A modular data analysis pipeline

for the discovery of novel RNA motifs

by

Justin Schonfeld

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Bioinformatics and Computational Biology

Program of Study Committee:
Daniel Ashlock, Co-major Professor
Dan Voytas, Co-major Professor
Dean Adams
Susan Carpenter
Karin Dorman

Iowa State University

Ames, Iowa

2006

Graduate College
Iowa State University

This is to certify that the doctoral dissertation of

Justin Schonfeld

has met the dissertation requirements of Iowa State University

Signature was redacted for privacy.

Co-major Professor

Signature was redacted for privacy.

Co-major Professor

Signature was redacted for privacy.

For the Major Program

DEDICATION

I would like to dedicate this thesis to Judge and Susan. Thank you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

This dissertation presents a modular software pipeline that searches collections of RNA sequences for novel RNA motifs. In this case the motifs incorporate elements of primary and secondary structure. The motif search pipeline breaks up sets of RNA sequences into shortened segments of RNA primary sequence. The shortened segments are then folded to obtain low energy secondary structures. The distance estimation module of the pipeline then calculates distances between the folded bricks, and then analyzes the resulting distance matrices for patterns.

An initial implementation of the pipeline is applied to synthetic and biological data sets. This implementation introduces a new distance measure for comparing RNA sequences based on structural annotation of the folded sequence as well as a new data analysis technique called non-linear projection. The modular nature of the pipeline is then used to explore the relationships between several different distance measures on random data, synthetic data, and a biological data set consisting of iron response elements. It is shown that the different distance measures capture different relationships between the RNA sequences. The non-linear projection algorithm is used to produce 2-dimensional projections of the distance matrices which are examined via inspection and k-means multiclustering. The pipeline is able to successfully cluster synthetic RNA sequences based only on primary sequence data as well as the iron response elements data set. The dissertation also presents a preliminary analysis of a large biological data set of HIV sequences.

CHAPTER 1. Overview

This dissertation develops a modular software pipeline for the location of patterns in sets of RNA sequence data. The pipeline is implemented and tested on a variety of data sets including both synthetic and real data. The modular nature of the pipeline is explored and exploited by the use of several different distance measures and data analysis techniques on these data sets.

This research is presented in six parts: an introductory overview of the RNA motif search problem, a survey of relevant literature, a description of an initial implementation of the motif search pipeline, an examination of several distance measures, an examination of some of the available data analysis methods, and a concluding chapter which looks at a preliminary application of the pipeline to real data and includes conclusions and future work.

1.1 Introduction to RNA Motif Search

It is becoming increasingly apparent that patterns in both coding and non-coding RNAs are indicators of significant biological function. Finding patterns in RNA is difficult. We need to develop computational methods for two reasons: to deal with large amounts of data and to recognize complex patterns. We need methods to direct biological experiments. Problems with existing methods include: a lack of flexibility, long running times, and lack of easy availability.

RNA motif search is a relatively new area of research; so while there are many approaches to solving the problem there has, as of yet, been little effort to compare them. There is no standard test data set for evaluating search algorithms or any agreed upon measure for success.

This dissertation presents a modular pipeline for exploring sets of RNA sequences for novel RNA motifs. The modular nature of the pipeline allows it to incorporate many of the known approaches for RNA motif detection.

The Problem

In this section we define both the RNA motif finding problem and RNA motifs.

Definition 1.1.1 RNA Motif: *A recurring pattern containing both primary sequence and secondary structure components.*

Instances of an RNA motif can often be produced by folding many distinct primary sequences. An RNA motif can be either very broad or very specific, having many instances or only a few.

Definition 1.1.2 Primary Sequence Pattern: *A sequence of symbols from the alphabet of all possible characters and partial or full wildcards. The primary sequence alphabet and its associated wildcards are listed in Table 1.1.*

Definition 1.1.3 Secondary Structure: *A secondary structure for an RNA sequence is a specification of which bases are paired in the folded sequence. There are three canonical base pairs. Two, the Watson-Crick base pairs, are C-G and A-U. The third canonical base pair, the wobble pair, is G-U.*

Definition 1.1.4 Secondary Structure Pattern: *A sequence of members from the set of secondary structure elements. Secondary structure elements are composed of contiguous paired bases called helices (stems) and single-stranded regions called bulges, hairpin loops, internal loops, or multi-branched loops depending on their context[1, 2]. The final secondary structure element, pseudoknots, are formed by interlocking helices[3]. Descriptions of each element are listed in Table 1.1, and examples are shown in Figure 1.1.*

One of the difficulties of searching for RNA motifs is a lack of clarity in determining which kind of RNA motif is being searched for. One commonly applied adjective is "conserved," which implies a search for motifs which do not change much. Another adjective that is used is "common," implying a search for motifs that occur frequently. The ideal search, however, is not for common or conserved motifs, but for motifs which capture the set of sequences which fold into a structure that performs a specific biological function. Such a motif is difficult to verify. Note that biological activity is not a binary quantity but can vary by degrees with minor changes in structure. Also, as with the ribosomal RNAs, context can determine function as much or more than structure.

Definition 1.1.5 RNA Motif Search:

Given: *A set* **S** *of RNA sequences.*

Find: *A motif* **M** *that appears in* **S** *significantly more often than random expectation in association with a given biological activity.*

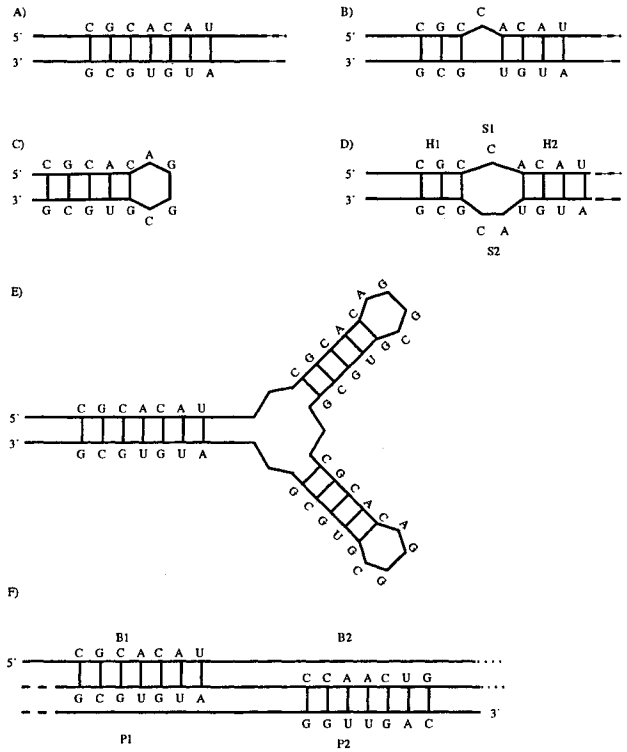

Figure 1.1 Examples of the secondary structure elements: A) Helix (Stem), B) Bulge, C) Hairpin loop, D) Internal loop, E) Multi-branch loop, and F) Pseudoknot

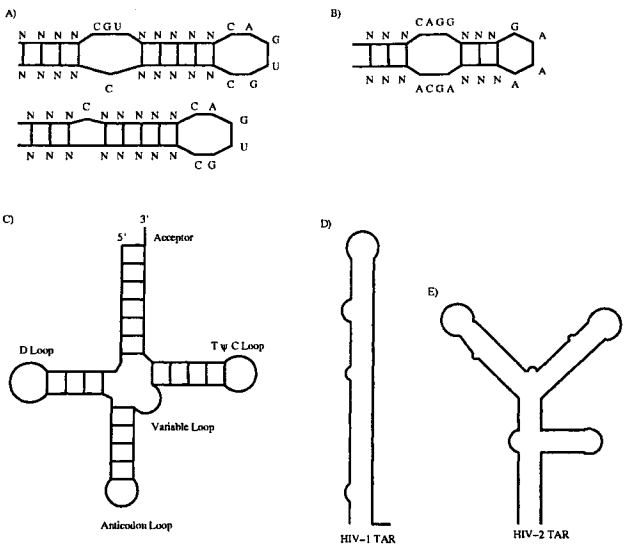

Figure 1.2 Examples of the secondary structure motifs: A) the two forms of the IRE motif, B) the fourth domain of the SRP motif, C) tRNA motif, D) the HIV-1 TAR motif, and E) the HIV-2 TAR motif.

Characters	Symbol	Type
A	A	Character
G	B	Character
C	C	Character
U	D	Character
A,G	E	Partial Wildcard
A,C	F	Partial Wildcard
A,U	G	Partial Wildcard
A,G,C	H	Partial Wildcard
A,G,U	I	Partial Wildcard
A,C,U	J	Partial Wildcard
G,C	K	Partial Wildcard
G,U	L	Partial Wildcard
G,C,U	M	Partial Wildcard
C,U	N	Partial Wildcard
A,G,C,U	O	Full Wildcard

Table 1.1 Primary Sequence Alphabet.

In recent years researchers have found quite a few RNA motifs known to play important biological roles. These include the Iron Response Element (IRE), the SRP (Signal Recognition Particle), and the TAR (Transactivation Response) element. This section gives examples of each of these and the tRNA motif.

Example 1: Iron Response Element

The Iron Response Element (IRE) is an RNA motif which is involved in iron regulatory pathways. IREs bind to the Iron Regulatory Proteins (IRP-1 and IRP-2). The two forms of the IRE proposed in the literature are shown in Figure 1.2 [4, 5].

The IRE is a common target for testing RNA motif discovery software. It is small (28 or 31 nucleotides), well documented, and has two forms. One form has an internal loop, while the other has only a bulge.

Example 2: TAR Element

The Transactivation Response (TAR) element binds to the Tat (Transcription trans-activator) protein in HIV and plays a role in regulating gene expression. The HIV-1 TAR element is a straightforward hairpin-loop with three bulges. The HIV-2 TAR element is more complex, containing several multi-branch loops. Examples of both are shown in Figure 1.2, see also [6].

Example 3: tRNA

Secondary Structure Element	Description
Helix(stem)	Two complementary anti-parallel strands.
Bulge	A single stranded region which interrupts one side of a helix.
Hairpin loop	A single stranded region connecting the two strands of the same helix.
Internal loop	Two non-pairing single stranded regions, S1 and S2, each of which joins one strand of the helix H1 to a strand of the helix H2.
Multi-branch loop	A single stranded region that begins and ends at the same helix with multiple helices interrupting it.
Pseudoknot	Two helices which interlock in the form: .. B1 .. B2 .. P1 .. P2 ...

Table 1.2 Secondary Structure Elements.

Transfer RNAs (tRNAs) are small sequences, 70-90 bases, which bond to amino acids and convey them to growing polypeptide chains. tRNAs make a relatively easy RNA motif search target because they are small, well studied, and highly conserved. See Figure 1.2 for an example of tRNA secondary structure.

Example 4: SRP domain IV

The Signal Recognition Particle (SRP) is an RNA molecule that binds to proteins with the signal-peptide. The SRP helps them travel through the cell to the endoplasmic reticulum or plasma membranes. The SRP contains several secondary structure elements which are separated into domains. An example of the fourth domain can be seen in Figure 1.2.

Related Problems

There are two related problems which show up quite often in the literature and need to be addressed before we continue: predicting RNA secondary structure and finding a known (as opposed to novel) RNA motif in a set of sequences.

RNA Secondary Structure Prediction

RNA secondary structure prediction is the problem of predicting the base pairings that occur in stably folded RNA from the primary sequence. Secondary structure prediction is an integral part of finding RNA motifs since a large component of an RNA motif is secondary structure. It has been shown experimentally that for small ($\leq$100 bases) sequences we can predict base pairing with $\approx$86% accuracy. For larger sequences, that drops to $\approx$55% accuracy [1].

Methods for predicting RNA secondary structure include: energy minimization algorithms, kinetic folding algorithms, evolutionary algorithms, and comparative analysis. For a comparative study of the different methods see [7].

Checking for the occurrence of a known RNA Motif

The second related problem is determining if a known RNA motif is present in a given primary sequence. This is made more difficult by the lack of a formal representation for RNA motifs. An emerging standard is that used by the software package RNAMotif[8]. Several of the methods for finding novel RNA motifs surveyed here make use of either RNAMotif or the motif representation used by RNAMotif. There are several other software packages which take different approaches including ERPIN[9] and RNABOB[10].

CHAPTER 2. Review of Literature

2.1 RNA Motif Search

Framework for Analysis

To better understand the diverse approaches to the RNA motif finding problem, a hierarchical framework is used to classify them. The first classification principle is the type of computer algorithm used in the motif search. Types are evolutionary algorithms (EAs), dynamic programming algorithms, and text indexing algorithms. I further group the EA based approaches first by whether they are searching for a motif, and, if so, then whether or not they are searching for a solely structural motif. The dynamic programming strategies are separated into those that search for sequence compatible structures and those that do not.

The analysis of each paper within the hierarchy is broken down into four parts. The first part of the analysis looks at which variation of the motif search problem the authors are trying to solve. The second part of the analysis describes the algorithm and motif representation they use. The third part gives a brief overview of their data sets, experiments, and evaluation criteria. The final part of the analysis contains comments on the papers and the algorithms, mentioning the strengths and weaknesses of each approach.

2.2 Evolutionary Algorithm Based Methods

Evolutionary algorithm (EA) based methods start with a population of putative solutions and use successive applications of selection, reproduction with variation, and replacement to improve the quality (called fitness) of the solutions. They continue evolving the population until either an adequate solution is found or time runs out.

An evolutionary algorithm works as follows:

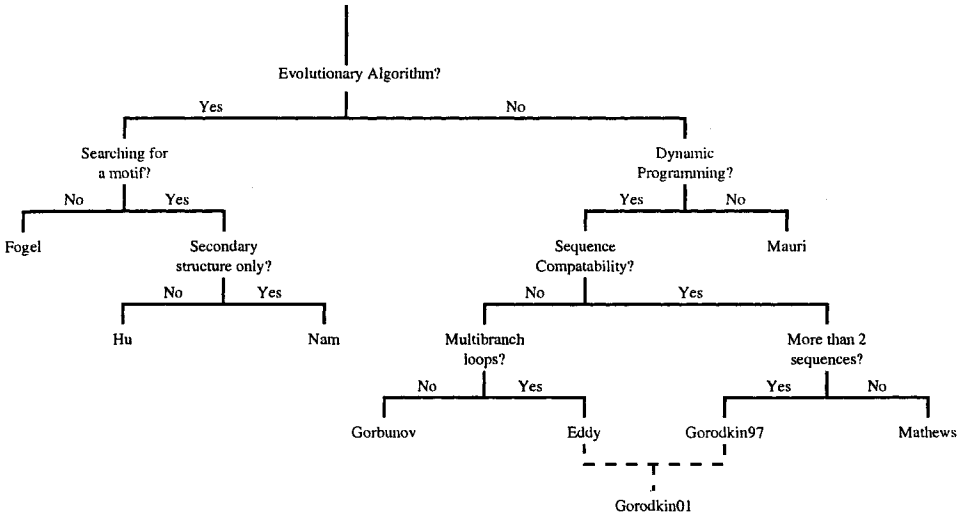

Figure 2.1 A hierarchical breakdown of RNA motif search approaches using differentiating features.

Create an initial population of solutions

Evaluate each solution for fitness

Repeat

Select members with high fitness to serve as parents

Duplicate the parents to make the children

Apply variation operators to the children, generating new solutions

Evaluate children for fitness

Replace members of the population with the children

Until Finished

Variation operators are usually classified as either mutation or crossover. A mutation operator makes small changes to a solution. A crossover operator recombines two solutions to generate two new solutions containing part of each original solution. There are several common methods for selecting parents including ranked, roulette, and tournament selection. Rank selection works by placing the members of the population in an ordered list based on their fitness values, with the least fit being number one. Each member is then assigned a probability of being chosen as a parent equal to their rank divided by the sum of the ranks of the members of the population. In roulette selection the members of the population are assigned probabilities proportional to their fitnesses. This requires

a non-negative fitness function. In tournament selection k members of the population are chosen at random, and the one with the highest fitness is the parent. For crossover this tournament procedure is performed twice, once for each parent.

For a more detailed discussion of evolutionary algorithms, see the text Optimization and Modeling with Evolutionary Computation [11].

Review of "Discovery of RNA structural elements using evolutionary computation" by Fogel et al.[12]

The authors present an evolutionary computation based search algorithm for finding the set of n RNA subsequences which most closely match each other from the set of all RNA subsequences which match a user-defined RNA motif descriptor. The user-defined motif descriptors use the RNAMotif format, see [8] for more details on the format. The algorithm uses tournament selection and four different variation operators that implement a range of changes, varying in magnitude from small to large, to enable search.

Each solution is represented as a set of n RNA subsequences. The subsequences are partitioned into components of two types, helix and loop (here meaning any single stranded region) by the motif descriptor. The fitness of a solution, the closeness of the members of the set, is determined by a weighted sum over two criteria: sequence similarity and sequence length. The weights are determined by the user.

Sequence similarity is calculated between a pair of components using a standard global alignment algorithm with a match scoring 5, a mismatch scoring -4, a gap opening scoring -16, and a gap extension penalty of -4. The resulting score is then multiplied by a weight depending on the component type (stem or loop), summed over all components, and rescaled to the range [0,1]. A total sequence similarity score for all pairs of subsequences in the set is then calculated and divided by the total possible number of pairings.

Sequence length for a pair of components is scored as 1 minus (difference in lengths between the two components/maximum difference in lengths for that component over all subsequences). This score is, as with sequence similarity, multiplied by a weight depending on the component type, summed over all components, and rescaled to the range [0,1]. A total sequence length score for all pairs is then calculated and divided by the total possible number of pairings.

The two total scores are then each multiplied by a weight, added, and then divided by the sum of the weights to give a final score for the set of subsequences. An outline of the algorithm is given in

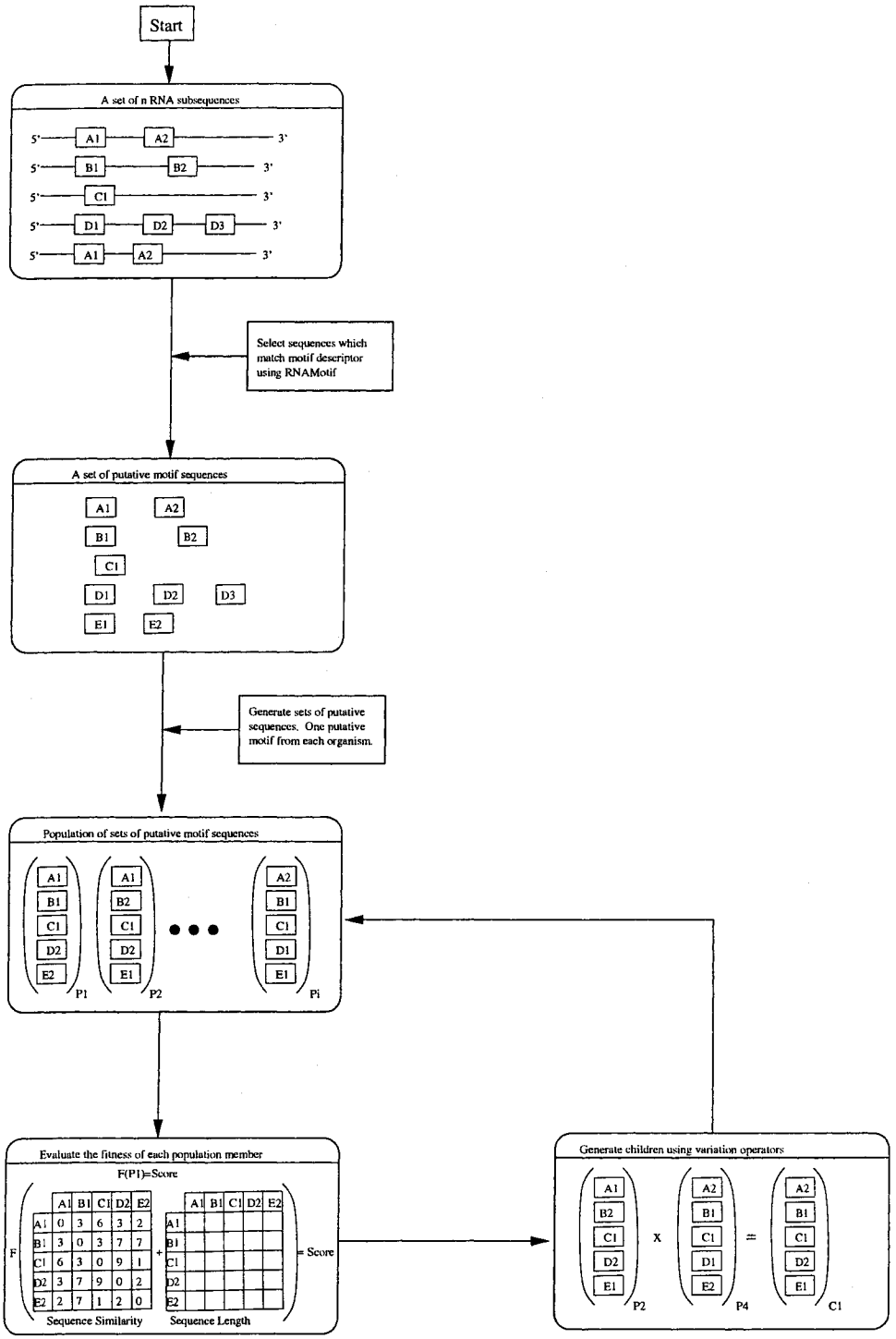

Figure 2.2 An outline of the algorithm presented by Fogel et al.

Figure 2.2.

The authors tested their algorithm on ten different data sets. The first four data sets were constructed from sequences containing an IRE motif. The last six data sets were constructed from sequences containing an SRP motif.

The first data set was based on seven full-length ferritin mRNA sequences and used a motif descriptor which fairly closely selected for the IRE structures predicted in the literature. The second and fourth data sets used the same seven full-length sequences but with increasingly generic motif descriptors. The third data set used an additional five full-length ferritin mRNA sequences.

The fifth through eighth data sets were generated using increasingly generic motif descriptors for the SRP motif on five full-length sequences for 4.5S/7S RNA. The ninth and tenth data sets used the original five full-length ferritin sequences but also the genomes for *S. pyogenes* M1 and *S. aureus* Mu50 respectively, as well as a new type of motif descriptor.

The authors reported that their algorithm was able to find, on each of the first eight data sets, a set of structures where every member (one from each RNA sequence) was consistent with the IRE motif found in the literature. For the ninth and tenth data sets, the data sets containing recently sequenced genomes, the authors reported finding sequences consistent with the IRE motif.

Comments on the paper:

Several implementation details and algorithm parameters are missing in the paper and the on-line supplementary material. This makes duplicating the authors' algorithm and experiments difficult. In addition several of the parameter choices, specifically giving stems a weight of 1.2 and single-stranded sequence components a weight of 1.0, are not adequately explained or consistent with the literature. Stems are known to vary in sequence more than the single-stranded regions without lessening the integrity of the secondary structure, so it seems odd that, in the sequence alignment portion of the "closeness" function, stem sequence fidelity is given more weight than single-stranded sequence fidelity.

It is also unclear what roles the sequence similarity and sequence length scores play respectively. Given the high gap opening and extension penalties, as well as the relatively small size of aligned components (0-10 bases), it is unlikely that internal gaps will show up in alignments. This implies the gap costs will act as a *de facto* sequence length penalty, which makes the role of a separate sequence length evaluation questionable.

Finally, the data sets tested contained, apparently, only two relatively simple RNA motifs. While the algorithm identified them handily, it is difficult to tell from these two samples whether the algorithm will prove equally effective on more complex RNA motifs such as the TAR element in viral RNA[6].

Comments on the algorithm:

The program requires a user-specified motif as a starting point for its search. The authors demonstrated that their algorithm could be used to find unknown instances of a known structure. It couldn't, however, be used for exploratory location of a novel significant motif.

The number of generations needed to find a quality solution is highly variable. In the experiments presented the algorithm finds a set of sequences consistent with those listed in the literature in anywhere from the 13th to the 115th generation. When searching for an unknown instance of a structure, it's unclear how many generation would be needed.

The approach described in this paper has several advantages over other approaches. It is relatively fast, allows for the same structural complexity in motifs as the RNAMotif descriptor language, and does not require lengthy preprocessing steps. With a retuned "closeness" function and more testing on complex RNA motifs, the approach has a lot of potential.

Review of "Prediction of consensus structural motifs in a family of coregulated RNA sequences" by Hu et al.[13, 14]

The authors present an evolutionary algorithm which searches for structural motifs that are present in a positive set of examples but not in a negative set of random sequences with the same base frequencies. The EA uses tournament selection applied to the entire population, with mutation and crossover. In each generation the least fit half of the population is replaced with children.

Motifs are represented as ordered lists of paired(stem) and unpaired(single-stranded) components. A motif matches a primary sequence if the criteria of each component is met in the primary sequence. Each motif begins and ends with an unpaired component and separates unpaired components with paired components. Unpaired, single-stranded, components contain a length range and will match any sequence which fits in that range. Paired components have a length range in addition to an integer representing the component they are paired to. Paired components match uninterrupted stems that fit inside their length ranges.

The mutation of an unpaired component randomly alters the length range. For a paired component both the length range and the component it is paired with are changed. The authors provide no other implementation details for mutation. Crossover exchanges either a single pair of unpaired components or two pairs of paired components. Again, no other implementation details are provided.

The fitness of a solution is evaluated according to the function $f(x, y) = \frac{2XY}{X+Y}$ where X is the number of positive examples containing the motif / the total number of positive examples and Y is the number

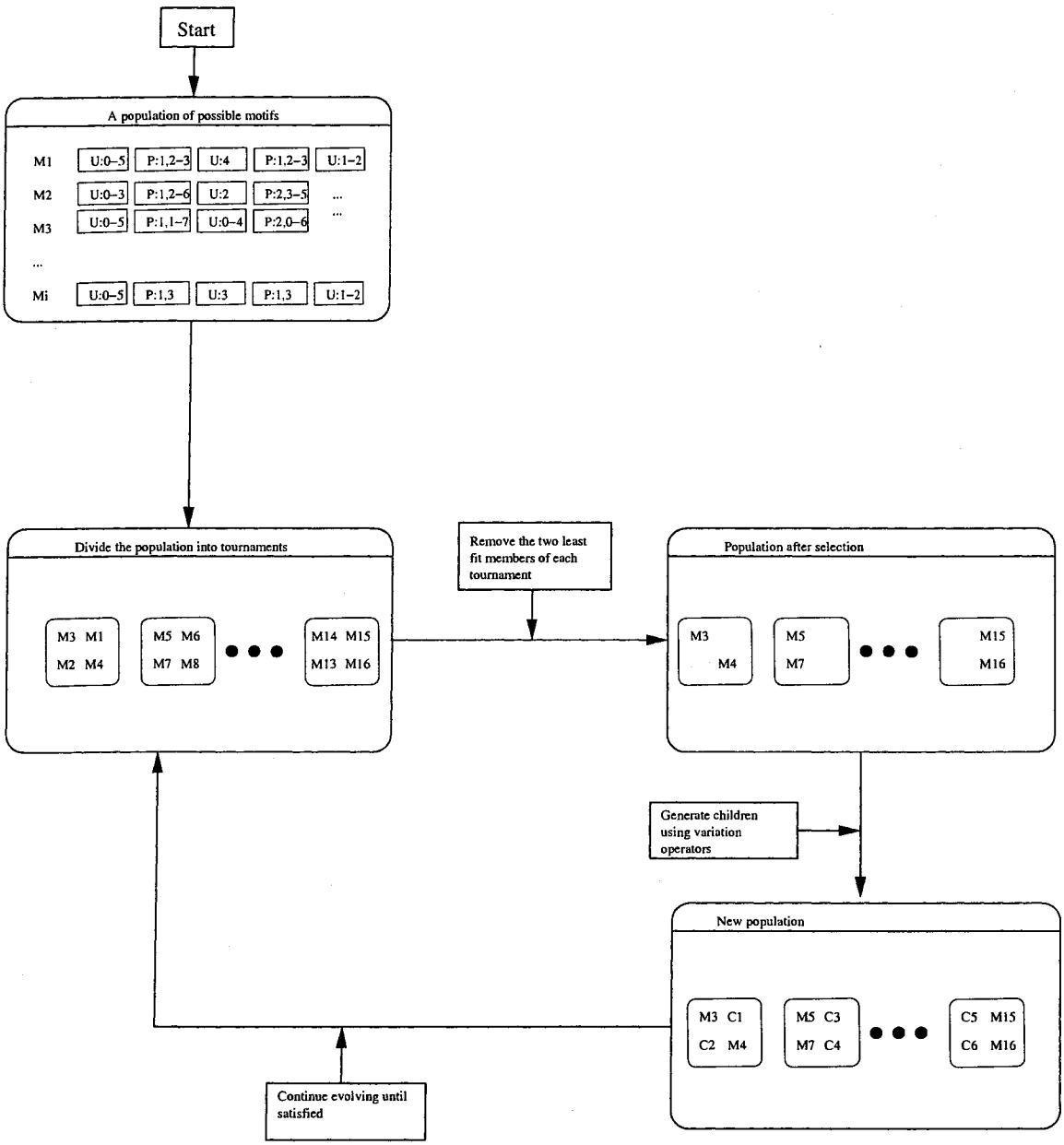

Figure 2.3 An outline of the algorithm presented by Hu et al.

of positive examples containing the motif / the total number of examples containing the motif. The negative test set is created by generating random sequences of equivalent number, length, and base frequency to the positive test set. The authors do not provide details on the algorithm they used to verify if a motif is matched in a sequence, but give the time complexity as $O(L^3)$ where L is the length of the longest sequence in the data set. Figure 2.3 provides an outline of the author's algorithm.

The authors evaluate their algorithm on three different data sets: a ribosomal RNA data set, an IRE-like data set, and a pseudoknot data set. The first two data sets are the same ones as those used by Gorodkin[15]. The IRE-like data set consists of IRE sequences whose stems have been altered to consist of random base pairs. The authors state they use the Matthews correlation coefficient[16] for comparing two RNA secondary structures as their performance metric. The correlation coefficient (Equation 2.1) considers the number of base pairs which are correctly predicted as paired(t_p), correctly not predicted as paired(t_n), incorrectly predicted as paired(f_p), and incorrectly not predicted as paired(f_n).

$$\frac{t_p t_n - f_p f_n}{\sqrt{(t_p + f_p)(t_p + f_n)(t_n + f_p)(t_n + f_n)}} \tag{2.1}$$

It is implied, but not explicitly stated, that they compare the best solution resulting from at most 50 generations of their EA with the structure predicted in the literature for each of the data sets. The paper does not, however, provide these structures.

For each of these data sets the authors performed four different experiments: 1) varying the probability of crossover, 2) varying the probability of mutation, 3) varying the size of the negative data set, and 4) varying the minimum and maximum possible lengths of the paired components. In the first two experiments the authors reported their algorithm found structures with a 0.99 correlation coefficient for the IRE-like data, 0.87 for the ribosomal RNA data, and 0.75 on the pseudoknot data. These values only showed slight variation as the rates of mutation and crossover were changed. As they increased the size of the negative data set, however, they reported the correlation coefficient for the pseudoknot data set increasing up to 0.83. They also reported that changing the min and max lengths for the paired components had little effect on the behavior of the algorithm. The relatively high correlation coefficients suggest the method is useful for finding structural motifs.

Comments on the paper: The authors do not provide the algorithmic details necessary to reproduce their work or a detailed enough analysis of their results to allow a fair assessment of the accuracy. It is not clear, for example, if there was any pattern to the bases that the algorithm failed to predict (or predicted incorrectly) for the pseudoknot data. Without that information it is impossible to tell if algorithm is making errors on irrelevant or structurally critical base pairs.

Comments on the algorithm: The main drawback to this approach is not the algorithm, but the limiting motif representation which does not account for the importance of primary sequence. It would be interesting to use this same general approach, but with an alternate motif representation.

Review of "Two-Step Genetic Programming for Optimization of RNA Common-Structure" by Nam et al.[17]

The authors present an evolutionary algorithm that optimizes RNA motifs for their ability to match a set of positive RNA sequence examples and their inability to match a set of negative RNA sequence examples. The EA contains two evolutionary steps, the first optimizes for structure, and the second optimizes for sequence. Each of the steps runs for a user-specified number of generations and uses rank selection to select the pool of parents. The replacement method is not specified.

The RNA motifs are instances of a context free grammar (CFG) whose words are a subset of the RNAMotif structure definition language. A grammar is a formal specification of a set of rules (called productions) for generating words in a language[18]. The language here is a subset of the RNAMotif structure definition language and each word is a valid RNAMotif descriptor. The grammar contains two productions f1 and f2. The f1 production represents a helix containing either an f1 or f2 production. The f2 production represents a single stranded region. Each helix has associated with it the variables minlen/maxlen, len, mispair, seq, and mismatch. Each single-stranded region has the variables minlen/maxlen, len, seq, and mismatch. In both cases only one of the variables minlen/maxlen and len is specified. The minlen/maxlen variable specifies a minimum length and a maximum length for the associated region. The len variable specifies a specific length for the associated region. Mispair specifies the maximum number of mispairings allowed in a helix. The seq variable is a motif in the primary sequence alphabet which the associated region must match. Mismatch specifies the maximum number of times the sequence can differ from the seq variable. These productions are successively iterated to produce words equivalent to RNAMotif descriptors.

The EA represents each motif (word in the grammar) as a tree structure with f1 productions as internal nodes and f2 productions as terminal nodes. The authors have several additional rules to prevent the formation of trees that contain redundancies or evaluate to words inconsistent with the grammar (invalid motifs). Figure 2.5 presents an example motif in each of its representations: a descriptor, a tree, and several RNA sequences.

The first evolutionary step utilizes both crossover and mutation. Crossover swaps two subtrees and mutation changes the value of one of the variables by a Poisson distributed random variable. Since all

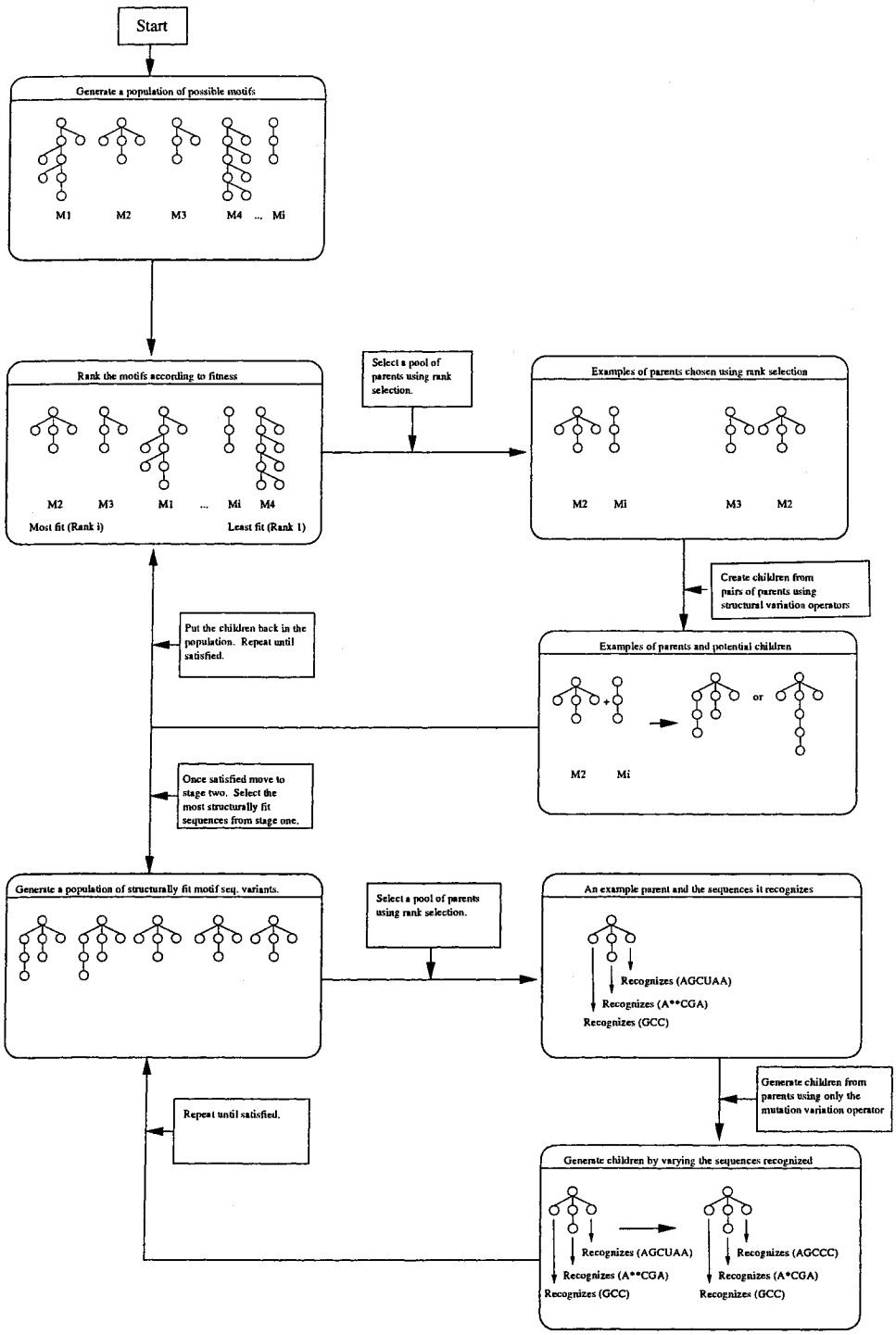

Figure 2.4 An outline of the algorithm presented by Nam et al.

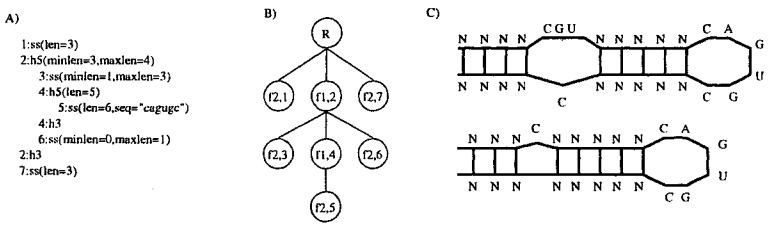

Figure 2.5 An example of a simple IRE motif descriptor of the type used by Nam et al. (A), it's associated tree (B), and two of the RNAs it represents (C).

of the variables whose values are to be changed by mutation are ≥ 0, it is likely the authors meant that mutation changes the value to one picked from a Poisson distribution. In addition a local search (using hill climbing) is performed during each generation. Implementation details are not provided. The second evolutionary step uses only mutation and changes only the sequence and mismatch variables. The mechanism for determining which variables are used at each loop and stem is not stated.

The fitness of a motif is determined by the number of positive sequences it correctly matches and negative sequences it correctly fails to match. Formally the fitness is defined as $\alpha Specificity + \beta Sensitivity - Complexity$. *Sensitivity* is the number of positive sequences matched over the total number of positive sequences, and *specificity* is the number of negative sequences not matched of the total number of negative sequence[19]. Variables α and β sum to 1 and are used to weight the relative importance of specificity and sensitivity. The complexity is a measure of the size of the trees and is inversely proportional to the total number of sequences in the data set squared. The author's algorithm is outlined in Figure 2.4.

For their experiments the authors use tRNA sequences from *Drosophila melanogaster* and eukaryotic 5S small rRNAs for both training and positive data sets. The authors describe the members of the negative data sets as "linear sequences extracted from mRNA and simple secondary structure elements ..." and give no additional details other than to list some of the simple secondary structure elements. The negative training data sets contained 200 of these sequences and the negative test data sets contained 290. The positive training sets contained 50 tRNAs and 50 rRNAs respectively, while the test sets contained 100 of each. In each experiment the evolutionary algorithm is run on the training data set, and then the best motif from the final generation of each of the runs is used on the test data set.

The first experiment was on the 5S small rRNAs; it used only the first evolutionary step, four runs, 100 motifs in the population, 30 generations, an α of 0.95 and a β of 0.05. The authors state that these parameter values produced optimal results. The resulting motif had a sensitivity of 0.83 and a

specificity of 0.86 on the test set. The second experiment was on the tRNAs; it used both evolutionary steps, an unspecified number of runs, 50 motifs in the population, 30 generations, an α of 0.9 and a β of 0.1. The resulting motif had a sensitivity of 0.84 and a specificity of 0.946.

Comments on the paper:

The authors left out both implementation and experimental details making their work difficult to evaluate. In addition much of the experimental data wasn't presented, and a link wasn't provided to a website. The negative data sets were only minimally specified, and any selection criteria used to pick the members of the positive data sets were also left unspecified.

Comments on the algorithm:

The authors describe the algorithm as a method for discovering putative noncodingRNAs, but it also has potential for discovering novel structures. If a region of RNA is believed to be functionally active, but the mechanism is unknown, this approach could be used to identify common structures in the active region that do not occur in the inactive region. The algorithm presented here, however, is missing much needed functionality such as the ability to encode pseudoknots.

2.3 Non-Evolutionary Based Methods

The non-population based methods can be separated into two categories: 1) dynamic programming algorithms, and 2) text indexing algorithms.

Dynamic Programming Algorithms

Dynamic programming is a technique which solves complicated problems by first solving many instances of a smaller problem. One common application of dynamic programming is aligning a pair of DNA sequences. This is solved by constructing a 2D matrix with one sequence on the Y-axis and the other on the X-axis. The algorithm then fills in each matrix entry (i,j) with the score of the best alignment of the two sequences up to that i and j. The smaller problem is to find the best alignment of the ith element of the first sequence and the jth element of the second given that you know the scores for the i-1 to j, j-1 to i, and i-1 to j-1 alignments. This is solved by returning the highest scoring of the three options: align i to j, align i to a gap, align j to a gap. For a more complete (and general) description of dynamic programming see [20]. For a more thorough discussion of dynamic programming as applied to multiple sequence alignment and secondary structure prediction see [21].

One of the most common uses of dynamic programming in RNA motif search is to solve a variation

of the RNA motif finding problem, referred to here as the sequence compatibility problem. In this problem an alignment and a consensus secondary structure are generated simultaneously from a set of RNA sequences and given a score. The score is a measure of the compatibility of the sequences. Sequences that align poorly and have little similar structure score poorly. Sequences with high sequence and structural similarity score well.

Review of "Search for Conserved Secondary Structures of RNA" by Gorbunov et al.[22]

The authors present a strategy for finding conserved secondary structures in a set of RNA sequence fragments, some of which may not contain the conserved structure. This paper presents some algorithmic details for the first four steps in their strategy and a sampling of results from experimental testing.

The strategy is as follows: 1) For each RNA sequence determine the set of likely stems; 2) For each stem in each sequence break the stem apart into its component left and right strands; 3) Locally align all pairs of left and all pairs of right strands using standard dynamic programming sequence alignment algorithms; 4) Score the stems and then rank them; 5) Generate a consensus secondary structure for each sequence from the high ranking stems; 6) Generate a multiple sequence consensus structure from the set of stem consensus structures, and 7) Repeat with modified parameters until satisfied.

For each of the first four steps the authors provide some, but not complete, algorithmic details. Likely stems are selected from the set of all possible stems by calculating the total energy of each stem and then saving the best X percent of the stems. The alignment score for two stems is calculated summing the local alignment scores of the pair of left strands and the pair of right strands. The score is then adjusted by accounting for the stem-loop, internal loops, and flanking regions. The exact method is not specified. A pair of stems which are sufficiently distant are given a large negative score. The final score for a stem is the sum of its pairwise scores with all other helices. Sufficiently low scoring stems are removed from consideration for the consensus structure.

The authors applied an implementation of their algorithm on three data sets: 18 tRNA sequence fragments from *Escherichia coli*, the tRNA sequence fragments including flanking sequences, and 39 sequence fragments containing the RNA secondary structure element[23].

The authors present results for the tRNA sequences showing that in several instances they were able to predict the four stems of the tRNA secondary structure correctly. In most instances the algorithm missed one or more of the stems, often the D stem. For the RFN data the results presented were less promising, ranging from 0 to 82% accurate prediction of the correct stems.

Comments on the paper: The authors provide only an outline of their strategy relying on previous work to explain the details and inconsistencies. Considered by itself the paper is missing key details, such as how to build the consensus secondary structures and many experimental parameters. The results section suffers from the same problem, as only a part of the results are presented.

Comments on the algorithm: It is impossible to fairly evaluate the algorithm without knowing the missing details. The idea of predicting stems and then combining them into a consensus structure has potential, but the algorithms need to be presented more completely.

Review of "RNA sequence analysis using covariance models" by Eddy et al.[24]

The researchers present a data structure, called a RNA covariance model (RCM), that represents both a sequence alignment and a consensus structure for a set of RNA sequences. They present two dynamic programming algorithms: the first for constructing an RCM from a set of sequences and the second for aligning a sequence to an RCM.

An RCM is a stochastic context free grammar that is represented as a tree. The words of the grammar are RNA sequences, each with an associated probability value. For each base pairing position in a stem an RCM stores 16 probabilities, one for each possible pair of bases (AA,AG,AC,AU,...,UU) that could occur at that position. Additionally, since the RCM represents a set of sequences, it also stores the probabilities that the base pair is constant, inserted, deleted, really a left strand bulge, or really a right strand bulge. The tree structure serves as a consensus secondary structure for the set of sequences.

To construct, in the authors' words "train," an RCM the authors start with a random (or externally produced) alignment of a set of RNA sequences. They then use a variation of the Nussinov/Zuker dynamic programming RNA folding algorithm to predict an initial consensus secondary structure for the set of sequences. Then, using the Viterbi approximation to Baum-Welch ([25, 26]) expectation maximization, they iteratively adjust the probabilities for both base and structural occurrences. The adjusted probabilities are then used to create a new alignment which is used to create a new structure. The entire process is repeated until both the structure and probabilities do not change significantly from one iteration to the next. The criteria for determining whether or not a significant change took place were not provided.

The authors also present a dynamic programming algorithm for globally aligning a sequence to an RCM. A multiple sequence alignment is produced by aligning each sequence to the same RCM. A modified version of the RCM alignment algorithm can be used to search through a set of RNA sequences

for the subsequences which best align to an RCM.

The authors tested the alignment, consensus structure prediction, and search abilities of the algorithms and representation on the 1993 compilation of aligned tRNA sequences[27]. They removed redundant and "noncanonical" sequences leaving them with a total of 1415 sequences. The authors removed 100 of the sequences to use as a cross-validation data set. From the remaining sequences 100 were chosen at random to create one training set, and an additional 100 with a high degree of sequence dissimilarity were chosen to create a second training set. The authors created four models: one trained on a trusted alignment of each data set and one trained on an unaligned version of each data set. They called the implementation of their software used for these experiments COVE.

The consensus secondary structure predicted by each of the unaligned models was completely consistent with the known tRNA-Phe secondary structure. All four models were used to create multiple sequence alignments for the test data sets. Results varied from 90-94% agreement with the trusted alignment. Percent agreement was measured as the number of base pairs correctly predicted by the algorithm.

Comments on the paper: The paper leaves out implementation details concerning the RNA covariance models. These details can be found in the authors' text[28]. Additionally the model is only tested on a small subset of secondary structure elements, those found in tRNAs.

Comments on the algorithm: The main weakness of the algorithm, is its dependence on the sequences entered being closely related. The algorithm does not have a mechanism for filtering out unrelated sequences. If a set of unrelated sequences are input, this approach will be unable to find motifs in them. The unrelated sequences poison the training process. It should be noted, however, that there are many instances where you already know that the sequences you are analyzing are closely related and so this is not a serious problem.

Review of "Finding the most significant common sequence and structure motifs in a set of RNA sequences" by Gorodkin et al.[15]

The authors present an algorithm for greedily finding local alignments of multiple RNA sequences where the alignment is of both primary sequence and secondary structure. The algorithm uses dynamic programming to find an alignment which maximizes both the number of sequence matches and the number of shared base pairs. The algorithm combines the Smith-Waterman [29] algorithm for local sequence alignments with a variation of the Nussinov and Jacobson[30] sequence folding algorithm. It uses a 4D matrix with each sequence on two axes. Sequences are compared against themselves in the

calculation of maximal base pairs and against each other when calculating the best alignment. The recursion compares the ith and jth bases of sequence A with the kth and lth bases of sequence B to determine which of fifteen ways to place up to three gaps will result in the best score. The recursion iteratively fills out the matrix starting from the center diagonal and proceeding outwards. The scoring matrix is a 25x25 matrix that gives scores for all combinations of bases and gaps.

To simplify the algorithm the authors do not consider either branching structures or pseudoknots. This allows the scoring algorithm to run in $O(L^4)$ where L is the length of the longest sequence being compared. To build a multiple alignment of a set S of RNA sequences the authors use a greedy iterative comparison process. As an initial step all pairs of sequences are aligned to produce a set of 2-tuples. Then each 2-tuple is aligned against all of the remaining sequences not already part of it to yield 3-tuples. This process is repeated until only $|S|$-tuples containing all the sequences remain. To keep the running time and space requirements of the algorithm reasonable, only a small fraction (the authors typically use 30) of the best tuples are saved each iteration. The final time complexity of the algorithm is $O(L^4|S|^4)$ where $|S|$ is the number of sequences being aligned.

The authors analyzed four different data sets using their algorithm and three other publicly available programs: CLUSTALW[31, 32], COVE[24], and tools from the Vienna RNA package[33]. The first data set contains hairpin loop structures. The second data set contains pseudoknots. The third data set contains conserved sequences. The fourth contains multi-branch loops. CLUSTALW produced multiple sequence alignments; COVE was used to perform global alignments and search for a consensus structure, and the Vienna RNA package tools were used to predict RNA secondary structures and calculate edit distances between them. The authors summarized the success of using each approach to analyze the data.

When evaluating their algorithm the authors compared the structural alignments predicted by FOLDALIGN, an implementation of their algorithm, to the consensus structures predicted in the literature. For each of four data sets the authors reported that their software found the consensus structure or at least one consistent with the most common consensus structure in instances where there were more than one.

Comments on the paper: For the most part the paper is concise and well written. The description of the method for constructing N-tuples as well as the strategy the authors use to find which of the N-tuples represents the strongest signal could have been clearer.

Comments on the algorithm: The algorithm, while potentially better at finding alignments (both sequence and structural) than contemporary publicly available software packages, has several severe

limitations. First, due to its simplified nature, it cannot find either branching structures (multi-loops) or pseudoknots. Second, even though it is highly simplified, the running time is still excessive for use on large (80+ sequences with 300+ bases each) datasets.

Review of "Discovering common stem-loop motifs in unaligned RNA sequences" by Gorodkin et al.[34]

The authors present a strategy for combining the FOLDALIGN and COVE software packages to find RNA multiple sequence alignments and motifs. The authors use FOLDALIGN to generate initial sequence alignments which they use as starting points for the COVE software. This strategy is then evaluated for both of the tasks: multiple sequence alignment and secondary structure motif location. The multiple sequence alignment task was evaluated on a set of 34 archaea 16S ribosomal sequences. The secondary structure motif location task was tested on a set of artificially constructed IRE-like sequences.

The authors evaluated the sequence alignment task by looking at an information score reported by COVE. The structural predictions were evaluated using the Mathews correlation coefficient (See equation 2.1).

The authors reported that alignments made by COVE with FOLDALIGN were significantly better than those made by COVE alone. They also demonstrated that FOLDALIGN was able to locate the IRE motifs in the IRE-like data set. Then using the alignment produced by FOLDALIGN they were able to generate COVE models which could scan a larger set of sequences and successfully locate the IRE motifs in them.

Comments on the paper: The information score produced by COVE is an indirect measure. It's unclear what it actually means.

Comments on the algorithm: Combining the two algorithms helps in reducing the speed issues associated with FOLDALIGN and the global alignment nature of COVE. The combined model, however, is still unable to detect pseudoknots and has trouble with branching structures as FOLDALIGN cannot detect them initially.

Review of "Dynalign: An Algorithm for Finding the Secondary Structure Common to Two RNA Sequences" by Mathews et al.[35]

The Dynalign algorithm takes as input two unaligned RNA sequences and produces a single common structure which minimizes the free energy of both sequences with a gapped alignment.

The algorithm presented is a four dimensional dynamic programming algorithm that runs in $O(M^3N^3)$ time. M is the maximum insertion size, and N is the length of the shorter sequence. The maximum insertion size is a guess at the maximum distance between aligned nucleotides. While the algorithm can be extended to align several sequences, doing so increases run time to $O(M^{(2|S|)}N^3)$ where $|S|$ is the number of sequences. The algorithm is a simplified version of the algorithm proposed by Sankoff[21] for simultaneously solving the sequence alignment and sequence folding problems. Unlike the other simplified version of Sankoff's algorithm discussed in this review (Gorodkin et al. [15]) this one allows for branching structures.

The authors tested their algorithm on three different data sets: 13 tRNAs, 7 5S rRNAs, and 2 R2 3' Untranslated Regions (UTRs). For each data set the authors ran their algorithm on all pairs of sequences. They measured success as the percent of base pairs predicted in agreement with the structure obtained by comparative analysis. For the tRNA they achieved an average of 86.1% accuracy, and for the rRNAs they achieved an average of 86.4%. For the 2 R2 3' UTRs the prediction missed only two base pairs.

Comments on the paper: The paper is clear and well written.

Comments on the algorithm: The algorithm presented here has several drawbacks. It doesn't consider sequence similarity, except as it is implied by the rules for energy calculation. The algorithm is unable to align, in a reasonable time period, more than two sequences at a time. Finally, the algorithm does not consider pseudoknots. The algorithm can be extended to consider multiple sequences and search for pseudoknots, but doing so would make the running time so large even comparisons of a few small sequences would be impractical.

Text Indexing Methods

These are methods which treat RNA sequences as text strings in order to take advantage of text indexing structures to find patterns in the sequences.

Review of "Pattern Discovery in RNA Secondary Structure Using Affix Trees" by Mauri et al.[2]

The authors present two variations of an algorithm for finding conserved RNA motifs in a set of unaligned sequences by converting the sequences into text strings and using a text indexing structure, affix trees, to quickly search for patterns. The first variation takes a set of sequences and their predicted

structures as input and reports all motifs that occur at least n times. The second variation removes the requirement for the predicted structures.

The authors limit their search to non-pseudoknot secondary structures in order to preserve symmetry in base pairings which is exploited by the affix trees. In the first variation each secondary structure annotated sequence is converted into a text string as follows: 1) Each set of paired bases is replaced with an opening and closing parenthesis. The opening parenthesis replaces the base which occurs earlier in the sequence. 2) Each single stranded region (bulges, hairpin loops, internal loops, and external loops) is replaced with a symbol indicating its size and type.

An affix tree is a data structure which combines both a prefix and a suffix tree [36]. It is, in a sense, an efficient database of all prefixes and suffixes of the strings entered into it. This means it is possible to expand both forward and backward from a symbol in an affix tree accessing both the preceding and following symbols. The text strings generated above are combined in linear time into an affix tree.

The distance between two secondary structures is measured by three variables: the bulge number difference, the hairpin loop length difference, and the bulge loop length distance. The bulge number difference is the difference in the number of bulges and internal loops that occurs between two stems. The hairpin loop length difference is the sum of the differences between corresponding hairpin loops. The bulge loop length difference is the sum of the differences in length between corresponding bulges and internal loops. Each of these variables is governed by a user-defined maximum. When the maximum is exceeded for any one of the variables the hairpin stems are considered different and not to correspond as structural elements.

For each unique hairpin stem, which occurs at least once, the affix tree is searched in both directions to find the largest motif which occurs in at least n sequences. The algorithm runs in $O(|S|T)$time where $|S|$ is the number of sequences and T is the total length of the sequences. This variation of the algorithm also includes loop sequence comparison as a post-processing step.

The second variation of the algorithm uses the primary sequences as the text strings to place in the affix tree. The algorithm expands outward from all sequences over a given size which cannot base pair. It keeps track of possible sequence pairs and forms the stems as it goes.

The authors performed three experiments using an IRE data set and a data set of a highly conserved stem-loop found in Metazoan histone 3'UTR mRNAs. The IRE data set contained 20 5'UTR ferritin sequences (100 - 700 bases in length) and structure annotations done using MFold[37]. The histone data set consisted of 20 sequences each containing the simple six base-pair helix four base loop secondary structure.

In the first experiment the authors ran the first variation of their algorithm on the IRE data set and reported finding 19 of 20 instances of the secondary structure.

For the second experiment the authors ran the second variation of the algorithm on the histone data set. They stated that with an additional energy minimization criteria added they were able to find the correct structure in 19 out of 20 sequences with no false positives. The presence or absence of false positives in the first experiment was not reported.

In the final experiment the authors tested the second variation of the algorithm on the IRE data set. This time all 20 structures were found, without false positives or non-IRE motifs being found.

Comments on the paper: The authors only tested their algorithm on very simple secondary structure elements. It would have been interesting to see a more thorough test as the technique seems promising.

Comments on the algorithm: The algorithm presented here has a great deal of potential, but there are two weaknesses to this approach. It depends on symmetry in base pairing, which renders it unable to work with pseudoknots, and the current implementation only deals with stems. It is, however, quite fast, much more so than the algorithms relying on dynamic programming to do structural prediction.

Review of "Algorithms for pattern matching and discovery in RNA secondary structure" by Mauri et al.[38]

This paper presents a fleshed-out version of the second variation of the algorithm in the preceding paper. Minor changes are made to the methodology for distinguishing stems and the energy criterion for evaluating which stems to report as motifs.

The authors repeat the IRE experiment of the previous paper and add a new experiment using a secondary structure motif from the Signal Recognition Particle (SRP) RNA domain IV[39]. The authors report the algorithm is successful in identifying the correct domain for the 116 sequence data set.

Comments on the paper: The title of this paper is slightly misleading. While the paper briefly mentions other approaches to the RNA motif matching and discovery problem, they are not described in detail.

Comments on the algorithm: While minor improvements have been made, the main weaknesses of the algorithm remain unchanged. The authors discuss the possibility of post processing steps that look for pseudoknots and complex stems but do not present any details.

Paper	Alg.	Motif Type	Align	Motif	Run Time
Fogel	EA	RNAMotif	No	No	NA
Nam	EA	RNAMotif	No	Yes	NA
Hu	EA	SS only	No	Yes	NA
Mathews	DP	SS only	Yes	Yes	$O(M^{(2\|S\|)}N^3)$
Gorodkin97	DP	RNAMotif	Yes	Yes	$O(L^4\|S\|^4)$
Eddy	DP	RNAMotif	Yes	Yes	NA
Gorodkin01	DP	RNAMotif	Yes	Yes	$O(L^4\|S\|^4)$
Gorbunov	DP	SS only	No	Yes	?
Mauri	TI	RNAMotif	No	Yes	$O(\|S\|T)$

Table 2.4.1 *Overview of the different methods. "Alg." refers to the type of algorithm used. "Motif Type" refers to the type of RNA motif the algorithm searches for. "Align" refers to whether or not the algorithm produces as multiple sequence alignment. "Motif" refers to whether or not the approach predicts a motif. "Run Time" lists the run time of the algorithm when appropriate.*

2.4 Conclusions

Each of the approaches presented here has both strengths and weaknesses. The dynamic programming methods find algorithmically optimal solutions but have long run times. The EA methods are either poorly suited to novel motif search or suffer from flawed motif representations. Many of the papers surveyed present interesting ideas and search strategies but fail to describe them clearly. Some of the properties of each approach are summarized in Table 2.4.1.

One of the main challenges presented by the approaches discussed here is their lack of consistency with each other. With only a few exceptions each approach uses a unique data set to test its search algorithm and a unique criteria for evaluating success. These are both problems which will be solved as research into this field progresses.

The algorithmic challenges presented are more problematic. RNA secondary structure prediction remains a difficult task, and it is not clear that it will (or can) be solved soon. Even though several of the EA based methods and the text indexing method presented by Mauri et al. demonstrate an ability to find simple motifs while side-stepping secondary structure prediction, the ability of such approaches to find more complex motifs is not clear.

The ability to search for motifs containing pseudoknots also remains largely unaddressed. Pseudoknots are believed to play an important role in determining the biological activity of some RNAs (viral among others). Therefore more RNA motif search strategies which look for motifs containing pseudoknots need to be developed.

Finally, the inability to compare two motifs is a serious problem. Several of the approaches mentioned above, such as Mauri et al., present techniques for comparing stems and stem-loops. None of the approaches, however, develop and test a strategy for comparing more complex motifs containing multi-branch loops and pseudoknots. In order to create an effective automated novel motif search tool, a method for distinguishing between classes of motifs needs to be developed.

CHAPTER 3. Motif Search Pipeline

3.1 Introduction

This chapter presents a novel RNA motif search technique in the form of a modular data analysis pipeline. This pipeline takes RNA primary sequences as input and produces clusterings and non-linear projections of the structurally annotated sequences as output. The modular nature of the pipeline lends it a great deal of flexibility, which is necessary when searching for motifs in RNA. This pipeline was first presented in [40].

3.2 Outline of the Motif Search Pipeline

The RNA motif search pipeline has two main components: a framework which outlines how to search for common features and a set of interchangeable modules for each step of the framework. This division is not only convenient from an implementation perspective but permits the smooth substitution of different techniques for existing ones to permit comparison, improvement, and adaptation of the pipeline to specific tasks.

Suppose a set G of RNAs are thought to have a common biological activity. The framework outlines how to search G for common features. The stages of the framework, shown in Figure 3.1, are as follows.

- **M1** Each RNA sequence in G is fragmented into overlapping segments called *bricks*. The exact fragmentation is specified by the length of each fragment and the length of the overlap between segments. Any terminal partial bricks are discarded, so the members of G should contain what is thought to be the active region of the sequences.

- **M2** Each brick is folded to obtain one or more secondary structures. The number of structures per brick controls the broadness of the motif search. Using more folds has a substantial downstream computational cost but compensates for the difficulty of predicting the actual biological fold of a given brick.

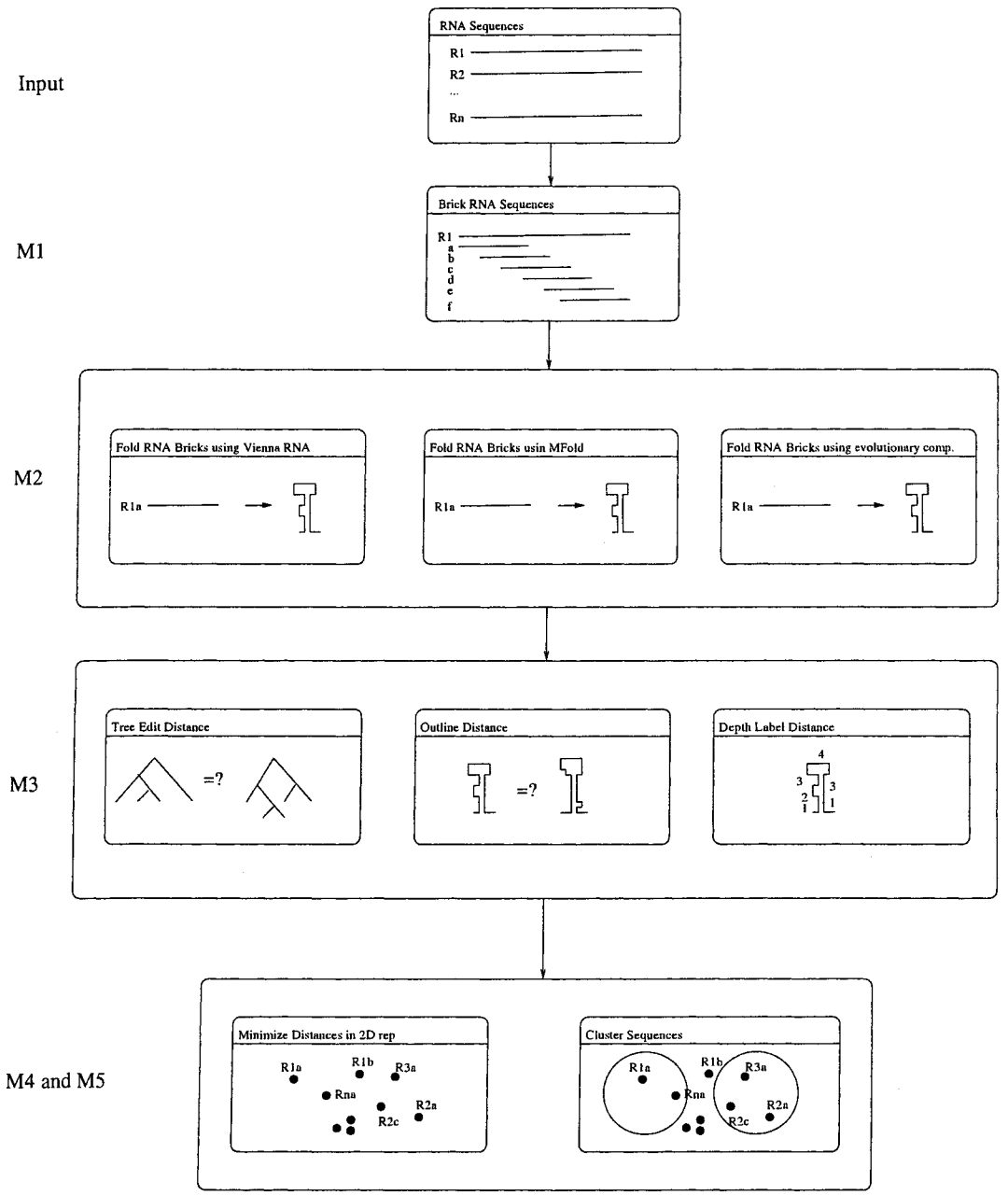

Figure 3.1 An outline of the motif discovery framework. Within a module, e.g. M2, alternative methods can be easily substituted, e.g. ViennaRNA, MFold, or evolutionary computation.

- **M3** A distance measure is applied to compute the pairwise distance between all pairs of bricks.
- **M4** A set H of points in Euclidean space is selected so that for each sequence $s \in G$ there is a corresponding point $P(s) \in H$. This set of points can be selected in several different ways. In this thesis we look at two methods: using Principle Coordinates Analysis (PCoA) ([41]) to find corresponding points with the same dimensionality as that of the distance matrix and using randomly initialized points with an EA algorithm.
- **M5** The points in H are examined to find clusters. This may be done on the high dimensional points with a clustering algorithm, or by first reducing the dimensionality of the points and then simply using the placement of points corresponding to sequences in the plane as a visualization. Clusters correspond to collections of similar RNA structures. Clusters possess simple descriptions, e.g. the sequences whose corresponding points are close to one given point in Euclidean space. These clusters are thus RNA motifs.

3.3 An Initial Implementation of the Pipeline

Data Sets

The pipeline was run on five data sets, four synthetic data sets, and an Iron Response Element (IRE) data set. The synthetic data was generated from sequence templates. Each sequence template was an ordered list of paired and unpaired sequences. For each of the paired sequences only one half of the pair was specified; the other half was supplied according to the Watson-Crick base pairing rules. All of the synthetic sequences were generated from three basic templates: A, D, and E (See Table 3.1). Template A encoded a simple hairpin loop (see Figure 3.2). Template D contained a single internal loop in addition to a hairpin loop. Template E contained two sequential hairpin loops. Variants were generated from each template by applying operations from the following set: increase stem length by one base, increase loop length by one base, decrease stem length by one base, decrease loop length by one base, mutate a complementary pair of bases in a stem, mutate a single base in a loop. The initial templates for each of the three core sequences are given in Table 3.1.

The first synthetic data set (SD-DS1) consisted of 100 sequence variants of template A in addition to the original template A sequence. The first 10 sequence variants were generated from one application of the variation operators. The next ten were generated with two applications, and so on up to ten applications of the variation operators. The second synthetic data set (SD-DS2) consisted of 50 variation

Template ID	Template	Element
A	L0S0L1S0L2	L0 AGCGCAACUACGAAA L1 GCACG L2 CCUAGACAUAAGUUUCGUAC S0 GGCGGAUCAG
D	L0S0L1S1L2S1L3S0L4	L0 UGGCC L1 GUACCG L2 CGGUCG L3 CGUGCAA L4 CGAGCGUAGUCUACGU S0 CGAC S1 UCUCG
E	L0S0L1S0L2S1L3S1L4	L0 GCUACC L1 CGUAGC L2 AUGCAUC L3 GUACG L4 GUCAUCGAUCGAUCCG S0 UCUCG S1 ACGCC

Table 3.1 The nucleotide sequence and pairing templates used in creating the synthetic RNA sequences. Template A encodes a single hairpin loop (S0). Template D encodes a hairpin loop with an internal loop (L1 and L3). Template E encodes two hairpins, one (S1) after the other (S0).

number 3 sequence variants of template A and 50 variation number 3 sequence variants of template D. The third and fourth synthetic data sets (SD-DS3 and SD-DS4) consisted of 50 variation number 3 variants each from A and E and 50 variation number 3 variants each from D and E respectively. The synthetic data sets are summarized in Table 3.2.

The IRE data set (ID-DS1) was composed of 10 ferritin mRNA's each containing at least one instance of the IRE motif with accession numbers: gi—507251, gi—286151, gi—191071, gi—213691, gi—214135, gi—16416388, gi—12802902, gi—15076950, gi—11545422, gi—6753911. The presence of the IRE motif in the first 100 bases of each sequence was verified using RNAMotif([8]).

Bricks

Each of the sequences generated for the synthetic data set was between 50 and 70 bases in length. Each sequence was truncated to length 50 by removing excess flanking sequence and represented as a

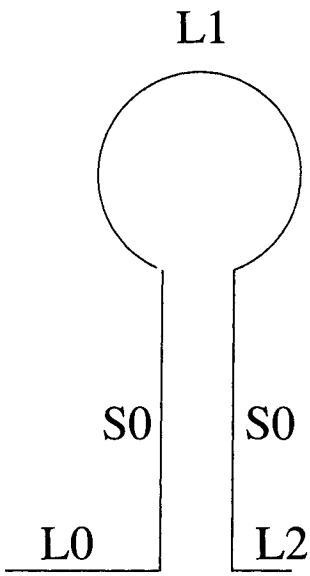

Figure 3.2 The secondary structure for template A. The prefix L indicates a loop element and the prefix S indicates a stem element.

ID	Templates	No. Seqs (No. Var. Operators)
SD-DS1	A	1(0), 10(1), 10(2), ..., 10(10)
SD-DS2	A, D	50(3), 50(3)
SD-DS3	A, E	50(3), 50(3)
SD-DS4	D, E	50(3), 50(3)

Table 3.2 The first four synthetic data sets. The first column contains the data set identifiers. The second column contains the templates used to construct each data set. The third column contains the number of variants created by applying the variation operators v times, where v is the number given in the parenthesis.

single brick.

The first 100 bases of each IRE sequence were converted into six bricks each containing 50 bases and spaced at increments of 10 bases. Of the 60 bricks, 21 were found to contain an instance of the IRE motif using RNAMotif. The two main factors to consider are brick size and increment size. A large brick size provides more contextual information for the folding algorithm but carries with it a cost in increased noise, increased computational run time, and poorer folding accuracy. If the brick size is too small then the pipeline will be unable to detect larger motifs and the foldings may be incorrect due to a lack of context. In this implementation a size of 50 was chosen for several reasons: it contains both the synthetic and IRE motifs, it's small enough to fold accurately, and its small enough to fold and calculate distances on in a matter of minutes.

A large increment size, greater than or equal to the brick size, risks losing data or clipping a motif off. If the increment is too small, then the run time increases dramatically, and the distance matrices become cluttered with overly similar sequences. The increment size of 10 was chosen to provide a tractable number of bricks.

RNA Folding

The bricks were folded using MFold version 3.2 [37]. The bricks based on the synthetic data sets were folded using constraint files. A constraint file was generated for each brick specifying which bases paired and which bases didn't pair. In essence, this forced the needed structure, for testing. For the bricks containing IREs MFold failed to produce folds consistent with the folds given in the literature. For each of the bricks containing an IRE one to two additional folds was calculated. If the brick contained a primary sequence segment consistent with both forms of the IRE then two additional folds were calculated, otherwise only a single additional fold was created. The additional folds were created using constraint files so that they matched the folds predicted in the literature.

Depth Annotation Distance

Depth annotation labels bases with their "depth" in the fold structure. This depth is computed by starting at the beginning of the primary sequence and assigning the first base a depth of zero. The depth annotation then traverses the bases of the primary sequence and secondary structure in the order given by the primary sequence. The depth is increased each time a stem is entered or left for the first time and decreased each time a stem is entered or left for the second time. Even depths denote loops; odd depths denote stems. Bulges and their corresponding gaps are treated as loops, able to begin or

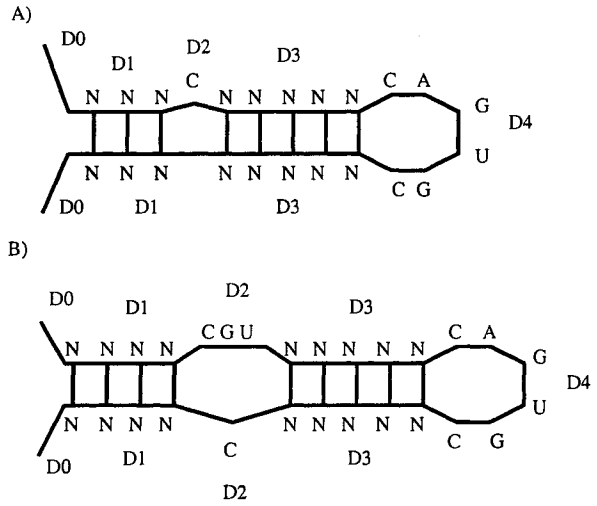

Figure 3.3 The depth annotated IRE structural motifs.

end stems. Stems are also annotated with a unique identifier to allow the annotation to distinguish between pseudoknots and nested stems. Two examples of depth annotation are shown in Figure 3.3.

Once the primary sequence has been depth-annotated, then distances are computed as weighted edit distances with dynamic programming([42]). The costs for the edit operations are as follows. The cost for changing a stem base into a loop base or vice versa is 8. Matching bases, both within a stem or both within a loop, have a cost of 0. Mismatching bases within a stem have a cost of 1; within a loop the mismatch cost is 3. Insertion or deletion of any character has a cost of 5. The cost matrix, except for the depth difference penalty, is given in Table 3.3. If a base was transformed into another that had a different depth, then five times the positive difference in the depths is added to the cost of that edit. The distance between two annotated primary sequences of length W is the minimum total cost, over all possible sequences of the edits described, that can turn one into the other. Aside from the weights, this is the standard notion of edit distance, the sum of the minimum (cost) edits that turn one string into the other.

A collection of parameters such as those above require justification. The matching of identical bases costing zero is clearly the right choice. Stems must base pair C-G and A-U, but, given that they pair, may be any base so the mismatch cost should be lower for bases in stems than those in loops, hence $3 > 1$ are chosen for those costs. Stems and loops should line up, hence the large cost, 8, of matching a stem base with a loop base; the cost is even higher because a stem and loop must have different depths. Differing depths reflect substantial disagreement in structure even when stem matches stem or loop matches loop, hence the premium for having distinct depths. The relative cost of a gap is not clear to

BT	CL	GL	AL	UL	CS	GS	AS	US
CL	0	3	3	3	8	8	8	8
GL	3	0	3	3	8	8	8	8
AL	3	3	0	3	8	8	8	8
UL	3	3	3	0	8	8	8	8
CS	8	8	8	8	0	1	1	1
GS	8	8	8	8	1	0	1	1
AS	8	8	8	8	1	1	0	1
US	8	8	8	8	1	1	1	0

Table 3.3 Cost matrix used in the calculation of the distance between two depth annotated sequences. The labels are in the form BT where B is the base and T is the type of sequence (loop or stem).

the authors and so was set to equal the "difference in depth" penalty.

For two bricks, all length W annotated substrings were compared and the smallest distance between any two of them was defined to be the distance between the bricks. The motivation for this definition of distance between bricks is that the distance should reflect the best match in the secondary structures of the target length. When the framework was conceived basing the distance between bricks on the best local alignment was contemplated. This is not, however, a well defined quantity; local alignments of different lengths would require substantial fudging to even have units that could be compared reasonably. The problem can be finessed by comparing all alignments of a fixed length, which then requires that some fixed length be chosen. If that length is too short, then perfect matches at that length become common, and the distance information becomes largely trivial. If the length is too long (longer than the secondary structures that make up the motif the system is intended to locate) then a comparison of (motif member+noise) to (motif member+noise) is made, adding noise to the signal. This suggests that a window length W that is a substantial fraction of the length of the motif, half or more, would be sensible. Since the system is intended to explore for *unknown* motifs, several window sizes are required. For the experiments presented in this chapter a window length of 27 was used.

Analysis Techniques

The goal of the analysis portion of the pipeline is to identify motifs through clustering, or present RNA sequences in such a manner that identification of the clusters is possible via observation. In this implementation a non-linear projection algorithm is used to generate 2D plots of the RNA bricks. The

plots are examined for clusters by hand. The algorithm minimizes the error between the n-dimensional space of the distance matrix and the 2D representation of the plot. The non-linear projection algorithm and other analysis techniques are described and explored in Chapter 5.

3.4 Experiments

The results of the pipeline are evaluated in two ways: first by checking that the depth annotation distance measure is a reasonable distance measure for RNAs and second by examining 2D plots produced by the non-linear projection algorithm.

The degree to which the distance measure captures differences in RNA structure and sequence was evaluated by comparing the distance matrices derived using the depth annotation distance measure with surrogate distance matrices. Two comparisons were performed: variants of increasing magnitude of a single template (SD-DS1) were compared with Surrogate Distance Matrix 1 (SDM1), and variants from distinct templates (SD-DS2, SD-DS3, and SD-DS4) were compared with one another using Surrogate Distance Matrix 2 (SDM2). The formats for the surrogate distance matrices and the synthetic distance data matrices are depicted in Figure 3.4.

The distances in the first surrogate distance matrix were designed to represent the "true" distances between pairs of sequences. Unfortunately, there is no definitive "true" distance between RNA sequences. The distance depends on the type of sequence being compared. To approximate the "true" distances between variants of the same template the sum of the number of applications of the variation operators used to generate each variant was used. For example, a variant generated via two applications of the variation operators to template A and a variant generated via five applications of the operators were assigned a distance of 7. This approximation makes two assumptions: that there is no smaller number of mutations that can transmute one variant into another, and that these mutations are reasonable base steps. This first assumption is reasonable given the small number of mutations used in these experiments. The probability that there is a shorter path between variants is unlikely. The second assumption is trickier. The types of changes that are allowable within a single motif vary from motif to motif. The small changes we implement are representative of the variations that show up in observed motifs.

The second surrogate distance matrix contained two distances: a small fixed distance (1) for pairs of sequence variants generated from the same template and a large fixed distance (100) for pairs of variants generated from different templates.

The correlation coefficients between the experimentally derived distance matrices and the surrogate distance matrices were calculated using the Pearson product-moment correlation coefficient and the significance was evaluated using the Mantel Test. The Mantel Test was run 1000 times with 100 row-column permutations each time.

For both the synthetic and the IRE data sets 2D plots were constructed using the non-linear projection algorithm. The algorithm was run 30 times on each data set and the projection with the least error was selected.

3.5 Results

The correlation coefficient and significance for the comparison of sequence template A and its variants are given in Table 3.5. The distance matrices were not highly correlated with SDM1. This suggests that the number of variations away from the original template is not a good surrogate for the distance. The correlation coefficients for the comparisons of the multi-sequence distance matrices and SDM2 are also given in Table 3.5. In each instance there is a very high degree of correlation between the surrogate distance matrix and the distance matrix. Thus the depth annotated distance is capturing the notion of different structures fairly well.

The non-linear projections for the synthetic data sets are given in Figures 3.6, 3.7, 3.8, and 3.9. The projection for SD-DS1 contains three clear clusters as well as several outlier points. The projections for SD-DS2 through SD-DS4 each show two highly distinct clusters. The clusters correctly group the sequence variants from distinct templates together. Only SD-DS4 contains a significant outlier. Examination of the outlier determined that the RNA brick was structurally distinct from the members of both the other clusters and that it originated from an error in the folding constraint generator used with MFold.

The non-linear projection for the IRE data is shown in Figure 3.10. The hand-folded IREs and the unconstrained folded bricks from sequences containing instances of IRE motifs form fuzzy clusters. The non-IRE containing bricks scatter throughout the plot and do not form clusters.

3.6 Conclusions

This initial implementation shows promise for the RNA motif search pipeline. The low correlation between SD-DS1 and SDM1 was not unexpected given the difficulty in estimating a "true" distance. The high correlation coefficients for SD-DS2 through SD-DS4 and SDM2 provide strong evidence that

	A1	A2	...	An	D1	D2	...	Dn
A1	0	D(A2,A1)		D(An,A1)	D(D1,A1)	D(D2,A1)		D(Dn,A1)
A2	D(A1,A2)	0		D(An,A2)	D(D1,A2)	D(D2,A2)		D(Dn,A2)
...			0					
An	D(A1,An)	D(A2,An)		0	D(D1,An)	D(D2,An)		D(Dn,An)
D1	D(A1,D1)	D(A2,D1)		D(An,D1)	0	D(D2,D1)		D(Dn,D1)
D2	D(A1,D2)	D(A2,D2)		D(An,D2)	D(D1,D2)	0		D(Dn,D2)
...							0	
Dn	D(A1,Dn)	D(A2,Dn)		D(An,Dn)	D(D1,Dn)	D(D2,Dn)		0

	AM1-1	AM1-2	...	AM1-10	AM2-1	AM2-2	...	AM2-10	...
AM1-1	0	2	2	2	3	3	3	3	
AM1-2	2	0	2	2	3	3	3	3	
...	2	2	0	2	3	3	3	3	
AM1-10	2	2	2	0	3	3	3	3	
AM2-1	3	3	3	3	0	4	4	4	
AM2-2	3	3	3	3	4	0	4	4	
...	3	3	3	3	4	4	0	4	
AM2-10	3	3	3	3	4	4	4	0	
...									

	A1	A2	...	An	D1	D2	...	Dn
A1	0	1		1	100	100		100
A2	1	0		1	100	100		100
...			0					
An	1	1		0	100	100		100
D1	100	100		100	0	1		1
D2	100	100		100	1	0		1
...							0	
Dn	100	100		100	1	1		0

Figure 3.4 The top matrix gives the format for the distance measure derived distance matrices. D(Ai,Aj) represents the distance between bricks Ai and Aj. The middle matrix gives the format for Surrogate Distance Matrix 1. The bottom matrix gives the format for Surrogate Distance Matrix 2.

Folds	Co. Coef.	P-Value
Fold A w/ Variants	0.56	<0.001
Fold A and Fold D	0.986	<0.001
Fold A and Fold E	0.935	<0.001
Fold D and Fold E	0.940	<0.001

Figure 3.5 The correlation coefficients and significances for the comparison of pairs of folds.

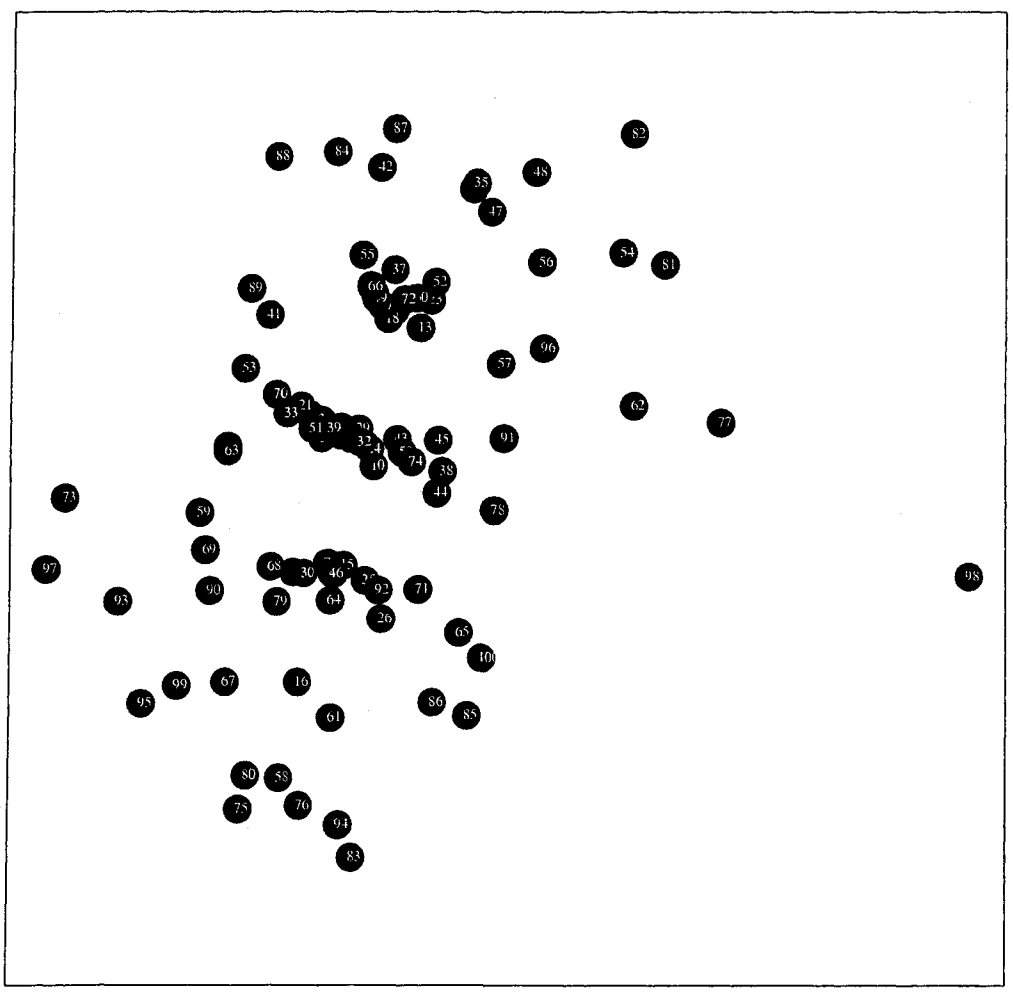

Figure 3.6 The best projection, out of 30 runs, for the first synthetic data set (SD-DS1).

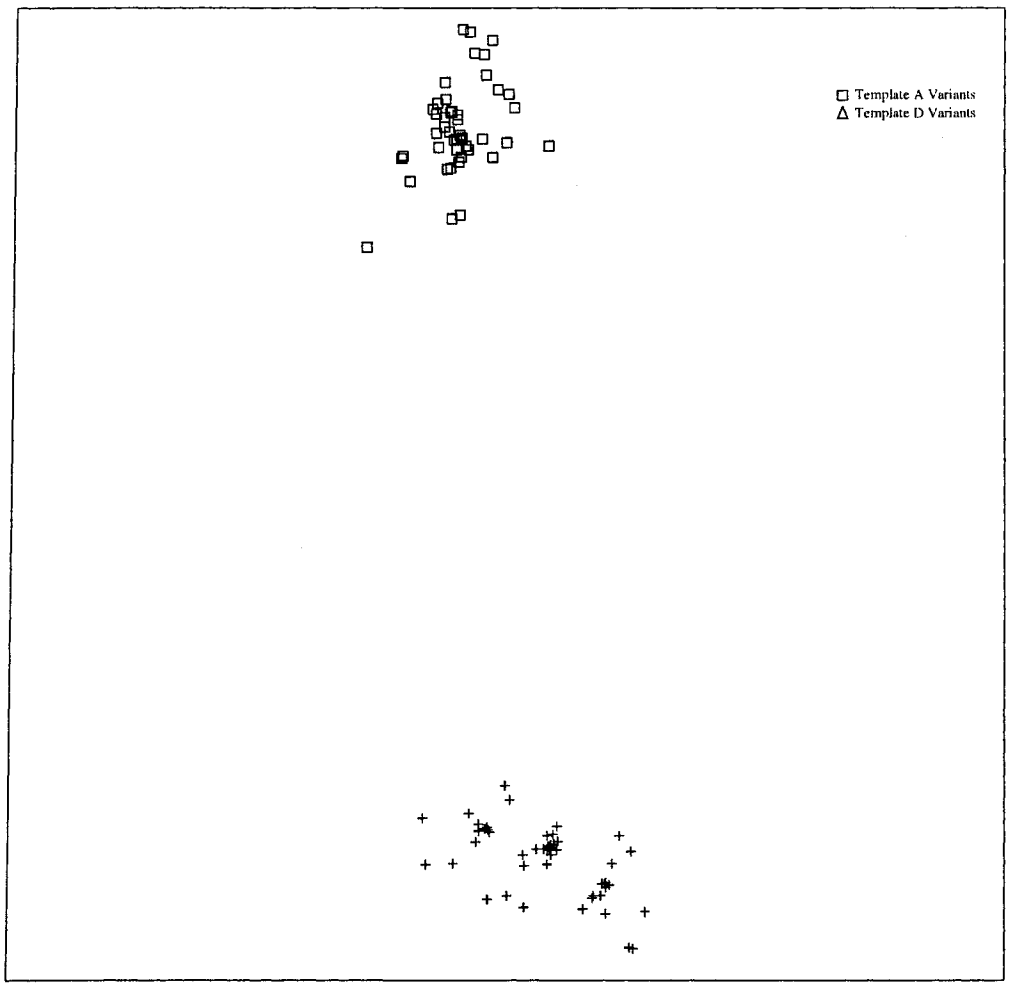

Figure 3.7 The best projection, out of 30 runs, for the second synthetic data set (SD-DS2). The squares represent variants of template A. The triangles represent variants of template D.

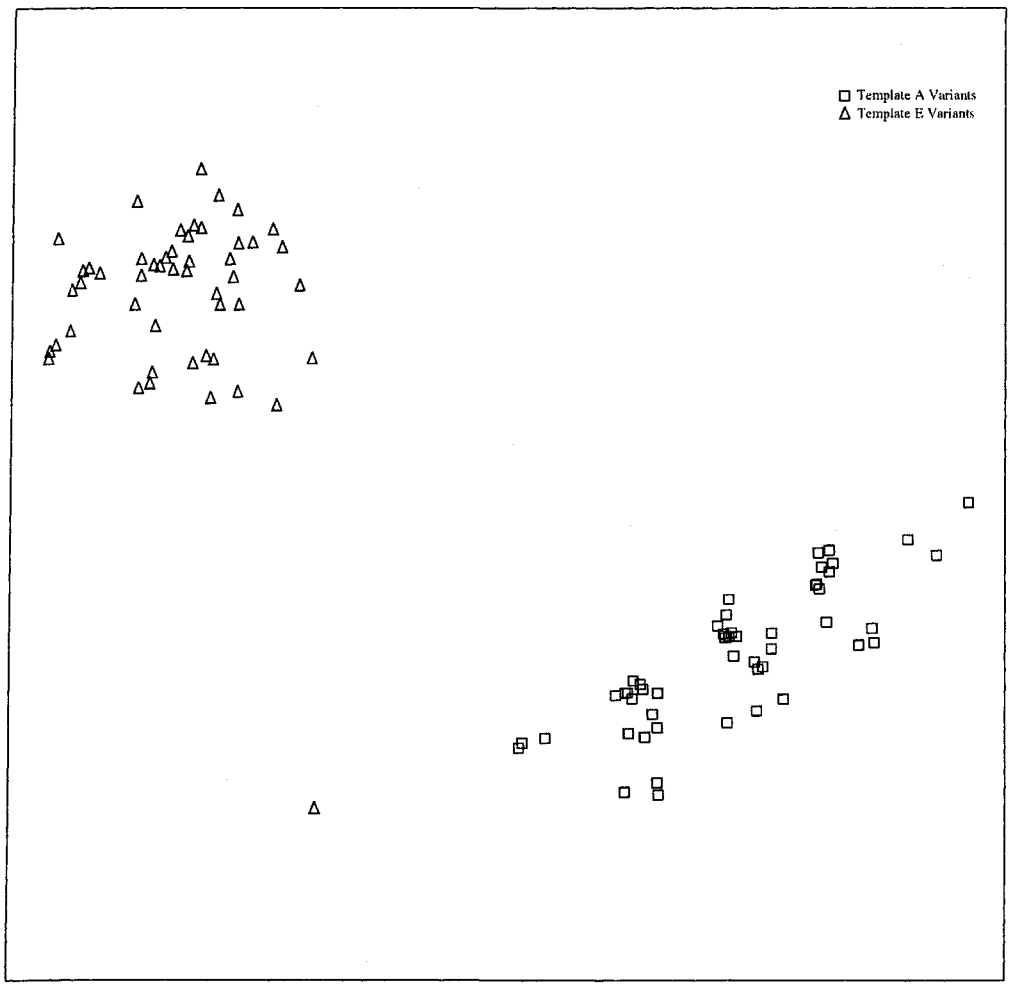

Figure 3.8 The best projection, out of 30 runs, for the third synthetic data set (SD-DS3). The squares represent variants of template A. The triangles represent variants of template E.

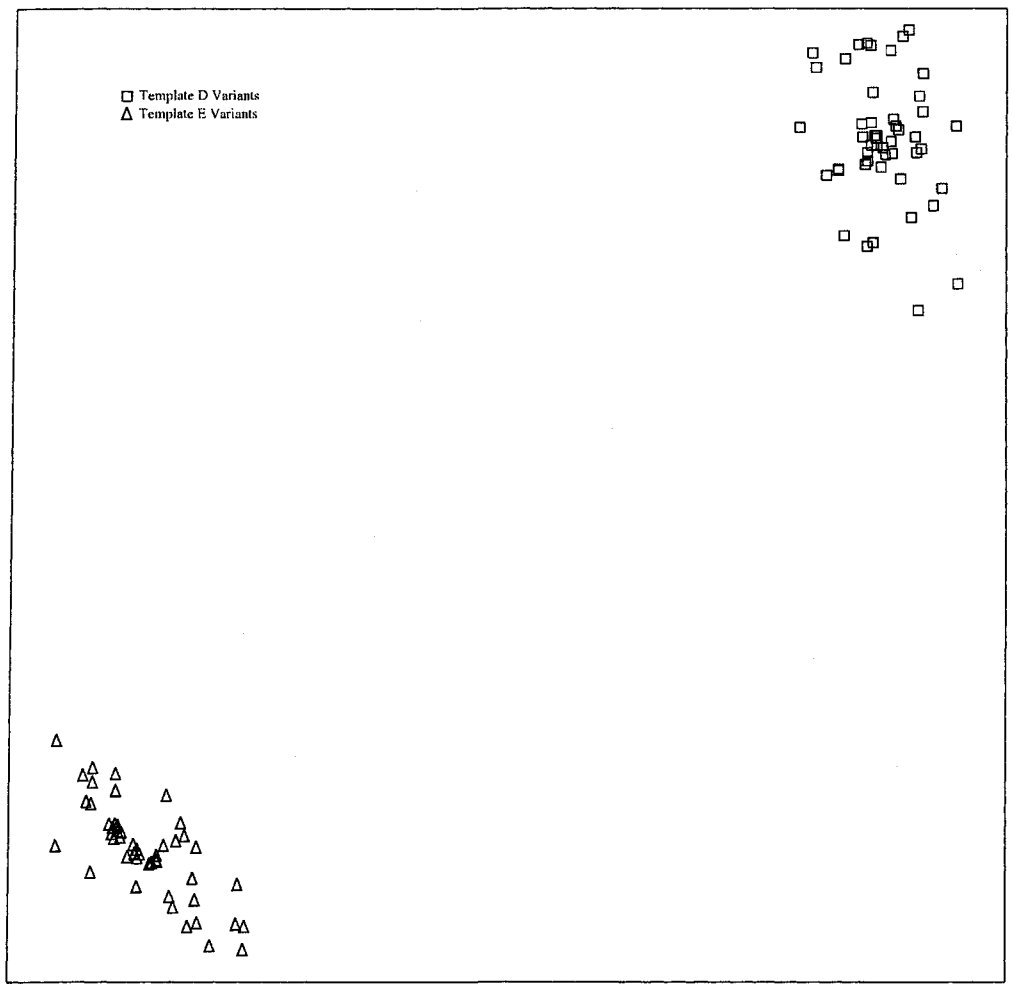

Figure 3.9 The best projection, out of 30 runs, for the fourth synthetic data set (SD-DS4). The squares represent variants of template D. The triangles represent variants of template E.

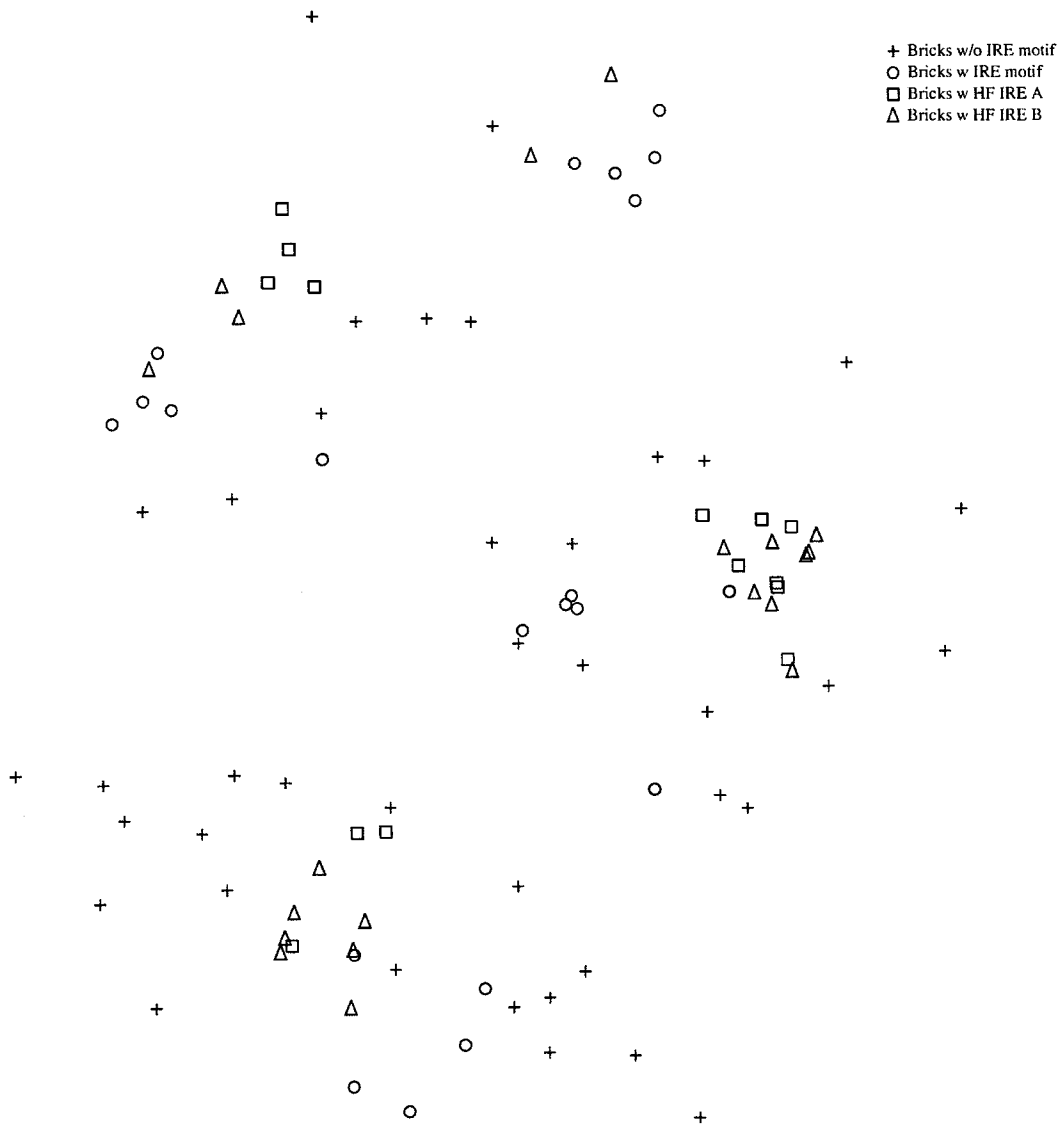

Figure 3.10 The best 2D projection of the IRE data.

the depth annotation distance measure is accurately representing significant structural difference of the type that can define a motif.

The non-linear projections of the synthetic data sets were also promising. The projection of SD-DS1 yielded the expected results. Since the depth annotation distance punishes structural differences more than sequential differences (in this implementation), the formation of several clusters is not surprising. Template variants which were only modified by the base mutation variation operators form clusters when placed in the same projection as template variants which were modified by the variation operators which changed the structure of the template. The non-linear projections of data sets SD-DS2 through SD-DS4 showed that the pipeline is capable of generating easily analyzed visual representations of patterns in the distance data.

The projection of the biological data also yielded patterns, however, much less strongly. The hand-folded bricks formed only fuzzy clusters. There are at least two potential sources for the fuzziness. The first is that the increment size may be too small. A small increment size generates lots of similar sequences which would tend to swamp the signal from the IREs. The other difficulty with the biological data is the inaccuracy inherent in the folds. In each brick containing an IRE the folding software failed to produce the correct fold without the use of constraint files. In previous experiments, not included here, the folding software was run on a variety of differently-sized bricks in order to determine how much context was required to produce the correct fold. In no instance was the correct fold found. There are, however, new folding techniques being developed regularly.

Each brick is given a unique identifier so that any potential clusters can be tested to see that their members come from different sequences.

The distance measure module of the framework is explored in greater depth in Chapter 4. The non-linear projection algorithm and other analysis methods are examined in Chapter 5.

CHAPTER 4. Motif Search on Synthetic Data

4.1 RNA Distances

The Distance Between two RNA structures

When determining the similarity of two strands of RNA it is necessary to consider the functional context. Comparing two strands of mRNA, for example, is best done by focusing on the sequence of residues. If, however, two ribosomal RNAs are to be compared, it is important to emphasize the secondary structure. Three different measures of distance between RNA structures are evaluated: the tree edit distance, the outline shape distance, and the depth labeled distance. Each of these exhibits a different level of emphasis of sequence versus structure. The tree edit distance counts the minimum number of tree edit operations necessary to convert one RNA, represented as a tree structure, into another. The outline shape distance takes each folded structure and represents it as a series of shape variables derived from the outline of the drawing of the RNA fold. The depth labeled distance uses dynamic programming to align string representations of the RNA sequences plus structural information.

While this thesis looks at three distinct distance measures, many other measures are available including base-pair metrics([43]) and mountain metrics([44]); testing of these is left for subsequent studies. Portions of this chapter were first presented in [45].

Outline Distance

The outline distance is a novel distance measure. The outline for each folded RNA is extracted as a set of points from the picture drawn by the MFold software package([37]). The outlines are then normalized for rotation, translation, and scale, and converted to a set of *shape variables* by the Elliptical Fourier Analysis (EFA) function in the Morpheus([46]) software package. A distance measure is derived from the Fourier coefficients by treating the coefficients associated with an outline as points in Euclidean space and applying the standard distance.

Tree Distance

One of the more common and well studied distance measures for RNA structure is the *tree edit distance.* Each folded RNA is represented as a tree, and the distance between trees is computed as the minimum number of edit operations necessary to convert one tree to the other. There are a wide variety of tree edit distances available. For this study the program RNAdistance included in the Vienna RNA Package toolset was used to generate distances between all pairs of bricks ([47, 48]).

Depth Annotation Distance

A detailed description of the depth annotation distance is given in Chapter 3. The cost matrix and window size used in these experiments are the same ones used in Chapter 3.

4.2 Data Sets

Three types of data are examined in this chapter: random collections of RNA bases, synthetic collections with several instances of five distinct RNA structures, and real biological data consisting of the known forms of the Iron Response Element ([4]).

Random Data

A random data set of fifty sequences each fifty nucleotides in length was created. The nucleotides were chosen uniformly at random. Each sequence was converted into a single brick.

Synthetic Data

The synthetic data used in these experiments was generated by creating variants of five structural templates. Three of the templates (A, D, and E) were presented in the preceding chapter. The two additional templates (B and C) are structurally identical to template A, but contain substantial sequence differences. Template B contains different loop sequences and Template C contains different stem sequences. Templates A, B, and C are listed in Table 4.1.

Nine synthetic data sets are examined. The first four data sets SD-DS1 through SD-DS4 are repeated from Chapter 3. SD-DS1 is constructed from variants of template A. SD-DS2 is constructed from 50 size 3 variants of template A and 50 size 3 variants of template D. SD-DS3 is constructed of 100 size 3 variants, 50 each from templates A and E. SD-DS4 is constructed of 100 size 3 variants, 50 each from

Template ID	Template	Element
A	L0S0L1S0L2	L0 AGCGCAACUACGAAA L1 GCACG L2 CCUAGACAUAAGUUUCGUAC S0 GGCGGAUCAG
B	L0S0L1S0L2	L0 ACGUCGCUACGGCGA L1 CAAGU L2 GUACGCCGUCAAGGCGUGCA S0 GGCGGAUCAG
C	L0S0L1S0L2	L0 AGCGCAACUACGAAA L1 GCACG L2 CCUAGACAUAAGUUUCGUAC S0 CGUACUCGAC

Figure 4.1 The nucleotide sequence and pairing templates used in creating the synthetic RNA sequences. Template A encodes a single hairpin loop(S0). Template B encodes the same structure as Template A with different loop sequences. Template C encodes the same structure as template A with a different stem sequences.

ID	Templates	No. Seqs (No. Var. Operators)
SD-DS5	A, B	50(3), 50(3)
SD-DS6	A, C	50(3), 50(3)
SD-DS7	B, C	50(3), 50(3)
SD-DS8	D	1(0), 10(1), 10(2), ..., 10(10)
SD-DS9	E	1(0), 10(1), 10(2), ..., 10(10)

Table 4.1 Synthetic data sets SD-DS5 through SD-DS9. The first column contains the data set identifiers. The second column contains the templates used to construct each data set. The third column contains the number of variants created by applying the variation operators v times, where v is the number given in the parenthesis.

templates D and E. The data sets SD-DS2 through SD-DS4 are designed to act as a test of the ability of the different distance measures to distinguish between sets of structurally distinct RNA sequences.

Three of the new data sets (SD-DS5, SD-DS6, and SD-DS7) each contain 100 size 3 variants. Fifty of the variants come from one template, and fifty come from another. SD-DS5 contains variants from templates A and B. SD-DS6 contains variants from templates A and C. SD-DS7 contains variants from templates B and C. The data sets SD-DS5 through SD-DS7 are designed to test the ability of the pipeline to distinguish between sets of RNA sequences with very similar structures and very different sequences.

The final two new data sets (SD-DS8 and SD-DS9) were constructed in the same manner as template A, but using templates D and E instead. These data sets were used to examine the ability of the different distance measures to capture small variations in sequence and structure. The new data sets are summarized in Table 4.1.

Biological Data

The Iron Response Element (IRE) is an RNA motif which is involved in iron regulatory pathways. IREs bond to the Iron Regulatory Proteins (IRP-1 and IRP-2). The two forms of the IRE proposed by the literature are shown in Figure 1.2 ([4, 5]). The IRE data set is composed of 10 ferritin mRNAs each containing at least one instance of the IRE motif. Accession numbers: gi—507251, gi—286151, gi—191071, gi—213691, gi—214135, gi—16416388, gi—12802902, gi—15076950, gi—11545422, gi—6753911. The presence of the IRE motif in the first 100 bases of each sequence was determined using RNAMotif ([8]). The first 100 bases of each sequence were converted into six bricks each containing 50 bases and spaced at increments of 10 bases. Of the 60 bricks, 21 contain an instance of the IRE motif. For each of the 21 bricks the fold produced by the MFold software failed to contain the IRE fold proposed by the literature. Each of the sequences was refolded using hand-designed constraints to produce a fold in agreement with the biological structure for IREs reported in the literature. When a sequence contained the potential to fold both the A and B forms, constraints were written to produce both folds.

4.3 Experimental Design

This experiment has two parts: an analysis of the distance matrices for the synthetic data sets and a projection of the distance matrices into 2 dimensions. The analysis of the distance matrices is achieved by calculating correlation coefficients (using the Pearson Product Moment correlation coefficient) and

their significance by using a Mantel Test on pairs of distance and surrogate distance matrices. The first three synthetic data sets are paired with the first surrogate distance matrix introduced in Chapter 3 (SDM1), and the remaining six data sets are paired with the second surrogate distance matrix (SDM2).

The algorithm used to locate projections is called the *non-linear projection (NLP) algorithm* and it is an evolutionary algorithm. It operates on a population of 100 tentative projections, each of which contains $2n$ real values, where n is the number of bricks. The algorithm uses size seven tournament selection, two point crossover, and a mutation operator that adds an unbiased bivariate Gaussian random variable to one of the points in the tentative projection. Both the x and y coordinates of the point undergo Gaussian perturbation. The standard deviation of this Gaussian variable is one-tenth the largest true distance between any two bricks. This choice automatically scales the algorithm to the particular problem, although the choice of one-tenth as a scaling factor is arbitrary. The NLP algorithm is discussed further in Chapter 5.

The algorithm is steady state([49]), placing the results of one instance of tournament selection back into the population before running the next instance of tournament selection. The algorithm is run until 100,000 instances of tournament selection have occurred.

4.4 Results

The Mantel Test was run once with 1000 trial permutations and 100 row-column permutations performed between each trial. The Mantel Test is a statistical test for evaluating the relationship between two different matrices of the same size([50]). The test reports a correlation coefficient for the two matrices and a P-value signifying how likely it would be to generate a random matrix with the same values that had a higher correlation coefficient. The Mantel Test was used to compare each (Data Set, Distance Measure) pair with a surrogate matrix. Data sets SD-DS1, SD-DS2, and SD-DS9 were compared to SDM1. The remaining six data sets were compared to SDM2.

The non-linear projection algorithm was run 30 times for each combination of data set and distance measure. All three distance measures yielded projections with some form of cluster in both the synthetic and biological data while demonstrating a gratifying absence of clusters in the random data.

Results for Random Data

Figures 4.3, 4.4, and 4.5 show the best (lowest RMS error) projection from each of the 30 runs for each distance measure for the random sequences. Random sequences were included in this study as a

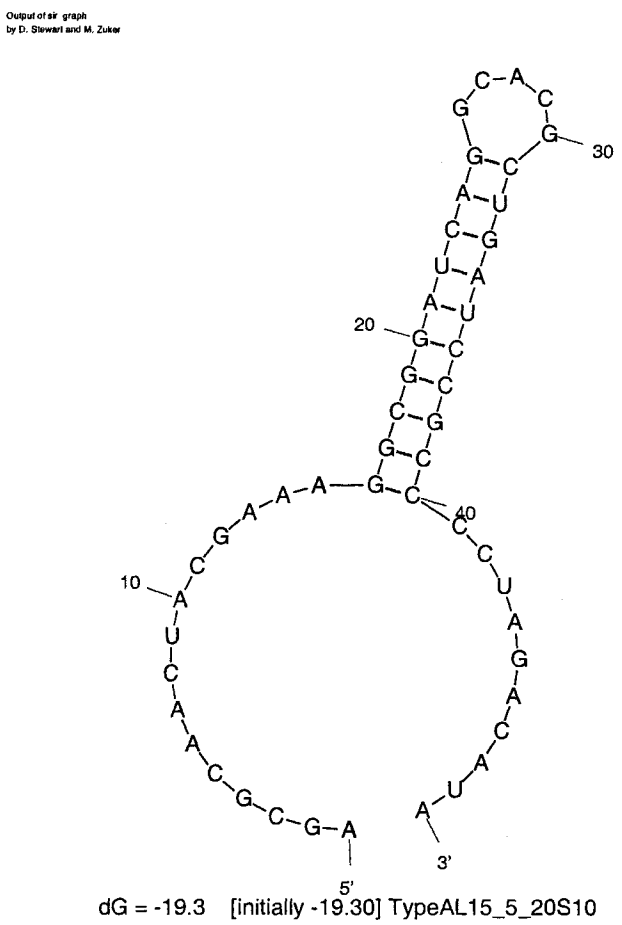

Figure 4.2 An example of the output from the MFold figure drawing software on a variant of template A.

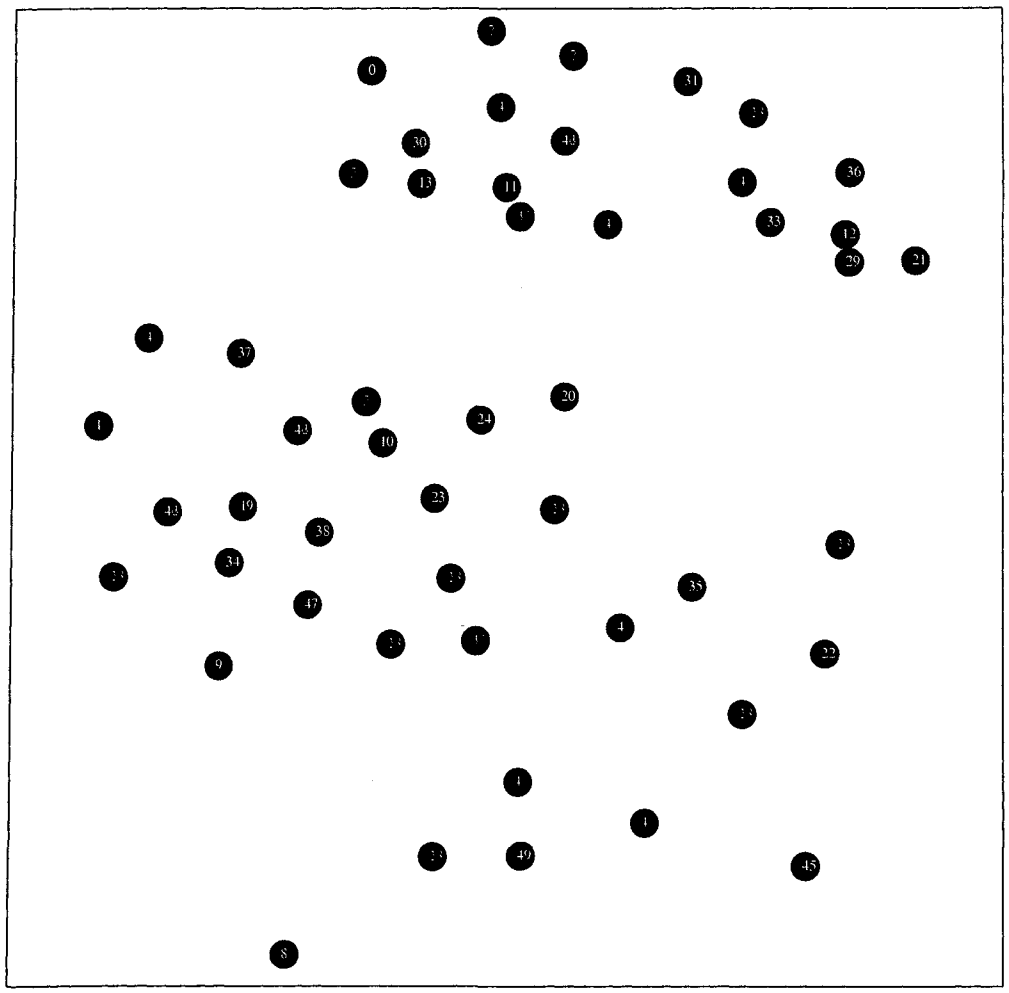

Figure 4.3 The best projection of the random data for the depth label distance measure.

control and, for a control, yielded the desired result: a lack of any discernible pattern. The outline-based distance measure produces a region of higher density in the lower-right part of the displayed region; given that a similar effect occurs with the biological data and, arguably, with the synthetic data for the outline-based distance measure it seems that this is simply the way the outline measure places random samples. Cursory examination of the folds of random objects suggests that the clump-plus scatter character of the outline based folds of random sequences is the result of an artifact of the process used to draw the outlines. The MFold software draws the sequences flanking the outermost stem in a large circular pattern (see Figure 4.4). This doesn't overwhelm the analysis, but it does create a noticeable bias.

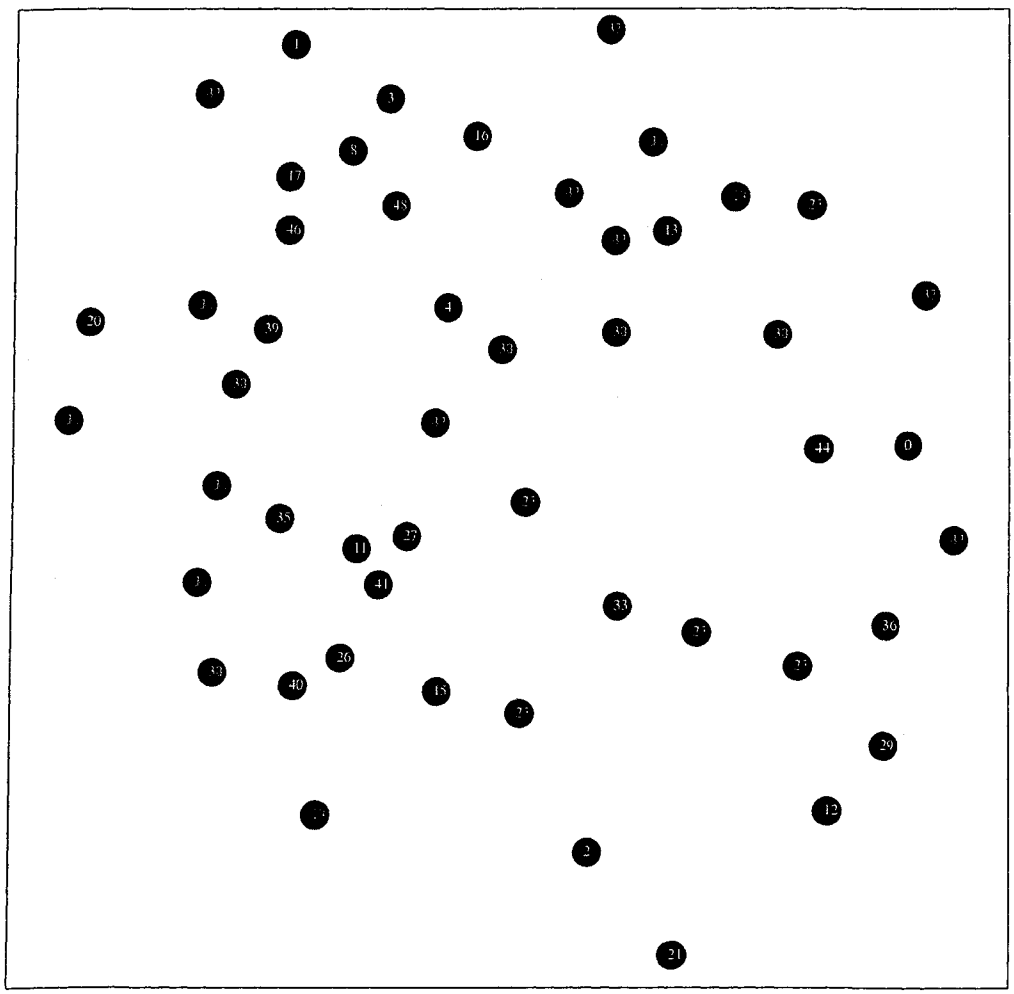

Figure 4.4 The best projection of the random data for the tree edit distance measure.

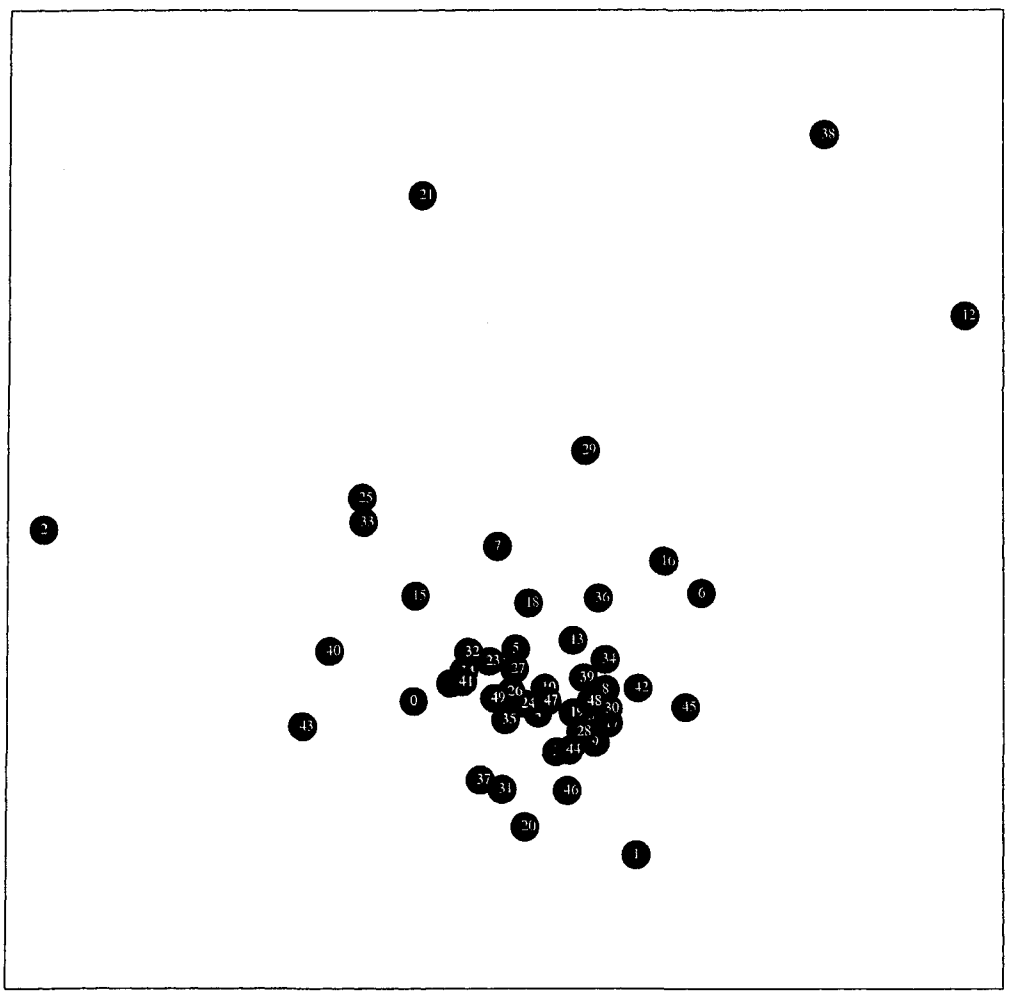

Figure 4.5 The best projection of the random data for the outline distance measure.

Data Set	Depth Label		Tree Edit		Outline	
	Corr. Co.	P-Value	Corr. Co.	P-Value	Corr. Co.	P-Value
SD-DS1	0.564	<0.001	0.458	<0.001	0.369	<0.001
SD-DS8	0.344	<0.001	0.439	<0.001	0.269	<0.001
SD-DS9	0.476	<0.001	0.379	<0.001	0.304	<0.001

Table 4.2 The correlation coefficients and P-values for each of the single template and variants data sets. Each distance matrix was compared with SDM1 using the Mantel Test.

Results for Synthetic Data

The three distance measures show distinct behaviors for each of the different data sets. The results for the comparisons of data sets SD-DS1, SD-DS8, and SD-DS9 with the SDM1 are show in Table 4.2. While none of the distance matrices correlates highly with the surrogate distance matrix, the depth annotated distance measure has the highest correlation coefficient for SD-DS1 and SD-DS9, while the tree edit distance has the highest for SD-DS8. The outline distance measure has the lowest correlation with the SDM1 out of the three distance measures for all three of the data sets.

The projections for SD-DS1 are given in Figure 4.6. The projections for SD-DS8 and SD-DS9 are omitted as they are not substantially different from the projections for SD-DS1. In the projection for the depth annotated distance, clusters are clearly identifiable. A closer inspection reveals that these clusters are not grouped by the number of variants used to generate them but instead by structural similarity. All of the projections contain a small number of outliers.

The correlation coefficients for the remaining six data sets (SD-DS2 through SD-DS7) are given in Table 4.3. The first three of these data sets (SD-DS2, SD-DS3, and SD-DS4) contain structurally and sequentially distinct sequences. The last three (SD-DS5 through SD-DS7) contain sequences which vary only in their sequences and not there structures.

All three distance measures yield very high correlation coefficients for data sets two through four. The depth label distance has the highest for each of the three comparisons, the tree edit distance the next highest, and the outline distance the least correlation. For data sets five through seven the tree edit and outline distance measures yield low correlation coefficients and non-significant p-values. The depth labeled distance yields relatively high correlation coefficients for data sets SD-DS5 and SD-DS7, the data sets which contain sequence differences in the loop regions. It also yields a low correlation coefficient for SD-DS6, where the difference between the two groups of variants is in the stem sequence.

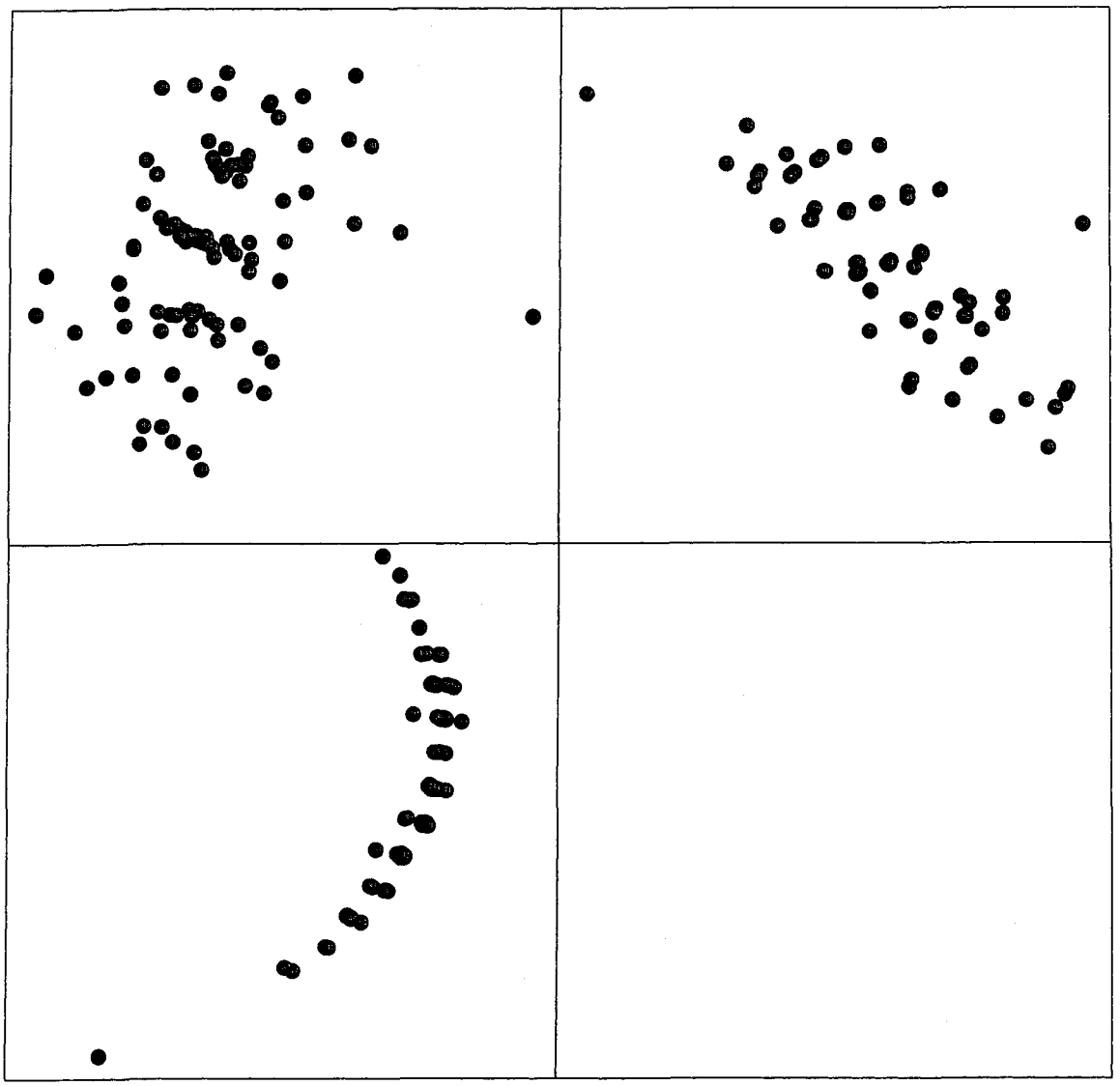

Figure 4.6 The lowest error projections of the three distance measures for the first synthetic data set (variants of template A). Starting with the top left panel and proceeding clockwise, the distance measures are: depth labeling, tree edit, and outline. Each dot represents a distinct variant.

Data Set	Depth Label		Tree Edit		Outline	
	Corr. Co.	P-Value	Corr. Co.	P-Value	Corr. Co.	P-Value
SD-DS2	0.986	<0.001	0.957	<0.001	0.931	<0.001
SD-DS3	0.935	<0.001	0.918	<0.001	0.852	<0.001
SD-DS4	0.993	<0.001	0.957	<0.001	0.716	<0.001
SD-DS5	0.673	<0.001	0.028	0.239	0.021	0.300
SD-DS6	0.277	<0.001	0.028	0.242	0.026	0.245
SD-DS7	0.796	<0.001	0.029	0.232	0.026	0.250

Table 4.3 The correlation coefficients and P-values for synthetic data sets two through seven. Each distance matrix was compared with SDM2 using the Mantel Test.

The projections for the second through fourth data sets show that all three distance measures cleanly partition the data. The depth label distance measure provides the tightest clusters.

The projections for the fifth through seventh data sets show that the depth label distance clearly partitions each data set. Both the tree edit and outline distance measures fail to partition the data.

Results for Biological Data

For the biological data there is better clustering of the biologically correct IRE folds of both types, shown in Figures 4.13, 4.14, and 4.15, with the distance derived from depth-annotated dynamic programming. The tree edit distance produces some groupings of the biologically correct folds, but not as much as the depth labeling. The outline based method produced a clump in which all four sorts of folds (non-IRE, M-fold default IRE, and type A and B biological IRE folds) seem well mixed. The clumping, again, appears to be the result of an artifact in the outline drawing process. Inspection showed the center of the clump, however, consisted primarily of the IRE-containing folds. The non-IRE folds also produced many outliers indicating a non-zero signal-to-noise ratio for the outline technique on the biological data. The biological data provided the clearest performance gradient for the three distance measures.

4.5 Conclusions

With human intervention providing correct biological folds for IREs, proof-of-concept for at least two and perhaps three distance measures within the context of the pipeline is evident. The substantial variation in performance among these distance measures strongly suggests that other distance measures,

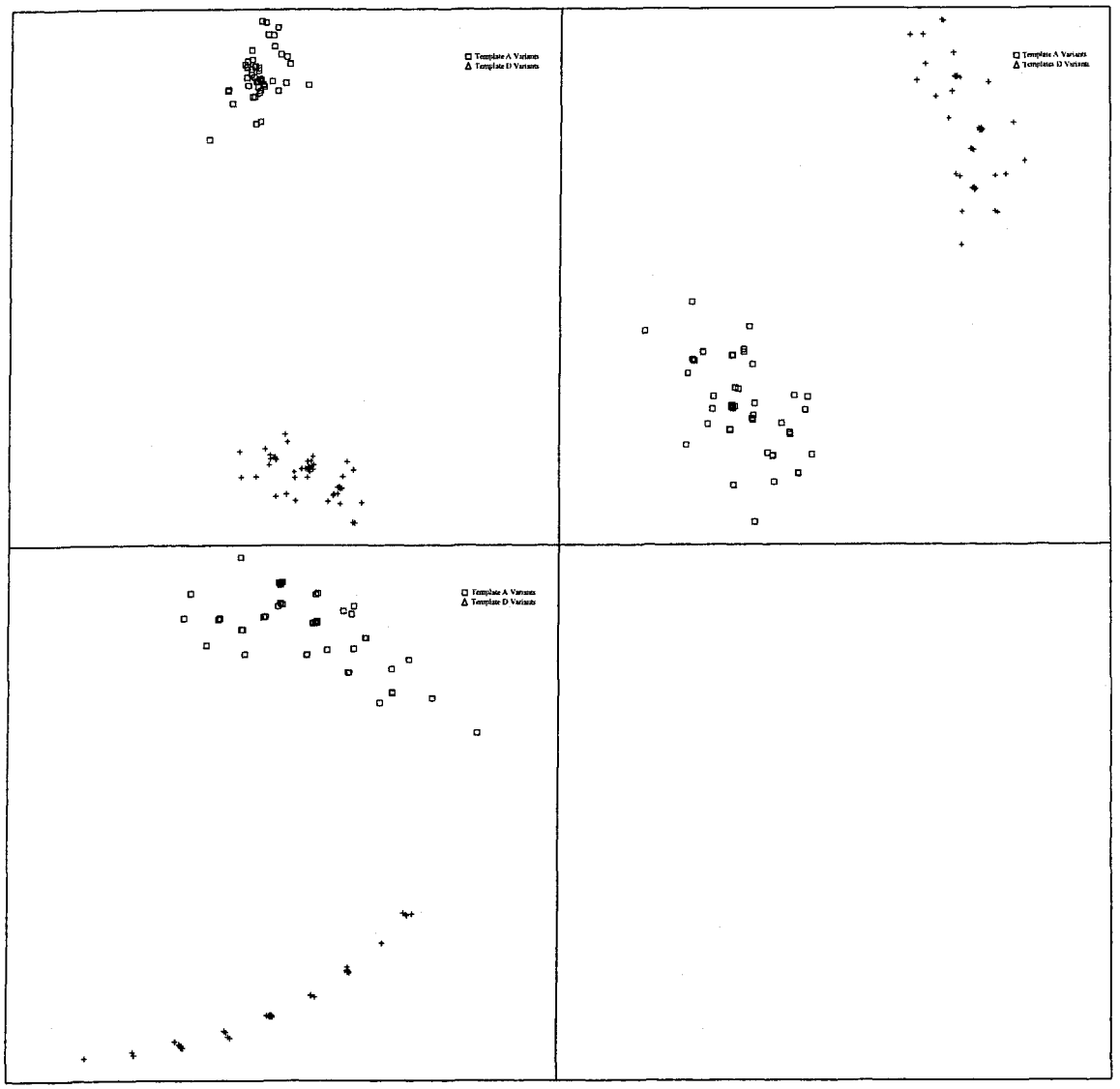

Figure 4.7 The lowest error projections of the three distance measures for the SD-DS2. Starting with the top left panel and proceeding clockwise, the distance measures are: depth labeling, tree edit, and outline. Crosses represent variants of template A and squares represent variants of template D.

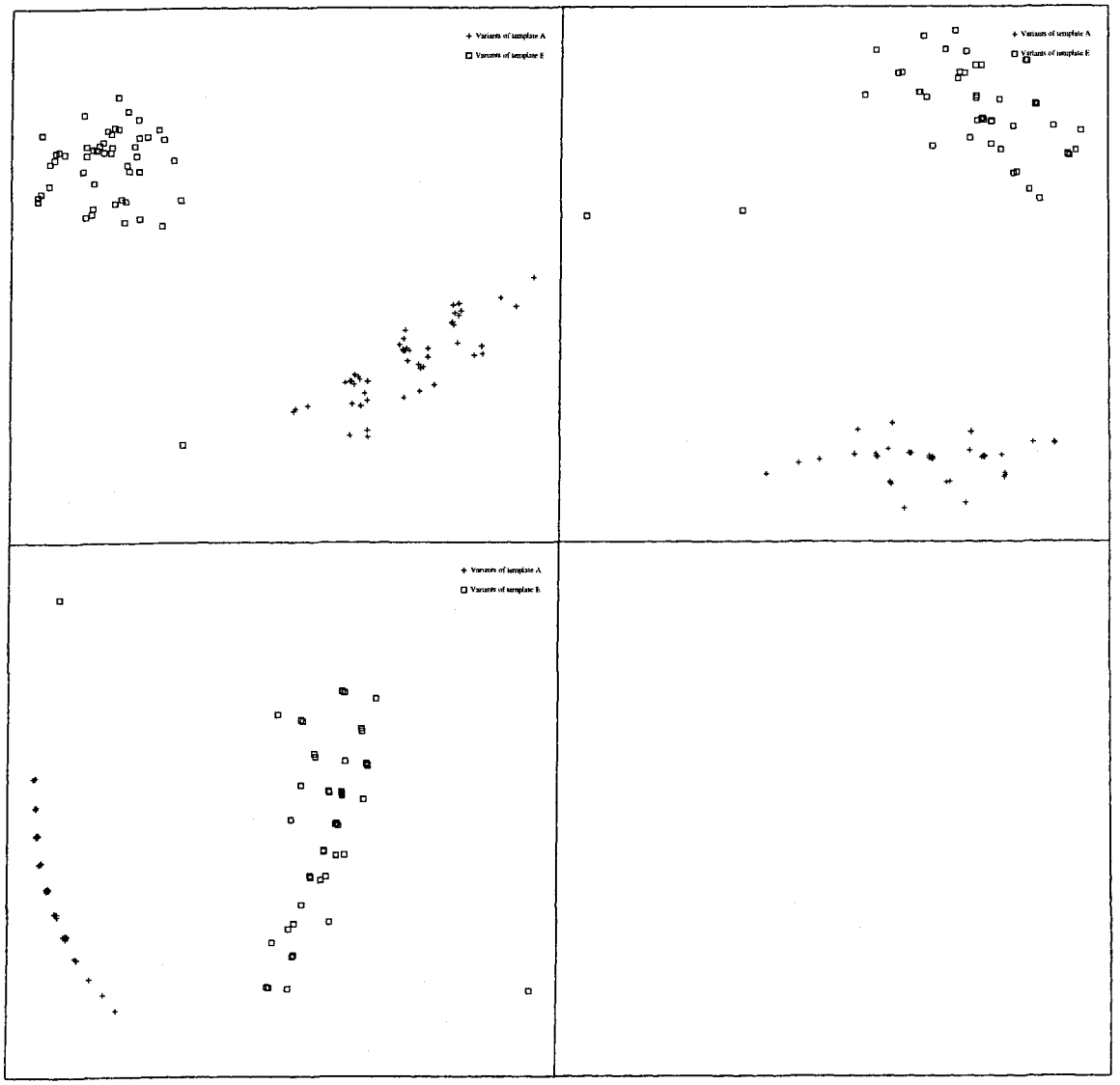

Figure 4.8 The lowest error projections of the three distance measures for the SD-DS3. Starting with the top left panel and proceeding clockwise, the distance measures are: depth labeling, tree edit, and outline. Crosses represent variants of template A and squares represent variants of template E.

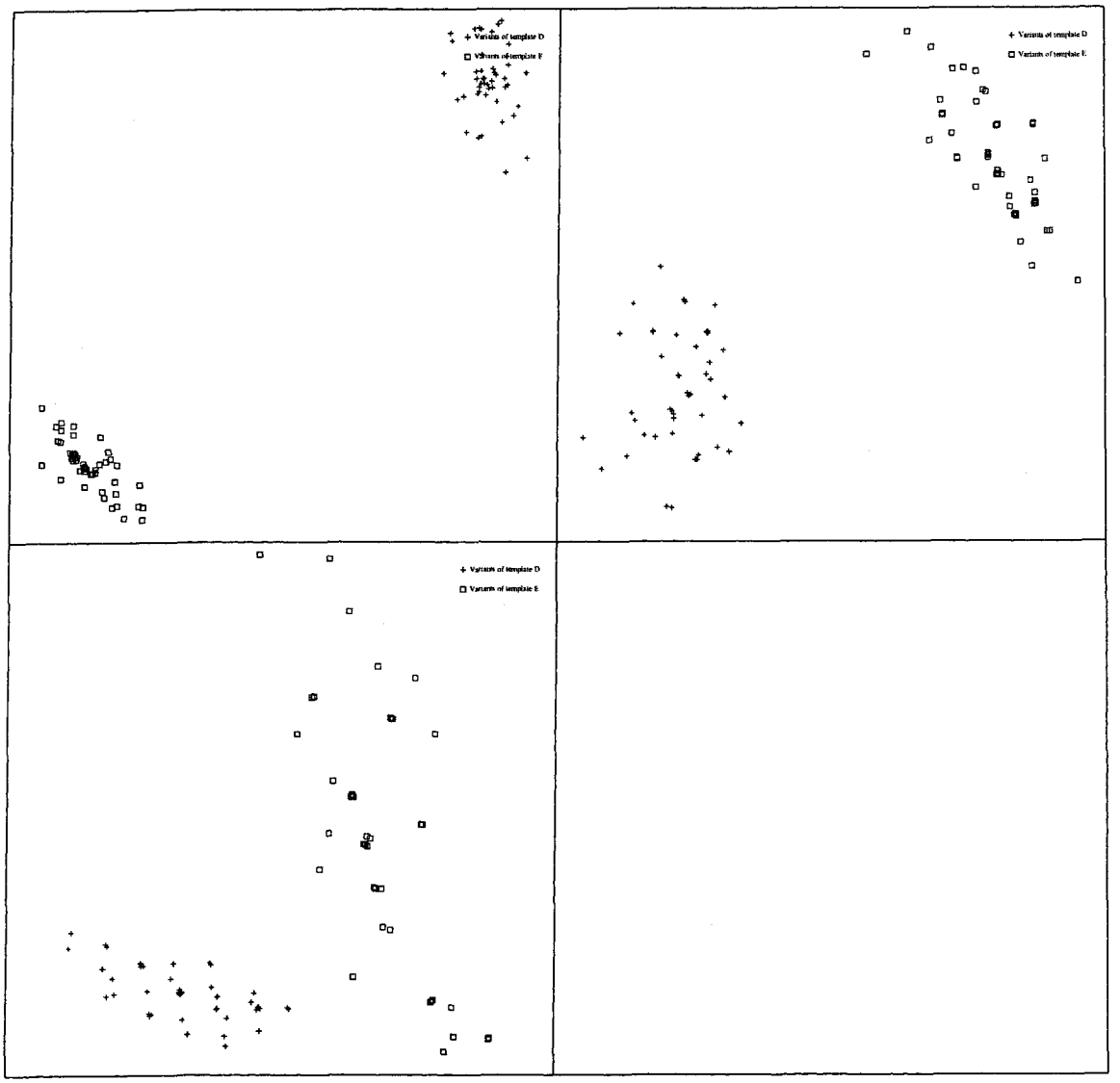

Figure 4.9 The lowest error projections of the three distance measures for SD-DS4. Starting with the top left panel and proceeding clockwise, the distance measures are: depth labeling, tree edit, and outline. Crosses represent variants of template D and squares represent variants of template E.

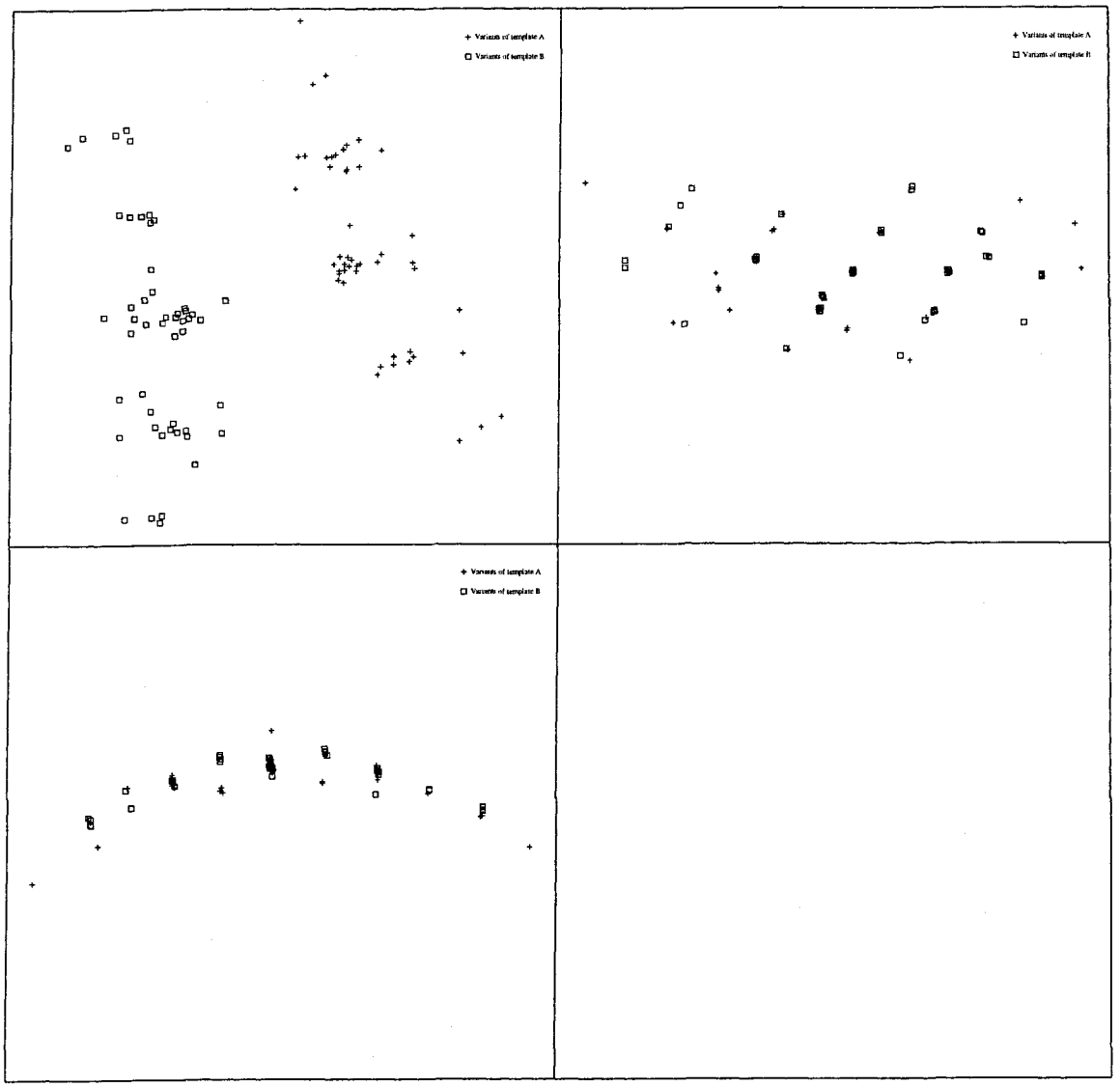

Figure 4.10 The lowest error projections of the three distance measures for the SD-DS5. Starting with the top left panel and proceeding clockwise, the distance measures are: depth labeling, tree edit, and outline. Crosses represent variants of template A and squares represent variants of template B.

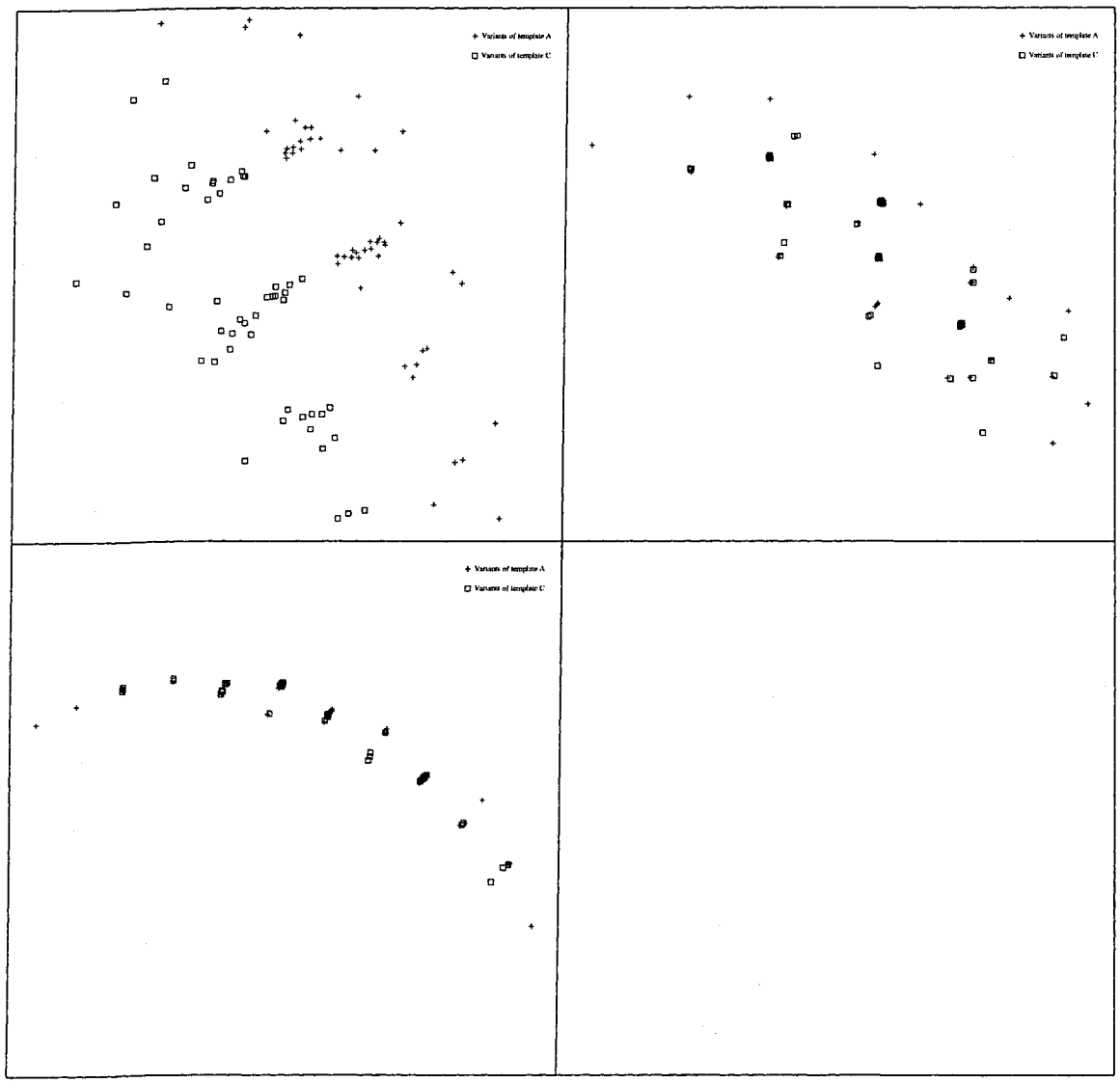

Figure 4.11 The lowest error projections of the three distance measures for SD-DS6. Starting with the top left panel and proceeding clockwise, the distance measures are: depth labeling, tree edit, and outline. Crosses represent variants of template A and squares represent variants of template C.

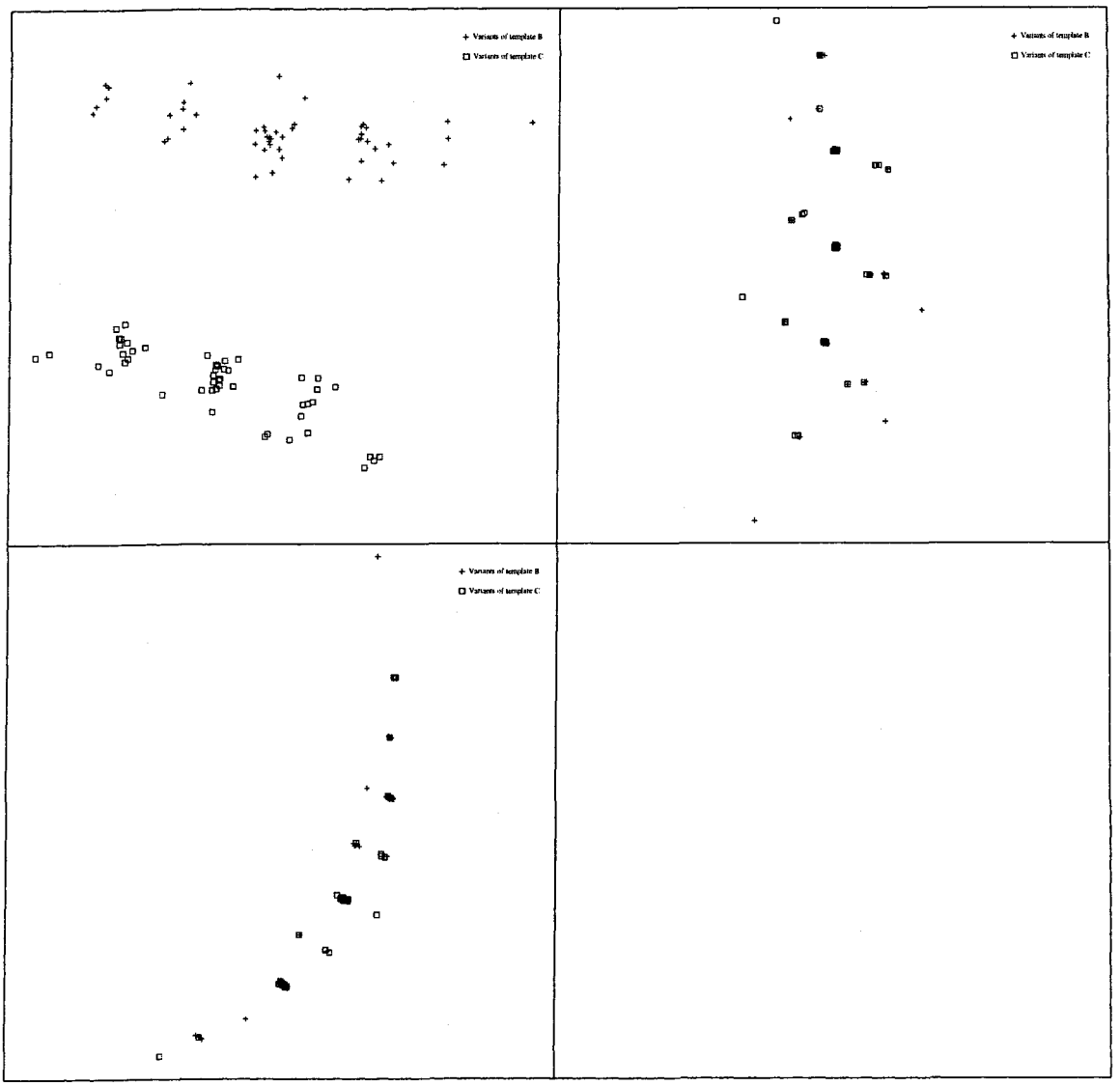

Figure 4.12 The lowest error projections of the three distance measures for SD-DS7. Starting with the top left panel and proceeding clockwise, the distance measures are: depth labeling, tree edit, and outline. Crosses represent variants of template B and squares represent variants of template C.

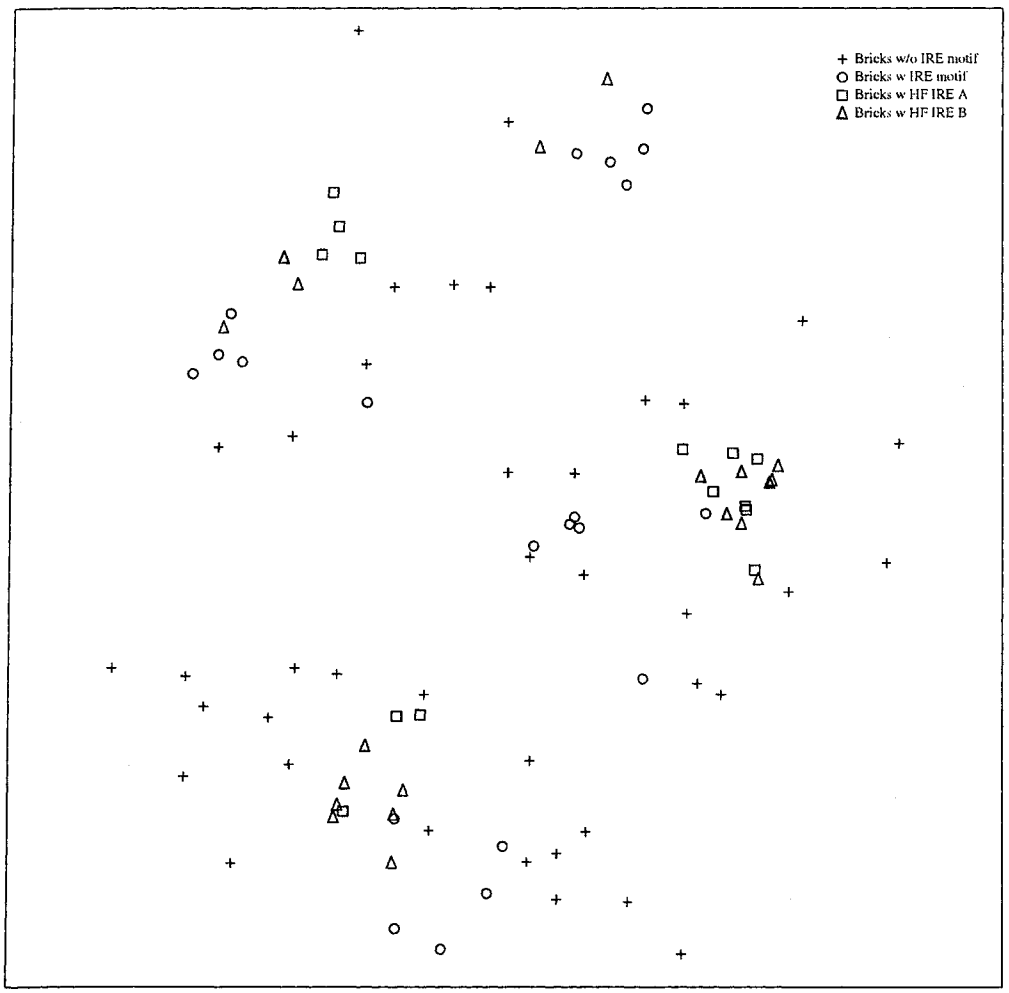

Figure 4.13 The lowest error projection of the IRE data for the depth labeling distance measure. Crosses denote bricks with no IRE, circles denote bricks with M-fold default IRE folds, squares denote correct biological type A IRE folds, and triangles indicate correct biological type B IRE folds.

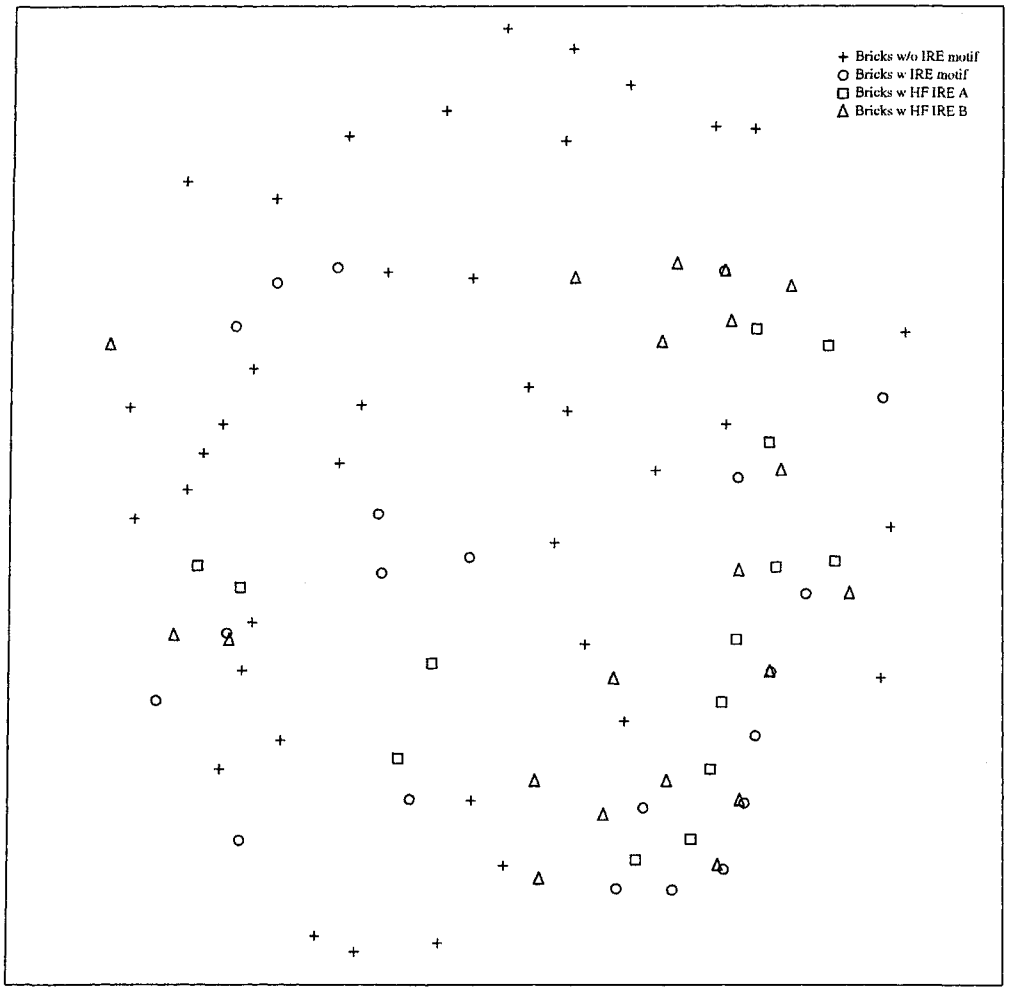

Figure 4.14 The lowest error projection of the IRE data for the tree edit distance measure. Crosses denote bricks with no IRE, circles denote bricks with M-fold default IRE folds, squares denote correct biological type A IRE folds, and triangles indicate correct biological type B IRE folds.

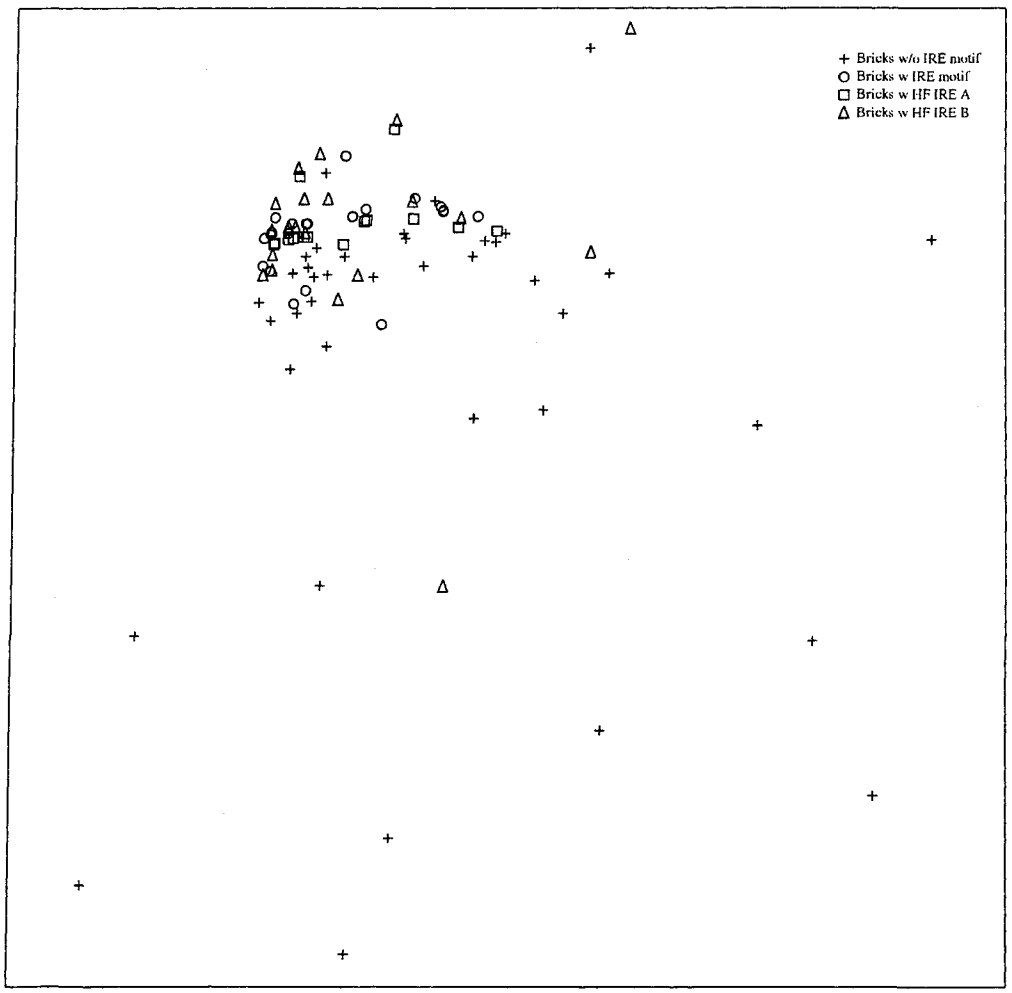

Figure 4.15 The lowest error projection of the IRE data for the outline distance measure. Crosses denote bricks with no IRE, circles denote bricks with M-fold default IRE folds, squares denote correct biological type A IRE folds, and triangles indicate correct biological type B IRE folds.

including ones obtained by parameter tuning of the cost matrix for the dynamic programming algorithm, should be examined. This should be done both in hopes of finding a better measure and also to provide a context in which to judge measures. It should be noted that the measure which performed the "best" was also the most computationally expensive.

Now that actual patterns are forming within the projections, it is time to bring statistical and clustering methods to bear to delineate the actual motifs within the projections. Projection in this study was to two dimensions to permit easy human inspection, but the evolutionary algorithm used to produce the projections is itself dimension-agnostic and can project to any number of dimensions less than the number of folds minus one. This means that if statistical or exploratory methods are used the parameter "number of dimensions to project into" becomes available for tuning to obtain improved performance. Alternatively, we can use PCoA to find the correct number of dimensions. This is explored further in the next chapter.

In [51] several suggestions were made for improving the non-linear projection algorithm. These include initializing projections with principal components analysis layouts, basing fitness only on smaller distances so that longer distances do not excessively distort the relationship of compact clusters, and the usual tuning of population size and rate of application of the variation operators. The appearance of actual patterns in biological data will permit useful evaluation of the impact of such improvements in the non-linear projection algorithm. It is perhaps worth noting that the non-linear projection algorithm often finds multiple distinct projections. The same projection, up to rotation and translation, was also located several times in many cases; this suggests that local optima capable of trapping the evolutionary algorithm are relatively sparse. This is both surprising and convenient from an application perspective.

The Necessity of Using Multiple Folds

Both the scatter of the synthetic sequences associated with template A and the failure at the folding step for IRE-sequences in [40] strongly suggest that producing incorrect (non-biological) folds is a major failure mode for the RNA motif search pipeline. A method for preventing this has already been suggested: the use of multiple folds per brick. This technique will multiply computational cost in many steps by the (average) number of folds used per brick. The non-linear projection step, which involved a computation that is quadratic in the number of bricks, will suffer the most from the use of multiple folds per brick. In spite of this, the lack of a reliable method of computing the natural biological folds for RNA sequences suggests this step is necessary. The hope that RNA folds would at least be *consistently* incorrect and still permit motif location has already been dealt a blow by the failure of IREs, folded

with M-fold default settings, to properly cluster. Although, the incorrectly folded bricks (shown as circles in Figures 4.13, 4.14, and 4.15) do form one small cluster in the center of the depth labeling based projection they do not form clusters that are visible to inspection elsewhere in that or any of the other projections.

Evaluation of Distance Measures with Random Data

The substantial non-uniform density produced for random sequence data by the outline measure, juxtaposed with the nice flat distributions obtained for the depth-labeled and tree edit distances suggest that a test for uniform distribution on random data might be a good preliminary evaluation technique for *any* RNA-structure measure. While its lack of uniformity does not exclude the outline-based technique, it is important that this bias be noted before interpreting results on a non-uniform data set, e.g. any interesting biological data set.

CHAPTER 5. Analyzing High Dimensional Data

5.1 Introduction

This chapter examines two techniques, applied both individually and jointly, for locating patterns in RNA distance matrices: non-linear projection and clustering.

Non-linear projection

Non-linear projection (NLP) is a variation of *non-metric multidimensional scaling*(MDS) ([41]). MDS is used both to locate and display minimally distorted scatter plots of high dimensional data and to experimentally locate the "natural" dimension of a data set. The "natural" dimension is the lowest one in which a projection with little or no distortion of inter-point distances is possible. MDS uses a greedy hill climber with restart to locate n-pace plots with minimal distortion, while the non-linear projection algorithm uses an evolutionary algorithm. One of the several forms of the stress metric for goodness-of-fit of a projection of data to a lower dimension used in MDS is computationally equivalent to the Root Mean Square (RMS) error measure used in the work presented here. A stress metric is a heuristic measure of how well a given projection captures a distance matrix data set.

Principle Coordinates Analysis (PCoA)

Principle Coordinates Analysis is a method for generating points in a Euclidean space whose inter-point distances match those of a distance matrix. If the matrix being matched is non-metric, then a correction may be required. One simple correction is to add a fixed constant to each value of the matrix. Non-metric matrices violate the triangle inequality. The triangle inequality says that if a, b, and c are the sides of a triangle and c is the hypotenuse then $a + b \geq c$. For a non-metric matrix, situations occur where $a + b < c$, creating a bad triple. A bad triple can be "fixed" by adding a constant δ to each member of the matrix so that $(a + \delta) + (b + \delta) \geq (c + \delta)$. For each non-metric matrix there exists some smallest δ which when added to each member of the matrix closes all open triangles.

Clustering

This study uses a k-means multi-clustering algorithm ([52]) to cluster the data. The main advantage of multi-clustering is that it doesn't require the user to specify a fixed number of clusters, but instead produces a *cutplot* which provides evidence as to the "natural" number of clusters in the data. Description of the k-means and multi-clustering algorithms are given below.

Algorithms

Non-Linear Projection (NLP)

The non-linear projection algorithm that produces the projection function P is an evolutionary algorithm. The algorithm acts on a population of tentative projections, stored as chromosomes (data structure) that contain $2n$ real numbers, where n is the number of bricks. The real numbers are interpreted in pairs as the coordinates of the points assigned to the bricks. The coordinates of the point that the first brick projects to are the first two values in the chromosome in the order x,y. The second brick is projected to a point whose x and y coordinates are the third and fourth values in the chromosome, and so on. The fitness of a chromosome is the sum, over all pairs of bricks, of the squared difference between a given RNA-structure distance between the bricks and the Euclidean distance between the points those bricks are projected to. This statistic is the RMS error between the two matrices. This fitness is minimized with the goal of finding an assignment of bricks to points so that the distances between the points match the distances between their corresponding bricks. The parameter settings for the NLP algorithm are the same as they were in Chapter 4. The default behavior of the algorithm is to use randomly initialized points. Each point is assigned an X and Y value between 0 and α * the largest distance in the distance matrix, where α is a user specified fudge factor.

k-means Clustering

Algorithm 1 k-means

Input: *1) A set S of points in $\mathbb{R}^n$*

2) A desired number k of clusters

3) A bound B on the number of cycles permitted

Output: *A category function*

$$C : S \to \{0, \ldots, k-1\}$$

Choose k distinct points in S as initial cluster centers.
Repeat
  Assign each point to the cluster whose center
    it is closest to, breaking ties at random.*
  Recompute cluster centers as the average of all
    points in the cluster.
Until (no points change their cluster assignment or
  B cycles have occurred+)
Report the assignment of points to clusters as C.

** for real-valued data such ties seldom occur*
+ for real-valued data B is seldom required

The output of Algorithm 1 is a category function,

$$C : S \to \{0, \ldots, k-1\}.$$

If two points i and j have the property that $C(i) = C(j)$, then we say that i and j are *in the same cluster.* We also say that i *is in cluster number* $C(i)$. The category function C is a convenient mathematical way of summarizing the clusters. It gives the number of the cluster containing a point.

Multi-clustering

Multi-clustering takes advantage of the fact that any k-means clustering performed with an excessively large number (k) of clusters yields useful information about which pairs of points should be associated. Re-running the k-means algorithm yields potentially *different* information about which points should be associated.

Informally, k-means based multi-clustering proceeds as follows. Pick some number N of clusterings to perform. Pick a distribution D of possible numbers of clusters. The algorithm performs N clusterings, selecting the number of clusters in a given clustering from D. Before clustering, the algorithm initializes a set of pairwise connection strengths for each pair of points with an initial strength of zero. Whenever a k-means clustering places two points in a cluster together, the algorithm increases their connection strength by 1. We then divide all the connection strengths by the number of clusterings performed to

yield connections strengths in the interval $[0, 1]$. After all the clustering is done and the final connection strengths have been computed, a cutoff value C is chosen, and only connections with strength exceeding C are retained. If we view the surviving connections as edges of a combinatorial graph ([53]) that has the data items as vertices, then the clusters are the connected components of this graph.

When performing k-means clustering it is possible to compute the tightness of clusters and use this to select a "good" clustering from many attempts. Multi-clustering yields a nice tool for allowing the user to see if there is a natural number of clusters. We call this tool the *cut plot*. The cut plot is a function that maps possible cut values onto the number of connected components that would result if the given cut value were used. The shape of the cut plot yields information about "natural" numbers of clusters. Flat areas in the cut plot indicate that the number of clusters is stable for that range of cut values. Large flat areas indicate a stable or "natural" number of clusters. This description of multi-clustering is taken from ([52]). Examples of cut plots appear in Figure 5.2 and Figure 5.3. It is worth noting that the first cluster in the k-means algorithm given here is cluster zero.

5.2 Experiments

This chapter contains three distinct experiments, each designed to evaluate different aspects of the data analysis portion of the pipeline. The first experiment verifies that the projections produced by the NLP algorithm correlate highly with the distance matrices upon which they are based. The second experiment looks to see how well the projections cluster using a k-means multi-clustering algorithm. The third experiment tests whether it is more effective to cluster the projections or the n-dimensional point representations of the distance matrices.

5.2.1 Experiment 1: Evaluation of the distance between projected points

Description

This experiment examines how well the Euclidean distances between the points in the 2D plane, selected by NLP, correlate to the depth labeled distances for those same RNA bricks. The (X,Y)-coordinates for each point are taken from the projection with the smallest RMS error between depth annotated distances and projected distances out of thirty projections. A Euclidean distance matrix is generated for each data set and compared to the depth annotated distance matrix using the Pearson Product Moment correlation coefficient and the significance of the comparison is assessed using a Mantel test. The analysis is performed on the synthetic datasets SD-DS1 through SD-DS9.

Data Sets	Depth Label	
	Corr. Co.	P-Value
SD-DS1	0.956	<0.001
SD-DS8	0.974	<0.001
SD-DS9	0.920	<0.001

Table 5.1 The correlation coefficients and P-values for SD-DS1, SD-DS8, and SD-DS9. Each distance matrix was compared with DM1 using the Mantel Test.

Template	Depth Label	
	Corr. Co.	P-Value
SD-DS2	0.9980	<0.001
SD-DS3	0.9889	<0.001
SD-DS4	0.9982	<0.001
SD-DS5	0.963	<0.001
SD-DS6	0.954	<0.001
SD-DS7	0.978	<0.001

Table 5.2 The correlation coefficients and P-values for data sets SD-D2 through SD-DS7. Each distance matrix was compared with DM2 using the Mantel Test.

Results

All pairs of distance matrices were highly correlated and the correlations were found to be significant.

Discussion

This experiment provides strong evidence that the points in the non-linear projections accurately represented the distances between the RNA bricks.

5.2.2 Experiment 2: Clustering on Non-Linear Projections

5.2.2.1 Description

This experiment looks at how the projected points from four synthetic data sets (SD-DS2, SD-DS5, SD-DS10, and SD-DS11) cluster. Data sets SD-DS10 and SD-DS11 are described in Table 5.3. Each of the data sets (see Figure 5.1) was analyzed using k-means multi-clustering.

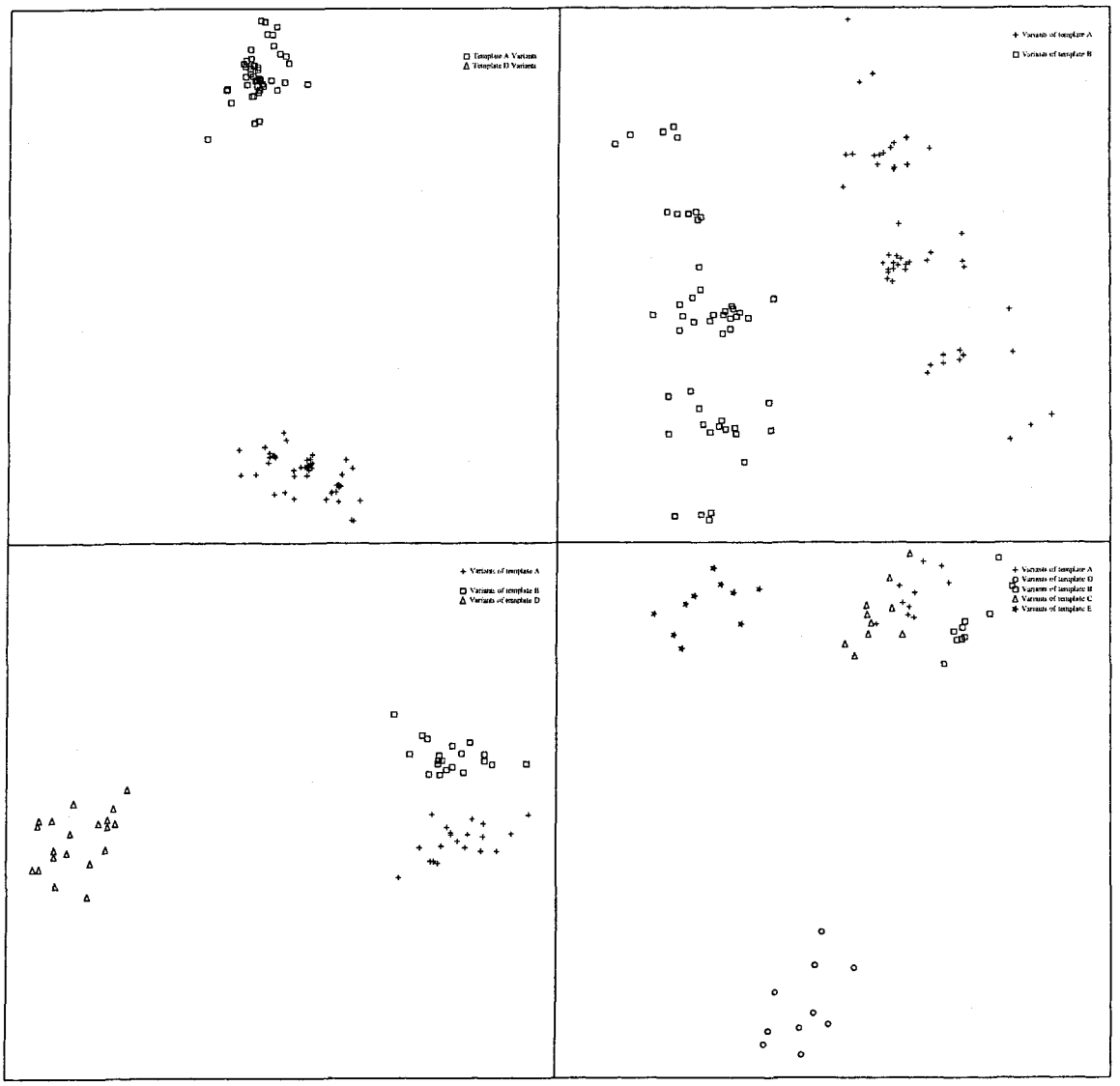

Figure 5.1 The least error projections for the four data sets used in Experiments 2 and 3 calculated from depth labeled matrices. They are, going clockwise from the top-left, SD-DS2, SD-DS5, SD-DS10, and SD-DS11.

ID	Templates	No. Seqs (No. Var. Operators)
SD-DS10	A, B, D	20(5), 20(5), 20(5)
SD-DS11	A, B, C, D, E	10(5), 10(5), 10(5), 10(5), 10(5)

Table 5.3 Synthetic data sets SD-DS10 and SD-DS11. The first column contains the data set identifiers. The second column contains the templates used to construct each data set. The third column contains the number of variants created by applying the variation operators v times, where v is the number given in the parenthesis.

Data Set	No. Clusters	Misclustered Points
SD-DS2	2	0 out of 100
SD-DS5	2	0 out of 100
SD-DS10	3	0 out of 60
SD-DS11	5	1 out of 50

Table 5.4 The number of misclustered points for each of the data sets given the specified number of clusters.

5.2.2.2 Results

The cut plots for SD-DS2, SD-DS5, SD-DS10, and SD-DS11 are given in Figure 5.2. Each of the cut plots contains a broad flat region around the correct number of clusters, although in the cut plot for SD-DS11 it is a relatively small region.

For each data set the multiclustering algorithm found the correct clusters (See Table 5.4). For SD-DS2 and two clusters the algorithm correctly clustered variants of A with variants of A and variants of D with only other variants of D. For SD-DS5 and two clusters the algorithm correctly partitioned variants of A from variants of B. For SD-DS10 and three clusters each set of variants was assigned to its own cluster. In SD-DS11 with five clusters once again the algorithm found the correct clusters.

Discussion

The second experiment showed that for the synthetic data sets the multi-clustering algorithm was able to find the "natural" number of clusters and to cluster the variants appropriately. When too many or too few clusters were specified the algorithm handled it appropriately. On data sets SD-DS2 and SD-DS5, forcing the algorithm to find three clusters caused it to split one of the variants into two clusters.

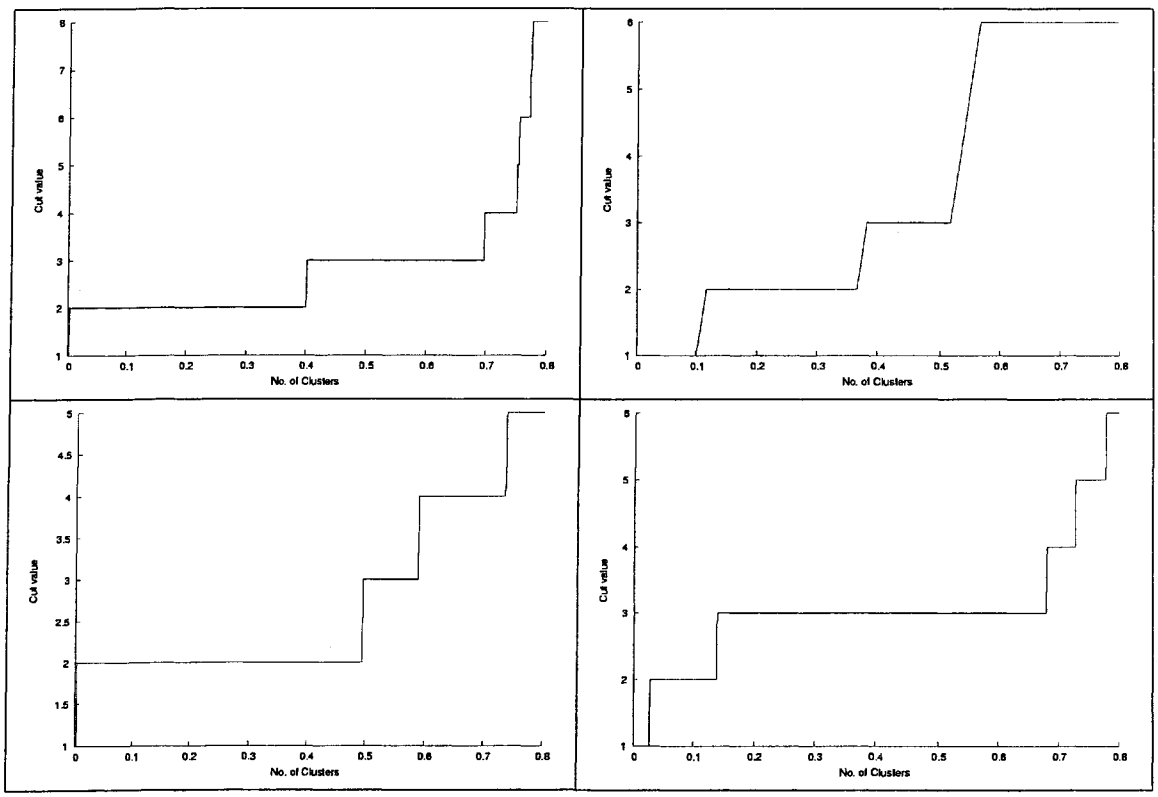

Figure 5.2 The cut plots for SD-DS2, SD-DS5, SD-DS10, and SD-DS11.

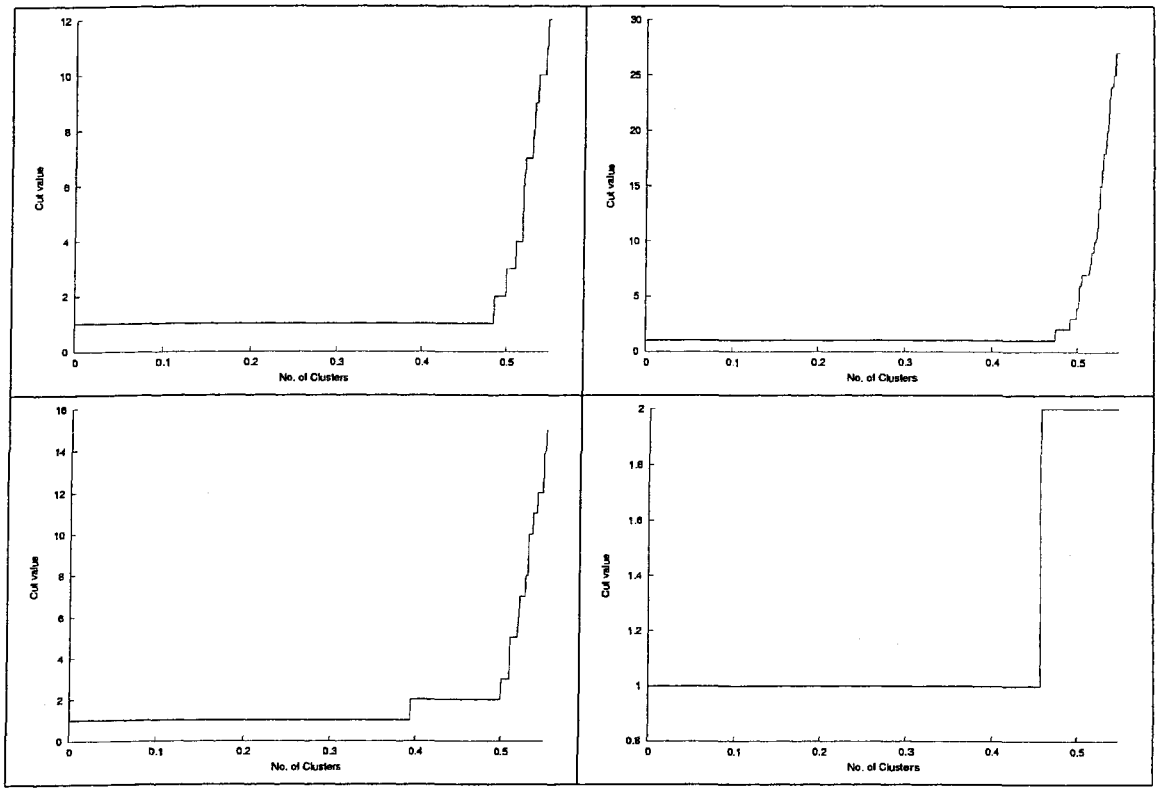

Figure 5.3 The cut plots for SD-DS2, SD-DS5, SD-DS10, and SD-DS11.

Specifying two clusters for SD-DS10, where there were naturally three, caused the algorithm to place variants of template A and variants of template B in the same cluster, while variants of template D made up the second cluster. Specifying only three clusters for SD-DS11 caused variants of templates A, B, and C to be grouped together.

5.2.3 Experiment 3: Clustering on PCoA Points

Description

This experiment looks at how well the points generated from those same four data sets cluster using PCoA based clusters.

Results

The cut plots for the multiclustering algorithm derived from the PCoA generated points contained only trivial flat spots suggesting no "natural clusters". They are shown in Figure 5.3.

The multiclustering algorithm failed to find the correct clustering for any of the data sets (See Figure

Data Set	No. Clusters	Misclustered Points
SD-DS2	2	49 out of 100
SD-DS5	2	49 out of 100
SD-DS10	3	38 out of 60
SD-DS11	5	NA

Table 5.5 The results of clustering points generated by PCoA.

5.5). When forced to find two clusters for data sets SD-DS2 and SD-DS5, the algorithm formed one cluster containing 99 of the points and a second containing only one. A similar pattern was observed for SD-DS10. There was no cut-value for which five clusters could be obtained from SD-DS11.

Discussion

The results of this experiment suggest that the small fixed increment δ added to the distance matrices to make PCoA practical also distorts the data so that the clustering can't pick out clear patterns. One problem was that smallest value for δ that closed all the triangles was frequently many times the original minimum distance. In one example distances in the original matrix ranged from 4 to 144 and the required δ equaled 180. This hypothesis was checked by running the same experiment on a distance matrix derived from SD-DS2 using the outline distance measure which automatically satisfied the triangle inequality. The multi-clustering algorithm was then able to correctly partition the data into two clusters. Going straight from the distance data to the multi-clustering has the advantage of cutting out the computational cost of producing the non-linear projections. This suggests it is worthwhile to explore alternative adjustments to the depth labeling distance matrices which don't distort the data so badly.

CHAPTER 6. An Application to Novel Data, Conclusions, and Future Work

6.1 Introduction

This chapter looks at an application of the RNA motif search pipeline to a large, previously unexplored, biological data set. It also contains general conclusions about the pipeline and a discussion of potential future directions for the research.

6.2 Application to a Novel Data Set

One of the primary goals of this research is to develop a tool for exploring sets of RNA sequences for patterns. This section applies the RNA motif search pipeline to a set of human immunodeficiency virus-1 (HIV-1) sequences. Each of the sequences contains a putative crossover point (COP) at which recombination of two viral types occurred. The RNA motif search pipeline is applied to a set of RNA sequences drawn from around and including each COP with the hope of finding patterns which may lead to information about the mechanism of recombination in HIV.

A large number of HIV-1 sequences were analyzed by Karin Dorman using a dual change-point model for detecting recombination ([54]) and eighty-two putative crossover points were located. Upper and lower bounds for the location of the crossover point were computed at 95% confidence intervals.

Pipeline Implementation Details

Eighty-two RNA sequences were analyzed. Only the regions around the COP points were extracted and explored in this study. From each sequence the region from fifty bases before the lower bound to fifty bases after the upper bound was extracted. Two different brick lengths and increment sizes were tested. With a brick length of 75 and an increment size of 25 this created 325 RNA bricks. A brick length of 100 and an increment size of 50 generated 132 RNA bricks. The brick length and increment size as well as the amount of sequence to extract around each COP were chosen to provide a tractable

Distance Matrix	Brick Length	Increment Size	Distance Measure
HD-DS1	75	25	Depth labeling
HD-DS2	75	25	Tree edit
HD-DS3	100	50	Depth labeling
HD-DS4	100	50	Tree edit

Table 6.1 The four HIV-1 RNA sequence based data sets.

number of bricks for use with MFold and the depth labeled distance. By substituting in different modules this search can be expanded and sped up substantially.

The bricks were folded with Mfold 3.2. Only the optimal folds were saved in order to obtain a computationally tractable set of sequences. Distances were calculated using both the depth annotated distance and the tree edit distance creating a total of four distance matrices (See Table 6.1). For the depth annotated distance a window length of 50 was used. Thirty projections were generated using NLP for each distance matrix.

Results and Discussion

The least RMS error projections for HD-DS1 and HD-DS3 are shown in Figure 6.1 and Figure 6.2. There are a number of possible reasons for the lack of strong clusters ranging from inaccurate folding of the sequences and a incorrectly tuned distance measure to the possibility that there simply aren't any strong motifs in the region examined. In the case of all of these problems, with the exception of the last, the modular nature of the pipeline will allow us to address them and improve the search subsequently.

6.3 Conclusions and Discussion about the RNA Motif Search Pipeline

The experiments detailed in the previous Chapters and the associated published works show that this modular pipeline has a good deal of potential. It is able to easily find, display, and cluster constructed patterns that appear in synthetic data. The pipeline also has some success on doing the same for biological data sets when given the correct foldings and tuned properly. Although, as was shown in the previous section, many more parameter studies are needed to determine effective settings for use in analyzing real data. Experience suggests these will probably vary from data set to data set.

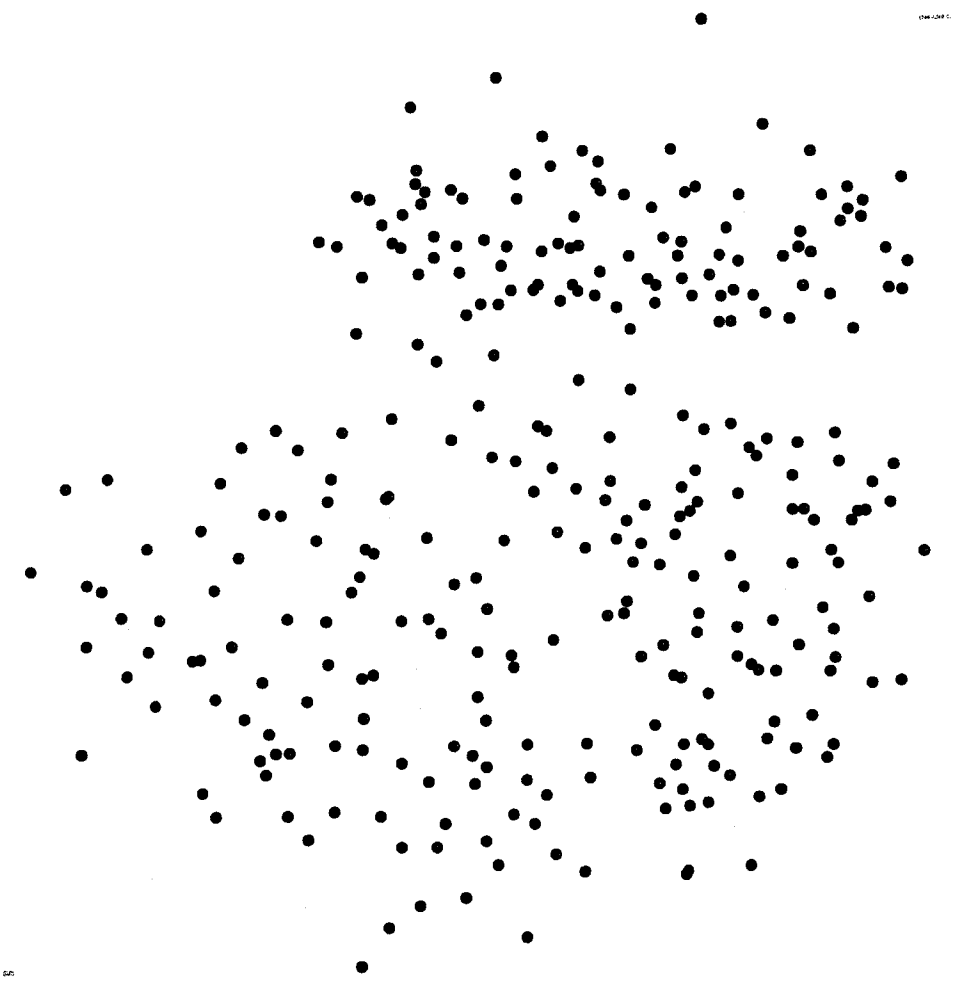

Figure 6.1 The least RMS error projection out of 30 projections for HD-DS1.

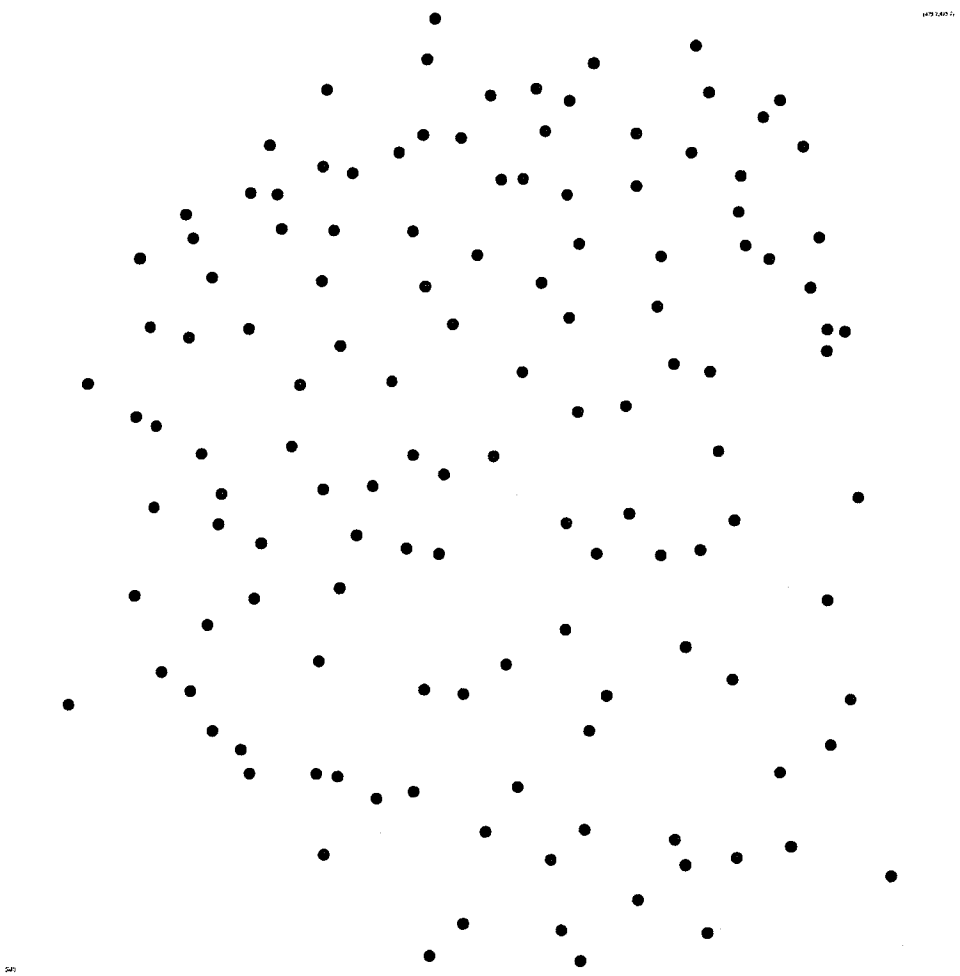

Figure 6.2 The least RMS error projection out of 30 projections for HD-DS3.

6.4 Future Work

One of the main advantages of this pipeline is it's modular nature. This allows the pipeline to constantly adapt and improve. One natural direction to pursue is the inclusion of other motif search methods in the pipeline. The sequence compatibility methods discussed in Chapter 2 ([15, 34, 35]) could easily be adapted to replace modules M3 and M4 of the pipeline.

Many of the most accurate RNA folding methods and distance measures have a high computational cost the pipeline could easily be restructured to use low accuracy low computational cost methods on a large data set and then high accuracy high cost methods on promising subsets.

The output of the motif search pipeline can be used to select small "interesting" subsets of large data sets. These could serve as input to the algorithms developed by Fogel and Nam. Clusters could be examined to determine general structural patterns to be used as input to Fogel's algorithm. For Nam's algorithm clustering could be used to divide the data into positive and negative data sets.

RNA Folding

While the current generation of folding techniques has a relatively low accuracy, 80%, for even small sequences, < 100 bases, ([1]) improved folding techniques are being developed regularly([7]). These include evolutionary computation based methods ([55, 56]) and sequence comparison based methods ([34, 57]). Many of these techniques have the additional benefit of being as fast or faster than current methods. Several of them also correct for the main problem with current methods, that the minimum free energy structure is not the biologically correct structure.

Distance Measures

The distance measures introduced in this thesis are only a few of many. Other distance measures include variations of the tree edit metric ([58, 59]) and the mountain metrics ([44]).

Additionally both of the novel distance measures developed in this thesis could be refined to provide a more accurate representation of distances between RNAs. The outline distance measure can be modified to include a sliding window like the depth labeling distance. This would increase the computational cost, but reduce the bias caused by artifacts in the folding process. Another refinement to the outline distance measure would be to develop or incorporate a standardized outline drawing method that works directly from a list of paired bases. This would prevent identical structures which were drawn by different algorithms from having distinct outlines.

Data Analysis

The data analysis portion of the pipeline also has a great deal of room to grow. In addition to PCoA other methods such as Principle Components Analysis could be used to reduce the dimensionality of the space and make it easier to examine.

Non-linear projection

The non-linear projection algorithm, while requiring less calculations than MDS, is still relatively slow. A parameter study of the algorithm would be useful to determine the smallest population size necessary, the point at which evolution can be halted, and whether tournament selection or another type of EA would give the best results.

Filtering

Another method for improving the readability of the projections would be to look at post-processing. One of the consistent behaviors observed in this thesis is the tendency for random sequences or unrelated sequences to scatter relatively evenly throughout the non-linear projections. By applying a filter which removes isolated points the projections can be cleaned up. One approach would be to remove any point with less than n neighbors with a radius r. This processing could be done on either the non-linear projection or the original distance data matrix.

APPENDIX A. Source Code

A.1 Bricking Algorithm

```
////////////////////////////////////////////////////////////
//                                                        //
// Fasta.cpp                                              //
// Date: 3-26-06                                          //
// Author(s): Justin Schonfeld                            //
// Description: Takes a set of RNA sequences and breaks them //
//              into bricks of a fixed size.              //
//                                                        //
////////////////////////////////////////////////////////////

#include<stdio.h>
#include<math.h>
#include<list>

using namespace std;

#define IFN "forj_75.fasta"
#define MFN "mfscript.mf"
#define BFN "s2input"
#define LBLEN 200
#define SQLEN 4000
#define BRKSZ 100 // Should be even
#define INC 50
#define NBRK 10

struct Sequence {
```

```
  char lbl[LBLEN];
  char seq[SQLEN];
};

////////////////////////////////////////////////////////////////
//                                                            //
// FUNCTIONS                                                  //
//                                                            //
////////////////////////////////////////////////////////////////
bool readFasta(list<Sequence> &seqs){
  // BEGIN declare local variables //
  int i;
  char c;
  FILE *in;
  Sequence aseq;
  list<Sequence>::iterator iseq;
  // END   declare local variables //

  // BEGIN code //
  if ((in = fopen(IFN, "r")) == NULL) {
    printf("ERROR! Target input file could not be opened.\n");
    return false;
  }

  for (i = 0; i < LBLEN; i++)
    aseq.lbl[i] = '\0';
  for (i = 0; i < SQLEN; i++)
    aseq.seq[i] = '\0';

  c='X';
  while (c != EOF) {
    // run until you hit a carrot '>'
    if (c == '>') {
      seqs.push_back(aseq);
      iseq=seqs.end();
```

```
        iseq--;
        // read in the first line
        i = 0;
        while (((c = fgetc(in)) != EOF) && c != '\n') {
         (*iseq).lbl[i] = c;
        // If i exceeeds LBLEN last character is
        // continuously replaced
        if (i < LBLEN - 1)
          i++;
        }
        // add the string terminator charachter
        (*iseq).lbl[i] = '\0';
        // read in the second line 'SEQUENCE'
        i = 0;
        while (((c = fgetc(in)) != EOF) && c != '>' ) {
        (*iseq).seq[i] = c;
        // If sequence length exceeeds SEQLEN last charachter is
        // continuously replaced
        if (i < SQLEN - 1 && c!='\n')
          i++;
        }
        // add the string terminator charachter
        (*iseq).seq[i] = '\0';
      }
      if (c!='>')
        c = fgetc(in);
    }

    fclose(in);

    return true;
    // END   code //
}

int main(void) {
```

```
// BEGIN declare local variables //
int i,j,k,len;
int brk;
char bname[100];
list<Sequence> seqs;
list<Sequence>::iterator iseq;
FILE *outb;
FILE *outm;
FILE *binl;
// END   declare local variables //

// BEGIN code //
// read in the fasta sequences
readFasta(seqs);

if ((outm = fopen(MFN, "w")) == NULL) {
  printf("ERROR! Mfold macro file could not be opened.\n");
  return false;
}
if ((binl = fopen(BFN, "w")) == NULL) {
  printf("ERROR! Brick list file could not be opened.\n");
  return false;
}

//  inc=round((double) BRKSZ/ 2.0);
for (iseq=seqs.begin(),k=0;iseq!=seqs.end();iseq++,k++) {
  brk=0;
  i=0;
  len=strlen((*iseq).seq);
  printf("LEN: %d\n",len);
  while ((i+BRKSZ)<len && brk<NBRK) {
    sprintf(bname,"brick_%.3d_%.5d_%.5d",k,i,i+BRKSZ);
    fprintf(outm,"mfold SEQ='%s.fasta' AUX='%s.fasta.aux'\n",bname,bname);
    fprintf(binl,"%s %3d %3d\n",bname,k,brk);
    brk++;
```

```
      if ((outb = fopen(strcat(bname,".fasta"), "w")) == NULL) {
        printf("ERROR! Brick output file could not be opened.\n");
        return false;
      }
      fprintf(outb,">%s\n",(*iseq).lbl);
      printf("%s\n",(*iseq).lbl);
      for (j=i;(j<len)&&(j<(i+BRKSZ));j++) {
        fprintf(outb,"%c",(*iseq).seq[j]);
        if (((j%60)==0)&&(j!=0))
          fprintf(outb,"\n");
        printf("%c",(*iseq).seq[j]);
      }
      fprintf(outb,"\n");
      printf("\n\n");
      fclose(outb);
      i+=INC;
    }
  }

  fclose(outm);
  fclose(binl);

  return 0;
  // END   code //
}
```

A.2 Depth Labeling Algorithm

```
////////////////////////////////////////////////////////////
//                                                        //
// Fasta2.cpp                                             //
// Date: 3-26-06                                          //
// Author(s): Justin Schonfeld                            //
// Description: Label a set of RNA sequences with structural //
//              information.                              //
```

```
//                                                              //
//////////////////////////////////////////////////////////////////

#include<stdio.h>
#include<math.h>
#include<list>

using namespace std;

#define IFN "s2input"
#define OFN "output.lab"
#define OFNB "bracket.out"
#define EOFN "energies.out"
#define BRKSZ 100

struct Sequence {
  char seq[BRKSZ];
  int  tnum[BRKSZ]; //loop == 0, stem == 1
  int  dnum[BRKSZ];
  int  snum[BRKSZ];
};



int main(void) {
 // BEGIN declare local variables //
 int  i,j,nf;
 int  ia,ib,ic,id,ie;
 int  ibrk,iseq;
 int  dnum;   //depth number
 int  snum;   //stem number
 int  instem;
 int  pspv;   // previous stem pair value
 int  inpoints;
 char c;
```

```
 char ca;
 char infname[100];
 char ifname[100];
 char ofname[100];
 char numx[20];
 char numy[20];
 char line[500];
 double eng;
 FILE *infl;  // input file name list
 FILE *in;
 FILE *out;
 FILE *outb;
 FILE *outx;
 FILE *eout;
 Sequence aseq;
 // END   declare local variables //

 // BEGIN code //
 if ((infl = fopen(IFN, "r")) == NULL) {
   printf("ERROR! List of input file names could not be opened.\n");
   return -1;
 }
 if ((out = fopen(OFN, "w")) == NULL) {
   printf("ERROR! cannot open file for writing.\n");
   return -1;
 }
 if ((outb = fopen(OFNB, "w")) == NULL) {
   printf("ERROR! cannot open output file 2 for writing.\n");
   return -1;
 }
 if ((eout = fopen(EOFN, "w")) == NULL) {
   printf("ERROR! cannot open output file 3 for writing.\n");
   return -1;
 }
 fscanf(infl,"%d",&nf);
```

```
for (i=0;i<nf;i++) {
  fscanf(infl,"%s %d %d",infname,&iseq,&ibrk);
   sprintf(ifname,"%s_1.ps",infname);
  sprintf(ofname,"sbr%03d%02d.xy",iseq,ibrk);
  if ((in = fopen(ifname,"r"))==NULL) {
    printf("ERROR! Input brick file '%s' for ps.\n",ifname);
    return -1;
  }
  if ((outx = fopen(ofname, "w")) == NULL) {
    printf("ERROR! cannot open output file 2 for writing.\n");
    return -1;
  }
  inpoints=0;
  while(fgets(line,500,in)!=NULL) {
    if ((inpoints==2) && (strstr(line,"setlinewidth")!=NULL))
      break;
    else if (inpoints==2) {
      sscanf(line,"%s %s",numx,numy);
      fprintf(outx,"%s %s\n",numx,numy);
      fgets(line,500,in);
      fgets(line,500,in);
    }
    if (strstr(line,"setlinewidth")!=NULL) {
      inpoints++;
    }
  }
  fclose(outx);
  fclose(in);

  if ((in = fopen(strcat(infname,".ct"),"r")) == NULL) {
    printf("ERROR! Input brick file '%s' could for ct.\n",infname);
    return -1;
  }
  for (j=0;j<BRKSZ;j++) {
    aseq.seq[j]='*';
```

```
  aseq.tnum[j]=-1;
  aseq.dnum[j]=-1;
  aseq.snum[j]=-3;
}
dnum=0;
snum=0;
pspv=-1;
instem=0;
// skip the first line
c='x';
while (c!='=')
  c=fgetc(in);
fscanf(in,"%lf",&eng);
fprintf(eout,"%d %10.8f\n",i,eng);
while (c!='\n')
  c=fgetc(in);

//    fprintf(outb,">");
// for each additional line
for (j=0;j<BRKSZ;j++) {
  fscanf(in,"%d %c %d %d %d %d",&ia,&ca,&ib,&ic,&id,&ie);
  printf("%3d %c %3d %3d %3d %3d\n",ia,ca,ib,ic,id,ie);
  if (id>0) {
    if (id<ia)
      fprintf(outb,")");
    else
      fprintf(outb,"(");
  } else fprintf(outb,".");
  aseq.seq[j]=ca;
  // if we are in a stem
  if (id>0) {
    // label the type as stem
    aseq.tnum[j]=1;

    // are we just starting the stem
```

```
        if (instem==0) {
          instem=1; // set instem
          pspv=id;
          if ((j+1)<id) { // new stem
            dnum++;
            if (aseq.snum[j]==-3&&aseq.snum[id-1]==-3) {
              snum++;
              printf("new stem:%2d  %3d\n",j,snum);
              aseq.snum[j]=snum;
              aseq.snum[id-1]=snum;
    }
            aseq.dnum[j]=dnum;
  }
          else {          // old stem
            dnum--;
            aseq.dnum[j]=dnum;
          }
}
        else { // continuing in a stem
          // check for stem continuity
          if (abs(id-pspv) > 1) {  // stem discontinuous
            if (id<j&&j<pspv) { // pseudoknot discontinuity
              aseq.dnum[j]=dnum;
            }
            else if (pspv<j&&j<id) {
              if (aseq.snum[j]==-3&&aseq.snum[id-1]==-3) {
                snum++;
                printf("crisscross:%2d  %3d\n",j,snum);
                aseq.snum[j]=snum;
                aseq.snum[id-1]=snum;
      }
              aseq.dnum[j]=dnum;
    }
          else { // bulge discontinuity
            if ((j+1)<id) { // new stem
```

```
                if (aseq.snum[j]==-3&&aseq.snum[id-1]==-3) {
                  snum++;
                  printf("bulge:%2d  %3d\n",j,snum);
                  aseq.snum[j]=snum;
                  aseq.snum[id-1]=snum;
}
                dnum+=2;
      }
              else {
                dnum-=2;
      }
              aseq.dnum[j]=dnum;
            }
            pspv=id;
  }
          else if (abs(id-pspv) == 1) { // stem continuous
            aseq.dnum[j]=dnum;
            if (aseq.snum[j]==-3&&aseq.snum[id-1]==-3) {
              aseq.snum[j]=snum;
              aseq.snum[id-1]=snum;
    }
            pspv=id;
          }
          else
            printf("ERROR!  STEM DISCONTINUITY DETECTED!!\n");
}
      }
      else {  // not in a stem
        aseq.tnum[j]=0;
        if (instem==1) { // just left a stem
          instem=0; // reset instem
          if (j<pspv) { // leaving new stem
            dnum++;
          }
          else {
```

```
            dnum--;
   }
          aseq.dnum[j]=dnum;
 }
        else { // haven't just left a stem
          aseq.dnum[j]=dnum;
 }
      }
    }
    fclose(in);
    for(j=0;j<BRKSZ;j++) {
      fprintf(out,"%3d %3d %3d %c %2d %3d %3d\n",iseq,ibrk,j,aseq.seq[j],
              aseq.tnum[j],aseq.dnum[j],aseq.snum[j]);
    }
    fprintf(outb,"\n");
  }
  fclose(out);
  fclose(outb);
  fclose(eout);
  fclose(infl);
  return 0;
  // END   code //
}
```

A.3 Depth Labeling Distance Computation Algorithm

```
////////////////////////////////////////////////////////////////
//                                                            //
// Distance.cpp                                               //
// Date: 3-26-06                                              //
// Author(s): Justin Schonfeld                                //
//            Dan Ashlock                                     //
// Description: Calculate the depth label distance between a  //
//              set of RNA sequences.                         //
//                                                            //
```

```
////////////////////////////////////////////////////////////////

//Include the relevant includes
#include<cstdlib>
#include<ctime>
#include<iostream>
#include<fstream>
#include<cmath>
#include<cstdio>
#include<cstring>
#include<cctype>

//Characters:   CL GL AL TL CS GS AS TS
//               0  1  2  3  4  5  6  7

using namespace std;

#define len 100
#define target 50
#define depthd 5
#define gap 5
#define bricks 213

//cost matric for basic character types
int cost[8][8]={
  {0,3,3,3,8,8,8,8},
  {3,0,3,3,8,8,8,8},
  {3,3,0,3,8,8,8,8},
  {3,3,3,0,8,8,8,8},
  {8,8,8,8,0,1,1,1},
  {8,8,8,8,1,0,1,1},
  {8,8,8,8,1,1,0,1},
  {8,8,8,8,1,1,1,0}
};
```

```
int seq[bricks][len];  //sequence data
int dpt[bricks][len];  //sequence depth data
int dpm[len+1][len+1]; //dynamic programming matix
int sn[bricks];        //sequence index number
int bn[bricks];        //brick number

void readData(char *fn); //read in the data in Justin's format

int ccost(int a,int b,int da,int db); //compute the cost for two characters

int dist(int *a,int *b,int *da,int *db,int ofsa,int ofsb);

int mindist(int *a,int *b,int *da,int *db);

main(){

int i,j,ofs,min,max;
int D[bricks][bricks];
int use[bricks];
fstream aus;
int Q;

 readData("output.lab");

 Q=0;
 for(i=0;i<bricks;i++){
   use[i]=(bn[i]<20);
   if(use[i])Q++;
 }
 for(i=0;i<bricks;i++)D[i][i]=0;
 for(i=0;i<bricks-1;i++)if(use[i]){
   min=20*len;
   max=0;
   for(j=i+1;j<bricks;j++)if(use[j]){
     D[i][j]=mindist(seq[i],seq[j],dpt[i],dpt[j]);
```

```
      D[j][i]=D[i][j];
      if((D[i][j]<min)&&(i!=j))min=D[i][j];
      if(D[i][j]>max)max=D[i][j];
    }
    cout << i << " " << min << " " << max << endl;
  }

  aus.open("d27.web",ios::out);
  aus << "%Fitness web display file" << endl;
  aus << "%Lines beginning with a % are ignored" << endl;
  aus << "%" << endl;
  aus << "%Number of optima" << endl;
  aus << Q << endl;
  aus << "%dimension of space" << endl;
  aus << 3 << endl;
  aus << "%optima positions if known" << endl;
  for(i=0;i<Q;i++)aus << "0 0 0" << endl;
  aus << "%lines with fitness and frequency" << endl;
  for(i=0;i<bricks;i++) {
    if(sn[i]==10) aus << sn[i] << " " << "200" << endl;
    else if(sn[i]==11) aus << sn[i] << " " << "250" << endl;
    else if(use[i])aus << sn[i] << " " << bn[i]*20+10 << endl;
  }
  aus << "%now the distance matrix seperated by spaces." << endl;
  for(i=0;i<bricks;i++)if(use[i]){
    aus << D[i][0];
    for(j=1;j<bricks;j++)if(use[j]){
      aus << " " << D[i][j];
    }
    aus << endl;
  }


}
```

```
void readData(char *fn){

int i,j,k,d;
char buf[1000];
fstream inp;

  inp.open(fn,ios::in);

  cout << "Reading " << fn << endl;

  for(i=0;i<bricks;i++){
    for(j=0;j<len;j++){
      inp.getline(buf,999);
      k=0;
      //find and compute the sequence number
      while(!isdigit(buf[k]))k++;
      d=atoi(buf+k);
      sn[i]=d;
      cout<<"sn,bn: "<<d<<", ";
      //find and compute the brick number
      while(isdigit(buf[k]))k++;
      while(!isdigit(buf[k]))k++;
      d=atoi(buf+k);
      cout<<d<<endl;
      bn[i]=d;
      //get the character index
      while(isdigit(buf[k]))k++;
      while(!isdigit(buf[k]))k++;
      d=atoi(buf+k);
      //double check the character index
      if(d!=j)cout << i << " Alert " <<d << " " << j << endl;
      //find the base
      while(buf[k]!=' ')k++;
      while(buf[k]==' ')k++;
```

```
      switch(buf[k]){
      case 'C' : seq[i][j]=0;
break;
      case 'G' : seq[i][j]=1;
break;
      case 'A' : seq[i][j]=2;
break;
      case 'T' : seq[i][j]=3;
break;
      }
      //now find the type 1=stem 0=loop
      k++;
      while(!isdigit(buf[k]))k++;
      d=atoi(buf+k);
      seq[i][j]+=d*4;  //add in stem factor
      //cout << j << " " << seq[i][j] << " ";
      //now find the depth
      k++;
      while(!isdigit(buf[k]))k++;
      dpt[i][j]=atoi(buf+k);
      //cout << dpt[i][j] << endl;
    }
  }

  cout << "Data read" << endl;

  inp.close();

}

int ccost(int a,int b,int da,int db){//compute the cost for two characters

int r;

  r=da-db;
```

```
    if(r<0)r=-r;
    return(r*depthd+cost[a][b]);

 }

 int min(int a,int b){

    if(a<b)return(a);else return(b);

 }

 int dist(int *a,int *b,int *da,int *db,int ofsa,int ofsb){

 int i,j,aa,bb,cc;

   for(i=0;i<target+1;i++)dpm[i][0]=dpm[0][i]=i*gap;
   for(i=1;i<target+1;i++)for(j=1;j<len+1;j++){
     aa=dpm[i][j-1]+gap;
     bb=dpm[i-1][j]+gap;
     cc=dpm[i-1][j-1]+ccost(a[i+ofsa],b[j+ofsb],da[i+ofsa],db[j+ofsb]);
     dpm[i][j]=min(min(aa,bb),cc);
   }

   return(dpm[target][target]);

}

int mindist(int *a,int *b,int *da,int *db){

int i,j,d,l;


  d=-1;
  for(i=0;i<len-target;i++)for(j=0;j<len-target;j++){
    l=dist(a,b,da,db,i,j);
```

```
    if(l!=0){
      if((d==-1)||(l<d))d=l;
    }
  }
  return(d);
}
```

A.4 Non-linear Projection Algorithm

```
////////////////////////////////////////////////////////////////
//                                                            //
// Drawer.cpp                                                 //
// Date: 3-26-06                                              //
// Author(s): Justin Schonfeld                                //
//            Dan Ashlock                                     //
// Description: Draws a non-linear projection for a set of    //
//              high dimensional data.                        //
//                                                            //
////////////////////////////////////////////////////////////////

//Include the relevant includes
#include<cstdlib>
#include<ctime>
#include<iostream>
#include<fstream>
#include <cmath>
#include <cstdio>
#include <cstring>

using namespace std;

#include "ps.h"
#include "pbm.h"

//number of members in the population
```

```
#define popsize 100
//tournament size for selection
#define tsize 7
//population initialization technique
//0=random 1=random projection
#define PIT 0
//Mask type 0=none, 1=reciprocal, 1=kneighbor
#define MASK 0
//number of neighbors to use for nearest neighbor mask
#define K 3
//number of runs to perform
#define runs 5
//number of mating events per run
#define mevs 100000
//reporting interval, number of mating intervals between report calls
#define RI 100
//Mutation size as a fraction of scale
#define msize 0.2
//Make pictures?
#define pictures 1
//Picture size
#define Pz 2000
//circle min and max
#define cmin 12
#define cmax 12
//tolerance for no stress
#define tol 0.05
//fudge factor
#define fudge 1.2
//lines?
#define drawlines 0

int nopt;            //number +of optima
int dim;             //dimension of the space
int *freq;           //frequency of the optima
```

```
double **psn;          //position of the optima
double *fito;          //fitness of the optima
double **pop;          //population
double **dis;          //distance matrix
double **mask;         //mask array to enable different types of fitness
double fit[popsize]; //population member fitness
double scale;          //rough scale of distance
double hist[runs];       //Ad hoc histogram

void readData(char *fn); //read in a .web data file

void initpop();  //initialize a population

int select();  //tournament select an agent

double fitness(double *layout); //compute the masked fitnes

void matingevent(); //perform a mating event

void report(ostream &aus,int mev);

void savebest(ostream &aus,int run);

main(int argc,char **argv){

int run,mev;
fstream rpt,bes;
char fn[60];
int i,j;

 if(argc==1){
   cout << "web <infile>" << endl;
   return(0);
 }
```

```
    srand48(33120781); //seed the random number generator

    readData(argv[1]);

    cout << "Read data " << endl;

    bes.open("best.lyt",ios::out);

    for(i=0;i<runs;i++)hist[i]=0.0;

    for(run=0;run<runs;run++){
      cout << "Run " << run << endl;
      initpop();
      sprintf(fn,"run%d.dat",run);
      rpt.open(fn,ios::out);
      for(mev=0;mev<mevs;mev++){
        matingevent();
        if((mev+1)%RI==0)report(rpt,mev);
      }
      rpt.close();
      savebest(bes,run);
    }

    bes.close();



    psDoc H("histo.eps",0,0,640,480);
    H.histPS(hist,runs,27.0,30.0,0.05,50);


}

void initpop(){//initialize a population
```

```
 int i,j;

   switch(PIT){
   case 0://totally random initialization
          for(i=0;i<popsize;i++)
          for(j=0;j<2*nopt;j++){
            pop[i][j]=drand48()*scale;
 }
  }
  for(i=0;i<popsize;i++){
    fit[i]=fitness(pop[i]);
    //cout << fit[i] << " ";
  }
  //cout << endl;
}

void nextline(char *line,istream &inp,int len){


  do {
    if(inp.eof()){
      strcpy(line,"");
      return;
    }
    inp.getline(line,len);
  }while(line[0]=='%');

}

void readData(char *fn){

fstream inp;
char buf[10000];
int i,j,k;
```

```
  inp.open(fn,ios::in);
  nextline(buf,inp,9999);
  nopt=atoi(buf);
  nextline(buf,inp,9999);
  dim=atoi(buf);
  freq=new int[nopt];
  fito=new double[nopt];
  psn=new double*[nopt];
  dis=new double*[nopt];
  mask=new double*[nopt];
  for(i=0;i<nopt;i++){
    psn[i]=new double[dim];
    dis[i]=new double[nopt];
    mask[i]=new double[nopt];
  }
  for(i=0;i<nopt;i++){
    nextline(buf,inp,9999);
    k=0;
    for(j=0;j<dim;j++){
      while(buf[k]==' ')k++;
      psn[i][j]=atof(buf+k);
      if(j==dim-1)break;
      while(buf[k]!=' ')k++;
    }
  }
  for(i=0;i<nopt;i++){
    nextline(buf,inp,9999);
    fito[i]=atof(buf);
    k=0;
    while(buf[k]!=' ')k++;
    freq[i]=atoi(buf+k);
  }

  scale=0.0;
  for(i=0;i<nopt;i++){
```

```
    nextline(buf,inp,9999);
    k=0;
    for(j=0;j<nopt;j++){
      while(buf[k]==' ')k++;
      dis[i][j]=atof(buf+k);
      if(dis[i][j]>scale)scale=dis[i][j];
      if(j==nopt-1)break;
      while(buf[k]!=' ')k++;
    }
  }

  scale*=fudge;

  //create the population array
  pop=new double*[popsize];
  for(i=0;i<popsize;i++)pop[i]=new double[2*nopt];

  //create the mask array
  for(i=0;i<nopt;i++)for(j=0;j<nopt;j++)mask[i][j]=1.0;

}

int select(int &die){//tournament select an agent

int i,rs,dx;
double fitb,fitw;

  die=dx=(lrand48()%popsize);
  fitw=fitb=fit[dx];
  for(i=1;(i<tsize)||(die==dx);i++){
    rs=lrand48()%popsize;
    if(fit[rs]<fitb){
      dx=rs;
      fitb=fit[rs];
    }
```

```
    if(fit[rs]>fitw){
      die=rs;
      fitw=fit[rs];
    }
  }
  return(dx);
}

double fitness(double *layout){//compute the masked fitnes

int i,j;
double d,dx,dy,delta,accu;


  accu=0.0;
  for(i=0;i<nopt-1;i++)for(j=i+1;j<nopt;j++){
    dx=layout[2*i]-layout[2*j];
    dy=layout[2*i+1]-layout[2*j+1];
    d=sqrt(dx*dx+dy*dy);
    delta=d-dis[i][j];
    accu+=delta*delta*mask[i][j];
  }
  return(sqrt(2*accu/nopt/(nopt-1)));
}

double Gauss01(){

  return(cos(2*M_PI*drand48())*sqrt(-2*log(drand48())));

}

void matingevent(){//perform a mating event

int p1,p2,d1,d2,c1,c2,sw,i,psn;
double dx,dy,th,rd;
```

```
//select distinct parents and clild slots
p1=select(d1);
do {
  p2=select(d2);
} while((p1==p2)||(d1==d2)||(p1==d1)||(p1==d2)||(p2==d1)||(p2==d2));

//pick crossover points
c1=lrand48()%(2*nopt);
c2=lrand48()%(2*nopt);
//order the crossover points
if(c1>c2){sw=c1;c1=c2;c2=sw;}

//perform the crossover
for(i=0;i<c1;i++){
  pop[d1][i]=pop[p1][i];
  pop[d2][i]=pop[p2][i];
}
for(i=c1;i<c2;i++){
  pop[d1][i]=pop[p2][i];
  pop[d2][i]=pop[p1][i];
}
for(i=c2;i<2*nopt;i++){
  pop[d1][i]=pop[p1][i];
  pop[d2][i]=pop[p2][i];
}

//Gaussian radial point mutation on first child
psn=lrand48()%nopt;
th=drand48()*2*M_PI;
rd=scale*msize*Gauss01();
pop[d1][2*psn]+=rd*cos(th);
pop[d1][2*psn+1]+=rd*sin(th);

//Gaussuan radiual point mutatio on second child
```

```
    psn=lrand48()%nopt;
    th=drand48()*2*M_PI;
    rd=scale*msize*Gauss01();
    pop[d2][2*psn]+=rd*cos(th);
    pop[d2][2*psn+1]+=rd*sin(th);

    //update the fitness information
    fit[d1]=fitness(pop[d1]);
    fit[d2]=fitness(pop[d2]);

 }

 void report(ostream &aus,int mev){

 double mu,sg,bs,ci;
 int i;

   mu=sg=0.0;
   bs=fit[0];
   for(i=0;i<popsize;i++){
     mu+=fit[i];
     sg+=(fit[i]*fit[i]);
     if(fit[i]<bs)bs=fit[i];
   }
   mu/=((double)popsize);
   sg/=((double)popsize);
   sg-=(mu*mu);
   sg=sqrt(sg);
   ci=1.96*sg/sqrt((double)popsize);
   //cout << mev+1 << " " << mu << " " << sg << " " << bs << " " << ci << endl;
   aus<< " " << mu << " " << sg << " " << bs << " " << ci << endl;

}

void savebest(ostream &aus,int run){
```

```
int i,bdx;
double bft;
char fn[60];

bft=fit[0];
bdx=0;
for(i=1;i<popsize;i++){
  if(fit[i]<bft){
    bdx=i;
    bft=fit[i];
  }
}

i=((int)(100.0*bft-50.0));
hist[run]=bft;

for(i=0;i<nopt;i++){
  aus << pop[bdx][2*i] << " " << pop[bdx][2*i+1] << endl;
}
aus << bft << endl;

if(!pictures)return;

sprintf(fn,"Fitness%d.eps",run);

psDoc Q(fn,0,0,Pz,Pz);
double dx,dy,d,xmin,xmax,ymin,ymax,scale;
double faux,fdx,fdy;
double omax,omin;
int j,h,k,hh,kk,dia;

xmax=xmin=pop[bdx][0];
ymin=ymax=pop[bdx][1];
omax=omin=fito[0];
```

```
for(i=0;i<nopt;i++){

  if(fito[i]>omax)omax=fito[i];
  if(fito[i]<omin)omin=fito[i];

  if(pop[bdx][2*i]<xmin)xmin=pop[bdx][2*i];
  if(pop[bdx][2*i]>xmax)xmax=pop[bdx][2*i];
  if(pop[bdx][2*i+1]<ymin)ymin=pop[bdx][2*i+1];
  if(pop[bdx][2*i+1]>ymax)ymax=pop[bdx][2*i+1];

}

dx=xmax-xmin;
dy=ymax-ymin;
if(dx>dy)scale=dx; else scale=dy;

//make the stress lines
if(drawlines)for(i=0;i<nopt-1;i++)for(j=i+1;j<nopt;j++){
  h=((int)((pop[bdx][2*i]-xmin+(scale-dx)/2)/scale*(Pz-2*cmax-10)))+cmax+5;
  k=((int)((pop[bdx][2*i+1]-ymin+(scale-dy)/2)/scale*(Pz-2*cmax-10)))+cmax+5;
  hh=((int)((pop[bdx][2*j]-xmin+(scale-dx)/2)/scale*(Pz-2*cmax-10)))+cmax+5;
  kk=((int)((pop[bdx][2*j+1]-ymin+(scale-dy)/2)/scale*(Pz-2*cmax-10)))+cmax+5;
  fdx=pop[bdx][2*i]-pop[bdx][2*j];
  fdy=pop[bdx][2*i+1]-pop[bdx][2*j+1];
  faux=sqrt(fdx*fdx-fdy*fdy);
  if(faux-dis[i][j]>tol*scale){//stretched
    Q.setgray(75);
    Q.setlinewidth(1);
    Q.moveto(h,k);
    Q.lineto(hh,kk);
    Q.stroke();
  } else if(dis[i][j]-faux>tol*scale){//compressed
    Q.setgray(25);
    Q.setlinewidth(3);
```

```
      Q.moveto(h,k);
      Q.lineto(hh,kk);
      Q.stroke();
    } else {//non-stressed edge
      Q.setgray(50);
      Q.setlinewidth(2);
      Q.moveto(h,k);
      Q.lineto(hh,kk);
      Q.stroke();
    }

  }

  //make the optima
  for(i=0;i<nopt;i++){
    h=((int)((pop[bdx][2*i]-xmin+(scale-dx)/2)/scale*(Pz-2*cmax-10)))+cmax+5;
    k=((int)((pop[bdx][2*i+1]-ymin+(scale-dy)/2)/scale*(Pz-2*cmax-10)))+cmax+5;
    dia=((int)((fito[i]-omin)/(omax-omin)*(cmax-cmin)))+cmin;
    Q.setgray(0);
    Q.FillStar(h,k,dia,20,1);
    sprintf(fn,"%d",i);
    Q.setgray(99);
    Q.sayAT(h-3,k-3,fn);
  }

  Q.setgray(0);
  Q.sayAT(cmax,cmax,"(0,0)");
  sprintf(fn,"(%4.1f,%4.1f)",scale,scale);
  Q.sayAT(Pz-cmax-60,Pz-cmax-10,fn);

sprintf(fn,"Frequency%d.eps",run);
psDoc QQ(fn,0,0,Pz,Pz);

    //make the stress lines
  if(drawlines)for(i=0;i<nopt-1;i++)for(j=i+1;j<nopt;j++){
```

```
    h=((int)((pop[bdx][2*i]-xmin+(scale-dx)/2)/scale*(Pz-2*cmax-10)))+cmax+5;
    k=((int)((pop[bdx][2*i+1]-ymin+(scale-dy)/2)/scale*(Pz-2*cmax-10)))+cmax+5;
    hh=((int)((pop[bdx][2*j]-xmin+(scale-dx)/2)/scale*(Pz-2*cmax-10)))+cmax+5;
    kk=((int)((pop[bdx][2*j+1]-ymin+(scale-dy)/2)/scale*(Pz-2*cmax-10)))+cmax+5;
    fdx=pop[bdx][2*i]-pop[bdx][2*j];
    fdy=pop[bdx][2*i+1]-pop[bdx][2*j+1];
    faux=sqrt(fdx*fdx-fdy*fdy);
    if(faux-dis[i][j]>tol*scale){//stretched
      QQ.setgray(75);
      QQ.setlinewidth(1);
      QQ.moveto(h,k);
      QQ.lineto(hh,kk);
      QQ.stroke();
    } else if(dis[i][j]-faux>tol*scale){//compressed
      QQ.setgray(25);
      QQ.setlinewidth(3);
      QQ.moveto(h,k);
      QQ.lineto(hh,kk);
      QQ.stroke();
    } else {//non-stressed edge
      QQ.setgray(50);
      QQ.setlinewidth(2);
      QQ.moveto(h,k);
      QQ.lineto(hh,kk);
      QQ.stroke();
    }

}

//make the optima
for(i=0;i<nopt;i++){
  h=((int)((pop[bdx][2*i]-xmin+(scale-dx)/2)/scale*(Pz-2*cmax-10)))+cmax+5;
  k=((int)((pop[bdx][2*i+1]-ymin+(scale-dy)/2)/scale*(Pz-2*cmax-10)))+cmax+5;
  //    dia=7+2*((int)(log((double)freq[i])/log(2.0)));
  QQ.setgray(0);
```

```
   //    QQ.FillStar(h,k,dia,20,1);
   if (freq[i]==0)
     QQ.glyph(h,k,freq[i],4);
   else if (freq[i]==1)
     QQ.glyph(h,k,freq[i],4);
   else if (freq[i]==2)
     QQ.glyph(h,k,freq[i],6);
   else if (freq[i]==3)
     QQ.glyph(h,k,4,4);
   //    sprintf(fn,"%d",i);
   QQ.setgray(99);
   // QQ.sayAT(h-3,k-3,fn);
}

QQ.setgray(0);
//  QQ.sayAT(cmax,cmax,"(0,0)");
sprintf(fn,"(%4.1f,%4.1f)",scale,scale);
//  QQ.sayAT(Pz-cmax-60,Pz-cmax-10,fn);

QQ.glyph(Pz-cmax-110,Pz-cmax-30,0,4);
sprintf(fn,"Bricks w/o IRE motif ");
QQ.sayAT(Pz-cmax-100,Pz-cmax-33,fn);
QQ.glyph(Pz-cmax-110,Pz-cmax-45,4,4);
sprintf(fn,"Bricks w IRE motif");
QQ.sayAT(Pz-cmax-100,Pz-cmax-48,fn);
QQ.glyph(Pz-cmax-110,Pz-cmax-60,1,4);
sprintf(fn,"Bricks w HF IRE A");
QQ.sayAT(Pz-cmax-100,Pz-cmax-63,fn);
QQ.glyph(Pz-cmax-110,Pz-cmax-75,2,6);
sprintf(fn,"Bricks w HF IRE B");
QQ.sayAT(Pz-cmax-100,Pz-cmax-78,fn);
//PBM stuff

pbm P(Pz,Pz);
```

```
 //make the optima
  P.clear(255,255,255);
  for(i=0;i<nopt;i++){
    h=((int)((pop[bdx][2*i]-xmin+(scale-dx)/2)/scale*(Pz-2*cmax-10)))+cmax+5;
    k=((int)((pop[bdx][2*i+1]-ymin+(scale-dy)/2)/scale*(Pz-2*cmax-10)))+cmax+5;
    dia=((int)((freq[i]-omin)/(omax-omin)*(cmax-cmin)))+cmin;
    P.SetColHSV(7*freq[i]%360,100,100);
    P.Disk(h,k,7);
  }
  sprintf(fn,"pic%d.ppm",run);
  P.writeXVC(fn);

}
```

BIBLIOGRAPHY

[1] P. G. Higgs, "Rna secondary structure: physical and computational aspects," *Quaterly Reviews of Biophysics*, vol. 33, no. 3, pp. 199–253, 2000.

[2] G. Mauri and G. Pavesi, "Pattern discovery in rna secondary structure using affix trees," *CPM 2003, LNCS 2676*, pp. 278–294, 2003.

[3] C. W. A. Pleij, "Rna pseudoknots," *Current Opinion in Structural Biology*, vol. 4, pp. 337–344, 1994.

[4] M. W. Hentze and L. C. Kuhn, "Molecular control of vertebrate iron metabolism: mrna-based regulatory circuits operated by iron, nitric oxide, and oxidative stress," *Proc. Natl. Acad. Sci. USA*, vol. 93, pp. 8175–8182, 1996.

[5] B. R. Henderson, E. Menotti, C. Bonnard, and L. C. Kuhn, "Optimal sequence and structure of iron-responsive elements," *Journal of Biological Chemistry*, vol. 269, no. 26, pp. 17 481–17 489, 1994.

[6] J. Coffin, S. Hughes, and H. Varmus, *Retroviruses*, first edition ed. Cold Spring Harbor, 1997.

[7] P. P. Gardner and R. Giegerich, "A comprehensive comparison of comparative rna structure prediction approaches," *BMC Bioinformatics*, vol. 5, 2004.

[8] T. J. Macke, D. J. Ecker, R. Gutell, D. Gautheret, D. A. Case, and R. Sampath, "Rnamotif, an rna secondary structure definition and search algorithm," *Nucleic Acids Research*, vol. 29, no. 4735, pp. 4724–4735, 2001.

[9] D. Gautheret and A. Lambert, "Direct rna motif definition and identification from multiple sequence alignments using secondary structure profiles," *Journal of Molecular Biology*, vol. 313, pp. 1003–11, 2001.

[10] S. R. Eddy, "rnabob," St. Louis, Missouri, 1996.

[11] D. Ashlock, *Optimization and modeling with evolutionary computation*, first edition ed. Springer, New York, 2005.

[12] G. B. Fogel, V. W. Porto, D. G. Weekes, D. B. Fogel, R. H. Griffey, J. A. McNeil, E. Lesnik, D. J. Ecker, and R. Sampath, "Discovery of rna structural elements using evolutionary computation," *Nucleic Acids Research*, vol. 30, no. 23, pp. 5310–5317, 2002.

[13] Y. Hu, "Prediction of consensus structural motifs in a family of coregulated rna sequences," *Nucleic Acids Research*, vol. 30, no. 17, 2002.

[14] ——, "GPRM: a genetic programming approach to finding common rna secondary structure elements," *Nucleic Acids Research*, vol. 31, no. 13, 2003.

[15] J. Gorodkin, L. J. Heyer, and G. D. Stormo, "Finding the most significant common sequence and structure motifs in a set of rna sequences," *Nucleic Acids Research*, vol. 25, no. 18, pp. 3724–3732, 1997.

[16] B. W. Matthews, "Comparison of the predicted and observed secondary structure of the t4 phage lysozyme," *Biochim. Biophys. Acta*, vol. 405, pp. 442–451, 1975.

[17] N. J.W., J. J.G., A. Y.S., and Z. B.T., "Two-step genetic programming for optimization of rna common-structure," *Applications of Evolutionary Computing Lecture Notes in Computer Science*, vol. 3005, pp. 73–83, 2004.

[18] P. Linz, *An introduction to formal languages and automata*, second edition ed. D.C. Heath and Company, 1996.

[19] R. Sokal and F. Rohlf, *Biometry*, third edition ed. W.H. Freeman and Company, 1995.

[20] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, second edition ed. MIT Press, 2001.

[21] D. Sankoff, "Simultaneous solution of the rna folding, alignment and protosequence problems," *SIAM Journal of Applied Mathematics*, vol. 45, no. 5, pp. 810–825, 1985.

[22] K. Y. Gorbunov, A. A. Mironov, and V. A. Lyubetsky, "Search for conserved secondary structures of rna," *Molecular Biology*, vol. 37, no. 5, pp. 723–732, 2003.

[23] A. Vitreschak, D. Rodionov, A. Mironov, and M. Gelfand, "Regulation of riboflavin biosynthesis and transport genes in bacteria by transcriptional and translational attenuation," *Nucleic Acids Research*, vol. 30, pp. 3141–3151, 2002.

[24] S. R. Eddy and R. Durbin, "Rna sequence analysis using covariance models," *Nucleic Acids Research*, vol. 22, no. 11, pp. 2079–2088, 1994.

[25] A. Krogh, M. Brown, I. Mian, K. Sjolander, and D. Haussler, *J. Mol. Biol.*, vol. 235, pp. 1501–1531, 1994.

[26] L. R. Rabiner, *Proc. IEEE*, vol. 77, pp. 257–286, 1989.

[27] S. Steinberg, A. Misch, and M. Sprinzl, *Nucleic Acids Res.*, vol. 21, pp. 3011–3015, 1993.

[28] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison, *Biological sequence analysis: Probabilistic models of proteins and nucleic acids*, first edition ed. Cambridge University Press, 1998.

[29] T. Smith and M. Waterman, *J. Mol. Biol.*, vol. 147, pp. 195–197, 1981.

[30] R. Nussinov and A. Jacobson, *Proc. Natl. Acad. Sci. USA*, vol. 77, pp. 6309–6313, 1980.

[31] D. Higgins, A. Bleasby, and R. Fuchs, *Comput. Appl. Biosci.*, vol. 8, pp. 189–191, 1991.

[32] J. Thompson, D. Higgins, and T. Gibson, *Nucleic Acids Research*, vol. 22, pp. 4673–4680, 1994.

[33] I. Hofacker, W. Fontana, P. Stadler, P. Bonhoeffer, M. Tacker, and P. Schuster, *Monatshefte fur Chemie*, vol. 125, pp. 167–188, 1994.

[34] J. Gorodkin, S. L. Stricklin, and G. D. Stormo, "Discovering common stem-loop motifs in unaligned rna sequences," *Nucleic Acids Research*, vol. 29, no. 10, pp. 2135–2144, 2001.

[35] D. H. Mathews and D. H. Turner, "Dynalign: An algorithm for finding the secondary structure common to two rna sequences," *Journal of Molecular Biology*, vol. 317, pp. 191–203, 2002.

[36] M. G. Maaβ, "Linear bidirectional on-line construction of affix trees," *CPM 2000, LNCS 1848*, pp. 320–334, 2000.

[37] M. Zuker, D. Mathews, and D. Turner, "Algorithms and thermodynamics for rna secondary structure prediction: A practical guide." in *RNA Biochemistry and Biotechnology*, J. Barciszewski and B. Clark, Eds. Kluwer Academic Publishers, 1999.

[38] G. Mauri and G. Pavesi, "Algorithms for pattern matching and discovery in rna secondary sturcture," *Theoretical Computer Science*, vol. 335, pp. 29–51, 2005.

[39] H. Lutcke, "Signal recognition particle(srp), a ubiquitous initiator of protein translocation," *European Journal of Biochemistry*, vol. 228, no. 3, pp. 531–550, 1995.

[40] D. Ashlock and J. Schonfeld, "Depth annotation of rna folds for secondary structure motif search," in *Proceeding of the 2005 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology.* Piscataway NJ: IEEE Press, 2005, pp. 38–45.

[41] P. Legendre and L. Legendre, *Numerical Ecology, 2nd English Edition.* New York: Elsevier, 1998.

[42] J. Meidanis and J. C. Setabal, *Introduction to Computational Molecular Biology.* Pacific Grove, CA: Brooks-Cole, 1997.

[43] M. Zuker, "The use of dynamic programming algorithms in rna secondary structure prediction," in *Mathematical Methods for DNA Sequences*, M. Waterman, Ed. Boca Raton: CRC Press, 1989, pp. 159–184.

[44] V. Moulton, M. Zuker, M. Steel, R. Pointon, and D. Penny, "Metrics on rna secondary structures," *Journal of Computational Biology*, vol. 7, pp. 277–292, 2000.

[45] J. Schonfeld and D. Ashlock, "Evaluating distance measures for rna motif search," 2006, accepted to the 2006 Congress on Evolutionary Computation.

[46] D. E. Slice, "Morpheus et al.: software for morphometric research," Stony Brook, New York, 1998.

[47] B. A. Shapiro, "An algorithm for comparing multiple rna secondary structures," *CABIOS*, vol. 4, pp. 381–393, 1988.

[48] B. A. Shapiro and K. Zhang, "Comparing multiple rna secondary structures using tree comparison," *CABIOS*, vol. 6, pp. 309–318, 1990.

[49] G. Syswerda, "A study of reproduction in generational and steady state genetic algorithms." in *Foundations of Genetic Algorithms.* Morgan Kaufmann, 1991, pp. 94–101.

[50] N. Mantel, "The detection of disease clustering and a generalized regression approach," *Cancer Research*, vol. 27, 1967.

[51] D. Ashlock and J. Schonfeld, "Nonlinear projection for the display of high dimensional distance data," in *Proceedings of the 2005 Congress on Evolutionary Computation*, vol. 3. IEEE Press, 2005, pp. 2776–2783.

[52] D. A. Ashlock, E. Y. Kim, and L. Guo, "Multi-clustering: avoiding the natural shape of underlying metrics," in *Proceedings of the 2005 Conference on Artificial Networks in Engineering*, vol. 15. ASME Press, 2005, pp. 453–461.

[53] D. B. West, *Introduction to Graph Theory.* Upper Saddle River, NJ 07458: Prentice Hall, 1996.

[54] V. N. Minin, K. S. Dorman, F. Fang, and M. A. Suchard, "Dual multiple change-point model leads to more accurate recombination detection," *Bioinformatics*, vol. 21, no. 13, pp. 3034–3042, 2005.

[55] A. Deschenes, K. C. Wiese, and J. Poonian, "Comparison of dynamic programming and evolutionary algorithms for rna secondary structure prediction," in *Proceedings of the 2004 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology.* IEEE Press, 2004, pp. 214–222.

[56] A. Deschenes and K. C. Wiese, "Using stacking-energies (inn and inn-hb) for improving the accuracy of rna secondary structure prediction with an evolutionary algorithm - a comparison to known structures," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2004).* IEEE Press, 2004, pp. 598–606.

[57] D. K. Chiu and T. Kolodziejczak, "Inferring consensus structure from nucleic acid sequences," *Computational Applied Bioscience*, vol. 7, pp. 347–352, 1991.

[58] J. Allali and M. Sagot, "A new distance for high level rna secondary structure comparison," *IEEE/ACM Transaction on Computational Biology and Bioinformatics*, vol. 2, no. 1, pp. 3–14, 2005.

[59] M. Hochsmann and T. Toller, "Local similarity in rna secondary structures," in *Proceedings of the IEEE Bioinformatics Conference 2003 (CSB 2003).* IEEE Press, 2003, pp. 159–168.

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this thesis: Dr. Ashlock, Dr. Adams, Dr. Dorman. Dr. Voytas, and Dr. Carpenter. Thank you for all your help and encouragement. This work was supported in part by NIH grant GM068955.