Computational Tools for Including Specificity in Protein Design

by

Daniel J. Park

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2002

Author .... [illegible] ....
Department of Electrical Engineering and Computer Science
May 24, 2002

Certified by .... [illegible] ....
Amy E. Keating
Assistant Professor of Biology
Thesis Supervisor

Accepted by .... [illegible] ....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Computational Tools for Including Specificity in Protein Design

by

Daniel J. Park

Submitted to the Department of Electrical Engineering and Computer Science
on May 24, 2002, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Research in computational protein design has produced a number of algorithms that are empirically effective for selecting amino acid sequences that fold to desired protein structures. In comparison, the problem of designing proteins for desired functionality (e.g. binding specificity) has not been as well explored. This M. Eng. thesis aims to develop software for this purpose. Algorithms for sequence selection have been adapted from known algorithms for designing fold stability. The result of the thesis is a computational toolkit for biologists who wish to reengineer the structure and properties of protein folds or interfaces.

Thesis Supervisor: Amy E. Keating
Title: Assistant Professor of Biology

Acknowledgments

Several individuals have contributed greatly to the work of this thesis:

- **Amy Keating** provided the motivation, guidance, funding, and relevant background material for this project, as well as feedback for this written document.
- **Jiangang Chen** provided the computational chemistry work and produced and refined the energy function data sets that served as program input. Any biological applicability of this software is a result of his efforts.

Contents

List of Figures

Chapter 1

Introduction

Proteins are functional molecular units in biological organisms [2, 5]. They manufacture other proteins, regulate and replicate DNA, transport objects within and between cells, selectively catalyze biochemical reactions, convert various forms of energy, and perform other complicated tasks. They are the focus of much research in biology.

Proteins are composed of amino acid residues, of which there are twenty different kinds. All amino acids contain a backbone component and a variable "side chain". The backbone component is the same in every amino acid. At the simplest level, a protein is just a strand of connected amino acids and can be uniquely identified by the sequence of the side-chain identities.

However, proteins are never linear in nature. Instead, the strand packs into a three dimensional structure that is energetically more stable in a process called protein folding. Both the backbone and side-chain groups have conformational flexibility. Folded structures can combine with other structures to form larger complexes. For a fixed environment, the folded structure is entirely a function of the sequence of amino acid side chains in the original strand [1]. A change in the original sequence often results in a structural and/or functional change. Depending on the mutation, the change may be insignificant or drastic.

The "protein folding problem," predicting what structure a given sequence will fold to, is computationally difficult because of the sheer number of possible three dimensional structures. The reverse problem, the "protein design problem," is also computationally hard. This is the problem of selecting amino acid sequences that fold to desired proteins. This thesis explores and extends the research that has been done on the protein design problem.

The protein design problem is both theoretically interesting and biologically applicable. Reengineering proteins can potentially allow researchers to alter the binding properties, and thus, the functionality, of gene regulators, antibodies, cell receptors and inhibitors. Consequently, protein design has the potential to aid the discovery of new therapeutics as well.

1.1 Thesis Structure

This chapter will cover background on the computational protein design problem, discuss issues related to specificity design and state the motivation for this thesis.

Chapter 2 discusses the design and implementation of the software. Chapter 3 discusses testing and validation.

1.2 Problem Introduction

1.2.1 Protein Design Difficulties

The goal in protein design is to select an amino acid identity and side-chain conformation at each position of the protein. This specifies both the sequence and the structure of the design. The protein design problem is difficult for a number of reasons:

- **Many possible sequences** - algorithms tend to scale poorly, as the search space of amino acid combinations scales exponentially.
- **Many possible folds** - the fold of a protein is determined by the backbone conformation. The number of possible three dimensional folds for a given sequence is infinite. The calculation must identify a sequence that favors one desired fold over all others.
- **Many possible side-chain conformations** - amino acids contain flexible side chains that can pack into many different conformations. We must account for these conformations when attempting to calculate energies, but again, this increases the combinatorial size of the problem.
- **Energy functions** - our calculation of the energies of structures is approximate.
- **Folding kinetics not well understood** - our understanding of the biophysics behind the process by which a protein folds is imperfect.
- **Objective function** - it is not entirely clear what we want to optimize, quantitatively. For naturally occuring proteins, evolution optimizes a very complex function (one that includes stability, biological function, lack of biological harmfulness, etc).

1.2.2 A Framework for Computational Protein Design

To make computational protein design tractable, it is common to simplify the problem in a number of ways. If we use a fixed backbone fold and select amino acids and side-chain conformations to optimize the energy of this single target structure, we no longer have to worry about issues of alternate folds, folding kinetics or complicated objective functions. This simplified problem is more computationally manageable and has produced successful designs in recent years.

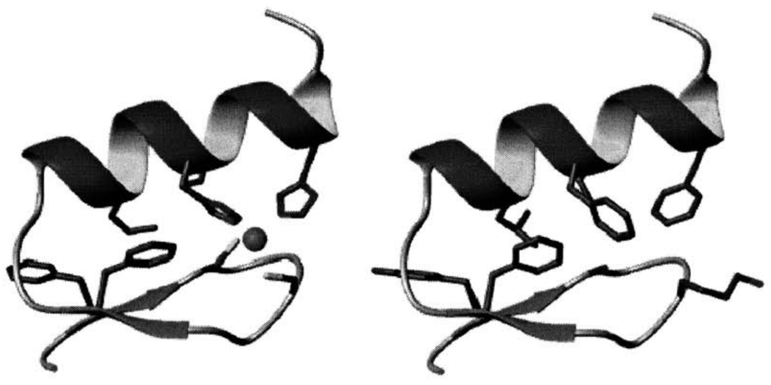

Figure 1-1: Zinc finger design. Left: wild-type (PDB code: 1ZAA). Ball depicts Zn^{2+} at core. Right: redesigned protein.

A now-classic example of this approach involves the redesign of a small zinc finger.[7] This is a protein that does not fold in the absence of zinc. Dahiyat & Mayo selected and packed a new sequence of amino acids onto to the original protein backbone. The newly designed protein had a structure very similar to that of the original target, as determined by NMR spectroscopy, even in the absence of zinc.

The "side-chain packing" problem emerges as a special case of protein design. In addition to assuming a fixed backbone fold, we also assume a fixed amino acid sequence. The problem that remains is determining the side-chain conformations—a subproblem of the protein design problem that corresponds to protein side-chain structure prediction.

1.2.3 Modeling Assumptions

Some additional assumptions we typically make:

Water molecules removed - in nature, proteins exist in some environment that affects their folding. Most often that environment is mostly water. Because modeling hundreds or thousands of small water molecules in highly variable positions is computationally expensive, we cannot include them explicitly in the model.

Rotamers - the torsion angles between atoms in the amino acid side chains can be used to specify side-chain geometry. These are allowed to have any value. Since searching an infinite continuum of torsion angles is computationally infeasible, we restrict the torsion angles to a small number of discrete values, resulting in a finite number of side-chain conformations at any position. Each of these conformations is called a "rotamer." In nature, torsion angles are found to be biased towards a small number of finite angles [35]. These are the angles we choose for our "rotamer library."

Linear and pairwise energy functions - Many search algorithms heavily exploit the ability to decompose an energy function describing the protein in a linear and pairwise manner. Define $E_k(a_1, \dots, a_n)$ as the energy of the sequence $a_1, \dots, a_n$ on backbone k. We would like to decompose it in this way:

$$E_k(a_1, \dots, a_n) = \varepsilon_k + \sum_i \varepsilon_k(i_r) + \frac{1}{2} \sum_i \sum_{j \neq i} \varepsilon_k(i_r, j_s) \tag{1.1}$$

where ε_k is the energy between backbone atoms and fixed residues that are not being designed, $\varepsilon_k(i_r)$ is the interaction energy between a selected rotamer r_i at position i with the backbone atoms and fixed residues, and $\varepsilon_k(i_r, j_s)$ is the interaction energy between two selected rotamers at distinct positions i and j.

All these assumptions and approximations serve to make the problem more tractable computationally.

1.2.4 Energy Function

We must choose a representation of the potential energy of a particular state, given all of the selected amino acids and side chain positions. Energy functions that accurately model the physical system typically include contributions from van der Waals interactions, electrostatic forces, torsion angles, bond lengths and bond angles. This becomes further complicated when corrections need to be made for various modeling assumptions. In particular, in order to accurately describe electrostatics in a polar solvent without modeling each water molecule, approximate solvation energy terms must be introduced. Also, some energy terms cannot be easily decomposed in a linear or pairwise manner and doing so would itself be an approximation [14].

The Keating Lab and others are developing effective energy functions that correct for the modeling assumptions. However, research in this area is not within the scope of this thesis.

1.2.5 Objective Function

What exactly do we want to optimize and how do we want to quantify it? If we assume that a single target state is being designed for, then we want to select the amino acids that maximize the probability of occupying the target state. A 2-D representation of the rough energy surface that characterizes protein fold space is shown in Figure 1-2.

The most physically justifiable objective function to maximize is P_{k_0}, the probability of occupying the desired state k_0 given the selected sequence. This probability can be computed from the energy function via the Boltzman relation:

$$\max P_{k_0},\ P_{k_0} = \frac{e^{-\beta E_{k_0}}}{\sum_k e^{-\beta E_k}} \tag{1.2}$$

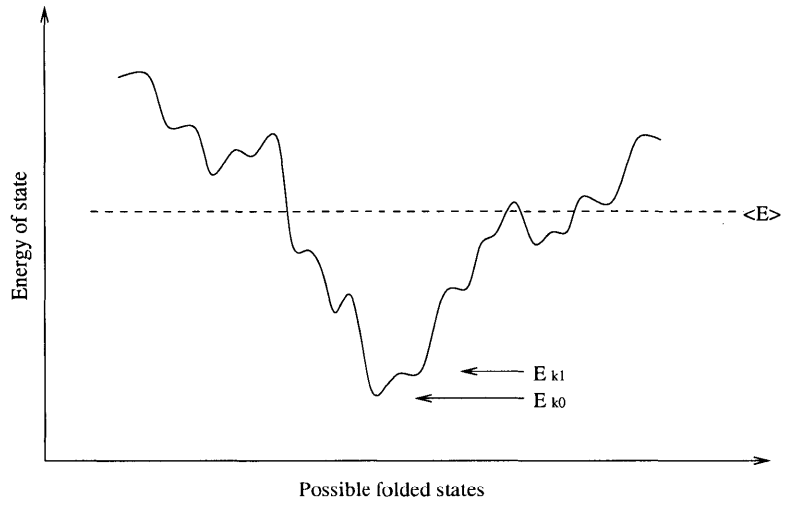

Figure 1-2: Example energy landscape for a given amino acid sequence

where the summation occurs over the set of *all* possible states. This summation is dominated by two major components:

- A small number of low energy states - the exponential exaggerates energy differences to the extent that the terms corresponding to low energy states are much larger than those corresponding to high energy states.
- A large number of high energy (unfolded) states - since there are many more high energy states, the sum of these states contributes to the summation as well.

The non-linearity of the objective function in (1.2) can present problems. We can try to make it appear more linear by optimizing the negative log:

$$\min(-\log P_{k_0})/\beta,\ (-\log P_{k_0})/\beta = E_{k_0} + \frac{1}{\beta}\log\sum_k e^{-\beta E_k} \tag{1.3}$$

Unfortunately this objective function is still non-linear, which presents problems for some search algorithms. Even worse, it requires an enumeration through all possible states. Saven reviews a number of metrics [36] that simplify this function:

1. $\min E_{k_0}$ = the potential energy of the desired state. Advantages: easy to compute and results in a linear and pairwise objective function. Justification: sequences with low E_{k_0} often have high energies on other backbones. However, this sacrifices the ability to design against undesired states and does not take into account energy differences between different sequences in the unfolded state.

2. $\max \Delta_{0,1} = E_{k_1} - E_{k_0}$ = energy gap between minimum energy fold (k_0) and "second best" energy fold (k_1). Justification: $\sum_k e^{-\beta E_k}$ in (1.3) will be dominated by the lowest energy E_k aside from k_0, which is E_{k_1}. This reduces the entire expression to the minimization of $E_{k_0} - E_{k_1}$, alternatively the maximization of $E_{k_1} - E_{k_0}$. However, E_{k_1} is still difficult to find because k_1 is difficult to find and it still ignores the unfolded states.

3. $\max \Delta = \langle E_k \rangle_{k \in U} - E_{k_0}$ = difference between the energy of the target fold (E_{k_0}) and the mean energy of all unfolded states ($\langle E_k \rangle$). Justification: expand the sum in $\frac{1}{\beta} \log \sum_k e^{-\beta E_k}$ in (1.3) as described in [36, p. 3122]. The largest term is $\langle E_k \rangle_{k \in U}$. Difficulty: U is large and hard to define. Variations: average over a small number of user-specified unfolded states or a single representative state. Justification: the user may know which unfolded states are of design interest.

4. $\min \Delta G = \Delta H - T \Delta S$ = free energy of folding, which includes both potential energy and entropy (S). Justification: this is equivalent to the original objective function, $\max P_{k_0}$. Difficulty: entropy term makes function non-linear and cumbersome to calculate. Also, since this is ultimately a difference in free energies between the target and unfolded state, it requires a model for the unfolded state.

Decisions: most researchers have chosen to optimize for folding stability, specifically the appoximation of Δ, where only a crude approximation of a single, representative unfolded state is used to model $\langle E_k \rangle_{k \in U}$. Developing algorithms that explicitly account for more undesired states will require exploration of some of these alternate quantifications of $\max P_{k_0}$.

1.2.6 Search Algorithms

Some choices we must make in the design of the algorithm:

- **Speed/runtime** - we want our algorithms to run in a reasonable amount of time.

- **Optimality** - is our algorithm guaranteed to find the optimal solution(s)? What guarantees can we prove about the output of the algorithm?

- **Theoretical basis** - how sound is the algorithm theoretically? Can we prove anything about its output, its worst case run time, its average runtime or any invariants? Or is the algorithm supported by purely empirical results and testing?

- **Physical basis** - how sound is the algorithm physically? Is it optimizing physically or chemically meaningful values (energies, probabilities, etc) or, again, is it supported only empirically?

The above factors may vary for different design problems, thus, for side-chain packing purposes, we chose to implement a variety of algorithms described in literature and make all of them available.

Randomized Algorithms

Monte Carlo is a stochastic algorithm that randomly initializes points in the search space and then repeatedly tests small random modifications to improve the objective function [19, 27]. The Monte Carlo main loop consists of two basic steps: 1) random walk and 2) decision to accept or reject. The decision criteria is as follows: if the random walk results in an energetically favorable change, accept it. Otherwise, accept the move with a certain probability. The probability is a function of the energy difference and a temperature factor. The Simulated Annealing approach is similar, but adds a gradual decay to the temperature factor [38]. This allows the algorithm to make higher risk moves early in the cycle and conservative ones towards the end.

The Monte Carlo algorithm does not guarantee optimality of its results, but longer runs of Monte Carlo will increase the likelihood that low energy solutions will be found (optimal or close to optimal). Various tricks, such as the quench technique, can improve the energy of the final side-chain pack configuration [40].

Genetic Algorithms perform a similar search, but run on populations of configurations, instead of single configurations [8]. Configurations also replicate and compete with each other in a processes intended to mimic Darwinian natural selection. The best configurations at any given time during the search will occasionally swap segments in a meiosis-like fashion in order to gather favorable subsets of rotamers into the same configuration.

Genetic Algorithms, though straightforward to implement, have not been included in our software as a side-chain packing algorithm. Comparisons have shown that, for the side-chain packing problem, the optimality of results generated by Genetic Algorithms do not differ significantly from those generated by Monte Carlo [40].

Dead End Elimination

Dead end elimination algorithms act to prune poor rotamers from the search space. They do so by examining a particular rotamer at a position on the backbone and determining whether any other rotamers at that position are always more favorable, regardless of the configuration of the remainder of the protein [9].

$$E(i_r) - E(i_s) + \sum_{j \neq i} \min_t \left[E(i_r, j_t) - E(i_s, j_t)\right] > 0 \tag{1.4}$$

Equation 1.4 describes the elimination criterion for the first order singles DEE method [13]. If an i_r and i_s exist that satisfy this equation, we say i_r has been eliminated by i_s by the DEE criterion and remove i_r from the search space.

There are many variations of DEE algorithms, from the original Desmet formulation to more rigorous criteria. In particular, we have implemented: zeroth order singles [9], first order singles [13], singles with conformational splitting [34, 29], "magic bullet" pairs [15], and first order pairs [23] with coefficient of extrema speedups [15].

All of these implementations provide a guarantee: no rotamer in the global minimum energy conformation (GMEC) will ever be eliminated. DEE is not guaranteed to significantly reduce the search space, and it is not guaranteed to terminate quickly, but DEE will never eliminate the GMEC. Hence, it provide a means for an exact search for small problems.

DEE is effective at reducing the size of the search space by many orders of magnitude, but often does not reduce it to a single solution. For this reason, it is often paired with a branch and bound algorithm to complete the rest of the search.

Branch and Bound

A* is a branch and bound search algorithm that attempts to descend through a search tree in an intelligent search order [25]. It relies on effective ways to estimate the optimal energy with a lower bound. The closer this estimator is to the optimal energy, the less steps the A* algorithm takes to complete. However, highly accurate estimator functions are slow and increase the amount of time spent on each step of the A* algorithm.

A* is one of the few algorithms that is guaranteed to find the GMEC. However, it is not guaranteed to terminate in a reasonable amount of time. Its runtime can scale exponentially for larger problems. Even worse, its memory consumption can also scale exponentially. For this reason, A* is generally only used on smaller, more tractable problems.

Self-Consistent Mean Field

These methods are based on constructing a probability matrix as a function of energies, and an energy matrix as a function of probabilities as shown in equation 1.6 [21, 26, 22].

$$\begin{aligned} P'(i_r) &= \frac{e^{-E_{i_r}/kT}}{\sum_s e^{-E_{i_s}/kT}} && (1.5) \\ E'(i_r) &= E(i_r) + \sum_j \sum_s P(j_s) E(i_r, j_s) && (1.6) \end{aligned}$$

These two conditions can be iteratively run to convergence to compute final energies. This formulation assumes a pairwise independence of probabilities that is not realistic. As a results, SCMF methods are not guaranteed to find optimal solutions. Runtime is extremely fast, however, scaling effectively linearly, even for large problems.

Semidefinite/Linear Programming

Very recently, the protein design problem has been formulated as an integer program [10] and as a semidefinite program [4]. These programs are general theoretical models for optimization problems and various algorithms exist for solving or approximating such programs.

1.3 Past Work

Though significant successes have been achieved in designing protein stability (e.g. Mayo [7]), a number of design problems point to the need for an approach that accounts for specificity. Sharma et al. designed a peptide to preferentially heterodimerize with a target protein [37]. It was important in this design process to explicitly disfavor competing states such as the peptide homodimer. In another project, DeGrado et al. designed a peptide receptor for calcineurin [11, 28]. This work also required a way to disfavor competing undesired states. More generally, the need for negative design becomes apparent whenever one is considering competing states that have high structural similarity to the target state.

Research in the area of specificity design has mostly progressed using simplified lattice models of proteins [20]. Approaches to pursuing specificity in the all-atom context that is relevant to the design of real proteins are comparatively unexplored.

1.4 Problem Specification

1.4.1 Problem Statement

We will develop algorithms for the computational design of proteins that output a set of selected amino acids chosen to optimize *for* a number of desired folds and *against* a number of undesired folds. The algorithms take as input the backbone structures of each desired and undesired fold. By including bound protein-protein complexes as folds, one can use such algorithms to redesign a particular protein for desired interactions and against undesired interactions. Some notes about our new problem formulation and how it relates to the stability problem:

- The list of undesired folds must be finite and small. Our algorithms will not search the infinite continuum of unfolded states for each amino acid sequence. This assumes that the user can characterize all of the "particularly important" undesired folds.

- This differs from the folding stability objective function in that multiple desired and undesired states are explicitly described. Since most stability algorithms design for only a desired state, the binding specificity problem can be seen as a generalization of the folding stability problem.

- The search space of the new objective function differs from the old one. Folding stability algorithms often search the space of *rotamer* possibilities (see section 1.2.3). However, rotamers have no universal meaning among different backbone conformations. Thus, binding specificity algorithms search the space of *amino acid* possibilities at each position.

- Side-chain placement is a subproblem of the binding specificity problem. During the search, the algorithms will compare energies of selected amino acid sequences on given backbones. This energy greatly depends on the packing of the side chains of each amino acid. The energy of the sequence will be the energy of the best side-chain packing of that sequence. Fortunately, the side-chain placement problem is a special case of the the well-researched folding stability problem. Thus, established algorithms we have used for optimizing folding stability can be used to solve the subproblem of side-chain placement within our specificity algorithms.

1.4.2 Exploring Objective Functions

Ultimately, the choice of objective function will depend on what the user wants to optimize. The choice of objective function also affects the choice of search algorithm. This section will explore a few suggested mathematical representations of objective functions.

First, define "state" as a way to arrange a given number of residues with a fixed stoichiometry. Define U as the set of all undesired states and D as the set of all desired states. E_k is the energy of state k and is computed as described in section 1.2.4. What follows are some suggestions for linear and pairwise objective functions:

1. Maximize the energy gap between the probability weighted average of the desired states and the average of the undesired states. This is a parallel of function 3 in section 1.2.5 (Δ). We can quantify it in the following way:

   $$\max f,\ f = \sum_{k \in U} c_k E_k - \sum_{k \in D} c_k E_k \tag{1.7}$$

   where $\sum_{k \in U \cup D} c_k = 1$. The most sensible coefficients to assign are the Boltzman probabilities :

   $$c_k = p_k = \frac{e^{-\beta E_k}}{\sum_{i \in U \cup D} e^{-\beta E_i}} \tag{1.8}$$

   Unfortunately, the p_k are not constant with respect to E_k, so our objective function is no longer linear. Instead we can choose arbitrary constants $\{c_k\}$, but it is not clear how to choose these values in a meaningful, non-empirical way.

2. Maximize the probability of occupying desired states. This parallels function 4 in section 1.2.5 (P_{fold}). For example: if a heterodimeric binding between proteins A and B is desirable, but homodimeric bindings are undesirable, then the desired state can be specified as AB and the undesired states as AA and BB. This can be captured in the equilibrium equation

$$AA + BB \rightleftharpoons AB + AB$$

To maximize the probability of occupying the right half side of the equilibrium, we maximize P_{AB} where

$$P_{AB} = \frac{e^{-2E_{AB}}}{e^{-E_{AA}-E_{BB}} + e^{-2E_{AB}}} = \frac{1}{e^{2E_{AB}-E_{AA}-E_{BB}} + 1}$$

To maximize P_{AB}, we need to drive the exponent towards $-\infty$. This gives us the following function:

$$\max f,\ f = E_{AA} + E_{BB} - 2E_{AB}$$

Or more generally:

$$\max f,\ f = \sum_{k \in U} c_k E_k - \sum_{k \in D} c_k E_k \tag{1.9}$$

Where the $\{c_k\}$ correspond to the weights in the balanced equation. This objective function is equivalent to maximizing the difference in stabilities of the two competing states under the assumption that they are equal in energy when unfolded (a commonly made assumption). It becomes more difficult to quantify, however, when all of the desired binding properties of a protein cannot be captured in a single equilibrium equation. For example, if the input included an undesired unfolded state $U_A + U_B$, we might have multiple equilibrium equations:

$$AA + BB \rightleftharpoons AB + AB$$

$$U_A + U_B \rightleftharpoons AB$$

This would give us multiple functions to maximize:

$$f_1 = E_{AA} + E_{BB} - 2E_{AB}$$

$$f_2 = E_{U_A} + E_{U_B} - E_{AB}$$

Which roughly corresponds to:

$$f_1 : \text{optimize for specificity}$$

$$f_2 : \text{optimize for stability}$$

As before, any method for unifying these criteria into a single objective function seems arbitrary:

$$\max f,\ f = \lambda_1 f_1 + \lambda_2 f_2 + \ldots + \lambda_n f_n \tag{1.10}$$

1.4.3 Adapting Search Algorithms

We must choose at least one algorithm for selecting sequences for specificity design. This section details the considerations made in determining which of the algorithms used in stability design (see section 1.2.6) to adapt to the specificity problem.

Randomized Algorithms

Monte Carlo and genetic algorithms are a natural choice and can be adapted easily to the new problem. Irbäck et al. have explored a Monte Carlo approach to the specificity problem using a lattice model [20]. These algorithms can optimize almost any objective function that can be stated mathematically, and therefore don't require a linearization of P_{k_0}. However, these functions give us no guarantees on the optimality of the output. This may be one of the most promising leads, however, as the larger search space size of the specificity problem may make most other algorithms impractical.

Self Consistent Mean Field

The literature on applying self-consistent mean field (SCMF) algorithms to the binding specificity problem is very recent and deals with adapting these statistical mechanics algorithms. Harbury proposes an SCMF approach to optimize a sequence for a single desired state and against a number of undesired states [18]. Saven also explores a self-consistent approach using a lattice model [42]. As before, the SCMF formulation should provide a quick solution, but suffers from the assumption that the probabilities are pairwise independent and thus provides no guarantees on the optimality of the output. Empirical testing would be necessary to determine its usefulness.

Dead End Elimination

DEE is an algorithm that works surprisingly well in the stability problem, but in designing for specificity, there are reasons to think it may not work well at the amino acid level. Using DEE at this level would require the elimination of all conformations of an amino acid on all given backbones as opposed to single rotamers on a single backbone. The latter problem is more tightly constrained and leads to a significant number of eliminated rotamers. The specificity problem has fewer constraints and may lead to fewer eliminations. However, since DEE is an empirically-justified algorithm, the only way to know is to test its performance. The challenge: we need to determine a pairwise formulation of the energy functions.

Branch and Bound

Since this algorithm doesn't scale very well, it is uncertain how it will handle the larger search space for the specificity problem. Its success, as before, will depend on whether we can write good energy estimators. Also, branching at the amino acid, rather than rotamer, level complicates the search.

Semidefinite/Linear Programming

Modeling the problem this way is challenging. Some questions to consider: can we linearize the energy function? Will we attempt to find an approximation algorithm [12] or attempt to find an exact solution in a theoretically unbounded amount of time [10]? Is an approximation algorithm even possible [4]? Since we haven't dealt with this algorithm before, we will not initially implement it for sequence selection.

Algorithm choices

Only the Monte Carlo algorithm will be implemented at first. Future search algorithms can be added later.

1.5 Summary

Algorithms for effective *in silico* protein design have undergone most of their development in recent years. A number of empirically effective algorithms exist for identifying amino acid sequences that fold to desired states.

With respect to the folding stability problem, designing proteins for binding specificity is comparatively uncharted. There are a number of design problems that can most effectively be addressed with a specificity design approach. This thesis aims to implement software that can address these.

The purpose of this thesis is to develop a computational toolkit for biologists who wish to reengineer the binding behavior of proteins or protein interfaces.

Chapter 2

Software Design

This chapter details the design and implementation of the protein design software. The documentation is current as of version **1.0.1**.

2.1 Design

2.1.1 Formalized Program Specification

This program will provide a user with a number of algorithms to search the amino acid solution space for certain protein design problems. The user will be allowed to specify several parameters of the search without recompiling the program. These parameters include, but are not limited to:

1. different protein energy data sets.

2. algorithm(s) to use for amino-acid sequence selection and any associated options.

3. algorithm(s) to use for side-chain packing (rotamer selection) and any associated options.

4. a user definable objective function to maximize during the search.

The guarantees and limitations of the program will correspond to the guarantees and limitations of the various algorithms used (expected runtimes, guarantees on convergence or global minimum energy, etc). All required energy data will be provided by the user. Program output will include a human and machine readable representation of the solutions found. The program will run non-interactively (`stdin` is not touched) and make use of parallel computing hardware.

Input files: control file

All user parameters are read through the control file. The control file is read line by line. Valid end of line markers are `CR` (0x13), `LF` (0x10) and '`#`' (hash). Blank lines are ignored. Any contiguous string of whitespace (space, tab) is treated as a single space.

The control file must contain several blocks that may be placed in any order. These blocks are: `options`, `objective-function`, `spec-sequence`, `dee-sequence`, and one or more `backbone` blocks. See Appendix A.1.1 for an example control file.

Input files: energy files

For each backbone there must be a corrseponding energy file. The file is binary and contains an 8-byte Intel-endian double for each rotamer at each position, followed by a double for each pair of rotamers. The order of entries is as follows. Self energies:

$$E(1,1), E(1,2), \ldots, E(1,n_1), E(2,1), \ldots, E(2,n_2), \ldots, E(p,n_p)$$

Where the number of positions is p and the number of rotamers at any given position i is n_i. This is immediately followed by pairwise interaction energies as follows:

$$\begin{aligned} &E((1,1),(2,1)), E((1,1),(2,2)), \ldots, E((1,1),(2,n_2)), \ldots, E((1,1),(p,n_p)),\\ &\ldots, E((1,2),(2,1)), E((1,2),(2,2)), \ldots, E((1,2),(2,n_2)), \ldots, E((1,2),(p,n_p)),\\ &\ldots, E((1,n_1),(2,1)), E((1,n_1),(2,2)), \ldots, E((1,n_1),(2,n_2)), \ldots, E((1,n_1),(p,n_p)),\\ &\ldots, E((2,1),(3,1)), E((2,1),(3,2)), \ldots, E((2,1),(3,n_3)), \ldots, E((2,1),(p,n_p)),\\ &\ldots, E((p-1,n_{p-1}),(p,n_p)) \end{aligned}$$

Output files

The program produces several similar output files. The main output file (specified by the main-out-file command in the options block of the control file) contains the rank ordered list of top scoring solutions. Each line contains the score followed by the amino acids selected at each design site. The first line is the highest scoring solution found and the nth line is the nth highest scoring solution.

The program produces a backbone output file for each defined backbone (specified by the output command in the backbone block). Each line contains the energy, amino acid and rotamer choices for the sequence in the corresponding line of the main output file. An example main output file is provided in Appendix A.2.1.

2.1.2 Program Control Flow

The program control flow is depicted in Figure 2-1. The search algorithm generates an amino acid sequence that ultimately results in a score; the algorithm uses the score in informing its next sequence choice. This score is generated by a user-defined objective function that depends on the energies of the amino acid sequence on each desired and undesired backbone state. Determining these energies requires the use of user specifiable side-chain packing algorithms.

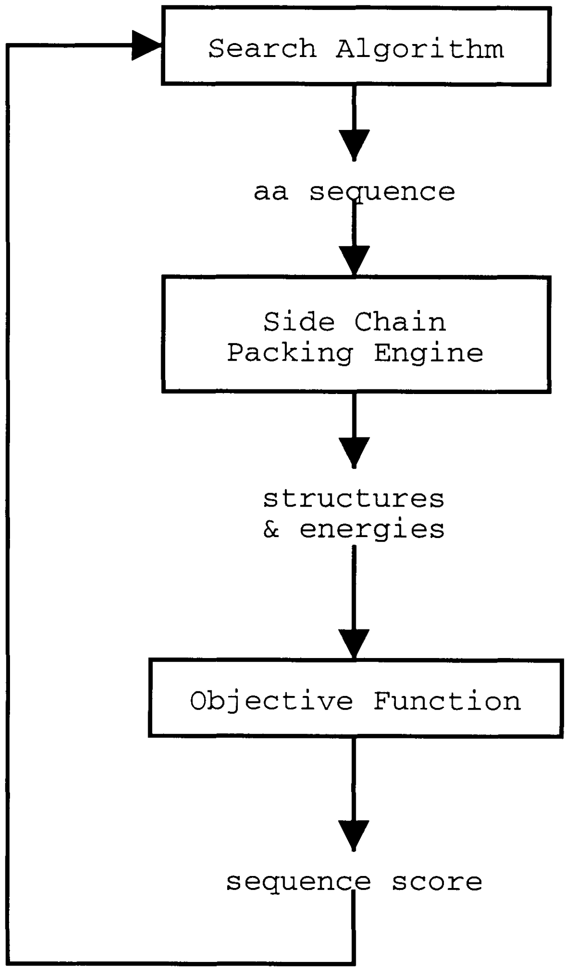

Figure 2-1: Program flow chart

2.1.3 Parallelization

The protein design problem is broken into two separate subproblems: the amino acid sequence search and the side-chain packing problem (see section 1.4.1). Similarly, the software can be thought of as divided into two major pieces: the sequence search engine and the side-chain packing engine.

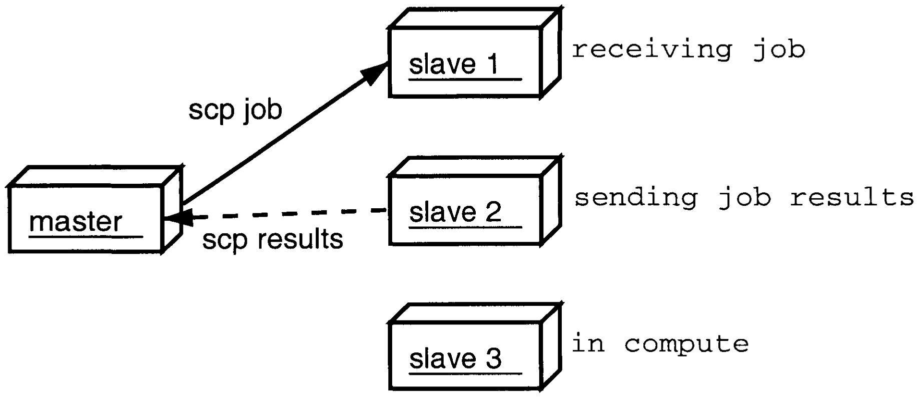

Figure 2-2: Parallel job distribution

Since the sequence optimization algorithms call side-chain packing functions, the speed of sequence optimization is directly dependent on the speed of side-chain packing. Thus, the most straightforward way to exploit a parallel computing environment is to run multiple side-chain packing engines. The division of labor is depicted in Figure 2-2. For N nodes in a computing cluster, there will be one node dedicated to amino acid search algorithms and $N - 1$ nodes acting as side-chain packing engines. The master node sends a full sequence string for a given backbone to a free slave node. The slave node then computes the optimal side-chain conformations and returns the lowest energy and associated structure.

Inter-node communication occurs only between the master node and its slave nodes. During sequence selection, communication is slave node initiated. The handshake is as follows:

1. (master) Waits for any slave requests

2. (slave) Initiates data exchange (DXCHG) request. If slave has recently completed a side-chain packing job, the DXCHG packet indicates that data is about

to follow.

3. (slave) If results are available, send it.

4. (master) If data was received, it is saved on the master.

5. (master) If a new job is available, send job to slave, otherwise, send a WAIT command. If the search algorithm is complete, send a SHUTDOWN command.

6. (slave) If job received, compute the side-chain pack. If WAIT received, pause briefly and repeat from top. If SHUTDOWN received, exit program.

2.1.4 Data Objects and Relationships

Figure 2-3 contains a UML representation of the relationships between objects that the program needs to track. Boxes indicate a set of objects, lines represent relations between members of two sets. The punctuation markings indicate the multiplicty of the relation, where '?' indicates "at most one" (0 or 1), '!' indicates "exactly one" (1), '+' indicates "at least one" (1 or more), and the absence of punctuation indicates any multiplicity (0 or more). Thus, we see that every amino acid is related to at most one seq_position, and that every seq_position is related to at least one amino acid. And since every position is related to exactly one seq_postion, we can also infer that every position must be associated with at least one amino acid (through seq_position) and that each of those amino acids has exactly one aa_name. Furthermore, since this is a description of software design, we can also infer that the program contains a bug if we ever find that a position is associated with no amino acids during runtime.

Some relations have names (ie, pos_xlate relates the sets seq_position and position), but most do not. Some names are directional. For example, every pair is related to exactly two rotamers, one called i,r and the other j,s. The name "i,r" is directional in that it refers to the rotamer, not the pair, in the relation.

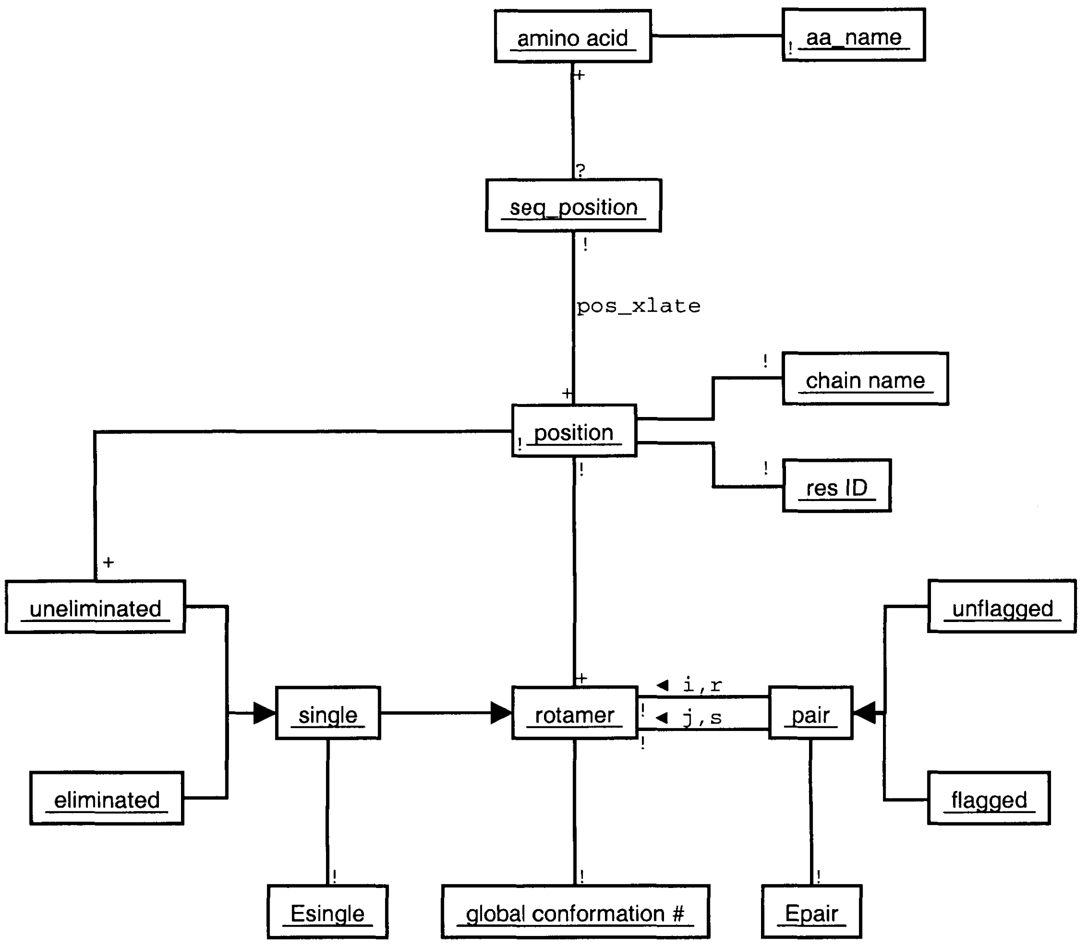

Figure 2-3: Program object model: state data

Lines with solid triangles have a special meaning. They indicate a partition relationship. The figure indicates that uneliminated and eliminated are subsets of single and completely partition it. This means that all elements of single are either elements of uneliminated or eliminated, but not both. This also means that single is the same set—or another name for—rotamer. Notice that the following assertion can be made from this model: a position is required to be associated with at least one rotamer, some of which may be eliminated, but at least one of which must be uneliminated. If there is ever any position that has no uneliminated rotamers during runtime, there is a bug in the program.

2.1.5 Code Modules and Structure

The module dependency diagram (MDD) details the overall structure of the code and can be seen in Figure 2-4. Its purpose is to describe the intended relationships of the modules and submodules of code. These relationships are part of the module specifications. To change dependencies in this diagram is to change the overall design and the specifications of the functions in the affected modules.

Arrows denote a dependency of the implementation of one module on the *specifications* (not implementations) of another—a spec change to a function in one module may require implementation changes in all modules that point to it. Note that every module within the side-chain packing package is dependent on the specifications of the data module in the backbone package.

To add new side-chain packing algorithms to the program, one creates a new function and adds it to an existing module (e.g. dee-pairs) or creates a new module, if appropriate. This new function may use the data module functions (if it uses functions from any other existing module, it represents a significant program design change). The scp-ctl module must then be modified to incorporate the new function:

side-chain packing engine.

1. Random initialization. If C is the specified number of cycles, Generate C randomly initialized sequences and enqueue for scoring. Create a temperature array of size C and seed with initial temperatures.

2. Wait for result. Wait for a sequence to finish scoring.

3. Decision. If the returned score is higher than the previous score for this cycle, or if this is the first score computed for this cycle, accept the new sequence. Otherwise, accept the new sequence with probability $P = \exp -\Delta E/kT$.

4. Random walk. At a random position, a residue is replaced with a new, randomly chosen amino acid. Unless the number of steps for this cycle have completed, enqueue the new sequence for scoring. Anneal temperature by a small amount.

5. Repeat from step 2. This loop repeats until the job queue empties.

2.2.2 DEE Pseudocode

DEE is a pruning algorithm for side-chain packing. There are many variants described in the literature, with widely varying runtimes. This section will not describe every variant of the DEE algorithm, but will illustrate the most commonly used form we call "first order singles." It is first described in [13].

Let i and j be positions on the backbone, r and u be rotamers at these positions respectively, and t be a rotamer on i that attempts to eliminate r. The first order singles algorithms is as follows.

```
for all positions i
  for all uneliminated rotamers r on i
    for all uneliminated rotamers t on i where r != t
```

```
      X = E(i,r) - E(i,t)
      for all positions j where j != i
        D = +inf
        for all uneliminated rotamers u on j
          d = E((i,r),(j,u)) - E((i,t),(j,u))
          D = min(d, D)
        end-for u
        X = X + D
      end-for j
      if X > Ecut, eliminate rotamer r on i
    end-for t
  end-for r
end-for i
```

User-specifiable options include: the variant of DEE being executed, whether to repeat the algorithm to convergence, and the number of splits (for conformational splitting only).

2.2.3 A* Pseudocode

A* is a branch and bound algorithm that is described in [25]. Partial solutions, that is, solutions with some fixed rotamers and some undefined, are stored in a heap along with two quantities: g and h. g is the energy of all fixed elements in the partial solution. h is an estimate of the energy of the undefined elements. There is one requirement for h: it must never overestimate the energy of the undefined elements. That is, the energy of any possible combination of unassigned rotamers must be greater than or equal to h. The estimated total energy of a partial solution is then $f = g + h$.

A* operates by extracting the partial solution with minimum f from the heap. It then selects one of the undefined positions and expands it by placing each allowed rotamer successively in that position. Each of the new partial solutions generated this way has exactly one more fixed rotamer than before, and the entire set of new partial solutions with estimated energies is inserted back into the heap. This cycle then repeats until a complete solution (a solution with no undefined rotamers) is extracted. The algorithm is illustrated below:

```
insert a node with no fixed rotamers into heap
while no complete solutions have been found
  node = extract min node from heap
  i = the first undefined position in node
  for all valid rotamers r on i
    create newnode where
      newnode.fixedrots = union of node.fixedrots and (i,r)
      newnode.g = node.g + E(i,r) + sum E((i,r), node.fixedrots)
      newnode.h = energy estimate of remaining undefined positions
    insert newnode into heap
  end-for r
end-while
```

Since the energy estimator implementation can have a problem-dependent effect on runtime, we allow the user to specify one of two energy estimator functions: the Leach & Lemon estimator [25] or the Mayo estimator [16].

The order in which positions are expanded has also been shown to have an effect on runtime. The user may specify one of two ordering heuristics: Leach & Lemon ordering [25] or Mayo ordering [16].

2.3 Implementation Methodology

After the software was designed, implementation proceeded from bottom to top. Each step of the way, modules were tested to ensure they met their specifications. Debugging of C pointer problems was accomplished with the dmalloc debugging library (`dmalloc.com`). Higher level or "holistic" tests of the software are outlined in the next chapter.

2.4 Current Status

The software continues to be maintained at the Keating Lab in the MIT Biology department.

Chapter 3

Validation

The remainder of this document aims to validate the software and establish whether it achieves its intended purpose. This chapter details a number of tests we have constructed and what we can determine from the results of those tests.

Though empirical testing of software cannot prove its correctness, we can gain a greater confidence in its functionality and, in particular, its biological utility from the results.

3.1 Background: Coiled Coils

All of the following tests involve modeling coiled coil proteins. The coiled coil is the simplest and probably most common protein-protein interaction motif [30, 3]. Because a considerable amount is known about coiled coils [6], they provide an ideal case for testing computational models against established biochemical properties.

A coiled coil consists of a number of α-helices supercoiled around one another giving a superhelical bundle. Coiled coils have been shown to exist as dimers, trimers, tetramers and pentamers. The amino acid sequence can be easily mapped to the coiled coil's three dimensional helical fold: every seven residues maps to two turns of the helix. Because of this pattern, researchers often label each residue based on its position in the heptad with a letter *a* through *g* (see Figure 3-1).

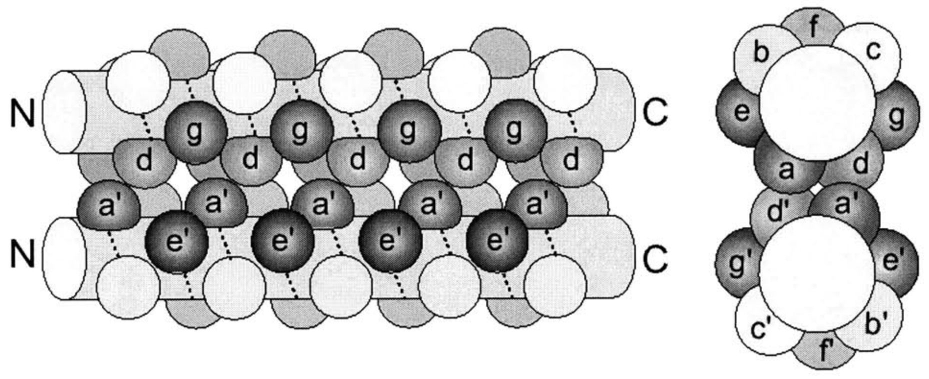

Figure 3-1: A typical coiled-coil dimer. Left: lengthwise view. Right: axial view. Each cylinder corresponds to one α-helix. The superhelical twist is omitted for clarity. Graphic taken from [39].

It has been well established that the *a* and *d* positions greatly influence how coiled coils interact with each other. In particular, the residues at these positions have been shown to influence the oligomerization state (the number of α-helices in the bundle), helix orientation (parallel vs. antiparallel) and homo- vs. heterooligomerization [17, 31, 33].

Coiled coils present a particular need for specificity design. Because most coiled coils bear a structural resemblance to each other, designing for a desired interaction may inadvertently make a competing interaction more favorable. For example, when designing to stabilize a coiled coil dimer, the undesired trimeric form may be stabilized as well. Ultimately, we would like to test our program's ability to address this problem.

3.2 Test Case 1: Side-Chain Packing

Our first aim was to validate the side-chain packing engine. All of the code used in our side-chain packing engine was ported from a previous program we had written to design proteins for stability. Since the old software has already been tested, we need to verify that the new software produces the same structures and energies when run under the same conditions.

If we construct a test case that involves a fixed amino acid sequence (no sequence selection), protein design is reduced to the side-chain packing problem. Since there are no amino acids to select, the only task remaining to either program is finding the side-chain conformations that minimize energy.

3.2.1 Inputs

The specificity design program and the old stability design program were both run with the same backbone data, the same amino acid sequence, and the same packing algorithms. The only variable was the program itself.

Backbone data

We chose to repack residues at the interface of a heterodimeric coiled coil complex of interest to our lab (Fos-Jun, Protein Data Bank code: 1A02), pictured in Figure 3-2. The following *a* and *d* residues were repacked. Fos: 23, 26, 27, 29, 30, 33, 34, 36, 37, 40, 41, 43, 44, 47, 48, 50, 51. Jun: 21, 24, 25, 27, 28, 31, 32, 34, 35, 38, 39, 41, 42, 45, 46, 48, 49.

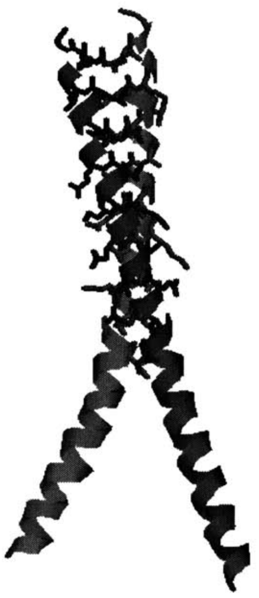

Figure 3-2: Fos-Jun heterodimer (1A02). The α-helix backbones (ribbons) form a supercoiled bundle. Side chains (sticks) are shown for the residues being repacked or redesigned in Test Cases 1 and 2.

Amino acid sequence

Both programs performed side-chain packing only—the amino acid sequence was fixed to the wild type sequence. Since there is only one amino acid sequence in the search space, a search algorithm for sequence selection need not be specified.

Packing algorithms

In each series of tests, we used the same side-chain packing algorithms. Only deterministic algorithms (A* with DEE, SCMF) were tested, so identical results can be expected from both programs.

3.2.2 Expected Results

Both programs should produce identical structures and energies when the same packing algorithm is used.

3.2.3 Actual Results

For each side-chain packing algorithm tested, both programs produced the same structures and energies. The resulting energies are listed below. Note that the DEE/A* solution is guaranteed to be the optimal solution.

	energy (kcal/mol)	
Search Algorithm	new program	old program
DEE and A*	-53.2102	-53.2102
SCMF, init flat	+56.2674	+56.2674
SCMF, init zero	-52.4571	-52.4571

This verifies the DEE/A* and SCMF components of the new side-chain packing engine.

3.3 Test Case 2: Stability Design and Search Algorithm

Our next aim was to validate the ability of the new software to design proteins for stability. Since the side-chain packing engine was validated by the previous test case, this is effectively a test of the new software's search algorithm for sequence selection.

Since most interesting problems in specificity design involve an extremely large search space, our primary sequence selection algorithm is Monte Carlo, a randomized algorithm that has no guarantees on the optimality of its solutions. We want to establish that Monte Carlo can minimally find a set of good (close to optimal) solutions on a small problem.

Since the old design software is only used to address the stability design problem—a comparitively smaller search problem, it is equipped with slower, but exact, search algorithms such as the A* algorithm. Though this algorithm does not scale well, it is guaranteed to provide the optimal solutions for small problems. This output can be used to measure the effectiveness of our Monte Carlo sequence selection algorithm.

The test case we construct must be small enough for the old design program to fully explore so it can provide a list of the most optimal sequences in the search space (using the A* algorithm). Comparing the results to our Monte Carlo selection output should validate the effectiveness of the search algorithm and, equivalently, the ability of the new software to design proteins for stability.

3.3.1 Inputs

Backbone data

The same Fos-Jun interface will be used as in the previous test case. The same residues will be repacked, and in addition, the following residues will be redesigned: Fos: 40, 41, 43, 44. Jun: 38, 39, 41, 42. These positions comprise the *a, d, e* and *g* positions of one heptad on each side of the Fos-Jun interface.

Amino acid search space

Because the search space must be small enough for A* to search, amino acids with a large number of side-chain conformations (e.g. LYS, ARG, GLU, GLN) were excluded. The exception is at Fos positoins 41 and 43, where the wild type sequence includes GLU and GLN residues. The search alphabet was restricted to the following:

Design position	allowed amino acids
Fos 40 (LEU)	ALA VAL LEU ILE PHE TRP ASN THR SER TYR HIS CYS MET ASP
Fos 41 (GLN)	ALA VAL LEU ILE PHE TRP ASN THR SER TYR HIS CYS MET ASP GLU GLN
Fos 43 (GLU)	ALA VAL LEU ILE PHE TRP ASN THR SER TYR HIS CYS MET ASP GLU GLN
Fos 44 (ILE)	ALA VAL LEU ILE PHE TRP ASN THR SER TYR HIS CYS MET ASP
Jun 38 (LEU)	ALA VAL LEU ILE PHE TRP ASN THR SER TYR HIS CYS
Jun 39 (ALA)	ALA VAL LEU ILE PHE TRP ASN THR SER TYR HIS CYS
Jun 41 (THR)	THR VAL LEU ILE PHE TRP ASN ALA SER TYR HIS CYS
Jun 42 (ALA)	ALA VAL LEU ILE PHE TRP ASN THR SER TYR HIS CYS

The wild type residues for Fos-Jun are listed in the left column. For sequence selection, the new program ran Monte Carlo for 8 cycles at 800 steps each with linear temperature annealing from 250K to 150K. These parameters were chosen through experimentation: final scores seemed to converge around 400 steps of Monte Carlo. Other details are not covered here.

Side-chain packing

This search space (10^{34} packed structures) is small enough for the old program to completely search with the DEE and A* algorithms. The new program also used DEE and A* for side-chain packing.

3.3.2 Expected Results

Monte Carlo has a high probabilty of finding an optimal or close to optimal solution for small problems. If the program cannot effectively solve a problem this simple, it will be hard to apply to more complicated problems. Successful stability design is a necessary (but not sufficient) function before the software can be used for specificity design.

3.3.3 Actual Results

The twelve most optimal solutions (as determined by the old program's A* algorithm) are listed below:

Energy (kcal/mol)	Fos 40, 41, 43, 44	Jun 38, 39, 41, 42
-47.62725	LEU LEU ALA ILE	LEU TRP LEU LEU
-47.59601	LEU LEU ALA ILE	LEU PHE LEU LEU
-47.46449	LEU LEU ALA ILE	LEU LEU LEU LEU
-47.45449	LEU LEU ALA ILE	LEU HIS LEU LEU
-47.43629	LEU LEU MET ILE	LEU PHE LEU VAL
-47.39257	LEU LEU MET ILE	LEU HIS LEU VAL
-47.31653	LEU LEU MET ILE	LEU LEU LEU VAL
-47.27693	LEU LEU ALA ILE	LEU TYR LEU LEU
-47.27542	LEU LEU MET ILE	LEU TRP LEU VAL
-47.26645	LEU LEU LEU ILE	LEU TRP LEU LEU
-47.24457	LEU LEU ALA ILE	LEU TRP ILE LEU
-47.21290	LEU LEU ALA ILE	LEU PHE ILE LEU

The top six solutions found by the Monte Carlo search algorithm are listed below:

Energy (kcal/mol)	Fos 40, 41, 43, 44	Jun 38, 39, 41, 42
-47.62725	LEU LEU ALA ILE	LEU TRP LEU LEU
-47.24457	LEU LEU ALA ILE	LEU TRP ILE LEU
-47.21290	LEU LEU ALA ILE	LEU PHE ILE LEU
-47.08372	LEU LEU TRP ILE	LEU TRP LEU TRP
-46.69713	LEU LEU TRP ILE	LEU TRP ILE TRP
-46.66999	LEU LEU TRP ILE	LEU PHE LEU TRP

3.3.4 Analysis

The Monte Carlo algorithm achieved its primary objective: it successfully found the global minimum (the same sequence and energy as solved by the A* algorithm).

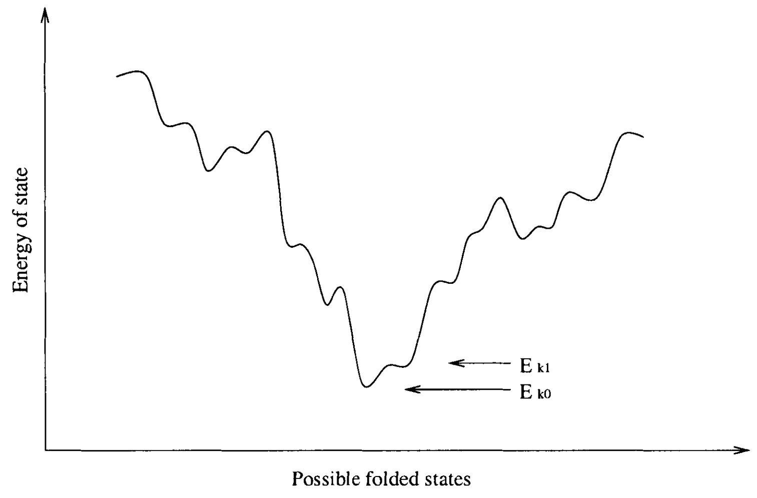

Figure 3-3: Energy contour of the amino acid search space

The search was not as successful in finding the ensemble of sequences that were close-to-optimal. The search algorithm did not find the second best or third best solution, or any solution up to the tenth best. Indeed, in the top twelve solutions, Monte Carlo only found solutions 1, 11 and 12. This is not of great concern, however, as it was not designed for finding close-to-optimal solutions. Monte Carlo spends most of its search time in local minima and mostly moves from one minimum to the next in the hopes that it eventually finds the global minimum (see Figure 3-3). The algorithm spends less time in regions of the search space where the energy contour is more "sloped," and more time in the energy wells. If the global minimum resides in a "steep" enough well, the algorithm may easily skip over a number of close-to-optimal solutions. For the purposes of our test case, the software has succeeded in finding the optimal sequence in solving a protein stability design problem.

3.4 Test Case 3: Specificity Design

The motivation of this thesis was to be able to design proteins with specificity for one fold over others. The final aim was to test whether the software meets this objective. Since no other available software addresses the same problem, we cannot simply compare our results to those of another program. Instead, we construct a test scenario that is biologically well understood and compare our results to the results of previous experiments.

This kind of test requires both the proper functioning of the program as well as accurate modeling of the energy functions and various parameters that are provided as input to the program. Thus, it serves as a holistic test of our entire computational protein design platform.

3.4.1 Test Problems

We applied the program to the following two problems: 1) design a coiled coil that forms a dimer and not a trimer, 2) design a coiled coil that forms a trimer and not a dimer. In addition, we would like to test whether our specificity design approach offers better results than a stability design approach, so we also ran the following two control tests: 3) design a stable coiled coil dimer, 4) design a stable coiled coil trimer.

3.4.2 Expected Results

We will compare our computational results to known rules about coiled coil dimers and trimers. In particular, Harbury [17] has shown that the placement of beta-branched residues (e.g. ILE, VAL) at *d* positions in coiled coils highly disfavors the formation of dimers. Also, the placement of a destabilizing ASN at the centermost *a* position disfavors the formation of trimers. We hope to recapture these patterns in the program output. In particular, the following results would be consistent with what is known experimentally:

1. Design for dimer, against trimer: ASN at the center *a* position (the only position it is considered at), and no ILE or VAL at *d* positions.
2. Design for trimer, against dimer: no ASN at the center *a* position, and ILE or VAL at *d* positions.
3. Design for dimer stability: no ASN at the center *a* position, and no ILE or VAL at *d* positions.
4. Design for trimer stability: no ASN at the center *a* position.

3.4.3 Inputs

Backbone data

The X-ray structures of the GCN4 dimer (PDB code: 2ZTA) and that of its trimeric variant (PDB code: 1GCM) were used as dimer and trimer backbones (see Figure 3-4). Five *a* and four *d* positions were redesigned on each helix of the homodimer and homotrimer: 2, 5, 9, 12, 16, 19, 23, 26, 30. In addition, eight *e* and *g* positions were repacked to allow for flexibility near the design sites: 6, 8, 13, 15, 20, 22, 27, 29. Without such flexibility, the original crystal structure conformation (encoded in the PDB input files) may strongly bias the design outcome towards the wild type sequence.

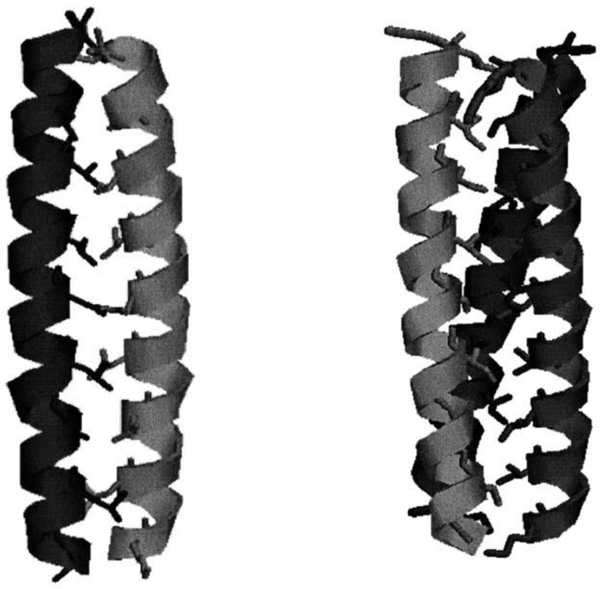

Figure 3-4: GCN4-p1 homodimer (2ZTA, left) and GCN4-pII homotrimer (1GCM, right). Side chains are shown for the *a* and *d* positions being redesigned.

Objective function and energy function

In the first two tests, amino acids were selected to maximize the energy difference between the dimer and trimer states. In the two control tests, amino acids were selected to maximize the energy difference between the desired (dimer or trimer) state and the unfolded state. The energy function included the following terms: van der Waals, torsion, electrostatics and solvation. The unfolded state was modeled as a hypothetical coiled coil with no interactions between side chains or between side chains and non-local backbone atoms.

Amino acid search space

To mimic Harbury's experiments, the search alphabet was restricted to the following:

Design position	allowed amino acids
2	LEU VAL ILE MET
5	LEU VAL ILE
9	LEU VAL ILE
12	LEU VAL ILE
16	LEU VAL ILE ASN
19	LEU VAL ILE
23	LEU VAL ILE
26	LEU VAL ILE
30	LEU VAL ILE

A Monte Carlo search was run for 5 cycles at 400 steps each with linear temperature annealing of 250K down to 150K. Again, these parameters were chosen after some experimentation, the details of which are not discussed here.

3.4.4 Actual Results

E_d, E_t, and E_u refer to the energies of the dimer, trimer, and unfolded states respectively (as reported by the side-chain packing engine). Energy differences are in kcal/mol.

The following are the top ten results when designing for a dimer and against a trimer (higher scores are better):

$-\frac{1}{2}E_d + \frac{1}{3}E_t$	a	d	a	d	a	d	a	d	a	
-13.12702	LEU	LEU	VAL	LEU	ASN	LEU	VAL	LEU	VAL	
-13.23110	MET	LEU	VAL	LEU	ASN	LEU	VAL	LEU	VAL	GCN4-pI
-13.38548	VAL	LEU	VAL	LEU	ASN	LEU	VAL	LEU	VAL	
-13.55549	ILE	LEU	VAL	LEU	ASN	LEU	VAL	LEU	VAL	
-14.12365	LEU	LEU	ILE	LEU	ASN	LEU	VAL	LEU	VAL	
-14.22759	MET	LEU	ILE	LEU	ASN	LEU	VAL	LEU	VAL	
-14.32961	LEU	LEU	VAL	LEU	ASN	LEU	LEU	VAL	VAL	
-14.37963	VAL	LEU	ILE	LEU	ASN	LEU	VAL	LEU	VAL	
-14.43371	MET	LEU	VAL	LEU	ASN	LEU	LEU	VAL	VAL	
-14.43433	LEU	LEU	VAL	LEU	ASN	LEU	LEU	LEU	VAL	

For trimer, against dimer:

$-\frac{1}{3}E_t + \frac{1}{2}E_d$	a	d	a	d	a	d	a	d	a	
118.89571	ILE	ILE	LEU	ILE	ILE	ILE	ILE	ILE	ILE	
118.72865	VAL	ILE	LEU	ILE	ILE	ILE	ILE	ILE	ILE	
118.52003	LEU	ILE	LEU	ILE	ILE	ILE	ILE	ILE	ILE	
118.35151	MET	ILE	LEU	ILE	ILE	ILE	ILE	ILE	ILE	
117.46184	ILE	ILE	ILE	ILE	ILE	ILE	ILE	ILE	ILE	
117.29916	VAL	ILE	ILE	ILE	ILE	ILE	ILE	ILE	ILE	
117.08927	LEU	ILE	ILE	ILE	ILE	ILE	ILE	ILE	ILE	
116.91988	MET	ILE	ILE	ILE	ILE	ILE	ILE	ILE	ILE	GCN4-pII
116.78832	ILE	ILE	LEU	ILE	ILE	ILE	LEU	ILE	ILE	
116.62127	VAL	ILE	LEU	ILE	ILE	ILE	LEU	ILE	ILE	

The following are the top ten results of the control test, designing for dimer stability. $\frac{1}{3}E_t - \frac{1}{2}E_d$ was also computed for comparison to the previous tests and is listed in the leftmost column.

$-\frac{1}{2}E_d + \frac{1}{3}E_t$	$-\frac{1}{2}E_d + \frac{1}{2}E_u$	a	d	a	d	a	d	a	d	a
-16.59162	32.42005	LEU	LEU	ILE	LEU	LEU	LEU	VAL	LEU	VAL
-16.51736	32.12317	LEU	LEU	ILE	LEU	ILE	LEU	VAL	LEU	VAL
-15.61222	31.85642	LEU	LEU	VAL	LEU	LEU	LEU	VAL	LEU	VAL
-18.07370	31.59020	LEU	LEU	ILE	LEU	LEU	LEU	LEU	LEU	VAL
-15.52070	31.57688	LEU	LEU	VAL	LEU	ILE	LEU	VAL	LEU	VAL
-15.58839	31.53760	LEU	LEU	ILE	LEU	VAL	LEU	VAL	LEU	VAL
-18.51434	31.43499	LEU	LEU	ILE	LEU	LEU	LEU	VAL	LEU	LEU
-18.93166	31.42932	LEU	LEU	ILE	LEU	LEU	LEU	VAL	LEU	ILE
-17.84983	31.29698	LEU	LEU	ILE	LEU	ILE	LEU	LEU	LEU	VAL
-16.69654	31.22043	MET	LEU	ILE	LEU	LEU	LEU	VAL	LEU	VAL

Control test for trimer stability:

$-\frac{1}{3}E_t + \frac{1}{2}E_d$	$-\frac{1}{3}E_t + \frac{1}{2}E_u$	a	d	a	d	a	d	a	d	a
109.80618	55.36762	LEU	ILE	LEU	ILE	ILE	ILE	LEU	LEU	ILE
105.72143	55.34926	LEU	ILE	LEU	ILE	LEU	ILE	LEU	LEU	ILE
116.41265	55.19022	LEU	ILE	LEU	ILE	ILE	ILE	LEU	ILE	ILE
112.32781	55.17186	LEU	ILE	LEU	ILE	LEU	ILE	LEU	ILE	ILE
109.38733	54.95601	LEU	ILE	LEU	ILE	ILE	ILE	LEU	LEU	LEU
105.30258	54.93766	LEU	ILE	LEU	ILE	LEU	ILE	LEU	LEU	LEU
56.79563	54.78326	LEU	ILE	LEU	LEU	ILE	ILE	LEU	LEU	ILE
95.93941	54.76091	LEU	LEU	LEU	ILE	ILE	ILE	LEU	LEU	ILE
92.24740	54.74228	LEU	LEU	LEU	ILE	LEU	ILE	LEU	LEU	ILE
63.40212	54.60587	LEU	ILE	LEU	LEU	ILE	ILE	LEU	ILE	ILE

3.4.5 Analysis

The results capture many of the known rules about coiled coil dimers and trimers. In particular:

- ILE and VAL in *d* positions favor trimers experimentally. This effect is captured in our specificity results: ILE or VAL appear at *d* in all trimer solutions when we select for specificity and in very few of the dimer solutions (dimer solutions 7 and 9 contain VAL in the last *d* position).

- ASN at the center *a* position favors dimers experimentally. This effect is captured in our specificity results: ASN appears in all dimer solutions when we select for specificity and in none of the trimer solutions.

Moreover, the results of the control tests suggest that the specificity design approach was necessary to recover these properties. Specifically:

- LEU occurs at *d* positions frequently (8 of the top 10 solutions) when designing trimers for stability alone.

- ASN is not selected at the center *a* position when designing dimers for stability alone.

It can be noted that, for the purpose of designing for dimers and against trimers (and vice versa), the specificity approach produced better scores by 3.5 and 9 kcal/mol, respectively.

In addition to producing solutions that agree with experimentally established rules, the program actually found the original GCN4-p1 and GCN4-pII sequences within its top ten solutions. Since it is already known that GCN4-p1 forms dimers in the laboratory (and similarly, GCN4-pII forms trimers) [17], we know that, in this case, the program has produced output that contains experimentally validated solutions.

These results demonstrate the ability of the software to reproduce experimentally derived patterns about coiled coil oligomerization, as well as the necessity of using the specificty design approach to do so.

3.5 Conclusion

Computational protein design has been the focus of much recent research for several reasons: it is theoretically interesting, it is biologically applicable and it is medically relevant. Recent advances in protein design algorithms and the capabilities provided by advances in hardware have made the design problem more tractable.

Researchers have been particularly successful at optimizing proteins for fold stability. However, approaches to designing proteins for binding specificity have not been as well explored. The need for software that can address this problem has become increasingly apparent.

The product of this thesis research is a software package that fills the need for a specificity design tool. It adapts the randomized Monte Carlo algorithm for sequence selection and employs a number of well established algorithms for side-chain packing.

Testing of this software has shown the ability to capture known patterns about coiled coil oligomerization and has demonstrated the utility of the specificity design approach. Because it allows us to address problems that are poorly handled in a stability design approach, we believe this software offers new possibilities for protein design research.

Appendix A

File examples

The following sections are examples of the files described in Chapter 2.

A.1 Example input files

A.1.1 control file

```
#
# test control file for specificity design
#

###### program options ######

begin-options
# stdout-prefix must have absolute path name in a slave
# node-reachable place (ie, /beowulf/...)
stdout-prefix /beowulf/scratch/dpark/out.node
num-solutions 25
```

```
main-out-file out-main

aa_input aa_input

end-options


###### backbone definitions ######


#begin-backbone example-things

# rot_subset rot_subset

# sc-dee-prune

# sc-pack bnb_astar leach_pos leach_h*

# sc-pack rand_scmf  init-flat

# sc-pack rand_mc 10 1E5 4000 250 # cycles, steps, T_h, T_l

#end-backbone


begin-backbone dimer

rot_input2 dimer/rot_input2

rot_input3 dimer/rot_input3

rot_states dimer/rot_states.self

energies dimer/energies.bin

# rot_subset dimer/rot_subset

pos_xlate dimer/pos_xlate

output out-dimer

# sc-dee-prune

sc-pack bnb_astar leach_pos leach_h*

end-backbone
```

```
begin-backbone trimer
rot_input2 trimer/rot_input2
rot_input3 trimer/rot_input3
rot_states trimer/rot_states.self
energies trimer/energies.bin
# rot_subset trimer/rot_subset
pos_xlate trimer/pos_xlate
output out-trimer
sc-dee-prune
sc-pack bnb_astar leach_pos leach_h*
end-backbone



###### objective function definition ######

begin-objective-function # some "arbitrary" math function here
# x0 = prod x1 x2 x3 .....
# x0 = wsum 0.25 x1 0.25 x2 0.25 x3 0.25 x4 -1 x5
# x0 = exp x1

# x1 = exp d1
# x2 = exp d2
# x3 = wsum 1 x1 1 x2
# f = div x3 Q

f = wsum 2 trimer -3 dimer
```

```
end-objective-function

###### specificity control schedule ######

begin-spec-sequence
spec_mc 10 1000 2000 250
#spec_enum
end-spec-sequence

###### dee control schedule ######

begin-dee-sequence

  dee_singles_zeroth  repeat
  dee_singles_mb      once
  dee_singles_mb      once

  ## heavy singles, light pairs
  begin-loop LOOP-1 1e15 # LOOP-1 begin

    dee_singles_first    repeat
    dee_singles_split_s1 repeat tryall

    begin-alternate LOOP-1
      dee_singles_split_s2 once   fixedbylooger
      dee_singles_split_s2 once   fixedbymayo
    dee_singles_split_s2 repeat fixedbylooger
  end-alternate

  dee_pairs_mb    repeat

end-loop # LOOP-1 end

## heavier singles, heavy pairs
begin-loop LOOP-2 1e16 # LOOP-2 begin

  begin-alternate LOOP-2
    dee_pairs_first once
    dee_pairs_mb    repeat
  end-alternate

  dee_singles_first    repeat
  dee_singles_split_s1 repeat tryall

  begin-alternate LOOP-2
    dee_singles_split_s2 once   fixedbylooger
    dee_singles_split_s2 once   fixedbymayo
    dee_singles_split_s2 repeat fixedbylooger
  end-alternate

end-loop # LOOP-2 end
```

```
end-dee-sequence
```

A.1.2 aa_input file

The aa_input file parallels the rot_input file. Each design site is represented as a line containing a list of allowable amino acids in the search space.

```
ALA VAL ILE LEU
ALA VAL ILE LEU
ALA VAL ILE LEU
ALA VAL ILE LEU
ALA VAL ILE LEU
ALA VAL ILE LEU
ALA VAL ILE LEU
ALA VAL ILE LEU
ALA VAL ILE LEU
```

A.1.3 pos_xlate file

Since backbones may have an unequal number of repacking positions, this file (one for each backbone) describes the relationship between design sites and repacking positions. Each line of the file represents a repacking position for a particular backbone. The number indicates which design site maps to that position. In this example, each design site maps to two repacking sites on the backbone. This allows one to design a homodimer by mapping the same amino acid onto corresponding positions of the backbone.

```
1
2
```

```
3

4

5

6

7

8

9

1

2

3

4

5

6

7

8

9
```

A.2 Example output

A.2.1 out-main file

This is an example of the main output file described in Chapter 2.

```
261.00655 LEU ALA VAL ALA VAL ALA ILE LEU VAL

258.37538 LEU ALA VAL LEU VAL LEU ILE ALA ILE

257.39442 ILE ALA VAL LEU VAL ALA VAL ALA ILE

256.92510 LEU ALA ILE ALA ILE LEU ILE ALA VAL

253.00857 ILE ALA ILE LEU VAL ALA VAL ALA ILE
```

```
252.92679 ILE ALA ILE LEU ILE ALA VAL ALA ILE
252.78859 ILE ALA ILE ALA ILE ALA VAL ALA ILE
249.49349 ILE LEU ILE LEU ILE LEU ILE LEU ILE
249.34565 ILE LEU ILE ALA ILE LEU VAL ALA ILE
249.22933 LEU LEU VAL LEU VAL LEU ILE ALA ILE
248.55404 LEU ALA VAL ALA LEU ALA VAL LEU VAL
244.25409 LEU ALA LEU ALA ILE LEU ILE ALA VAL
244.25409 LEU ALA LEU ALA ILE LEU ILE ALA VAL
244.25409 LEU ALA LEU ALA ILE LEU ILE ALA VAL
244.11819 LEU ALA VAL ALA LEU ALA ILE LEU VAL
243.85756 LEU ALA ILE ALA LEU ALA ILE ALA VAL
241.98496 ILE ALA VAL LEU LEU ALA VAL ALA ILE
241.98496 ILE ALA VAL LEU LEU ALA VAL ALA ILE
241.94467 ALA ALA ILE ALA ILE LEU ILE ALA VAL
241.94467 ALA ALA ILE ALA ILE LEU ILE ALA VAL
240.85163 VAL ALA ILE ALA ILE LEU ILE ALA VAL
240.13572 LEU ALA ILE ALA LEU ALA ILE ALA ILE
239.06802 ILE LEU VAL LEU ILE ALA VAL ALA LEU
239.06802 ILE LEU VAL LEU ILE ALA VAL ALA LEU
238.30167 ALA ALA ILE ALA ILE LEU VAL ALA ILE
```

Bibliography

[1] C. B. Anfinsen. Principles that govern the folding of protein chains. *Science*, 181:223–230, 1973. Nobel Lecture.

[2] C. Branden and J. Tooze. *Introduction to protein structure.* Garland Publishing, 1999.

[3] P. Burkhard, S. V. Strelkov, and J. Stetefeld. Coiled coils: a highly versatile protein folding motif. *Trends in Cellular Biology*, 11:82–88, 2001.

[4] B. Chazelle and M. Singh. A semidefinite programming approach to the side-chain positioning problem. Dept. of Computer Science, Princeton University, personal communication, 2001.

[5] T. E. Creighton. *Proteins: structures and molecular properties.* W. H. Freeman and Co., 1993.

[6] F. H. C. Crick. The packing of alpha-helices: simple coiled-coils. *Acta Crystallographica*, 6:689–697, 1953.

[7] B. I. Dahiyat and S. L. Mayo. De novo protein design: fully automated sequence selection. *Science*, 278:82–87, 1997.

[8] J. R. Desjarlais and T. M. Handel. De novo design of the hydrophobic cores of proteins. *Protein Science*, 4:2006–2018, 1995.

[9] J. Desmet, M. De Maeyer, B. Hazes, and I. Lasters. The dead-end elimination theorem and its use in protein side chain positioning. *Nature*, 356:539–542, 1992.

[10] O. Eriksson, Y. Zhou, and A. Elofsson. Side chain positioning as an integer programming problem. *Lecture Notes of Computer Science*, 2149:128–141, 2001.

[11] G. Ghirlanda, J. D. Lear, A. Lombardi, and W. F. DeGrado. From synthetic coiled coils to functional proteins: automated design of a receptor for the calmodulin-binding domain of calcineurin. *J. Molecular Biology*, 281:379–391, 1998.

[12] M. X. Goemans and D. P. Williamson. Improved approximation algoirthms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42:1115–1145, 1995.

[13] R. F. Goldstein. Efficient rotamer elimination applied to protein side-chains and related spin glasses. *Biophysical Journal*, 66:1335–1340, 1994.

[14] D. B. Gordon, S. A. Marshall, and S. L. Mayo. Energy functions for protein design. *Current Opinion in Structural Biology*, 1999.

[15] D. B. Gordon and S. L. Mayo. Radical performance enhancements for combinatorial optimization algorithms based on the dead-end elimination theorem. *J. Computational Chemistry*, 19(13):1505–1514, 1998.

[16] D. B. Gordon and S. L. Mayo. Branch-and-terminate: a combinatorial optimization algorithm for protein design. *Structure*, 7(9):1089–1098, 1999.

[17] P. B. Harbury, T. Zhang, P. S. Kim, and T. Alber. A switch between two-, three-, and four-stranded coiled coils in gcn4 leucine zipper mutants. *Science*, 262:1401–1407, 1993.

[18] Pehr B. Harbury. draft. personal communication, 2001.

[19] L. Holm and C. Sander. Fast and simple monte carlo algorithm for side chain optimization in proteins: application to model building by homology. *Proteins: Structure, Function and Genetics*, 14:213–223, 1992.

[20] A. Irbäck, C. Peterson, F. Potthast, and E. Sandelin. Monte carlo procedure for protein design. *Physical Review*, 58, 1998.

[21] P. Koehl and M. Delarue. Application of a self-consistent mean field theory to predict protein side-chains conformation and estimate their conformational entropy. *J. Molecular Biology*, 239:249–275, 1994.

[22] P. Koehl and M. Delarue. Mean-field minimization methods for biological macromolecules. *Current Opinion in Structural Biology*, 6:222–226, 1996.

[23] I. Lasters, M. De Maeyer, and J. Desmet. Enhanced dead-end elimination in the search for the global minimum energy conformation of a collection of protein side chains. *Protein Engineering*, 8(5):815–822, 1995.

[24] I. Lasters and J. Desmet. The fuzzy-end elimination theorem: correctly implementing the side chain placement algorithm based on the dead-end elimination theorem. *Protein Engineering*, 6(7):717–722, 1993.

[25] A. R. Leach and A. P. Lemon. Exploring the conformational space of protein side chains using dead-end elimination and the a* algorithm. *Proteins: Structure, Function and Genetics*, 33:227–239, 1998.

[26] C. Lee. Predicting protein mutant energetics by self-consistent ensemble optimization. *J. Molecular Biology*, 236:918–939, 1994.

[27] C. Lee and S. Subbiah. Prediction of protein side-chain conformation by packing optimization. *J. Molecular Biology*, 217:373–388, 1991.

[28] A. Lombardi, J. W. Bryson, G. Ghirlanda, and W. F. DeGrado. Design of a synthetic receptor for the calmodulin-binding domain of calcineurin. *J. American Chemical Society*, 119:12378–12379, 1997.

[29] L. L. Looger and H. W. Hellinga. Generalized dead-end elimination algorithms make large-scale protein side-chain structure prediction tractable: implications for protein design and structural genomics. *J. Molecular Biology*, 307:429–445, 2001.

[30] A. Lupas. Coiled coils: new structures and new functions. *Trends in Biochemical Sciences*, 21:375–382, 1996.

[31] D. L. McClain, H. L. Woods, and M. G. Oakley. Design and characterization of a heterodimeric coiled coil that forms exclusively with an antiparallel relative helix orientation. *J. American Chemical Society*, 123:3151–3152, 2001.

[32] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, and A. H. Teller. Equation of state calculations by fast computing machines. *J. Chemical Physics*, 21(6):1087–1092, 1953.

[33] O. Monera, N. Zhou, P. Lavigne, C. Kay, and R. Hodges. Formation of parallel and antiparallel coiled-coils controlled by the relative positions of alanine residues in the hydrophobic core. *J. Biological Chemistry*, 271:3995–4001, 1996.

[34] N. A. Pierce, J. A. Spriet, J. Desmet, and S. L. Mayo. Conformational splitting: a more powerful criterion for dead-end elimination. *J. Computational Chemistry*, 21(11):999–1009, 2000.

[35] J. W. Ponder and F. M. Richards. Tertiary templates for proteins–use of packing criteria in the enumeration of allowed sequences for different structural classes. *J. Molecular Biology*, 193:775–791, 1987.

[36] J. Saven. Designing protein energy landscapes. *Chemical Reviews*, 101(10):3113–3130, 2001.

[37] V. Sharma, J. Logan, D. King, R. White, and T. Alber. Sequence-based design of a peptide probe for the apc tumor suppressor protein. *Current Topics in Biology*, 8:823–830, 1998.

[38] P. S. Shenkin, H. Farid, and J. S. Fetrow. Prediction and evaluation of side-chain conformations for protein backbone structures. *Proteins: Structure, Function and Genetics*, 26:323–352, 1996.

[39] M. Singh and P. S. Kim. Towards predicting coiled-coil protein interactions. *RECOMB: Annual Conference on Research in Computational Molecular Biology (ACM)*, 2001.

[40] C. A. Voigt, D. B. Gordon, and S. L. Mayo. Trading accuracy for speed: a quantitative comparison of search algorithms in protein sequence design. *J. Molecular Biology*, 299:789–803, 2000.

[41] L. Wernisch, S. Hery, and S. J. Wodak. Automatic protein design with all atom force-fields by exact and heuristic optimization. *J. Molecular Biology*, 301:713–736, 2000.

[42] Jinming Zou and Jeffery G. Saven. Statistical theory of combinatorial libraries of folding proteins: energetic discrimination of a target structure. *J. Molecular Biology*, 296:281–294, 2000.