THEN 3383
3366 S=R(K)
3367 IF S=0 THEN S=-SGN(A#)
3368 A#=A#*S
```

```
3369 IF A#<=SM#/10# THEN 3383
3370 C#=0#
3371 FOR I=2 TO ND+1
3372 B#=0#
3373 FOR J=2 TO ND+1
3374 B#=B#-A#(K,J)*B#(J,I)*S
3375 NEXT J
3376 C#=C#+B#*B#
3377 NEXT I
3378 IF W#>=A#/CDBL(SQR(C#)) THEN 3383
3379 W#=A#/CDBL(SQR(C#))
3380 RO=K+ND+1+ND+1
3381 Q#=C#
3382 SN=S
3383 NEXT K
3384 ER=2
3385 IF W#>1# THEN RETURN
3386 IF RO>0 THEN 3422
```

Makes a cut constraint from the objective function if all the constraints are satisfied by the current center point of the ellipsoid and loads it into the cut constraint array P#(ND+1).

```
3387 A#=0#
3388 FOR I=2 TO ND+1
3389 A#=A#+B#(I,1)*A#(1,I)
3390 NEXT I
3391 FOR I=2 TO ND+1
3392 B#=0#
3393 FOR J=2 TO ND+1
3394 B#=B#+B#(J,1)*Q#(J,I)
3395 NEXT J
3396 P#(I)=B#
3397 NEXT I
3398 FOR I=2 TO ND+1
3399 A#=A#+P#(I)*B#(I,1)
3400 NEXT I
3401 ER=1
3402 IF ABS(B#(1,1)-A#)<=.0000001 THEN RETURN
3403 B#(1,1)=A#
3404 FOR I=2 TO ND+1
3405 P#(I)=-A#(1,I)
3406 NEXT I
3407 FOR I=2 TO ND+1
3408 FOR J=2 TO ND+1
3409 P#(I)=P#(I)-2#*B#(J,1)*Q#(J,I)
3410 NEXT J
3411 NEXT I
```

```
3412 Q#=0#
3413 FOR I=2 TO ND+1
3414 A#=0#
3415 FOR J=2 TO ND+1
3416 A#=A#+P#(J)*B#(J,I)
3417 NEXT J
3418 Q#=Q#+A#*A#
3419 NEXT I
3420 W#=0#
3421 GOTO 3442
```

Loads into the cut constraint array P#(ND+1) the lower bound constraint if it has the greatest slack.

```
3422 IF RO>ND+1 THEN 3429
3423 P#(1)=M#(RO,2)
3424 FOR I=2 TO ND+1
3425 P#(I)=0#
3426 NEXT I
3427 P#(RO)=1#
3428 GOTO 3442
```

Loads into the cut constraint array P#(ND+1) the upper bound constraint if it has the greatest slack.

```
3429 IF RO>ND+1+ND+1 THEN 3438
3430 J=RO-ND-1
3431 P#(1)=0#
3432 IF M#(J,1)<>BI# THEN P#(1)=-M#(J,1)
3433 FOR K=2 TO ND+1
3434 P#(K)=0#
3435 NEXT K
3436 P#(J)=1#
3437 GOTO 3442
```

Loads into the cut constraint array P#(ND+1) the CP problem constraint if it has the greatest slack.

```
3438 K=RO-ND-1-ND-1
3439 FOR J=1 TO ND+1
3440 P#(J)=A#(K,J)*SN
3441 NEXT J
```

Transforms the ellipsoid.

```
3442 GOSUB 3800:REM TRAN-ELP
```

Returns to find a new center point.

```
3443 GOTO 3323
```

Ellipsoid Transformation Subroutine -- File TRAN-ELP

The previous subroutine determined the cut row. The transformation subroutine uses the cut row to reduce the volume of the ellipsoid. For details of the algorithm, see the section on the theory of Goffin's method.

```
3800 REM           * ELLIPSOID TRANSFORMATION SUBROUTINE *
3801 REM-------------------------TRAN-ELP-----------------------------

3802 A#=1#+(((1#+ND*W#)^2)/(24#*ND*ND))
3803 A#=A#*SQR(((ND*ND)*(1#-W#*W#))/((ND*ND)-1#))
3804 B#=(1#-SQR(((ND-1#)*(1#-W#))/((ND+1#)*(1#+W#))))/Q#
3805 G#=((1#+ND*W#)/(ND+1#))/SQR(Q#)
3806 FOR I=2 TO ND+1
3807 B#(1,I)=0#
3808 V#(I)=0#
3809 H#(I)=B#(I,1)
3810 NEXT I
3811 FOR I=2 TO ND+1
3812 IF P#(I)=0# THEN 3816
3813 FOR J=2 TO ND+1
3814 B#(1,J)=B#(1,J)-P#(I)*B#(I,J)
3815 NEXT J
3816 NEXT I
3817 FOR I=2 TO ND+1
3818 IF B#(1,I)=0# THEN 3822
3819 FOR J=2 TO ND+1
3820 V#(J)=V#(J)+B#(1,I)*B#(J,I)
3821 NEXT J
3822 NEXT I
3823 FOR I=2 TO ND+1
3824 B#(I,1)=B#(I,1)-G#*V#(I)
3825 FOR J=2 TO ND+1
3826 B#(I,J)=A#*(B#(I,J)-B#*(V#(I)*B#(1,J)))
3827 NEXT J
3828 NEXT I
3829 RETURN
```

Parabolic Subroutines -- Files PAR-TANA, PAR-TANL, or PAR-TANG

Same as for dual simplex method with nonlinear constraints.

Hyperbolic Subroutines -- Files HYP-TANA, HYP-TANL, or HYP-TANG

Same as for dual simplex method with nonlinear constraints.

Hyperbolic Sheet Subroutines -- Files SHT-TANA, SHT-TANL, or SHT-TANG

Same as for dual simplex method with nonlinear constraints.

Program Table of Contents

Table C-1 can be used to reconstruct the ellipsoidal method with

Table C-1. Ellipsoidal method with nonlinear constraints BASIC program table of contents

nonlinear constraints from the computer disk and to organize subroutines from previous code listings. Since BASIC code is dependent on program line numbers for subroutine branching, the line numbers must be maintained as listed above.

Ellipsoidal Method: Solutions to Example 12 Minimum Project Supervision Cost Problem

The results of the ellipsoidal method are summarized in table C-2. The data are arranged and displayed in the same format as Beale's method with nonlinear constraints solutions to example 12 in table 13.

Because of the iterations and time required to reach a solution with the ellipsoidal method, it becomes evident that the method is impractical for solving the minimum project supervision cost problem. What is worse, the rate of convergence of the method is dependent on the volume of the starting sphere which is in turn dependent on the number of variables in the problem.

Table C-2. Ellipsoidal method, using supporting planes derived with the line search algorithm, solutions to the example 12 minimum project supervision cost problem with hyperbolic man count and parabolic cost functions

Dur.	100	90	80	70	60
Itr.	5439	5352	5295	5308	5538
Sec	16386	17147	16079	15582	16186
Obj.	554.78019	554.78019	554.78019	556.79125	581.59038
T1	16.84991	16.84828	16.84958	15.63109	13.04575
T2	34.59347	34.59216	34.59351	32.57017	27.95653
T3	52.98501	52.98307	52.98486	50.08570	43.30533
T4	74.39797	74.39700	74.39859	69.99999	59.99999
M5	0.18873	0.18873	0.18873	0.19965	0.23091
M6	0.50245	0.50245	0.50244	0.53433	0.62415
M7	1.78042	1.78059	1.78045	1.91925	2.29959
M8	2.25433	2.25429	2.25429	2.36140	2.68262
M9	2.71864	2.71873	2.71866	2.85460	3.25758
M10	2.80204	2.80191	2.80194	3.01291	3.59396
M11	1.93717	1.93719	1.93716	2.03165	2.31331
M12	2.31257	2.31266	2.31257	2.45623	2.86158
M13	1.56391	1.56389	1.56388	1.65535	1.91675
M14	1.34412	1.34414	1.34411	1.42857	1.66666

Table C-2. Continued

Dur.	50	40	30	20	10
Itr.	5496	5475	5680	5745	6730
Sec.	16077	16043	16461	16722	19250
Obj.	654.85701	831.36714	1270.72454	2619.40333	10124.15667
T1	10.65908	8.41577	6.25999	4.15620	2.07484
T2	23.32208	18.67314	14.01120	9.34397	4.67232
T3	36.31350	29.17421	21.93596	14.64327	7.32509
T4	49.99999	39.99999	29.99999	19.99999	9.99999
M5	0.27538	0.34276	0.45587	0.68290	1.36516
M6	0.74968	0.93778	1.25087	1.87687	3.75398
M7	2.81450	3.56473	4.79233	7.21811	14.45890
M8	3.15880	3.89963	5.16047	7.71044	15.39956
M9	3.84869	4.76141	6.30933	9.43521	18.84818
M10	4.38388	5.54232	7.44044	11.20087	22.43072
M11	2.72857	3.37212	4.46543	6.67488	13.33269
M12	3.43022	4.28422	5.70971	8.56166	17.12210
M13	2.28769	2.84952	3.79106	5.68045	11.35624
M14	2.00000	2.50000	3.33333	5.00000	10.00000

APPENDIX D: BARANKIN AND DORFMAN METHOD

Beale's method and the ellipsoidal method provided a means of solving the minimum project cost problem with quadratic cost funtions. Another method for solving the QP problem which is derived directly from the Kuhn-Tucker conditions is the Barankin and Dorfman method[59].

Theory of Barankin and Dorfman Method

The QP problem can be written in matrix notation as:

$$\text{minimize} \quad \bar{c}'\bar{x}+\bar{x}'\bar{Q}\bar{x}$$

$$\text{subject to:} \quad \bar{A}\bar{x}<=\bar{a}$$

$$\bar{x}>=\bar{0}$$

The Lagrangian function can be written for the problem as:

$$\bar{c}'\bar{x}+\bar{x}'\bar{Q}\bar{x}+\bar{u}'*(\bar{A}\bar{x}-\bar{a})$$

from which the Kuhn-Tucker conditions are:

$$\bar{c}+2\bar{Q}\bar{x}+\bar{A}'\bar{u}>=\bar{0}$$
$$\bar{x}'(\bar{c}+2\bar{Q}\bar{x}+\bar{A}'\bar{u})=\bar{0}$$
$$\bar{x}>=\bar{0}$$
$$\bar{A}\bar{x}-\bar{a}<=\bar{0}$$
$$\bar{u}'(\bar{A}\bar{x}-\bar{a})=\bar{0}$$
$$\bar{u}>=\bar{0}$$

where $\bar{u}$ is the vector of Lagrangian multipliers.

Define two (2) more vectors $\bar{v}$ and $\bar{y}$ as:

$$\bar{v}=\bar{c}+2\bar{Q}\bar{x}+\bar{A}'\bar{u} \qquad \bar{y}=\bar{A}\bar{x}-\bar{a}$$

Then, the Kuhn-Tucker conditions can be rewitten as:

$$\bar{A}\bar{x}+\bar{y}=\bar{a}$$
$$2\bar{Q}\bar{x}+\bar{A}'\bar{u}-\bar{v}=-\bar{c}$$
$$\bar{x}'\bar{v}+\bar{y}'\bar{u}=0$$
$$\bar{x},\bar{y},\bar{v},\bar{u}>=\bar{0}$$

In this form, the problem is now linear with the exception of the cross products or the restricted pairs. In this form, the QP problem can also be solved by the restricted pairs method.

Barankin and Dorfman Method: Solutions to Example 17 Minimum Project Cost Problem

The Barankin and Dorfman method requires more variables than Beale's method so the ten (10) activity example 10 would exceeds the capacity of the BASIC program without overlays. To provide an example network for the method, a two (2) activity network shown in figure D-1 will be used.

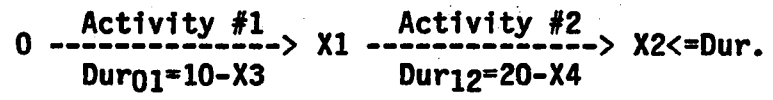


Figure D-1. Network Diagram

The QP problem for the minimum cost problem is then:

```
minimize    -X3-X4+10*X3^2+9*X4^2

subject to:  X1-(10-X3)>=0
             X2-(20-X4)-X1>=0
             X2<=Dur.
```

The Barankin and Dorfman restricted pairs CP problem for the minimum cost problem with quadratic activity costs is:

```
minimize      0

subject to:   -X1-X3+Y5=-10
              -X2-X4+X1+Y6=-20
               X2+Y7=Dur.
              -U8+U9-V11=0
              -U9+U10-V12=0
               20*X3-U8-V13=1
               18*X4-U9-V14=1
X1*V11+X2*V12+X3*V13+X4*V14+Y5*U8+Y6*U9+Y7*U10=0
   X1,..X4,Y5,..Y7,U8,...U10,V11,...V14>=0
```

The restricted pairs solution to the Barankin and Dorfman formulation of the minimum cost problem with quadratic costs functions is listed in table D-1 for six (6) different project durations. The same

Table D-1. Restricted pairs method solutions to example 17 minimum cost problem

Dur.	30	25	20	15	10	5
Itr.	6	15	15	14	12	19
Sec.	7	12	13	12	11	15
Obj.	0.00	0.00	0.00	0.00	0.00	0
X1	9.95	7.63157	5.26315	2.89473	0.52631	0
X2	29.89444	25.00	20.00	15.00	10.00	5
X3	0.05555	2.36842	4.73684	7.10526	9.47368	10
X4	0.05	2.63157	5.26315	7.89473	10.52631	15
Y5	0.00	0.00	0.00	0.00	0.00	0
Y6	0.00	0.00	0.00	0.00	0.00	0
Y7	0.10555	0.00	0.00	0.00	0.00	0
U8	0.00	46.36842	93.73684	141.10526	188.47368	199
U9	0.00	46.36842	93.73684	141.10526	188.47368	269
U10	0.00	46.36842	93.73684	141.10526	188.47368	269
V11	0.00	0.00	0.00	0.00	0.00	70
V12	0.00	0.00	0.00	0.00	0.00	0
V13	0.00	0.00	0.00	0.00	0.00	0
V14	0.00	0.00	0.00	0.00	0.00	0

model solved using Beale's method is listed in table D-2.

Table D-2. Beale's method solutions to the example 17 minimum cost problem

Dur.	30	25	20	15	10	5
Itr.	7	8	8	7	7	6
Sec.	3	4	3	3	3	2
Obj.	-0.05277	113.42105	463.68421	1050.78947	1874.73684	3000
X1	9.95	7.63157	5.26315	2.89473	0.52631	0
X2	29.89444	25	20	15	10	5
X3	0.05000	2.36842	4.73684	7.10526	9.47368	10
X4	0.05555	2.63157	5.26315	7.89473	10.52631	15

APPENDIX E: PROGRAM DIRECTORY

Table E-1. Program table of contents

File	Program lines	Page	Routines
MAIN-	0001-0999		Main routines
MAIN-CPM	0001-0064	20	Critical path method
MAIN-SMP	0001-0132	56	Dual simplex method
MAIN-KLT	0001-0075	100	Out-of-kilter method
MAIN-PDS	0001-0132	127	Primal-dual method
MAIN-BEA	0001-0147	160	Beale's method
MAIN-RST	0001-0153	261	Restart method
MAIN-GOM	0001-0137	288	Gomory's method
MAIN-BND	0001-0156	344	Branch and Bound method
MAIN-PRS	0001-0141	382	Restricted pairs method
MAIN-PDM	0001-0696	450	Precedence diagramming method
MAIN-CRN	0001-0138	479	Corner cut method
MAIN-ELP	0001-0140	504	Ellipsoidal method
INPT-	1100-1799		Input subroutines
INPT-ACT	1100-1133	24	Activity input subroutine
INPT-OBJ	1200-1220	61	Objective coefficient input subroutine
INPT-CON	1300-1328	63	Constraint coefficient input subroutine
INPT-TYP	1400-1440	65	Constraint type input subroutine
INPT-BND	1500-1523	67	Variable bounds input subroutine
INPT-QUD	1600-1628	166	Quadratic input subroutine
INPT-PRS	1700-1723	388	Restricted pairs input subroutine
UTIL-	1800-1999		Utilities
UTIL-OPT	1800-1803	22	Option line subroutine
UTIL-ERS	1850-1853	22	Erase option subroutine
UTIL-CON	1860-1866	22	Continue line subroutine
UTIL-CHX	1870-1882	23	Data check subroutine
REPT-	2000-2999		Reports
REPT-CPM	2000-2019	26	Critical path method report subroutine
REPT-CRV	2100-2125	105	Cost curve report subroutine
REPT-SMP	2200-2224	69	Simplex report subroutine

Table E-1. Continued

File	Program lines	Page	Routines
ALGR-	3000-3549		Algorithms
ALGR-KEY	3000-3057	200	Constraint key subroutine
	3100-3249		Branch and bound subroutines
ALGR-RST	3100-3122	266	Restart algorithm
ALGR-BND	3100-3215	352	Branch and bound algorithm subroutine
ALGR-DBK	3100-3219	367	Driebeek's penalty algorithm subroutine
ALGR-PRS	3100-3270	391	Restricted pairs algorithm subroutine
ALGR-CPR	3250-3274	357	Node compressing subroutine
	3300-3549		Main algorithm subroutines
ALGR-CPM	3300-3377	27	Critical path algorithm subroutine
ALGR-SMP	3300-3398	70	Dual simplex algorithm subroutine
ALGR-KLT	3300-3449	110	Out-of-kilter algorithm subroutine
ALGR-PDS	3300-3433	134	Primal-dual algorithm subroutine
ALGR-BEA	3300-3581	172	Beale's algorithm subroutine
ALGR-GOM	3300-3499	294	Gomory's algorithm subroutine
ALGR-CRN	3300-3434	495	Corner cut algorithm subroutine
ALGR-GOF	3300-3443	508	Goffin's algorithm subroutine
TRAN-	3550-3999		Transformation subroutines
TRAN-OBJ	3550-3584	139	Objective function transformation subr.
TRAN-QUD	3550-3563	180	Quadratic tableau transformation subr.
TRAN-CON	3600-3633	74	Constraint transformation subroutine
TRAN-INV	3700-3743	76	Basis inversion subroutine
TRAN-ELP	3800-3829	513	Ellipsoidal transformation subroutine
TRAN-RIV	3900-3960	78	Basis reinversion subroutine
	4000-4999		Supporting planes and cuts
PAR-TAN	4000-4099		Supporting plane for parabolic subroutines
PAR-TANA	4000-4034	206	Axial algorithm subroutine
PAR-TANL	4000-4034	209	Line search algorithm subroutine
PAR-TANG	4000-4018	211	Gordian algorithm subroutine
PAR-DEP	4100-4199		Deep cut for parabolic subroutines

Table E-1. Continued

File	Program lines	Page	Routines
PAR-DEPA	4100-4189	308	Axial algorithm subroutine
PAR-DEPL	4100-4189	311	Line search algorithm subroutine
PAR-DEPG	4100-4175	313	Gordian algorithm subroutine
HYP-TAN	4200-4299		Supporting plane for hyperbolic subroutines
HYP-TANA	4200-4221	215	Axial algorithm subroutine
HYP-TANL	4200-4221	217	Line search algorithm subroutine
HYP-TANG	4200-4226	219	Gordian algorithm
HYP-DEP	4300-4399		Deep cut for hyperbolic subroutines
HYP-DEPA	4300-4385	316	Axial algorithm subroutine
HYP=DEPL	4300-4385	318	Line search algorithm subroutine
HYP-DEPG	4300-4383	320	Gordian algorithm subroutine
SHT-TAN	4400-4499		Supporting plane for hyperbolic sheet subr.
SHT-TANA	4400-4428	225	Axial algorithm subroutine
SHT-TANL	4400-4425	228	Line search subroutine
SHT-TANG	4400-4430	231	Gordian algorithm
SHT-DEP	4500-4599		Deep cut for hyperbolic sheet subroutines
SHT-DEPA	4500-4593	323	Axial algorithm
SHT-DEPL	4500-4592	325	Line search subroutine
SHT-DEPG	4500-4587	328	Gordian algorithm
CBH-TANG	4600-4628	251	Supporting plane for cubic hyperbolic subr.
CBH-DEPG	4700-4782	331	Deep cut for cubic hyperbolic subroutine
CBS-TANG	4800-4832	254	Supporting plane for cubic sheet subroutine
CBS-DEPG	4900-4987	334	Deep cut for cubic sheet subroutine