1-1-2005

A genetic algorithm for simplifying the amino acid alphabet and predicting protein interactions.

Matthew Palensky

A GENETIC ALGORITHM FOR SIMPLIFYING THE AMINO ACID ALPHABET AND PREDICTING PROTEIN INTERACTIONS

A Thesis

Presented to the

Department of Computer Science

and the

Faculty of the Graduate College

University of Nebraska

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

University of Nebraska at Omaha

by

Matthew Palensky

January, 2005

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

THESIS ACCEPTANCE

Acceptance for the faculty of the Graduate College,
University of Nebraska, in partial fulfillment of the
requirements for the degree Master of Science,
University of Nebraska at Omaha.

Committee

Hesham Ali

William E. Taffesich

Chairperson

Date 2/28/2005

A GENETIC ALGORITHM FOR SIMPLIFYING THE AMINO ACID ALPHABET AND PREDICTING PROTEIN INTERACTIONS

Matthew Palensky, M.S.

University of Nebraska, 2005

Advisor: Dr. Hesham Ali

A central problem in creating simplified amino acid alphabets is narrowing down the massive number of possible simplifications. Since considering all possible simplifications is intractable, effectively creating simplified alphabets is essential. Genetic algorithms have been effective in providing near-optimal solutions for similar combinatorial problems with large solution spaces. This makes them a good candidate for creating simplified alphabets. Simplified amino acid alphabets could uncover hidden relationships in protein sequences, and in turn provide a valuable first step in solving protein-related microbiological problems. The project demonstrates the impact of reducing the alphabet in addressing an important open problem in microbiology, which is predicting protein-protein interactions. Various techniques for predicting protein-protein interactions exist, but are incomplete. No single method can effectively predict more than a small subset of interactions. Hence, a comprehensive listing all of a cells protein-protein interactions may require many complimentary approaches. The projects results indicate that genetic algorithms effectively create simplified amino acid alphabets, and these alphabets are a useful tool in predicting protein interactions.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Massive amounts of biological data have recently become available for analysis. As a result, the genes and encoded proteins of increasing numbers of organisms are becoming available. This flood of information presents tremendous opportunities, but much remains unknown. A major challenge in bioinformatics is to improve the quality of information that can be extracted from a biological sequence. Among the biological sequences, proteins are of particular importance. Proteins are generally represented as sequences of symbols, where each symbol represents one of the protein's amino acids. This set of symbols is known as the amino acid alphabet. There are 20 amino acids that can appear in a protein, so there are 20 symbols in the standard amino acid alphabet. For some biological problems it is useful to work with a reduced set of symbols. This is known as using a simplified amino acid alphabet. Grouping two or more amino acids together and representing them with the same symbol simplifies the amino acid alphabet since it can be represented with less than 20 symbols. The use of simplified amino acid alphabets is a technique employed to gain a better understanding of protein sequences. They have been used to predict protein interactions

[8], infer protein function [17], and to search for unanticipated patterns in proteins [3].

Simplified alphabets are typically created by grouping amino acids according to their biochemical and biophysical properties, such as hydrophobic / non-hydrophobic and polar / non-polar. In these cases, simplifications are effectively created manually. For example, creating a two-symbol alphabet where each amino acid is categorized as hydrophobic or non-hydrophobic is easily done by hand. Simplifications based on more complex criteria disallow manual creation. For example, suppose each amino acid is assigned a number indicating its degree of hydrophobicity. Creating a four-symbol simplification where each amino acid has a hydrophobicity measure close to the average hydrophobicity of its group is not reasonably done manually. Computational simplification methods for complex simplification criteria are emerging. Examples include grouping by contact potentials [17] and grouping according to evolutionary similarity [3].

1.1 Background

A major barrier to manually created simplifications is selecting from a massive number of possible simplifications. This is also a problem for computational simplification methods. There are 51×10^{12} possible simplified alphabets. Even for computational methods, it is not possible to efficiently examine each simplification. In computer science terms, creating simplified alphabets is equivalent to the clustering problem. The clustering problem is a well-researched topic. A cluster is a set of similar objects, where the definition of similarity is domain specific. There are many known clustering algorithms; however, an algorithm that optimally clusters one set of data may not

be effective on other data sets. Previous studies have used k-means [17] and branch-and-bound [3] clustering algorithms to create simplified alphabets.

The study described in [3] created a branch-and-bound algorithm that generates simplified amino acid alphabets based on evolutionary similarity. The benefit of this strategy is that branch-and-bound algorithms are guaranteed to find an optimal solution. The main drawback to a branch-and-bound simplification strategy, and branch-and-bound solutions in general, is long running times. For some sized alphabets the branch-and-bound algorithm takes hours to terminate. For example, their method took over 13 hours to find an optimal 6-symbol alphabet. If a different algorithm can more quickly create the same or similar simplification, it may be better suited for some applications.

Additionally, [3] proposed a method for evaluating simplified alphabets based on evolutionary similarity. Their method has not been proven effective in any biological domain. If a suitable domain were found, the simplified alphabets based on their method would be of value. Simplified alphabets have been used in predicting protein interactions. A study described in [8] utilized a six symbol simplified alphabet based on the biochemical similarity of amino acids to predict protein-protein interaction. Their simplified alphabet was found to have some value in protein interaction prediction. Hence, it is possible that simplified alphabets based on different criteria may provide additional insight into the protein prediction problem.

Predicting cellular protein interactions is an open problem in microbiology. Traditionally, these interactions have been uncovered through laboratory methods using genetic, biophysical, and biochemical techniques [16]. Although these methods have shown promise, they are incomplete, and protein-protein interactions remain un-

known. For example, 80,000 protein-protein interactions in yeast are known to exist, but only 2,400 interactions are detected by any single method [16]. This indicates that each method has strengths and weaknesses; they are effective at predicting certain types of interactions, but ineffective at predicting others. Laboratory methods also suffer from high error rates. It is estimated that as many as half the interactions predicted by these methods are false[16]. A comprehensive listing all of a cell's protein-protein interactions may require many complementary approaches. Thus new methods that detect protein-protein interactions would be of value.

To help meet this need, computational detection methods are becoming available. Many current computational methods attempt to infer unknown protein-protein interactions by examining known interactions. Identifying correlated sequence-signature pairs [14], sums of positive and negative forces along interacting proteins [8], and predicting targets of transcription factors through gene expression profiles [13] utilize this approach.

1.2 Motivation

Although simplified alphabets are a useful bioinformatics tool, the best methods for creating them remain unknown. The clustering problem is known to be NP-hard, which generally prevents the efficient generation of optimal solutions. Hence, a challenge in creating simplified alphabets is efficiently finding high-quality, or near-optimal, simplifications. Current studies utilizing simplified alphabets focus on finding optimal simplifications. Techniques emphasizing efficiency remain unexplored.

For detecting protein interactions, computational approaches are an attractive al-

ternative to laboratory methods. Compared to laboratory techniques, computational methods are relatively inexpensive, quicker, and are well positioned to take advantage of the flood of new biological data. Computational approaches can also complement experimental methods by identifying a smaller set of possible interactions for the experimental method to focus on. A main challenge in creating computational detection methods is that the number of possible protein-protein cell interactions is extremely large, and finding the proper criteria that narrows the possible interactions is difficult.

1.3 Overview

This study has two goals. The first goal is to create an algorithm that efficiently creates high quality simplified alphabets. Efficiency is measured by the time it takes for the simplified alphabet to be generated. Ideally, the algorithm should generate an optimal or near optimal solution within a few minutes. The quality of the genetic algorithms simplified alphabet consists of several characteristics. The first characteristic is how close its simplified alphabets are to an optimal solution. Next characteristic is how closely related the alphabets clusters are based on traditional cluster metrics. The last characteristic is their ability to aid in solving a biological problem. Several types of algorithms were initially evaluated to determine their suitability, including k-means, graph theoretic, and genetic algorithms. The initial evaluation found genetic algorithms are well suited for creating simplified alphabets. Genetic algorithms have proven successful in creating near-optimal solutions to combinatorial problems [12]. The simplification problem is combinatorial in nature; hence, genetic algorithms can potentially create high-quality simplified alphabets. Genetic algorithms attempt to solve problems by modeling the evolutionary process. They operate by combining

pieces from a set of potential solutions until a solution of sufficient quality emerges. Genetic algorithms have been successful in machine learning, neural network, and biological applications [12]. This study focuses on using genetic algorithms to create simplified alphabets.

The second goal is to identify a biological domain where simplified alphabets based on evolutionary history are insightful. [8] utilized simplified alphabets to predict interactions in *Saccharomyces cerevisiae*. Using their work as a starting point, this study explores how the genetic algorithms simplified alphabets can be applied to the protein prediction problem. The study's central idea is proteins that are evolutionarily similar to known interacting proteins will also interact. The genetic algorithm creates simplified alphabets based on the evolutionary similarity of amino acids. Hence, representing proteins with the genetic algorithms simplified alphabets may provide insight into the protein interaction prediction.

1.4 Methodology

The genetic algorithm's effectiveness is measured by how quickly it can find optimal or near optimal solutions. Since this study uses the method described in [3] to evaluate a simplified alphabets quality, their results provide a good baseline from which to measure the genetic algorithms results. [3] provides the optimal simplification for each alphabet size along with the time taken to generate the alphabet. The quality of the genetic algorithms simplified alphabets is measured by how close they are to the optimal simplification and how much faster they are generated.

This study proposes that proteins that are evolutionarily similar to known inter-

acting proteins will also interact. Rather than focusing on specific protein features, pairs of length four subsequences, where one subsequence comes from one of the interacting proteins and the other comes from the its interacting partner, that appear in interacting proteins are examined. Pairs of protein subsequences appearing at a higher than expected rate can be used to predict unknown interactions by looking for pairs of proteins where one protein contain one subsequence, and the other protein contains the other subsequence. This study utilizes a training set approach. Interactions are predicted by collecting data from a set of known protein interactions and using that information to predict new interactions. The study tests this method on a trusted set of 1003 known protein interactions from *Saccharomyces cerevisiae.* All the proteins in this set are represented with a simplified alphabet generated by the genetic algorithm. 500 of these interactions are then used as the training set. The data gathered from the training set is used to attempt to predict the remaining 503 interactions.

1.5 Thesis Organization

The study has two areas of concern: clustering and protein interaction. These issues are addressed in six chapters: background information on protein representation and clustering (chapter 2), state of the art methods in creating simplified alphabets and predicting protein interactions (chapter 3), the genetic simplification algorithm created for this study (chapter 4), applying simplified alphabets to the protein interaction prediction problem (chapter 5), and conclusions (chapter 6).

Chapter 2 provides an overview of the key ideas needed to understand the study. It begins by describing the basic structure of proteins and they can be programmatically

represented. Next, simplified alphabets are presented, including information on how to represent proteins with simplified alphabets. The chapter concludes by presenting commonly used clustering algorithms.

Chapter 3 contains the state of the art methods used to create simplified alphabets and computationally predict protein interactions. The chapter begins by presenting algorithms used in recent studies to create simplified alphabets along with their results. The following section analyzes these approaches and describes why further research is needed. Similarly, the proceeding sections describe and analyze computational approaches to the protein prediction.

Chapter 4 presents the genetic simplification algorithm created for this study. It begins by describing how genetic algorithms in general operate. Next, each component of the genetic algorithm is described and justified. Finally, the chapter concludes with an analysis of the quality of the simplified alphabets created by the algorithm.

Chapter 5 contains the protein interaction detection algorithm. The study's approach to predicting interactions is presented. Each step of the prediction algorithm is described and analyzed. Finally, the resulting predictions are listed and evaluated.

Chapter 6 contains conclusions resulting from the study, along with possible directions for future work.

Chapter 2

Definitions and Terminology

2.1 Proteins and Amino Acids

2.1.1 Proteins

Proteins are one of the most abundant cellular elements. They play a key role in many cellular processes. Their functions include catalyzing essential chemical reactions, providing a structure that holds cell parts together, and transporting molecules across cell boundaries [9]. Proteins consist of chains of amino acids, joined by a peptide bond [7]. There are 20 amino acids that can appear in a protein.

Proteins are characterized by four structural types: primary, secondary, tertiary, and quaternary. The most basic type is the primary structure. The primary structure is the sequence of amino acids that make up the protein. A protein's sequence of amino acids tend to twist into particular shapes [7]. The shape is known as the the protein's secondary structure. The shapes are either coils or a pleated sheet, known as

helixes and beta sheets, respectively. A single protein can contain multiple helix and beta shapes. These helixes and beta sheets interact to for form a three-dimensional structure known as the tertiary structure [7]. A protein's tertiary structure is known as its structural motif. These motifs are significant because they imply the protein's function [2]. The interaction of two or more protein's tertiary structures combine to form a quaternary structure. Figure 2.1 depicts the four protein structures.

2.1.2 Amino Acids

Amino acids are protein building blocks. Since there are 20 amino acids and a protein's primary structure is generally hundreds amino acids long, the number of possible primary structures is astronomical. Although amino acids have the same basic structure, they generally fall into several different categories. These categories include polar or non-polar, acidic or non-acidic, and ionized or non-ionized. Figure 2.2 lists the amino acids that can appear in a protein.

2.2 Protein Sequence Analysis

2.2.1 Representing Protein Sequences

The central dogma of molecular biology is that genetic information flows from DNA to RNA to proteins [10]. RNA encodes the sequence of amino acids that defines a protein. The sequence of amino acids determines the proteins three-dimensional structure, which in turn implies its function. This process is not reversible; however, proteins with similar motifs may have non-similar primary structures. Hence, protein sequence

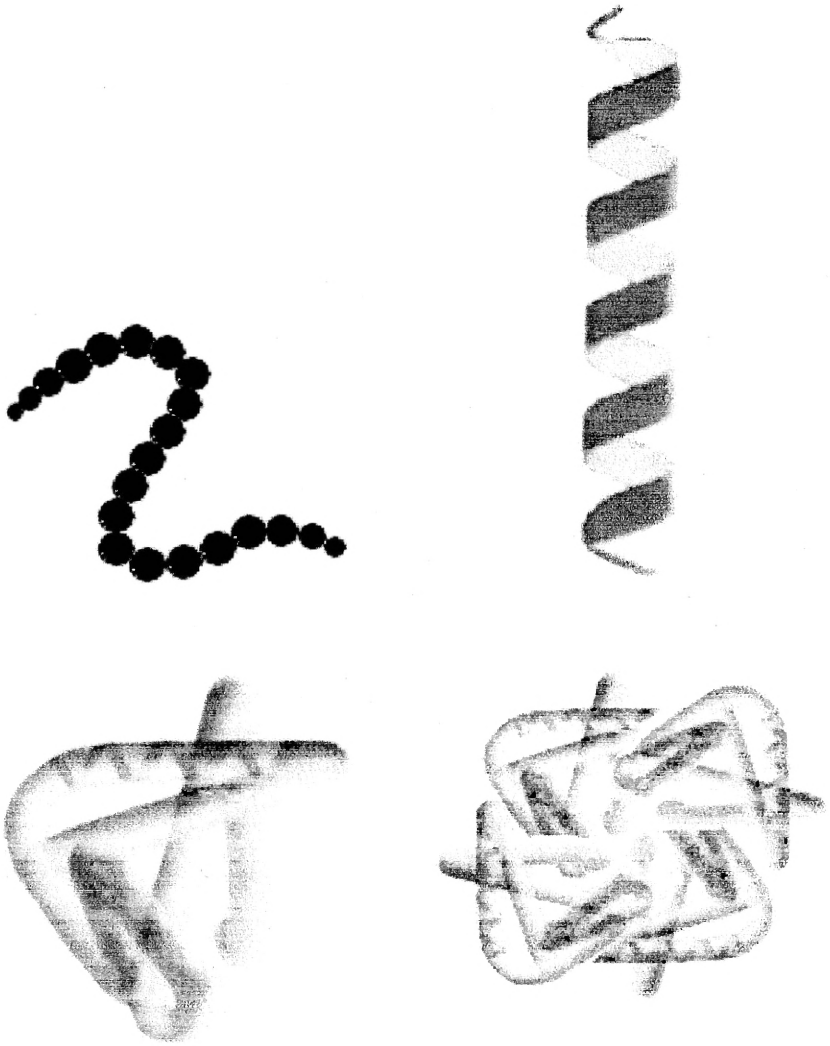
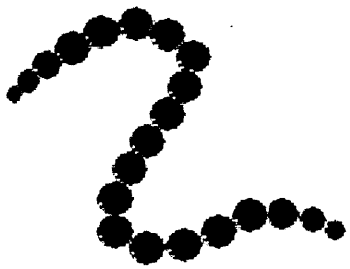


Figure 2.1: The four protein structures. Clockwise from the upper left is the primary structure, the secondary structure, the tertiary structure, and the quaternary structure.

Amino Acid	Abbreviation	Symbol
Alanine	Ala	a
Aspartate	Asp	d
Cysteine	Cys	c
Glutamate	Glu	e
Phenylalanine	Phe	f
Glycine	Gly	g
Histidine	His	h
Isoleucine	Ile	i
Lysine	Lys	k
Leucine	Leu	l
Methionine	Met	m
Asparagine	Asn	n
Proline	Pro	p
Glutamine	Gln	q
Arginine	Arg	r
Serine	Ser	s
Threonine	Thr	t
Valine	Val	v
Tryptophan	Trp	w
Tyrosine	Try	y

Figure 2.2: The amino acids that can appear in a protein. Each protein is associated with a one character symbol. The set of 20 symbols is known as the amino acid alphabet.



msqeirqnekisyriegpffiihlmnpdnlnalegedyiylg

Figure 2.3: A string representation of a protein's primary structure. Each character in the sequence represents an amino acid.

analysis focuses on the protein's primary structure. A protein's primary structure is easily represented by a string, where each of the string's characters represents an animo acid. Amino acids are represented by their one character symbol listed in Figure 2.2. Figure 2.3 depicts the string representation of a sample protein.

2.2.2 Determining Sequence Similarity

An open microbiology problem is determining a protein's three-dimensional structure and function given only its amino acid sequence. It is known that proteins with similar primary structures usually have similar motifs and function [10]. Hence, determining a protein's three-dimensional structure and function can be inferred by finding a protein with a known three-dimensional structure and function having a similar amino acid sequence. This makes protein sequence comparisons an essential aspect of protein analysis. Large efforts have been made to catalog protein sequences in databases to facilitate searches.

p_1: aymakiaa
p_2: aymsrsaa

p_1:	a	y	m	a	k	i	a	a
p_2:	a	y	m	s	r	s	a	a

Figure 2.4: Aligning two protein sequences.

A common method for measuring the similarity of two protein sequences is to align the sequences and note where they are similar. Figure 2.4 depicts the alignment of two protein sequences. In the figure, the sequences are the same at every position except the middle three. The more pair-wise matches the more similar the sequences.

A major challenge in comparing proteins is that they tend to change as they are copied from parent to child. Some amino acids are deleted, new amino acids are added, and some amino acids are substituted for others. The protein sequences in Figure 2.4 are easily aligned, but aligning actual sequences is more complicated. To illustrate one reason why, consider aligning the sequences in Figure 2.5. Since the figure's two sequences have different lengths, it is not obvious how the alignment should be done. Some of the questions this raises are:

- What is done with gaps between alignments? Are more or less gaps better?
- When an amino acid can not be aligned with the same amino acid, what amino acid should it be aligned with?
- Once an alignment is determined, how is similarity determined?
- How can a biologically-relevant alignment be generated?

p_1: aymkksriaryaa
p_2: aymsriaa

p_1:	a	y	m	k	k	s	r	i	a	r	y	a	a
p_2:	a	y	m	-	-	s	r	-	i	-	-	a	a

p_1:	a	y	m	k	k	s	r	i	a	r	y	a	a
p_2:	a	y	m	-	-	s	-	-	-	r	i	a	a

Figure 2.5: Two different alignmants of a pair of protein sequences. Which alignment is the correct one? How similar are these proteins?

Distinguishing between possible alignments is the goal of pairwise alignment methods. These methods can be automated, making searches through large data sets possible. The following three sections provide an overview of how alignments are evaluated. It is important to note that even though alignment methods exits, it is not always possible to automatically determine if an alignment is biologically relevant [6].

2.2.3 Pair-wise Protein Alignment

Essentially, comparing protein sequences is determining if they originated from a common ancestor [6]. Accounting for evolutionary history is typically done through pair-wise alignment matrices. Pair-wise alignment matrices are evolution in a nutshell [10]. They describe how amino acids tend to change over time. A pair-wise alignment matrix assigns each possible pair of amino acids a numeric value indicating the likelihood the amino acids will be substituted for one another over time. The higher the value, the more likely a substitution will occur. The values are determined by a statistical analysis of sets of evolutionarily similar proteins. Figure 2.6 shows

the BLOSSUM40 scoring matrix, a commonly used pair-wise scorong matrix.

-	a	r	n	d	c	q	e	g	h	i	l	k	m	f	p	s	t	w	y	v
a	5	-2	-1	-1	-2	0	-1	1	-2	-1	-2	-1	-1	-3	-2	1	0	-3	-2	0
r	-2	9	0	-1	-3	2	-1	-3	0	-3	-2	3	-1	-2	-3	-1	-2	-2	-1	-2
n	-1	0	8	2	-2	1	-1	0	1	-2	-3	0	-2	-3	-2	1	0	-4	-2	-3
d	-1	-1	2	9	-2	-1	2	-2	0	-4	-3	0	-3	-4	-2	0	-1	-5	-3	-3
c	-2	-3	-2	-2	16	-4	-2	-3	-4	-4	-2	-3	-3	-2	-5	-1	-1	-6	-4	-2
q	0	2	1	-1	-4	8	2	-2	0	-3	-2	1	-1	-4	-2	1	-1	-1	-1	-3
e	-1	-1	-1	2	-2	2	7	-3	0	-4	-2	1	-2	-3	0	0	-1	-2	-2	-3
g	1	-3	0	-2	-3	-2	-3	8	-2	-4	-4	-2	-2	-3	-1	0	-2	-2	-3	-4
h	-2	0	1	0	-4	0	0	-2	13	-3	-2	-1	1	-2	-2	-1	-2	-5	2	-4
i	-1	-3	-2	-4	-4	-3	-4	-4	-3	6	2	-3	1	1	-2	-2	-1	-3	0	4
l	-2	-2	-3	-3	-2	-2	-2	-4	-2	2	6	-2	3	2	-4	-3	-1	-1	0	2
k	-1	3	0	0	-3	1	1	-2	-1	-3	-2	6	-1	-3	-1	0	0	-2	-1	-2
m	-1	-1	-2	-3	-3	-1	-2	-2	1	1	3	-1	7	0	-2	-2	-1	-2	1	1
f	-3	-2	-3	-4	-2	-4	-3	-3	-2	1	2	-3	0	9	-4	-2	-1	1	4	0
p	-2	-3	-2	-2	-5	-2	0	-1	-2	-2	-4	-1	-2	-4	11	-1	0	-4	-3	-3
s	1	-1	1	0	-1	1	0	0	-1	-2	-3	0	-2	-2	-1	5	2	-5	-2	-1
t	0	-2	0	-1	-1	-1	-1	-2	-2	-1	-1	0	-1	-1	0	2	6	-4	-1	1
w	-3	-2	-4	-5	-6	-1	-2	-2	-5	-3	-1	-2	-2	1	-4	-5	-4	19	3	-3
y	-2	-1	-2	-3	-4	-1	-2	-3	2	0	0	-1	1	4	-3	-2	-1	3	9	-1
v	0	-2	-3	-3	-2	-3	-3	-4	-4	4	2	-2	1	0	-3	-1	1	-3	-1	5

Figure 2.6: The BLOSSUM40 scoring matrix. The matrix reports the alignment score of each possible amino acid pairs. The higher the score, the higher the likelihood that the amino acids tend to be substituted for each other over time.

Using a pair-wise alignment matrix, two proteins can be compared by aligning the protein's amino acids and summing their pair-wise alignment values. Protein pairs having high alignment scores tend to be similar, and protein pairs having low alignment scores dissimilar. This is part of the strategy used by BLAST, a popular protein comparison tool. Figure 2.7 shows how the alignment of two protein's is scored. When aligning two proteins, it is not always possible to align an amino

acid with another amino acid. For example, in the alignment depicted in Figure 2.5 the sequences have differing lengths, implying some amino acids will not be paired with another amino acid. These cases are called gaps, and pair-wise scoring methods account for this by assigning them low alignment scores.

a	r	d	c	n	t	h	a	Total
-1	-3	-2	-3	-1	-1	0	5	-6
e	p	c	m	e	f	e	a	-

Figure 2.7: The alignment of two proteins. The alignment score of each amino acid pair is taken from the BLOSSUM40 scoring matrix. The protein's alignment score is is the sum of all the aligned amino acid pair's scores. The higher the score, the more similar the proteins.

2.2.4 Pair-wise Alignment Algorithms

Scoring the alignment of two sequences with the same length is easily done if no gaps are allowed; however, allowing gaps significantly increases alignment complexity. There are

$$\binom{2n}{n} = \frac{(2n)!}{(n!)^2} \simeq \frac{2^{2n}}{\sqrt{\pi n}}$$

possible alignments of two sequences of length n [6]. This prevents proteins from being manually aligned. Well known alignment algorithms exits. The goal of pair-wise alignment algorithms is to create an alignment with the optimal, or highest, total alignment score. The algorithms assume that a protein pairs' optimal alignment is a biologically relevant one. Many alignment algorithms are based on dynamic programming. Dynamic programming techniques find global optimizations through

a divide-and-conquer strategy. Alignment algorithms using dynamic programming include the Needleman-Wunsch and Smith-Waterman algorithms [10].

2.3 Simplified Alphabets

A simplified amino acid alphabet is a partitioning of the amino acids into groups where each amino acid in a group is treated the same. When there are less than 20 groups, the amino is simplified since it can be represented with less than the standard 20 symbols. Figure 2.8 shows a simplified alphabet based on biochemical similarity.

{ivlm}{fyw}{hkr}{de}{qntp}{acgs}

Figure 2.8: A simplified alphabet based on the biochemical similarity of amino acids.

Simplified alphabets are used when for the purpose at hand, a group of amino acids share properties allowing them to be treated the same. Generally only a single property is considered. Thus, the art of creating simplified amino acid alphabets lies in knowing what can be ignored.

Representing a protein's primary structure with a simplified alphabet is similar to its non-simplified representation. The primary structure is represented by a string, but rather than each amino acid having a unique character, each amino acid group has a unique character. Each character in the protein's standard primary structure is replaced by its group's character.

Table 2.1 depicts an example character representation for the simplified alphabet in Figure 2.8. The left column shows the cluster, and the right column shows the

Cluster	Symbol
{ivlm}	B
{fyw}	Z
{hkr}	U
{de}	X
{qntp}	C
{acgs}	O

Table 2.1: A mapping of a cluster of amino acids to a symbol that represents the any amino acid in the cluster.

symbol used to represent all amino acids in the cluster. Figure 2.9 depicts how a protein's primary structure is represented with a simplified alphabet.

Standard Representation: msqeirqnekisyrie
Simplified Representation: BOCXBUCCXUBOZUBX

Figure 2.9: Representing a protein's primary structure with a simplified alphabet. Each amino acid symbol in the original sequence is replaced with the symbol representing the cluster to which the amino acid belongs.

2.4 Clustering Algorithms

A clustering is a transformation of a single set of data into multiple sets of similar data. Informally, clustering is an arrangement a group of data into multiple groups whose members are similar in some way. The similarity criterion depends on the data's properties of interest. For example, suppose a box contains 20 crayons of three different colors. The crayons can be clustered into three groups where each groups crayons have the same color. In this case the similarity criterion is color: two crayons belong in the same group if they are the same color. Challenges arise when one or more of the data elements do not fit into any of the available groups. In the crayon

example, this would occur if crayons of four different colors had to be placed in three groups. In these cases, elements must be placed in the group it is most similar to.

There are multiple ways to cluster a set of data. In the crayon example, there are over 500 million ways to partition the crayons into three groups. In order to select the solution that best represents the similarity criterion, a method to compare possible clustering schemes is needed. When evaluating a clustering of data, two principals guide the evaluation. First, the elements in each cluster should be strongly related to each other. Second, the clusters must be different from each other. This is commonly expressed in terms of "distance". The distance between the elements in each cluster should be small, which indicates the clusters elements are similar and belong in the same cluster. The distance between each cluster should be large, which indicates elements from different clusters are different from each other and should not be in the same cluster. The distance metric is typically the Euclidean distance between elements represented as data points on a graph.

Several well-known clustering algorithms exist. Each algorithm has strengths and weaknesses, and their suitability depends on the data set. The following sections describe some popular clustering methods.

2.4.1 Branch and Bound Clustering

The branch-and-bound algorithm is a general search method that finds the optimal solution to a given problem. It operates by dividing the search space into regions, known as branches. If a particular branch's score falls outside a specified boundary, all solutions in the branch are eliminated from consideration. The values used for scoring are dependent on the feature of interest. Otherwise, the branch further divided into

sub-branches and the process is repeated. The algorithm can be used for clustering by having it search through all possible clustering schemes.

Figure 2.10 depicts all the possible branch-and-bound branches for a search for the best clustering scheme of three data elements. The root of all braches is a single cluster containing a. Adding all possible clusters containing the original a cluster along with data element b creates two new branches. Adding the c element to the clusters created in the previous step creates more branches. Finally, all five possible clustering schemes are generated. Before creating a new branch, the algorithm checks if it falls within a specified threshold. If not, the branch is discarded and not further considered. For example, if the cluster ab fell outside the threshold then clusters ab, c and abc would not be generated.

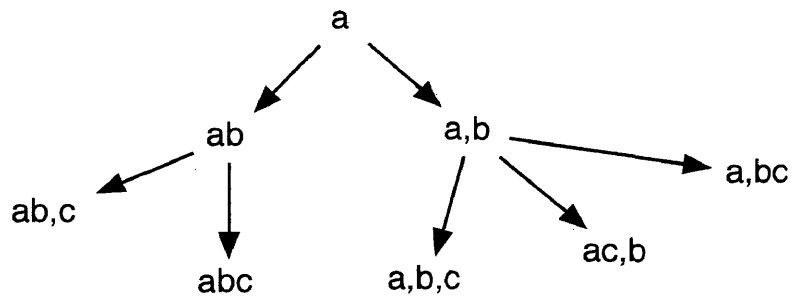


Figure 2.10: The tree-like structure of a branch and bound algorithm. The tree is rooted with a single data item. Branches are generated by creating partitions containing a nodes partitions along with a new single data item. After all items have been added, each leaf represents a possible partition of all the data items.

Like an exhaustive search, the branch-and-bound generally exhibits long running times when the search space is large since the elimination process leaves many possible solutions. The algorithm is suitable when an optimal solution is required, when a

solution of particular quality is required, or when a group of solutions within a certain threshold is required. Even though it exhibits long running times, the branch-and-bound algorithm can be significantly faster then an exhaustive search.

2.4.2 K-Means Clustering

The k-means algorithm clusters data by minimizing the distance between each data element and the geometric centroid of its cluster [1]. That is, each data element is placed into the cluster whose elements mean value is closest to the data element. The values depend on the data feature used for clustering. Since placing a new element into a cluster changes its mean value, this step is performed multiple times. The general structure of a k-means algorithm is:

- Until the *termination condition* is satisfied, loop:
    1. Assign each data element to the cluster such that the *distance* from the data element to the clusters center is minimized.
    2. Re-calculate each clusters median.
- End loop

The termination condition is generally a fixed number of iterations or a situation where no further data movement is possible. The distance is usually the Euclidean distance.

For example, the following k-means algorithm that clusters digital image pixels:

- Until 100 iterations are complete, loop:

1. Assign each pixel to the cluster such that the Euclidean distance between the pixels color and the median value of the clusters pixel colors is minimized.
2. Re-calculate each clusters median pixel color.

- End loop

K-means algorithms are easily implemented, making them a popular choice for clustering [1]. K-means algorithms can have long running times, but using the proper termination condition can significantly improve performance. This makes k-means a good choice when a near-optimal solution are sufficient.

2.4.3 Graph Theoretic Clustering

In graph clustering algorithms, the data elements are represented as a set of nodes and edges between nodes indicate the degree that two nodes are similar. The algorithms that derive clusters from this representation are often extensions of classic graph algorithms. Graph algorithms that find a minimum-cost spanning trees or cliques can be used to create clusters. Many of these algorithms are efficient [4], and are useful in clustering when approximate solutions are sufficient.

As an example, clusters can be found by extracting a graphs strongly connected components. In this scheme, the data elements are the graphs nodes and edges between nodes indicate a close relationship between the nodes. That is, an edge exists between two nodes if they are similar, and no edge exists indicate they are not similar. In graph theory, a group of nodes is strongly connected if any node in the set is reachable from any other node in the set. A graphs strongly connected

components can be found using a two-step depth first search algorithm [4]. Once the strongly connected components are found, their nodes are clustered together. Figure 2.11 depicts a directed graph with three strongly connected components. From these components the clusters {bce}, {ag}, and {hfd} are formed.

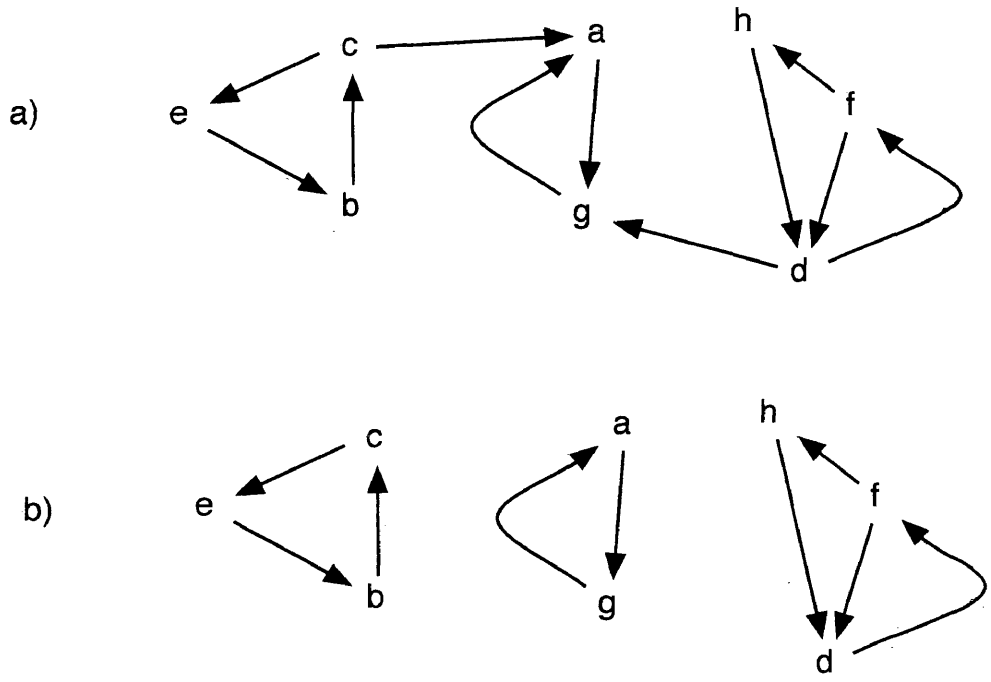


Figure 2.11: Clustering by finding a graphs strongly connected components. *a*) depicts the relationships between data elements. *b*) depicts the graph's strongly connected components, which will be used as data clusters.

Chapter 3

Background

3.1 State of the Art in Creating Simplified Alphabets

3.1.1 Branch-and-Bound Simplification

Although the simplification problem is not tractable, approaches finding optimal simplifications exist. A study was conducted on the effectiveness of branch-and-bound algorithms in creating simplified alphabets [3]. In the study, amino acids are grouped according to their evolutionary substitution rate. That is, amino acids that tend to be substituted for each other over time are grouped together.

The study evaluates a simplification's quality with a pair-wise substitution matrix based scoring method. When comparing two proteins using a pairwise alignment matrix, the sum of the pair-wise aligned amino acids represents the protein's simi-

larity. The study proposes that if proteins are represented with simplified alphabets, then pair-wise alignment scores will be less accurate, but better simplifications should score better than worse simplifications. Hence, the quality of a simplification can be evaluated by calculating the difference between the pair-wise alignment score of two proteins and the pair-wise alignment score of the same two proteins represented with a simplified alphabet. The alignment of the two proteins act as a baseline to which simplified alphabets can be compared.

Rather than using actual proteins for the baseline, the study utilizes two identical proteins containing 20 amino acids where each amino acid appears exactly once. These fictional proteins are easily aligned using no gaps. This scheme has the feature that each amino acid is paired with itself, and when using a simplified alphabet, each cluster is paired with itself. When aligning a cluster with itself, the alignment score is the average pair-wise score of all possible non-simplified alignments within the cluster. For example, suppose s and r are clustered together. Then when $\{sr\}$ is aligned with $\{sr\}$, there are four possible non-simplified alignments: s with s, s with r, r with s, and r with r. The average value of these pair-wise alignment scores is $5 - 1 - 2 + 9 \div 4 = 2.75$. The study defines the cost of a simplification as the difference between the sum of the alignment scores of the non-simplified alphabet minus the sum of the simplified alignment scores. The lower the cost, the closer the cluster alignment score is to the non-simplified alignment, and thus the higher the simplification's quality. The cost of an entire simplification is the sum of the simplification's clusters. Figure 3.1 describes this calculation.

The cost calculation is a mapping from a simplified alphabet to a number. The branch-and-bound simplification algorithm's goal is to find the simplification with the lowest cost value. Figure 3.2 depicts sample branches generated by the algorithm.

	a	r
a	5	-2
r	-2	9

		Score
{ar}	{ar}	2.75
{ar}	{ar}	2.75

$$\text{Cost} = (5 + 9) - (2.75 + 2.75) = 8.5$$

Figure 3.1: Determining a simplification's cost. The top table is the pair-wise alignment scores of *aa* and *rr*. The bottom table is the alignment scores of the same amino acids represented with a simplified alphabet. The simplification's cost is the difference between the non-simplified alignment and the simplified alignment.

Since each node contains a simplification, its cost can be calculated. Branched from nodes that can not contain an optimal simplification are eliminated from further consideration.

Table 3.1 shows the optimal alphabets for a subset of possible alphabet sizes. The algorithm quickly finds the optimal simplification for alphabets of sizes 2 and 3. The running time is significantly longer for alphabets of sizes 4 through 6, with a 6 symbol alphabet taking over 13 hours to find the optimal simplification.

3.1.2 K-Means Simplification

A central question in modeling proteins is how many amino acid residue types are needed to fashion a structured protein [17]. Research indicates at least five residue types may be required [5]. The study described in [17] created simplified alphabets based on the contact potentials of amino acids to explore this idea. Its goals are to

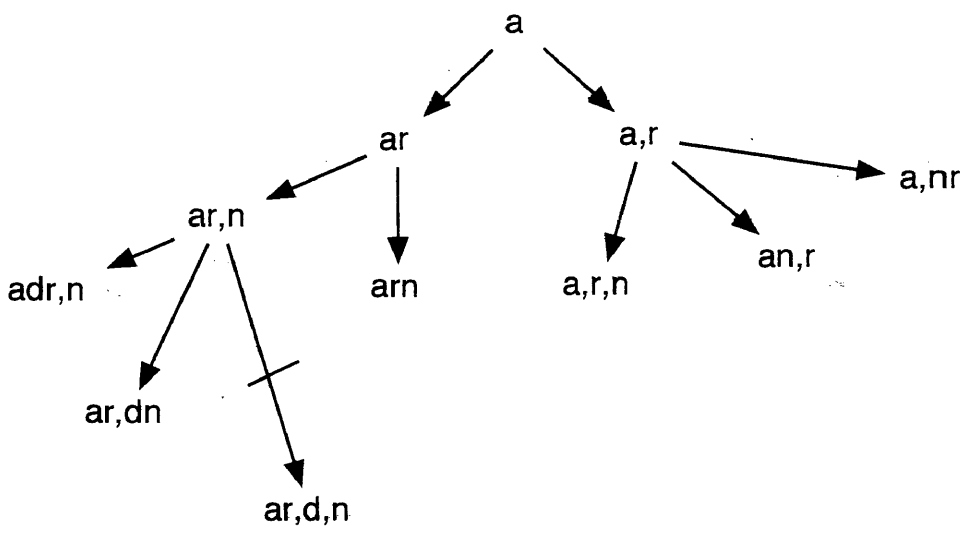


Figure 3.2: A section of the branches generated by the branch-and-bound algorithm. The line through the branch from the ar, n node to the ar, d, n indicates it cannot contain an optimal solution and will not be explored further.

determine how many residue types are needed to form a protein and to determine if clustering amino acids by their contact potentials accounts for the features needed to model proteins.

The central idea used to group the amino acids is that amino acids within a group should interact similarly and amino acids in different groups should interact differently. Amino acids grouped in a manner that does not meet this criterion are considered mismatched. The study implemented a k-means matrix that attempted to minimize the number of mismatches. The interactions are taken from the Miyazawa-Jernigan (MJ) matrix of contact potentials.

The studys k-means generated simplified alphabets are categorized by three differ-

Size	Best Possible Alphabet	Cost	Time
2	acdeghknpqrst,filmvwy	160.462	00:00:03
3	filmvwy,c,adeghknpqrst	140.583	00:11:12
4	w,adeghknpqrst,filmvy,c	120.917	02:42:00
5	filmvy,dehknqr,agpst,c,w	105.705	08:46:07
6	filmvy,h,c,w,agpst,deknqr	92.967	13:44:52

Table 3.1: The branch-and-bound simplifications for alphabets sized 2 through 6. The time required to generate the alphabets varies from just a few seconds to over 13 hours.

ent groups: polar, hydrophobic, and singlet. The number of mismatches is minimized at 20, 7, and 5 symbols. The k-means algorithm takes several hours to terminate. The study found that proteins modeled with the 7 and 5 symbol alphabets have good folding ability, indicating they are suitable for protein modeling.

3.2 State of the Art in Computationally Predicting Protein Interactions

3.2.1 Inferring New Interactions From Known Interactions

Until recently, detecting protein interactions was accomplished through laboratory techniques. Major laboratory methods include the yeast two-hybrid assay, mass spectrometry, and correlated mRNA expressions [16]. One drawback to laboratory detection methods is they are relatively slow and expensive. If the protein features essential to interactions can be examined outside the laboratory, computational techniques can

potentially detect interactions in a fast and cheap manner.

The success of laboratory techniques has uncovered trusted sets of protein interactions. These interactions can be viewed as training data, where key features of the interacting proteins are examined and used as a basis for predicting new interactions. This is usually accomplished by a statistical strategy; over-represented features in the interacting proteins are extracted, and new interactions are predicted by searching for proteins exhibiting those features. Statistical prediction methods tend to view protein sets as fully connected, undirected graphs, where vertices represent proteins and edges represent an interaction between two proteins [8]. Figure 3.3 depicts a small interaction graph.

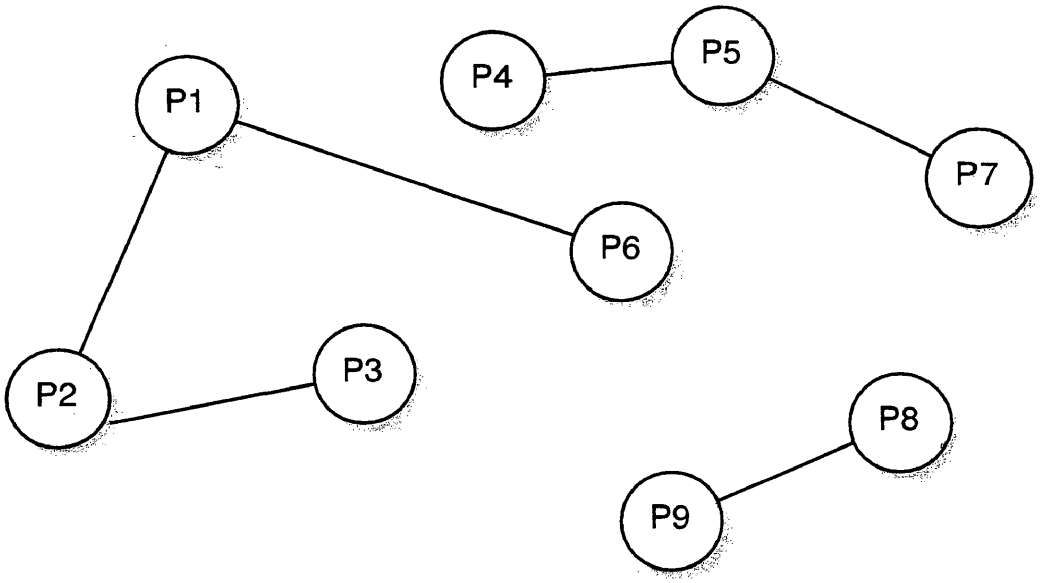


Figure 3.3: A protein interaction graph. Each node represents a protein and each edge an interaction. Actual interaction graphs can contain thousands of nodes and edges. Statistical detection methods generally assign a weight to each edge, indicating the likelihood of an interaction.

Rather than directly reporting the interactions in a set of proteins, statistical methods report the likelihood that two proteins will interact. In protein interaction graphs, this likelihood is represented by an edge weight. Two proteins are determined to interact if their edge weight is above a particular threshold. This makes statistical methods interaction predictors rather than absolutely determining which proteins will interact. The following sections describe the current methods utilizing this strategy.

3.2.2 Predicting Interactions with Sequence Signature Pairs

A study was done on the effectiveness of predicting interactions using protein sequence signatures as an interaction indicator [14]. A sequence signature is a mapping of a section of a protein's primary structure to a value representing a protein category. A protein can have multiple sequence signatures, so each protein can be characterized by its set of signatures. Figure 3.4 shows a sample mapping of a protein to its signatures. The figure is highly simplified. The study's proteins were classified through regular expressions, hidden Markov modes, and protein profiles [14].

The study proposes that pairs of sequence signatures, where one signature comes from one protein and the other comes from the protein it interacts with, occurring at a higher than expected rate in known interactions can be used to predict unknown protein interactions. The prediction is done by extracting the over represented signature pairs, then searching for protein pairs containing the signatures.

The study tested this idea by using a training set of known interactions as both a source of sequence signature data and as a prediction target. That is, the study attempted to predict the interactions in the training set using data collected from the training set. The training set consisted of a set of known interactions in *Sac-*

Sequence	Signature
iyv	A
cghi	B
arncq	X
fpstw	Y
afwt	Z

$$arncqilafwtiyvqqqwwafwt \rightarrow (A, X, Z)$$

Figure 3.4: Mapping proteins to sequence signatures. A sequence signature assigns protein sections to a category. A protein can then be characterized by the signatures it contains.

charomyces cerevisiae. This set consists of 2098 protein pairs. InterPro was used to assign signatures to proteins. InterPro, however, does not contain signature mappings for all the training set's proteins, and was only able to assign signatures to 1274 of the protein pairs. These protein pairs contain 434 unique signatures. From these signatures, a 434×434 matrix that reports how often each pair occurs was created. Figure 3.5 shows a pair-occurrence matrix for a set of sample interactions. Unexpectedly high rates of signature pairs were calculated using the ratio of the actual frequency and the expected frequency of each table entry. This value was expressed as a log-odds value. Sequence signature pairs with log-odds values over a certain threshold were marked as over-represented, and pairs of proteins containing these signatures are predicted to interact.

The authors used the over represented sequence signature pairs to predict the training set's interactions in in the following manner. A signature pair is selected from the set of over represented signature pairs. Two protein's are selected from the training set where one protein contains one of the signatures and the other protein

$$(A, X, Z) \leftrightarrow (A, B)$$
$$(A, B, X) \leftrightarrow (A, B, Z)$$
$$(Z) \leftrightarrow (X, Y)$$
$$(B, X, Y) \leftrightarrow (A, Z)$$

	A	B	X	Y	Z
A	2				
B	3	1			
X	3	2	0		
Y	1	0	0	0	
Z	1	2	3	2	0

Figure 3.5: The figure's upper part contains the interacting proteins signatures. The table shows the number of occurrences of each signature pair.

contains the other signature. This protein pair is predicted to interact. If no protein's containing the signatures is found, another signature pair is selected. After a protein pair is predicted to interact, it is excluded from the training set, the signature matrix is recalculated using the remaining proteins. This process is repeated until no predictions can be made. Using various log-odds thresholds, 94% to 97% of the training set's interactions were predicted.

3.2.3 Predicting Protein Interactions using Evolutionarily Conserved Features

A study attempted to predict protein interactions using evolutionarily conserved features occurring in interacting proteins [8]. Given a set of known protein interactions, the study proposes that pairs of evolutionarily conserved protein features occurring at unexpectedly high or low rates can predict unknown interactions.

Like the sequence signature study, the training set consisted of a set of known interactions in *Saccharomyces cerevisiae*. The set consists of 1714 proteins and 7735 interactions. The features were extracted using hidden Markov models of evolutionarily conserved protein domains from the Pfam database. The features are believed to be protein function related. 1015 unique domain types were identified. The authors use a subset of the training data to gather prediction information and attempt to predict the remaining training set's interactions. A probabilistic model was created that examines all pairs of features occurring in the interacting proteins and assigns a probability that a pair of proteins containing the feature pair will interact. All protein pairs in the training set containing a feature pair with an interaction probability over a particular threshold are predicted to interact.

For comparison purposes, the authors characterize the training set's proteins by all the length-4 amino acid subsequences appearing in each protein. Before the subsequences are extracted, the proteins are represented using a simplified 6-symbol alphabet based on biochemical similarity. Each length-4 subsequence is treated as a protein feature, and the same probability model used for evolutionarily conserved was used for the subsequences.

The study's results indicate that evolutionarily conserved features effectively predicted protein interactions within the training set. Although the length-4 subsequences were not as effective, they were found perform significantly better than chance.

3.3 Analysis of State of the Art

3.3.1 Simplified Alphabets

The methods for creating simplified alphabets to fall into two categories: 1) manual, or hand-crafted, simplification methods and 2) computational simplification methods. The division is based on the simplification criteria used. Simplifications based on simple amino acid properties tend to be done manually, while simplifications derived from complex criteria require computational methods. The main barrier to manual methods is the large number of possible simplifications that can be generated from the evaluation criteria. In these cases, a large number of solutions must be examined before an optimal simplification is found. These situations require computational support; however, even though computers can often quickly examine potential simplifications, the efficient generation of optimal solutions is not obviously obtained. This is evidenced by the current computational simplification methods. The branch-and-bound simplification algorithm described in [3] and the k-means algorithms described in [17] required several hours to terminate. Branch-and-bound algorithms, k-means algorithms, and clustering algorithms in general are well-researched areas computer science. That the simplification algorithms took so long to terminate indicates the efficient creation of simplified alphabets requires more than selecting a well-known algorithm. The efficient generation of simplified alphabets requires both selecting the proper algorithm and proper implementation of the algorithm.

The branch-and-bound simplification algorithm is targeted at finding optimal simplifications. The advantage of this approach over a bruit-force approach is that it can eliminate examining potential sets of solutions if it detects the set cannot contain an

optimal simplification. This is not enough to efficiently generate simplified alphabets. The k-means simplification algorithm is also set up to find optimal simplifications, and does not efficiently generate simplified alphabets. These algorithms are limited by their guarantee of finding an optimal solution. If this guarantee is removed, heuristics could be introduced. A heuristic is a problem solving strategy that is not guaranteed to solve a problem. Even though heuristics are fallible, they have been proven effective in solving problems with large search spaces [11]. Since there is a large number of possible simplified alphabets, heuristics can potentially help create simplified alphabets. The main advantage of heuristics is that they can provide quickly computed shortcuts that significantly reduce an algorithm's running time. The disadvantage is the shortcut could lead to a dead end, resulting in poor solutions. Discovering heuristics that lead to quality simplified alphabets would be of value since they could allow the efficient creation of simplified alphabets.

3.3.2 Predicting Protein Interactions

Alphabets generated by simplification algorithms are of little use unless they can be applied to a biological problem. This study applies simplified alphabets to the protein interaction problem. Current computational prediction methods focus on finding the statistical occurrence of a particular protein feature in known interactions and predicting new interactions by searching for proteins exhibiting the studied feature. This is the approach taken in the studies described in [16] and [8]. While both studies examine protein features that seem to have value in predicting interactions, neither study is conclusive. This mirrors what is true about prediction methods in general. No single method is effective at predicting all cellular interactions. A comprehensive listing may require the combination of many different approaches.

Although the studies in [16] and [8] are not conclusive, both provide a good framework for prediction methods based on over-represented protein features. That is, many different features can be studied utilizing their methodologies. In [8], proteins are represented by their sequence signatures. The number of possible signatures is sufficiently small enough to utilize easily implemented data structures and algorithms. For example, since only 434 signatures are possible the number of sequence signature pairs is easily and efficiently stored in a 434×434 table. This is particularly relevant to prediction methods based on simplified alphabets, since simplified alphabets can potentially reduce the complexity of representing protein features.

Chapter 4

The Simplification Algorithm

This chapter describes the genetic simplification algorithm created for this study. It begins with an overview of genetic algorithms, describing their key concepts and components. Next, the genetic simplification algorithm is described. The chapter concludes with the results of implementing the algorithm.

4.1 Genetic Algorithm Overview

In a sense, evolution is the process of probing through a massive search space, looking for the organisms best suited for a particular environment. The evolutionary process creates new organisms by combining and mutating current organisms in the hope that the new organisms will become better suited for their environment. Genetic algorithms use the evolutionary process as a model for solving problems. They are the computational equivalent of an adaptive system. While typical algorithms operate on a single solution, genetic algorithms operate on a set, or population, of solutions.

Like evolutionary systems, genetic algorithms create new solutions by combining parts of other solutions. New solutions are called child solutions, and the solutions it is composed of are the parents. This mechanism for creating new solutions makes genetic algorithms well suited for problems that are combinatorial in nature, since child solutions are essentially a different combination of the same elements.

The key components of a genetic algorithm are:

Fitness Function A function that determines the quality of a solution. The function generally evaluates to a numeric value where the higher the number the better the solution. The fitness function allows different solutions to be ranked according to how well they solve the problem.

Selection Function A function that selects a parent solution from the population. Its generally a random or weighted random function.

Crossover Operator An operation that creates a child solution from two parent solutions.

Mutation Operator An operation that mutates a child solution. Its used to inject diversity into the population

Genetic algorithms repeatedly create child solutions from parent solutions. Once a child solution is created, it replaces a solution existing in the population. The general steps in a genetic algorithm are listed below. Each iteration of the steps is called a generation.

1. Create a population of random solutions.

2. Using the fitness function, calculate the fitness of each solution.

3. Using the selection function, select two parent solutions.

4. Using the crossover operator, create a child solution from the parent solution.

5. Using the mutation operator, mutate the child.

6. Replace a randomly selected solution with the child solution.

7. Until a halting criterion is reached, go to step 2.

The procedure GENETIC-ALGORITHM, described in Algorithm 1, shows how these steps form an algorithm. The procedure accepts two parameters: the number of generations and the size of the population. The implementation of the CALCULATE-FITNESS, SELECT-PARENT, CROSSOVER, and MUTATE procedures are domain specific. Upon termination, the algorithm returns the population of solutions.

```
GENETIC-ALGORITHM(G, S)
1  population ← GENERATE-POPULATION(S)
2  for i ← 1 to G
3       do CALCULATE-FITNESS(population)
4          p1 ← SELECT-PARENT(population)
5          p2 ← SELECT-PARENT(population)
6          child ← CROSSOVER(p1, p2)
7          MUTATE(child)
8          ADD-CHILD(population, child)
9  return population
```

Algorithm 1: The general form of a genetic algorithm. New solutions are repeatedly created from a population of solutions.

4.2 The Genetic Simplification Algorithm

The algorithm creates simplified alphabets based on the evolutionary similarity of amino acids. That is, amino acids that tend to be substituted for each other over time should be clustered together. The algorithm has two main goals. First, the simplifications should be close to an optimal solution. Second, the algorithm should quickly produce a solution. The genetic algorithm creates populations of simplified alphabets where each solution has the same number of symbols, or clusters. The algorithm iterates for a specified number of generations. Each algorithm component is described in the following sections.

4.2.1 The Fitness Function

The fitness function attempts to measure the degree to which the amino acids in a simplified alphabet's clusters are evolutionarily similar to each other. Since pair-wise alignment matrices report evolutionary history, they provide a good basis for evaluating simplifications based on evolutionary similarity. The study described in [3] uses pair-wise alignment matrices to evaluate the simplifications created by their branch-and-bound algorithm. Rather than create a new simplification evaluation method, this study uses this evaluation method. Branch-and-bound algorithms are guaranteed to find optimal solutions, so the simplifications described in [3] provide a good measure of how close the genetic algorithm's simplified alphabets are to an optimal solution.

The fitness function has the form $F(c) = n$, where F is the fitness function, c is a cluster of amino acids, and n is numeric value. The fitness of a simplified alphabet is

$$\sum_{i=1}^{n} F(s_i)$$

where s is a simplified alphabet, s_i is the i-th cluster in s, and n is the number of clusters in s.

Algorithm 2 describes the fitness calculation. For input, the algorithm accepts a cluster of amino acids and returns the cluster's fitness. It assumes the existence of an algorithm named SCORE, which reports the pair-wise alignment of two amino acids. Steps 4 and 5 calculate the sum of the alignment scores when no simplification is performed. Steps 6 through 9 calculate the sum of the alignment scores of each pair of amino acids in the cluster. Step 10 calculates the cost of clustering the amino acids. The fitness of a simplified alphabet is calculated by summing the fitness of each of its clusters.

```
FITNESS(C)
 1  p ← 0
 2  a ← 0
 3  c ← 0
 4  for i ← 1 to Length(C)
 5        do p ← p + Score(C[i], C[i])
 6  for i ← 1 to Length(C)
 7        do for j ← i + 1 to Length(C)
 8              do a ← a + (2 × Score(C[i], C[j]))
 9                 c ← c + 1
10  return p − (a ÷ c)
```

Algorithm 2: The fitness algorithm. The fitness of a cluster is measured by calculating how close a pair-wise alignment score using the cluster is to a non-simplified alignment score.

4.2.2 The Crossover Operator

Traditional crossover operators rely on randomness. Forming new solutions by randomly combining the parts of the fittest members of a population is an effective strategy for many problems; however, there are drawbacks to a purely random approach to the simplification problem. The main issue is the placement of a few unrelated amino acids into a quality cluster can significantly reduce the clusters fitness. As a result, a populations fittest solutions do not necessarily have highly fit clusters, but rather have clusters containing fit sub-clusters. For example, the cluster $\{agnpqr\}$ is not itself of high fitness, but it contains the fit cluster $\{arn\}$. If these highly fit amino acid groups can be extracted from a cluster, then they can be used to effectively create child solutions. With this in mind, the genetic algorithms crossover operator identifies and combines the highly fit sub-clusters of the parent solutions.

A main challenge in finding highly fit sub-clusters is the searching through a potentially large number of possible sub-clusters within a cluster. This is especially problematic for highly simplified alphabets. For example, a cluster containing 10 amino acids can be partitioned in 115,975 ways. Calculating the fitness of so many sub-clusters would add significant time to the genetic algorithm. To avoid these calculations, the crossover operator splits each original cluster into two clusters and attempts to place each amino acid into the sub-cluster that is most related to itself. One approach to accomplishing this is to place each amino acid into the new cluster that results in the better cluster fitness score. For example, suppose amino acid a must be merged with either cluster {r} or cluster {n}. The fitness of cluster {ar} = 2.50 and the fitness of cluster {an} = 2.75, so a is placed in {r}. While this approach attempts to ensure the fitness of a single cluster is optimized, it may not result in

Step	Cluster	New Clusters	Sum of Costs
1.	$\{adqrtn\}$	{} {}	0
2.	$\{dqrtn\}$	{a} {}	0
3.	$\{qrtn\}$	{a} {d}	$0 + 0 = 0$
4.	$\{rtn\}$	{aq} {d}	$6.50 + 0 = 6.50$
5.	$\{tn\}$	{aqr} {d}	$14.67 + 0 = 14.67$
6.	$\{n\}$	{aqrn} {d}	$22.50 + 0 = 22.50$
7.	{}	{aqrn} {dn}	$22.50 + 6.50 = 29.00$

Figure 4.1: Splitting a cluster into two clusters. The process begins by creating two empty clusters. Each step puts an amino acid from the original cluster into the new cluster that results in the best overall fitness.

a better overall partitioning. For example, suppose r must be added to either $\{aqt\}$ or $\{dn\}$. The cost of $\{aqt\}$ is 13.33 and the cost of $\{dn\}$ is 6.50. Adding r to $\{aqt\}$ results in a cost of 22.50, while the cost adding it $\{dn\}$ is 16.17. The lower cluster cost is achieved by adding r to $\{dn\}$ with the overall cost is $16.17 + 13.33 = 30$. The overall cost of adding r to $\{aqt\}$ is $22.50 + 6.5 = 29$. In this case, a simplification will have a better fitness by adding r to $\{aqt\}$. Since a simplified alphabet's cost is the sum of the costs of its clusters, better simplified alphabet costs are attained by optimizing the cluster sums. Hence, the overall fitness of both clusters is considered instead of optimizing a single cluster. Figure 4.1 shows an example of how a cluster is split in two.

Algorithm 3 describes the process of finding the closely related amino acids within a cluster. Steps 1 and 2 create two new clusters. Steps 4 through 9 calculate the overall fitness of adding an amino acid to each cluster. The amino acid is added to the cluster that results in the best overall fitness of the new clusters in steps 10 through 13.

Before creating a child solution from two parent solutions, each parent's clusters

```
SPLIT-CLUSTER(C)
 1  c1 ← New-Cluster
 2  c2 ← New-Cluster
 3  for i ← 1 to Length(C)
 4        do a ← C[i]
 5           Add-Acid(c1, a)
 6           x ← Fitness(c1) + Fitness(c2)
 7           Remove-Acid(c1, a)
 8           Add-Acid(c2, a)
 9           y ← Fitness(c1) + Fitness(c2)
10           Remove-Acid(c2, a)
11           if x < y
12              then Add-Acid(c1, a)
13              else Add-Acid(c1, a)
14
15
16  return (c1, c2)
```

Algorithm 3: The split cluster algorithm. The algorithm splits a cluster into two separate clusters by maximizing the overall fitness of the two new clusters.

Parents	Acid	Child
p_1:{adeg}{hklnpq}{crst}{fimvwy} p_2:{fimd}{alr}{eghknpc}{qstvwy}	a	
p_1:{hknpq}{cst}{fimvwy} p_2:{fim}{hknpc}{qstvwy}	c	{adeglr}
p_1:{q}{fimvwy} p_2:{fim}{hknpc}{qvwy}	f	{adeglr}{sthknp}
p_1:{q} p_2:{hknpc}{q}	h	{adeglr}{sthknp}{fimvwy}
p_1:{q} p_2:{q}	q	{adeglr}{sthknp}{fimvwy}{hknpc}
		{adeglqr}{sthknp}{fimvwy}{hknpc}

Figure 4.2: Creating new simplifications from parent simplifications. Each row represents a step in the process. At each step, an amino acid is randomly selected. The parent clusters containing this amino acid are merged together and placed in the child solution. In the fifth step, the child solution already has the required number of clusters, so the remaining amino acid is placed in the child cluster that results in the fittest solution.

are split with the SPLIT-CLUSTER algorithm. Clusters from the modified parent solutions are used to form clusters in the child solution in the following manner. An amino acid is randomly selected. The clusters from the parent solutions that contain the selected amino acid are merged into a single cluster and placed into the child solution. When the child solution contains all 20 amino acids, it is complete. Figure 4.2 shows an example of this process.

There are two special cases. First, each amino acid can only appear once in the child solution. When the parent's clusters are merged into a child solution, the amino acids in the new child cluster must be eliminated from further consideration. For example, suppose p_1 = {ar}{nq} and p_2 = {an}. Merging clusters {ar} and {an} results in the cluster {arn}, which is added to the child solution. The amino acids

a, r, and n can no longer appear in the child solution, but it still exists in the {nq} cluster in p_1. When {nq} is merged to form a cluster in the child solution, n must be omitted from the cluster. To ensure this is the case, all amino acids that appear in a new child cluster are eliminated from the parent's clusters. So in the previous example, after the {arn} cluster is created, amino acids a, r, and n are removed from p_1 and p_2s clusters. So p_1s {nq} cluster becomes {q}.

The next special case is ensuring the child solution has the required number of clusters. An issue occurs in two cases. The first is when the child solution has the required number of clusters, but does not contain all the amino acids. In this case, the missing amino acids are randomly selected one at a time and placed in the childs cluster that results in the fittest solution. The second case is when the child solution has all 20 amino acids, but not the required number of clusters. In this case, randomly selected clusters are split into two clusters using the Split-Cluster Algorithm until the required number of clusters is reached.

4.2.3 The Selection Function

Although purely random approaches to the simplification problem are ineffective, many of the genetic algorithms components rely on randomness. The selection operator, which is used to select parent solutions and solutions to be replaced, randomly selects population members. Initially, a weighted selection functions was used. It was more likely to select the most highly solutions in the population. The weighted selection function did not increase the fitness of the simplifications, so a purely random selection function is used for performance reasons.

4.2.4 The Mutation Operator

The genetic algorithm does not use a mutation operator. The crossover operator generates sufficiently diverse child solutions without mutation. An operator that randomly moved some of the amino acids to different clusters was created, but it did not result fitter solutions.

4.2.5 Putting it all Together

The algorithm begins by generating a random population of solutions, each with a specified number of symbols. The selection function randomly picks two solutions to act as parents. The crossover operator generates a child solution from the parent solutions. A randomly selected solution is replaced by the child solution. The algorithm goes back to the parent selection step, and the process continues for a specified number of generations. Upon completion, the fittest solution in the population is reported as the best alphabet found.

4.2.6 Results

The genetic simplification algorithm was implemented in order to test its effectiveness. The program accepts the number of symbols, or clusters, in the desired alphabet as an input parameter. This number is referred to as the alphabets size. The size can be any number between 1 and 20. Regardless of the size, the algorithm uses a population of 10 solutions and runs for 1000 generations. A major goal of the thesis is to create a genetic algorithm that quickly creates quality simplifications. To that end, the quality of the results is measured by the quality of the simplifications and

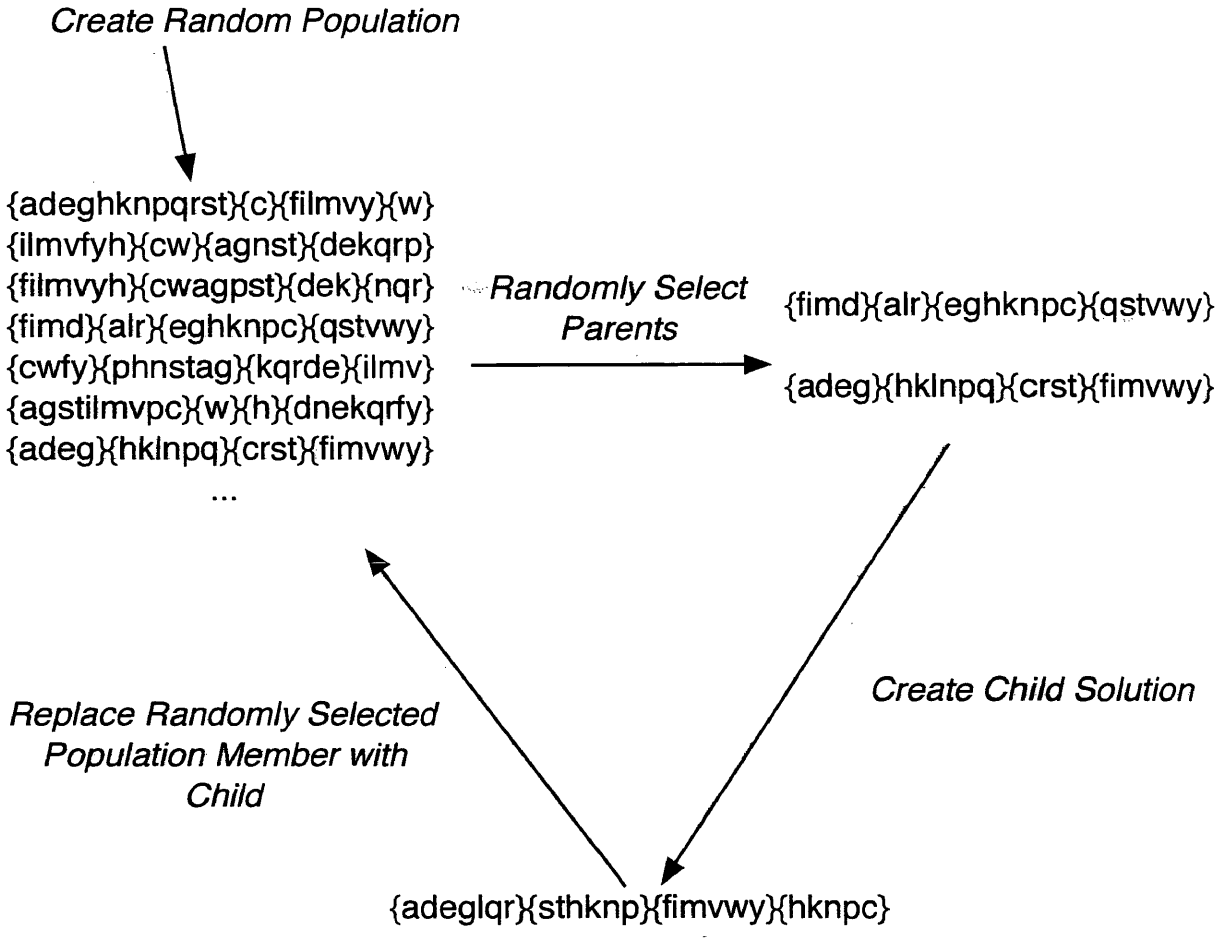


Figure 4.3: The entire genetic simplification algorithm. The algorithm begins by generating a population of random simplifications. Two simplifications are randomly selected as parents. From these solutions, a child simplification is created. This child simplification replaces a randomly selected simplification in the population. This process, excluding the initial population creation, is repeated for a specified number of iterations.

the algorithms running time.

It is useful to measure the alphabets generated by the simplification program against the optimal alphabets for the fitness function. Using a branch and bound algorithm [3] generated the best possible simplified alphabets using the evaluation method described in this study. Table 4.1 displays these alphabets along with the time taken to generate each alphabet. The branch and bound algorithm runs quickly for many sized alphabets, but for some alphabets take a long time to terminate. The longest running times are for alphabets of sizes 6 through 9. This is expected for alphabets of size 7, 8, and 9 since they contain largest number of possible alphabets. For example, there are 12×10^{12} possible size 9 alphabets, while only 5.8×10^{12} possible size 6 alphabets, which is much smaller than the possible 7, 8, and 9 alphabets.

Table 4.2 shows the alphabets generated by the algorithm. Since the algorithm is partially driven by randomness, a different alphabet is possible each time the algorithm is executed. Hence, measuring the quality of the genetic algorithms alphabets is best done by repeatedly running the algorithm and calculating the average cost of the best simplified alphabet found after each iteration. For each alphabet size, the table lists the best simplified alphabet found and its cost, along with the average cost of the best alphabet and the average time taken to generate the solution. The average cost and time were measured over 200 iterations of the algorithm.

On average, the genetic algorithm produces simplified alphabets that are close to the optimal alphabet for alphabets of all sizes. Regardless of size, the difference between an optimal alphabets cost and the genetic algorithms alphabet cost is 5 or less. For any sized alphabet, the genetic algorithm takes 1 second to complete. This represents a major performance improvement over the branch and bound algorithm.

Size	Best Possible Alphabet	Cost	Time
2	acdeghknpqrst,filmvwy	160.462	00:00:03
3	filmvwy,c,adeghknpqrst	140.583	00:11:12
4	w,adeghknpqrst,filmvy,c	120.917	02:42:00
5	filmvy,dehknqr,agpst,c,w	105.705	08:46:07
6	filmvy,h,c,w,agpst,deknqr	92.967	13:44:52
7	adgnst,ekqr,c,h,filmvy,p,w	81.000	11:57:31
8	adgnst,ekqr,c,h,ilmv,fy,p,w	69.167	05:42:47
9	agst,ilmv,p,c,w,h,dn,ekqr,fy	58.500	02:03:10
10	agst,kr,dn,c,eq,h,ilmv,fy,p,w	50.000	00:47:05
11	p,dn,g,w,ast,kr,ilmv,fy,h,c,eq	41.667	00:14:00
12	ast,kr,dn,c,eq,g,h,iv,lm,fy,p,w	35.167	00:04:13
13	ast,kr,n,d,c,eq,g,h,iv,lm,fy,p,w	28.667	00:01:07
14	ast,kr,n,d,c,q,e,g,h,iv,lm,fy,p,w	23.167	00:00:17
15	a,kr,n,d,c,q,e,g,h,iv,lm,fy,p,st,w	18.000	00:00:03
16	a,kr,n,d,c,q,e,g,h,iv,lm,f,p,st,w,y	13.000	00:00:00
17	a,r,n,d,c,q,e,g,h,iv,lm,k,f,p,st,w,y	8.500	00:00:00
18	a,r,n,d,c,q,e,g,h,iv,lm,k,f,p,s,t,w,y	5.000	00:00:00
19	a,r,n,d,c,q,e,g,h,iv,l,k,m,f,p,s,t,w,y	1.500	00:00:00

Table 4.1: The best possible simplified alphabets, as generated by the branch-and-bound algorithm.

Size	Best Possible Alphabet	Best Alphabet's Cost	Avg. Cost	Avg. Time
2	acdeghknpqrst,filmvwy	160.462	160.556	00:00:01
3	filmvwy,c,adeghknpqrst	140.583	141.561	00:00:01
4	w,adeghknpqrst,filmvy,c	120.917	124.42	00:00:01
5	filmvy,dehknqr,agpst,c,w	105.705	109.871	00:00:01
6	filmvy,h,c,w,agpst,deknqr	92.967	96.477	00:00:01
7	agpst,ilmv,fy,deknqr,c,w,h	81.133	84.585	00:00:01
8	ilmv,fy,h,c,w,agnst,dekqr,p	69.300	74.232	00:00:01
9	agst,ilmv,p,c,w,h,dn,ekqr,fy	58.500	63.900	00:00:01
10	c,w,fy,p,h,nst,ag,kqr,de,ilmv	50.000	54.990	00:00:01
11	p,dn,g,w,ast,kr,ilmv,fy,h,c,eq	41.667	46.741	00:00:01
12	fy,c,ast,kr,p,eq,w,g,d,ilmv,h,n	35.167	39.637	00:00:01
13	ast,kr,n,d,c,eq,g,h,iv,lm,fy,p,w	28.667	32.830	00:00:01
14	ast,kr,n,d,c,q,e,g,h,iv,lm,fy,p,w	23.167	27.053	00:00:01
15	a,kr,n,d,c,q,e,g,h,iv,lm,fy,p,st,w	18.000	20.823	00:00:01
16	a,kr,n,d,c,q,e,g,h,iv,lm,f,p,st,w,y	13.000	15.615	00:00:01
17	a,r,n,d,c,q,e,g,h,iv,lm,k,f,p,st,w,y	8.500	10.717	00:00:01
18	a,r,n,d,c,q,e,g,h,iv,lm,k,f,p,s,t,w,y	5.000	6.095	00:00:01
19	a,r,n,d,c,q,e,g,h,iv,l,k,m,f,p,s,t,w,y	1.500	2.255	00:00:01

Table 4.2: Best genetic algorithm generated simplified alphabets. The genetic algorithm generates simplifications more quickly than the branch and bound algorithm. The simplifications at least near optimal for all alphabet sizes.

The genetic algorithm can quickly generate alphabets of near optimal quality. Of particular importance are the alphabets of sizes 6 through 9. For these sizes, the genetic algorithm on average produces an alphabet within 5 cost points of the optimal solution. However, during the 200 executions used to generate the average, the genetic algorithm found the optimal simplification for sizes 6 and 9, and a simplification within a few tenths of a point from optimal for sizes 7 and 8. In fact, for any alphabet size the genetic algorithm found either the optimal solution or a solution extremely close to optimal.

4.2.7 Traditional Cluster Metrics

Although the genetic algorithm finds an optimal or near optimal alphabet in a few seconds, the method used to evaluate the alphabet's quality has not been shown to be effective in a biological application [3]. With this in mind, it is useful to evaluate the alphabet's amino acid clusters using general clustering metrics. In other words, are the distances between each cluster's amino acids small and the distance between each cluster large? To measure this, an algorithm that calculates the distance between amino acids according to their evolutionary substitution rate must be defined. Pair-wise substitution matrices report how likely a pair of amino acids are substituted for each other by assigning each possible pair a numeric value where the higher the number, the more like substitution will occur. This number can be used as a measure of distance between two amino acids. Pairs of amino acids having high pair-wise alignment scores should be clustered together and pairs having low scores should be in different clusters. Hence, the distance between amino acids in each cluster should be large and the distance between amino acids in different clusters should be small. This is a break with tradition, where large distances represent dissimilarity, but it

achieves the same result.

Since the pair-wise alignment score of two amino acids is used as a distance metric, the quality of an amino acid cluster can be represented as the average pair-wise alignment score of all possible amino acid pairs in the cluster. Algorithm 4 describes this process. Figure 4 receives a cluster as input and returns the average pair-wise alignment score of all possible pairs in the cluster. The procedure Score reports the pair-wise alignment score of two amino acids. In substitution matrices, a pair-wise alignment value of zero represents a division between pairs likely to be substituted for each other and pairs not likely to be substituted for each other. Thus, an average pair-wise score above zero is an indicator of a quality clustering.

```
INTERNAL-DISTANCE(C)
1  total ← 0
2  count ← 0
3  for i ← 1to Length(C)
4        do for j ← 2to Length(C)
5              do total ← total + Score(C[i], C[j])
6                 count ← count +1
7  dist ← total ÷ count
8  return dist
```

Algorithm 4: The internal cluster distance algorithm.

The distance between two clusters can be represented as the average pair-wise score between all possible amino acids pairs where one each amino acid in the pair comes from a different cluster. Algorithm 5 describes this process. An average pair-wise score below zero is an indicator that the amino acids in the different clusters tend not to be substituted for each other.

```
CLUSTER-DISTANCE(C, D)
1  total ← 0
2  count ← 0
3  for i ← 1to Length(C)
4        do for j ← 2to Length(D)
5              do total ← total + Score(C[i], C[j])
6                 count ← count +1
7  dist ← total ÷ count
8  return dist
```

Algorithm 5: The cluster distance algorithm.

Tables 4.3 and 4.4 depict the distances results for an optimal four-symbol alphabet and a four-symbol alphabet produced by the genetic algorithm. The pair-wise alignment scores are taken from the BLOSSUM40 scoring matrix. The values along the table's diagonal are the distance values for a particular cluster. The other values are the distance between two clusters. The table is symmetric, so redundant values are omitted.

-	{adeghknpqrst}	{c}	{filmvy}	{w}
{adeghknpqrst}	-0.50	-2.66	-2.20	-3.25
{c}	-	16.00	-2.83	-6.00
{filmvy}	-	-	1.33	-0.80
{w}	-	-	-	19.00

Table 4.3: 4-symbol alphabet distances

The two alphabets are similar. Two of the four clusters appear in both alphabets. The difference is the genetic algorithm has split the remaining amino acids into two

-	{adeghknpqrst}	{ilmv}	{fwy}	{c}
{adeghknpqrst}	-0.50	-2.20	-2.55	-2.66
{ilmv}	-	2.16	-0.50	-2.75
{fwy}	-	-	2.66	-4.00
{c}	-	-	-	16.00

Table 4.4: 4-symbol genetic algorithm distances

clusters and the optimal solution has all but one of the remaining amino acids in one cluster. The weakest cluster, {adeghknpqrst} appears in both alphabets. It has an average distance of -0.5, and although it is only slightly below zero, it is an indicator that it contains mismatched amino acids. All other clusters in both alphabets have scores above zero, so they appear to be of good quality. Additionally, The distance between different clusters is below zero in both alphabets. In the genetic algorithm's alphabet, the average distance between clusters {ilmv} and {fwy} is -0.50, a value close to zero. This could indicate that some of the amino acids in these clusters should be swapped. Similarly, in the optimal alphabet the average distance between clusters {filmvy} is -.80. It may not be possible to achieve flawless clustering for highly simplified alphabets. Although both alphabets have flaws, they appear to be good by traditional clustering metrics.

Chapter 5

Predicting Protein Interactions

The ultimate measure of a simplified alphabet's quality is its ability to function in a biological domain. To validate the quality of the simplifications produced by the genetic algorithm, this study applies simplified alphabets to the protein interaction problem. The genetic algorithm measures a simplification's fitness by the evolutionary similarity of its amino acids clusters. Similarly, this study uses the evolutionary similarity of amino acids to predict protein interactions. The key idea is that even though amino acids are substituted for each other over time, some aspect relevant to protein interaction was conserved. If this is the case then simplified alphabets can capture information essential to interactions. Hence, the genetic algorithms simplified alphabets may provide insight into protein interactions.

This study takes a statistical approach to the prediction problem, using features of the studies described in [8] and [14]. Over-represented protein features found in a set of known interactions are used to predict interactions occurring in the same domain. Rather than focusing on a specific protein feature, proteins are characterized

by the length-4 amino acid subsequences in their primary structure. For example, a protein with a primary structure of *arndcqe* has 4 length-4 subsequences: *arnd*, *rndc*, *ndcqe*, and *dcqe*. Pairs of protein blocks that occur at a higher than expected rate in known interactions can be used to predict unknown interactions by looking for pairs of proteins where one protein contain one block, and the other protein contains the other block.

The use of simplified alphabets reduces the complexity of this approach. For example, using a four symbol simplified alphabet, there are $4^4 = 256$ possible length-4 amino acid sequences and $256 \times 256 = 65,536$ possible subsequence pairs. With no simplification, there are $20^4 = 160,000$ length-4 amino acid sequences and $160,000 \times 160,000 = 25,600,000,000$ possible subsequence pairs. Other relevant complexity reductions are discussed in the forthcoming sections.

Predicting protein interactions is done through a four-step process:

1. Generate a simplified alphabet with the genetic algorithm
2. Represent all the protein with the simplified alphabet
3. Find the highly occurring pairs of length-4 subsequences in the training set
4. Predict new interactions with the highly occurring subsequence pairs

5.1 The Data Set

This study uses a set of known protein-protein interactions in *Saccharomyces cerevisiae* [15]. The interactions were identified through laboratory experiments using

high-throughput screening procedures. The set consists of 1003 interactions between 981 unique proteins. 500 of these interactions are randomly selected as the training set, and the algorithm attempts to predict the remaining 503 interactions. This data set is also used by [8] and [14], so their results provide a good baseline to measure this studys results. The proteins were obtained from the Munich Information Center for Protein Sequences (MIPS) public Internet site.

5.2 Representing Proteins with a Simplified Alphabet

The primary structure of all the proteins in the data set are represented using a 4-symbol simplified alphabet generated by the genetic algorithm. Since the genetic algorithm is driven by randomness, its simplified alphabets may be different each time it is executed. Given a simplified alphabet, translating the proteins primary structure is done though a two-step process. First, each of the simplified alphabets clusters is assigned a character. Second, all amino acid characters in each proteins primary structure is replaced with the character representing the cluster in which it appears.

Since the clusters making up a simplified alphabet have no ordering, the clusters in a simplified alphabet may be assigned different character depending on how it appears in the simplified alphabet. For example, suppose in a 4 symbol alphabet the first cluster is assigned A, the second B, the third C, and the fourth D. Then given the simplified alphabet $\{ailmv\}\{fwy\}\{c\}\{deghknpqrst\}$, $\{ailmv\}$ is assigned A, $\{fwy\}$ is assigned B, $\{c\}$ is assigned C, and $\{deghknpqrst\}$ is assigned D. Sup-

pose the same simplified alphabet has its clusters appearing in a different order, say $\{c\}\{ailmv\}\{deghknpqrst\}\{fwy\}$. Then $\{c\}$ is assigned A, $\{ailmv\}$ is assigned B, $\{deghknpqrst\}$ is assigned C, and $\{fwy\}$ is assigned D.

5.3 Extracting Subsequence Pairs

The first step in extracting the subsequence pairs is to find the subsequences in each protein. Each protein will have $(n - 4) + 1$ subsequences, where n is the length of the proteins primary sequence. On average, the primary structure of a protein from the training set has 425 amino acids, resulting in 422 possible subsequences. Storing the actual substrings occurring in each protein and performing calculations on substrings can be a time and storage burden on the algorithm. To avoid this overhead, the algorithm takes advantage of the fact that there are only 256 possible length-4 substrings. The algorithm assigns each possible length-4 subsequence a unique number between 1 and 256. Each protein contains a binary array with 256 entries, where each entry reports if the protein contains a particular subsequence. An array entry contains a 1 if the protein has the subsequence and a 0 if it does not.

Algorithm 6 describes this process. Steps 1 through 3 initialize the binary subsequence arrays elements to indicate the subsequence does not exist in the protein. The loop beginning in step 4 records each subsequence appearing in the protein. Step 5 extracts the length-4 subsequence. Step 6 retrieves the index assigned to the subsequence. Finally, step 7 uses the subsequences index to indicate the protein contains the subsequence.

The occurrences of subsequence pairs are stored in a 256×256 It would be difficult

```
EXTRACT-SUBSEQUENCES(S)
1  subsequences ← []
2  for i ← 1 to 256
3        do subsequences[i] ← 0
4  for i ← 1 to length(S) − 4 + 1
5        do sub ← SUBSEQUENCE(S, i, i + 4)
6           index ← SUB-SEQUENCE-INDEX(sub)
7           subsequences[index] ← 1
8  return subsequences
```

Algorithm 6: The extract subsequences algorithm.

record the occurrences in this manner with a standard alphabet as there would be $160,000 \times 160,000 = 25,600,000,000$ The binary arrays of each pair of interacting proteins in the training set are examined to find their subsequence pairs. Algorithm 7 describes the process of recording the subsequence pairs occurring in each interacting protein pair. It accepts the binary arrays from an interacting protein pair along with the occurrence table. The algorithm simply iterates through each binary array and records a match whenever both arrays report the subsequence occurs. Algorithm 7 requires $O(n^2)$ steps, where n is the size of the binary arrays. Since the binary arrays have 256 entries, recording the pair occurrences of a single pair of proteins takes on the order of $256 \times 256 = 65,536$ operations. Running the algorithm on a 500 pair training set requires on the order of $65,536 \times 500 = 32,768,000$ operations. Running the algorithm on a 500 pair training set requires on the order of $160,000 \times 160,000 \times 500 = 12,800,000,000,000$ operations.

```
RECORD-SUBSEQUENCE-PAIRS(p1, p2, table)
1  for i ← 1 to 256
2        do for j ← 1 to 256
3              do if p1[i] = 1 and p2[j] = 1
4                    then table[i][j] ← table[i][j] + 1
```

Algorithm 7: The record pairs algorithm.

5.4 Identifying High Occurrences of Subsequence Pairs

This study uses the method employed by [14] to find over represented subsequence pairs. In their study, correlated sequence-signature pairs occurring at a higher than expected rate are found by calculating the ratio of the actual and expected frequencies of the sequence-signature pairs. Similarly, this study finds subsequence pairs occurring at a higher than expected rate by calculating the ratio of the actual and expected frequencies of all the sequence-signature pairs appearing in the training set. This ratio is expressed as a log odds value. The log odds value is computed as $\log_2(\frac{F_{ij}}{F_i F_j})$ where F_{ij} is the observed frequency of a subsequence pair containing sequences i and j, F_i is the observed frequency of subsequence i, and F_j is the observed frequency of subsequence j. All log odds values over a particular threshold are marked as higher than expected.

Most subsequence pairs have a log odds threshold below zero. In fact, on average only about 1000 pairs will have a value greater than zero. Table 5.1 contains

Threshold	Avg. High occurrence Pairs
2	121
1	292
0	751
-1	15,241
-2	23,471

Table 5.1: Average high occurrence subsequence pairs per threshold

the average number of high occurrence subsequence pairs over a particular log odds threshold. The number of highly occurring subsequence pairs drops significantly when the threshold is moved from 1 to 0.

5.5 Predicting New Interactions

This study proposes that highly occurring subsequence pairs in known interacting proteins represented with simplified alphabets can be used to predict unknown interactions. This idea is explored by using the highly occurring subsequence pairs in the training set to predict the remaining 503 known interactions in the data set. New interactions are predicted by searching for protein pairs where one protein contains one of the highly occurring subsequences and the other protein contains the other highly occurring subsequence. That is, if (s_1, s_2) is a highly occurring subsequence pair, then if protein P_1 contains subsequence s_1 and protein P_2 contains subsequence s_2, then P_1 and P_2 are predicted to interact.

Algorithm 8 describes the prediction process. For input, the algorithm takes a set of subsequence pairs and a set of proteins. For each subsequence pair in the input

```
PREDICT-INTERACTIONS(H, S)
 1  p ← {}
 2  for i ← 1 to length(H)
 3        do done ← false
 4           while not done
 5                 do x ← protein-with-subsequence(H[i].first, S)
 6                    y ← protein-with-subsequence(H[i].second, S)
 7                    if x ≠ nil and x ≠ y
 8                       then add-pair(x, y, p)
 9                            remove(S, x)
10                            remove(S, y)
11                       else done ← true
12  return p
```

Algorithm 8: The prediction algorithm.

set, pairs of proteins from the data set are selected where one protein contains one of the subsequences and the other protein contains the other subsequence. Step 1 of the algorithm initializes the prediction set to empty. Beginning at step 2, a subsequence pair from the input set is selected, and pairs of proteins from the data set containing the subsequence pairs are selected and added to the prediction set. Once two proteins are predicted to interact, they are removed from further consideration. When no pairs of proteins in the data set contain the subsequence pairs, the next subsequence pair is selected. When there are no more subsequence pairs to consider the algorithm terminates.

5.6 Results

Table 5.2 contains the effectiveness of predicting interactions using various log odds thresholds. Since randomness of a component of the genetic algorithm and translating the proteins primary sequence using the simplified alphabet, the algorithms results vary. For each threshold in the table, averaging 100 runs of the prediction algorithm generated the values.

Threshold	Avg. Pred. Made	Avg. Correct	Avg. Predicted
2	43.20	18.10%	3.70%
1	92.00	42.40%	7.80%
0	111.60	50.10%	11.10%
-1	498.80	58.60%	57.10%
-2	503	82.00%	82.00%

Table 5.2: Prediction results

The effectiveness decreases in both accuracy and percentage of total interactions predicted as the log odds threshold decreases. The algorithm is most effective when using slightly lower than expected sequence signature pairs. A reason for this may be the low number of sequence signature pairs above higher thresholds. Most of the training sets sequence signature occur at a lower than expected rate. Even so, these sequence signature pairs appear to have effectiveness in predicting interactions.

Table 5.3 shows more details on the predictions made using log odds threshold of 2. The correct percentage of predictions varies from 100% to 54%. This demonstrates the algorithm may need to be run repeatedly in order to give the best results. When predicting unknown interactions, it is not possible to know when the best re-

sult is reached, so repeatedly running the algorithm would not be effective. This indicates the algorithm may best utilized by generating sets of possible interactions that laboratory methods can focus on.

Avg. Predictions	Avg. Correct	High Correct	Low Correct
500	410	500	270

Table 5.3: Prediction results with threshold over -2

Table 5.4 depicts the simplified alphabets most effective in predicting interactions. The third row in the table contains a simplified alphabet with a minimal cost, but this alphabet is not the most effective in making predictions. Simplified alphabets close to, but not equal to, the minimal cost seem to be more effective in making predictions.

Alphabet	Cost	Correct Predictions	Percentage Predicted
{ailmv}{fwy}{c}{deghknpqrst}	125.30	97.80%	97.20%

Table 5.4: Simplified alphabets effective in predicting interactions

The accuracy of the algorithms predictions compares favorably with other computational methods. The study conducted by (Sprinzak et al, 2001) was able to predict 94% of the interactions in the training set using correlated sequence signature pairs as an interaction indicator. Even though this study attempts to predict interactions outside the training set, it is able to come within 10 percentage points [14] in general, and specific simplified alphabets consistently predict at least 94% of the predictions.

Chapter 6

Conclusion

Improving the amount and quality of information that can be extracted from biological sequences is a main challenge in bioinformatics. These challenges may be met by building on multiple small concepts rather than through large-scale breakthroughs. This is especially true of the protein interaction problem. No single computational or laboratory methods can comprehensively detect all, or even most, cellular protein interactions. A comprehensive listing of all cellular interactions may require multiple prediction methods. Additionally, it is known that simplified amino acid alphabets can provide insight into microbiological problems, but the best methods for creating simplified alphabets is unknown. Research in these problem domains could provide insights leading to a better understanding of cellular processes.

This study describes how two concepts can be used in conjunction to gain further insight into the bioinformatics field. Specifically, how simplified alphabets can be used to predict protein interactions. The simplification algorithm described and implemented in this study demonstrate that genetic algorithms can efficiently cre-

ate optimal or near optimal simplified amino acid alphabets. Implementation results show that the time required to generate a simplification can be reduced from several hours to a second with little loss in quality. Furthermore, the genetic algorithm shows that randomness and heuristics can lead to quality simplifications.

In general, simplified alphabets can potentially reduce the complexity of some biological problems. Previous studies indicate that rather than using a specific protein feature to predict protein interactions, characterizing proteins by the subsequences of their primary structures has predictive value. This study demonstrates how simplified alphabets reduce the complexity of using amino acid subsequences for prediction. Additionally, initial results indicate the genetic algorithm's simplifications can be useful in predicting protein interactions.

These results point to paths of future research for both predicting protein interactions and simplified alphabets. For predicting protein interactions, several examples arise. First, since short subsequences of amino acids seem to be useful as an interaction indicator, certain amino acid patterns may occur with high than expected frequency. Analyzing these patterns could uncover key prediction data. Second, proteins tend to interact as part of a structural motif. In these motifs, the amino acids along one edge of the structure make direct contact. This implies amino acids at specific positions within a sequence are an important factor in interactions. Applying simplified alphabets to amino acids at key positions in a protein may lead to better predictions. Also, it is unknown whether the prediction process presented in this study is reversible. That is, can a simplified alphabets be derived from a training set of protein interactions. Finally, it is unknown whether interactions have transitive properties. That is, if protein a interacts with protein b, and protein b interacts with protein c, will a interact with c? For creating simplified alphabets, general simplifica-

tion heuristics are not known. The heuristic used to generate simplified alphabets is effective for simplifications based on the evolutionary similarity of amino acids, but it is not known if is effective in general.

Bibliography

[1] C.M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

[2] Philip Bourne and Helge Weissing. *Structural Bioinformatics.* Wiley, 2003.

[3] Nicola Cannata, Stefano Toppo, Chiara Romualdi, and Giorgio Valle. Simplifying amino acid alphabets by means of a branch and bound algorithm and substitution matrices. *Bioinformatics*, 18(8):1102–1108, 2002.

[4] Thomas Corman, Charles Leiserson, and Ronald Rivest. *Introduction to Algorithms.* McGraw Hill, 1994.

[5] S.R David. Look up this article. *Nature Structural Biology*, 4, 1997.

[6] Richard Durbin, Sean Eddy, Anders Krogh, and Graeme Mitchison. *Biological Sequence Analysis.* Cambridge, 2000.

[7] Eldon Enger, J. Kormelink, Frederick Ross, and Rodney Smith. *Concepts in Biology.* Wm. Brown, 7 edition, 1994.

[8] Shawn Gomez, William Noble, and Andrey Rzhetsky. Learning to predict protein-protein interactions from protein sequences. *Bioinformatics*, 19(15):681–692, 2003.

[9] John Ingraham and Catherine Ingraham. *Introduction to Microbiology.* Brooks Cole, 2000.

[10] Ian Korf, Mark Yandall, and Joseph Bedell. *BLAST.* O'Reilly, 2003.

[11] George Lugar and William Stubblefield. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving.* Addison Wesley, 3 edition, 1998.

[12] Melanie Mitchel. *An Introduction to Genetic Algorithms.* MIT Press, 1996.

[13] Jang Qian, Jimmy Lin, Nicholas Luscombe, Hianyuan Yu, and Mark Gerstein. Prediction of regulatory networks: genome wide identification of transcription factor targets from gene expression data. *Bioinformatics*, 19(15):1917–1926, 2003.

[14] Einat Sprinzak and Hanah Margalit. Correlated sequence-signatures as markers of protein- protein interaction. *Journal of Molecular Biology*, 311:681–692, 2001.

[15] P. Uetz, L. Giot, G. Cagney, T.A. Mansfield, RS Judson, D. Lockshon Knight JR, V. Narayan, M. Srinivasan, P. Pochart, A. Qureshi-Emili, Y. Li, B. Godwin, D. Conover, T. Kalbfleisch, G. Vijayadamodar, M. Yang, M. Johnston, S. Fields, and J.M. Rothberg. A comprehensive analysis of protein-protein interactions in saccharomyces cerevisiae. *Nature*, 403(6770), 2000.

[16] Christian von Mering, Roland Krause, Bernd Snel, Michael Cornell, Stephen G. Oliver, Stanley Fields, and Peer Bork. Comparative assessment of large-scale data sets of protein protein interaction. *Nature*, 417:399–403, 2002.

[17] Jun Wang and Wei Wang. A computational approach to simplifying the protein folding alphabet. *Nature Structural Biology*, 6(11), 1999.