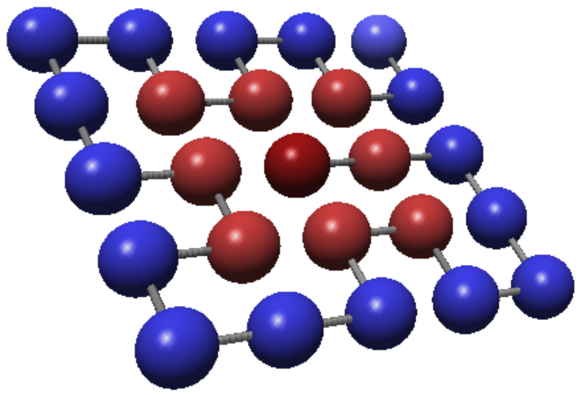
CHALMERS | GÖTEBORGS UNIVERSITET



Protein Folding

Implementation of Stochastic and Deterministic Algorithms using the two-dimensional HP Model for Computer Simulation of the Protein Folding Process

Bachelor's Thesis in Computer Science and Engineering

SIMON ANDERSSON
MALIN ANKER
TOBIAS FORSBERG
TOR HAMMAR
SEBASTIAN HERBERTSSON
ALEXANDER RADWAY

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, June 2013

Protein Folding

Implementation of Stochastic and Deterministic Algorithms using the two-dimensional HP Model for Computer Simulation of the Protein Folding Process

SIMON ANDERSSON
MALIN ANKER
TOBIAS FORSBERG
TOR HAMMAR
SEBASTIAN HERBERTSSON
ALEXANDER RADWAY

Examiner: Prof. Graham Kemp

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Cover: 2D conformation of an amino acid sequence, obtained by our implementation of an algorithm called Zipping and Assembly Mechanism by Dynamic Programming.

Department of Computer Science and Engineering
Göteborg, Sweden June 2013

Abstract

In nature, proteins are often found to instantly fold into the configuration with the lowest amount of free energy. Due to the complexity of this process, it is today still unknown exactly how it occurs. Since proteins have crucial roles in essentially all biological processes, this is an important problem within structural biology. Computational methods can be used to simulate the problem and a large variety of algorithms have been introduced in the field.

Using a widely studied abstraction of this problem, the two-dimensional HP model, this thesis studies the following combinatorial, stochastic and deterministic algorithms: Exhaustive Search, Random Walk, Monte Carlo, Ant Colony Optimization and Zipping and Assembly by Dynamic Programming. The mechanisms, efficiency and results of these algorithms are evaluated and discussed.

Even this simplified version of the problem proves to be challenging and gives interesting insights into principles of the protein folding problem.

Contents

Terms and Definitions

Amino acid - Chemical compound that serve as a building block for proteins. There are 20 different amino acids, each with a specific side chain which could be classified as either polar or hydrophobic.

Benchmark sequences - A list of sequences that have been used in previous literature, see Table 1.

Conformation - The current formation of a sequence.

Conformational space - All the possible conformations of a sequence.

Construction - A self-avoiding walk phase used in Ant Colony Optimization, similar to a Monte Carlo iteration.

Connected neighbors - Any two residues that are adjacent to each other in a sequence, i.e. position i and $i + 1$.

Covalent interaction - Chemical interaction between two atoms which involves the sharing of electrons. This type of intramolecular interaction is much stronger than intermolecular interactions, and is completely fundamental to the formation and structure of molecules.

HH interaction - The interaction between two hydrophobic residues that are topological neighbors.

HP model - Shorthand for "Hydrophobic-Polar protein folding model", which is a highly simplified model used for simulating protein folding, considering each amino acid as being either hydrophobic or polar.

Lattice - A recurring arrangement of points in space.

Native state - The state that a protein naturally adopts by itself. Here considered as the state that corresponds to the minimum energy score of a specific sequence.

H,P Residue - A hydrophobic or polar amino acid that has formed peptide interactions with other amino acids, in order to constitute a sequence. In this thesis, used as the basic units of a sequence, since the simplified HP model is applied.

Scoring function - Evaluation method determining the energy score of a folded sequence.

Self-avoiding walk - The placement of residues on a lattice, where a point can only be occupied by one residue.

Sequence - Here used as a shorthand for "amino acid sequence", a sequence consisting of amino acid residues.

Square lattice - A simplified 2D model of the conformational space in which a sequence can adopt its final conformation.

Topological neighbors - Any two residues, that are not considered connected neighbors, that are positioned next to each other on the lattice.

1 Introduction

The protein folding problem has been rated by Science as one of 125 most important unsolved puzzles in science today [1]. The three-dimensional structures of proteins are determined by the sequence of amino acids – the building blocks of proteins. There are 20 different amino acids participating in the constitution of proteins, all with different properties. After a protein is synthesized, it naturally folds and adopts a specific three-dimensional conformation, called the *native state.*[2]

In nature, this process happens very fast, often in just a few milliseconds, and a question that still is unanswered, is what principles that are used in nature when folding a protein to its native state. Even though this is an intensely studied problem, a solution for how this is done, is still not found.

To simulate all possible conformations that a sequence could form and then evaluate each of their energies based on certain criteria, is not possible for other than small sequences because of the enormous search space involved. There is a proposition, called the Levinthal's paradox, which states that if you were to obtain the correct conformation of a protein consisting of 100 amino acids, by sequentially examining all possible conformations, it would require a time longer than the age of the universe [3].

It has also been proposed that a protein will always adopt the state of lowest free energy. This proposal, together with the fact that the structure is wholly dependent on the order of the amino acids in the sequence, is called Anfinsen's dogma [4], and in this project we will use this assumption when evaluating the folded conformations of amino acid sequences.

In this project, the mechanisms of protein folding are being experimented with using major simplifications such as reducing the three-dimensional space to two dimensions and abstracting the 20 different amino acids to only hydrophobic (H) or polar (P) residues. Five different algorithms and an executable program are created using the Java framework. The algorithms are being evaluated both through individual testing and through comparisons to other algorithms.

1.1 Background

The biological function of the protein is dependent on its three-dimensional structure, which determines how the protein will interact with other molecules. Since proteins make up a major part of our cells and also have crucial roles in essentially all biological processes, the closer we get to understanding the protein folding problem the closer we get to understanding biological systems - including ourselves [5]. Therefore, apart from being a problem of considerable intellectual interest, improvement in the precision of algorithms used to find the native state of amino acid sequences could also lead to the discovery of new medicines. Several neurodegenerative diseases are caused by the improper folding of proteins, and in the case of Alzheimer's Disease for example, protein folding simulations have already helped the progress of the development of a viable drug [6].

1.1.1 Molecular Structure of Proteins

Proteins are the most structurally complex molecules known. Consisting of covalently bonded amino acids, proteins can form large macromolecules arranged in sophisticated three-dimensional conformations. The different amino acids are composed of two parts, one which is identical in all amino acids and constitute the so-called polypeptide backbone, and one part commonly referred to as the side chain, that is unique for each amino acid, and completely determines its properties.[2]

The different properties of the amino acid side chains allows non-covalent bonds to form between parts of the protein molecule, making it possible for the protein to fold into a more favorable energy state. Two important factors in determining which structure a protein will adopt, are the polar and hydrophobic properties of different amino acids. Since hydrophobic amino acids are not able to form hydrogen bonds, they tend to form clusters, in order to minimize their contact with polar molecules. Globular protein molecules often arrange themselves with these amino acids forming a hydrophobic core of the protein, whereas in membrane proteins the hydrophobic amino acids can appear on the surface. In this project, only globular proteins are considered, and the future use of the word *protein* in this thesis will hence refer to globular proteins.[2]

When analyses of three-dimensional protein structures have been carried out, it has been noticed that even though the structure as a whole, is unique, there are two distinct, local patterns that are repeated regularly. The reason to their abundance in many proteins, is that hydrogen bonds also can form between atoms in the polypeptide backbone, making parts of the protein adopt these specific structures. The first one discovered was the α-helix, where hydrogen bonds form between every fourth residue, making the protein adopt a helical structure. The second one, discovered shortly thereafter is the β-sheet. β-sheets actually exists in two different forms, and arises either when a backbone bends itself back and forth, or arranges itself in parallel sheets allowing hydrogen bonds to form between amino acids occurring with longer distances between them in the sequence.[2]

1.2 Reduction to the HP Model

The HP model is one of the most commonly used representations in the field of protein folding. It is based on the fact that hydrophobic interactions are considered to constitute the major force in the folding process [7]. It abstracts the given amino acid sequence into a sequence of only hydrophobic and polar residues. The model can make use of an imagined lattice for the placement of these residues, where each residue can be placed only on the empty sites of the lattice. Possible HP chain alignments are thus built-up through a self-avoiding walk on the lattice.

To evaluate the energy state of a conformation, a scoring function is introduced. Each appearance of two adjacent, but non-consecutive, H residues (hydrophobic residues) in a conformation leads to a decrement of the energy value by -1.0. A conformation with lower score is considered to be better.

The HP model has, since it was first implemented [8], been widely used to study the mechanisms of protein folding. Even though it is such a simplified model, similar structures are found on lattices using the HP model as can be found in nature. One of the most important similarities that is often seen is the forming of hydrophobic cores in order to shield the H residues from the water and to obtain a low energy score. For the HP model, if the sequence length is large enough, there is often just one state of lowest energy. We will refer to this state as the native state and its score as the optimal score.

The lattice model can be used either as a two- or three-dimensional representation of the conformational space. In our project we have chosen the two-dimensional representation in order to be able to focus more on the implementations of algorithms.

1.2.1 HP Model in Two Dimensions on a Square Lattice

A square lattice is designed to drastically simplify the protein folding problem. It does so by enforcing two limitations. The first being the reduction to two dimensions. The second is the discretization of this space to integer precision. Given these limitations it follows that a cell in the lattice is either free or occupied by an amino acid, that any amino acid can have at most four neighbors and that at most three of these are topological. See example of a conformation in Figure 1. Red spheres represent H residues and blue spheres P residues. The lighter color represent the first placed amino acid and the darker the last one.

Conceptually, an algorithm complying with the square lattice will submit to and try to utilize the limitations to gain some advantage.

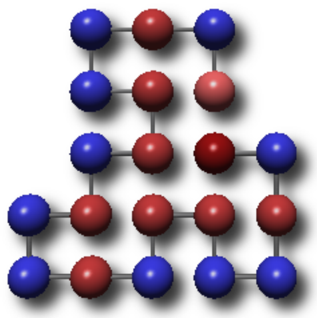


Figure 1: A residue sequence on a 2D lattice, red and blue spheres representing H and P residues respectively.

1.2.2 Relative Movement Scheme

Algorithms residing in the square lattice commonly employ a relative movement scheme since conformations are invariant with respect to rotations. The position of the two starting residues can be fixed without loss of generality. A conformation can therefore be represented by a sequence of relative movements $\in \{Left, Forward, Right\}$

1.2.3 Definition of the Simplified Protein Folding Problem

On a two-dimensional lattice, given a sequence consisting of hydrophobic and polar residues, find the conformation c of the best score according to a scoring function f.

2 Theory

This section starts by describing variants of the scoring function and then introduces some algorithmic approaches to solving the simplified protein folding problem. Besides giving an introduction to the algorithms and their differences, it explains why solving this problem with brute force is not a viable option.

2.1 Scoring Function

In order to evaluate the conformations, we have implemented a scoring function which gives a translation of the energy state for the conformation. As previously stated, a low score is preferred since this represents a low energy state. In the simplified HP model, the hydrophobic interactions are considered to be the driving force for the sequence to fold to its native-like conformation [9]. Therefore, many scoring functions only check for contacts between neighboring hydrophobic residues in the conformation to determine a score. The scoring function is objective and only evaluates completed conformations. The native conformation will often have a core of hydrophobic residues and a shell with polar residues and as many hydrophobic interactions as possible.

2.1.1 Basic Scoring Function

As the hydrophobic interactions are considered to be the driving force of the folding process in the HP model, a basic scoring function that only considers these interactions has been implemented, rewarding connected neighbors with a score of -1.0. This is the energy function we have used when designing the algorithms, to determine what interactions to reward. However, there is use for an extended scoring function to make the generated conformations more realistic. We make a suggestion of how such a function could be designed although we only use it in Section 4.1.

2.1.2 Extended Scoring Function

The extended scoring function will evaluate the folded sequence and generate scores regarding all interactions:

- *As an interaction between two H residues is very favorable and critical to get the sequence folded into the conformation we want, it will generate a score of -1.0.*

- *An interaction between two P residues is not as favorable as an interaction between two H residues, but in our simplified assumption of them having positive and negative groups, it is still favorable and will generate a score of -0.6.*

- *As we want the P residues to surround the H residues, an interaction between them will generate an energy score of -0.6.*

- *As we also want the P residues to be in the shell of the sequence, an interaction between a P residue and water or solvent is favorable. It will generate a score of -0.5.*

- *The sequence should not have a shell of H residues, since this is very unfavorable. An interaction between water or solvent and a H residue will therefore generate an energy score of +1.0.*

In order to evaluate these interactions in the folded sequence the scoring function will first make a representation of the two-dimensional lattice and place the residues at the correct place. The algorithms uses relative moves to place the residues, which is the movement from the previous placed residue to the most recent placed one. The representation of the model uses absolute moves, which is the direction of the sequence relative to the coordinate system. The scoring function then translates the movement of the sequence from relative moves into absolute moves to be able to check which connected neighbors each residue has, to generate the different energy scores.

2.2 Exhaustive Search

An Exhaustive Search algorithm deterministically explores all possible solutions to some problem statement. Subject to the square lattice, this means finding all possible self-avoiding walks for some protein. This section discusses the principles of ES on the square lattice.

ES works by recursively placing one residue from the protein at the time. If all residues have been placed, this classifies as a solution to the self-avoiding walk and coincidentally as a valid conformation in the HP lattice. The solution is enumerated and can be stored in memory. Otherwise, ES attempts to recursively place the head residue to the left, right and in front of the previous residue. If it is impossible for ES to place the head residue (in case all surrounding space is already occupied) ES will backtrack and try another direction for the previous residues until a solution is found. These properties ensures ES is complete and optimal, but runs in exponential time. For all but the two starting residues in an input sequence of length n, there are three possible directions to move, hence there are $O(3^{n-2})$ possible solutions to the simplified protein folding problem. There are a few possibilities for pruning, including the introduction of space constraints to find compact conformations only, but ES is still infeasible for anything but short sequences.

In reality there are far fewer self-avoiding walks, which is described in Section 3.2, but this combinatorial problem still classifies as NP-hard[7] and alternative algorithms are required.

2.3 Random Walk

In contrast to the Exhaustive Search, Random Walk is simply one solution, any solution that classifies as a self-avoiding walk. RW is a process in which the choice of direction is stochastic.
In its basic form the probabilities are equally distributed and not influenced by any heuristic. RW is guaranteed to find some valid solution, and while it makes no claim about the quality, it runs in $O(n)$. The use of RW is based on repetition. Say there are $M = |C^*|$ optimal solutions to a problem of a total of N possible solutions: Also suppose that these optimal solutions occurs uniformly in the solution space. Then RW will find a $c \in C^*$ with probability $P = \frac{M}{N}$. Naturally P could be very small, but this is countered by RW being fast.

Because of the sparseness of optimal solutions in the simplified protein folding problem, RW cannot be expected to deliver any good results on its own. However, in combination with a satisfying heuristic a Random Walk produces much better results. One such heuristic is used in the Monte Carlo algorithm.

2.4 Monte Carlo Algorithm

The Monte Carlo method is a common name for computational methods that rely on random sampling, often combined with some constraint. In this project a chain growth algorithm, based on anticipated importance sampling, is applied.

The implementation of the Monte Carlo algorithm is based on Random Walk, but also has the addition of heuristics favoring hydrophobic connections which stems from the definition of the scoring function. As in both RW and ES, this algorithm recursively places each residue one at a time. Before placing the residue, three positions relative to the most recently placed residue are considered; left, forward and right. Depending on the number of adjacent hydrophobic residues, a probability for each of these positions are calculated. This probability in combination with a pseudo-random number is used to determine the choice of site. If all adjacent sites would be occupied in some step, the algorithm will backtrack and choose the site with highest probability of those left, and continue doing so until one solution is obtained.

While this heuristic is quite intuitive, it is not always correct as the native state may require several non-favorable choices to be reached. Figure 2 shows an instance when the heuristic fails to find the optimal solution, partly because it turns left when leaving the upper area.

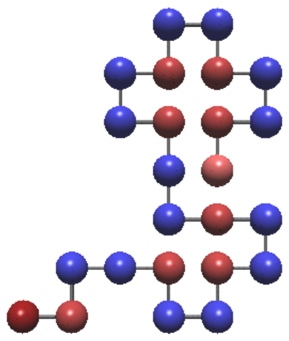


Figure 2: Failure caused by the greedy heuristics of Monte Carlo, in sequence number three from Table 1. See Figure 3 for the optimal solution.

2.5 Local Search

A Local Search algorithm will start with a solution state c, a conformation, and mutate it in search of a new state c' that is better. In general this mutation is any valid and reasonable operation that results in a new solution state c'. These two states are neighboring states and therefore called local.

A widely used analogy that also happens to be the name of an algorithm used to explain principles of Local Search is hill climbing. As any devoted alpinist – or perhaps protein folder hobbyist – knows, one must move in some direction in order to reach the top.

In order to climb a hill the simple heuristic is to move upwards, one step at a time. If the alpinist was blind and had no sense of direction however, he would need help to reach the top. If we equip the alpinist with a device which can relay the current height above the sea, then he or she might find the highest peak with its help. Intuitively, climbing this hill under these circumstances is what a local search algorithm does. It is also subject to issues of climbing the wrong hill, starting out at a flat surface or when walking one step at a time simply is not feasible.
In terms of HP conformations there are several mutating operations that can be performed. These range from micro-scale, safe operations which only modify a small part of the conformation while guaranteeing the validity of c' to larger types of moves that makes no such guarantees[10].

2.6 Ant Colony Optimization Algorithm

Like the name suggests, the idea behind ACO is to simulate ant-like behavior by letting agents explore an area and emit artificial pheromones for other agents to follow. The process of leaving traces in the environment which affect the behavior of other agents is called *stigmergy* and occurs naturally. Stigmergy does not require intelligent agents, but rather relies on principles of self-organisation by reinforcement of successful behavior. In ACO, the strength of the trail is proportional to the quality of the solution. The idea is that given enough time and explored area, it is probable the colony will find the best solution.

The ACO algorithm [11] [12] is a stochastic search method which combines four smaller algorithms: an Exhaustive Search (ES) a Random Walk (RW), a HH heuristic that will also serve in a Monte Carlo (MC) algorithm and a Local Search. In addition ACO provides means of indirect communication of information between these techniques by the use of a *pheromone matrix*. The following paragraphs describe these mechanisms that constitutes the ACO algorithm.

ACO loops through three major phases in which agents (ants) of different roles perform their colony tasks such as searching for food, finding a better routes and emitting pheromones.

In the *construction phase*, the ants are performing random walks, where probability of turning in a certain direction is determined by the HH interaction heuristic as well as pheromone trails left by other ants. The ants are not allowed to emit pheromone trails in this phase. When a walk is completed, it is given a score based on the HH interactions and stored in memory. In the *local search* phase, some fraction of the ants are attempting to further improve the best trails found so far. Finally, in the *update phase*, the most successful ants are asked to leave trails for future reference.

During construction, ants obey the rules of the self avoiding random walk while being influenced by earlier pheromone trails in addition to the HH interaction heuristic. The HH heuristic is identical to the one described in Section 2.4. The exact equation and its parameters is detailed in Section 3.5.1. The ACO random walk algorithm uses a slightly different backtracking procedure in which the ant instead of exhaustively trying every possible way out of a dead end simply retreat half way and try again.

The local search phase attempts to counter two different situations. One in which a strong trail dominates, but leads to a suboptimal solution in the area. The other in which a trail may lead to a good solution, but is perhaps not the best way to get there. Literature [11, 12] indicates the implementation is crucial to ACO, particularly to the quality of solutions.

Reference implementation [11] make use of a *long range move* in the local search phase which randomly selects a position and modifies the direction of the amino acid at that position. This operation is unsafe and in order to complete it some post-processing is required. For each of the following residues, it attempts to place it using the old direction. If this is not possible, it is placed using the HH heuristic, not taking pheromones into account.

In the later article [12] the importance of the local search phase is emphasized and a slight modification of the long range move is applied within an iterative first improvement procedure. This modified procedure is simpler and more effective, and is described in detail in Section 3.5.

During the update phase the environment is simulated by modifications to the *pheromone matrix*. The pheromone matrix can be read during all phases and facilitates the communication between ants at different times. During the update earlier trails evaporate by an inverse *persistence factor* and successful ants are allowed to write to the matrix.

2.7 Zipping and Assembly Mechanism by Dynamic Programming

As the name implies, ZAMDP is a Dynamic Programming approach to the protein folding problem that was proposed by Dill et al. [13, 14]. It is based on the Cocke-Kasami-Younger (CKY) algorithm for parsing sentences in a context free language.

There is substantial evidence that suggest that real physical proteins are folded in a hierarchical manner where small local structures such as α-helices and β-sheets are formed before the main structure [14]. If this is indeed the case, this provides a strong argument for why Dynamic Programming would be a suitable way to model the folding process.

The CKY algorithm works by splitting a given sentence into its constituent words and classifying them according the unit production rules of the context free grammar that defines the language. Words are then combined into longer and longer strings according to the rules of the grammar. For example, the words ”eat” and ”sushi” will in the initial phase of the algorithm be classified as a verb and a noun respectively. Both words can then be combined into a verb phrase, ”eat sushi”, since the grammar contains a rule stating that a verb phrase can be constructed from a verb and a noun. In this way the algorithm creates one or more binary parse trees which all represent different interpretations of the sentence. If at least one such tree exists where the root is a rule which in the grammar is defined as a ”start symbol” the entire sentence is valid.

The ZAMDP algorithm uses the same Dynamic Programming approach of ”local first, global later” as CKY but instead of combining strings into longer strings according to rules defined in a grammar, it folds short substrings of the sequence first and then assembles the best solutions found until the entire sequence is folded.

The first step in the algorithm is to split the sequence into short subsequences, each of which only contains one hydrophobic residue. These subsequences correspond to the words in the CKY algorithm. An exhaustive search is used to enumerate all possible conformations for these subsequences which typically are not that many since the sequences are short.

These folded sequences are then assembled like pieces of a jigsaw puzzle to form larger conformations. For every combination of two small conformations only those with the best score are stored and used in the next assembly level. When the whole sequence has been folded, the best solutions are returned.

In both CKY and ZAMDP this is accomplished by using a so called parse chart which is simply a matrix that in each cell stores the best solutions for its corresponding subsequence. If the original sequence is split into n parts the chart will be of the size $n \times n$ where each cell $chart[i][j]$ will contain the best conformations found for the subsequence from i to j. The chart is initialized by filling the diagonal cells ($chart[i][i]$ for $i = 1..n$) with the exhaustively enumerated solution for each initial subsequence i. The algorithm will then work its way up through each subsequent diagonal, $chart[i][i+1]$, $chart[i][i+2]$ and so on until it reaches the top right cell $chart[1][n]$ which will contain the best conformations for the entire sequence. Each cell in the chart except for the initial diagonal is filled by assembling all possible combinations of conformations from the cells $chart[i][i+k]$ and $chart[i+k+1][j]$ for $k = 0..j-i-1$.

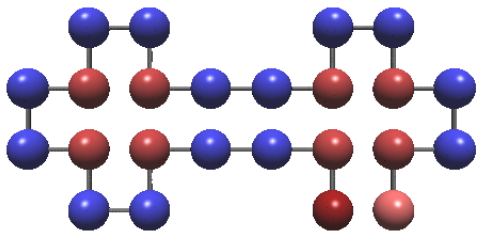
When the algorithm is finished the optimal conformations found for the given sequence will be the ones stored in the top right cell $chart[1][n]$.

2.8 Testing

The sequences from Table 1 have been used previously in literature [15]. They provide a variety in both sequence length and composition, and more importantly, they all have a known global minimum which make comparisons against our obtained optimal solutions possible. To acquire a more extensive testing, the algorithms will also be run with all possible sequences from a length of seven up to ten amino acids. For short sequences like this, Exhaustive Search is used to obtain the global minimum for all sequences. In this way the correctness of the algorithms can be critically evaluated for short sequences. For time efficiency testing, sequences from the length of 11 amino acids up to 20, are randomly sampled.

Sequence	Length	Global Min
1 $H_3P_2HPHPHP_2HPHPHP_2H$	20	-10
2 $HPHP_2H_2PHP_2HPH_2P_2HPH$	20	-9
3 $H_2P_2HP_2HP_2HP_2HP_2HP_2HP_2H_2$	24	-9
4 $P_2HP_2H_2P_4H_2P_4H_2P_4H_2$	25	-8
5 $P_3H_2P_2H_2P_5H_7P_2H_2P_4H_2P_2HP_2$	36	-14
6 $P_2HP_2H_2P_2H_2P_5H_{10}P_6H_2P_2H_2P_2HP_2H_5$	48	-22
7 $P_2H_3PH_8P_3H_{10}PHP_3H_{12}P_4H_6PH_2PHP$	60	-34
8 $H_{12}PHPHP_2H_2P_2H_2P_2HP_2H_2P_2H_2P_2HP_2H_2P_2H_2$ P_2HPHPH_{12}	64	-42

Table 1: Benchmark sequences.



H-H: -9.0

Figure 3: Sequence 3, $H_2P_2HP_2HP_2HP_2HP_2HP_2HP_2H_2$, an optimal solution.

3 Implementation

The project has revolved around the Java framework, with code being produced and tested on six different computers. In order to facilitate the compilation of our written code, we decided to use the source code management system Git [16]. Separating the framework into smaller modules enabled us to work separately on different algorithms and on other parts of the program – a way of working that has allowed us to always have a runnable and functional version of the program ever since the first working release. The program itself has gradually evolved from a state where only running one algorithm, ES, was possible to the final state, where the user is able to choose between different algorithms to execute and to select parameters unique for every algorithm. There is also a score along with each conformation.

The implementation of the scoring function, algorithms, testing methods and GUI are described in the following sections.

3.1 Scoring Function

When the algorithms have folded the residue sequence, the program will store all the data from the conformation in an immutable object called a "Fold". This is the object the scoring function will work with to generate a score for the conformation, it is not a tool to use in heuristics of the algorithms. Therefore, it will not evaluate subsequences within the conformations during the folding process.

First, the scoring function will create a representation of the lattice and place the conformation on it. Then, the scoring function starts with the first residue of the sequence and checks the neighboring places on the lattice with a constraint to 90 degrees for interactions with other residues, ignoring the topological neighbors. The scoring function will therefore always check on two neighboring places in the lattice except for the first and last amino acids in the sequence, which will be checked at three neighboring places.

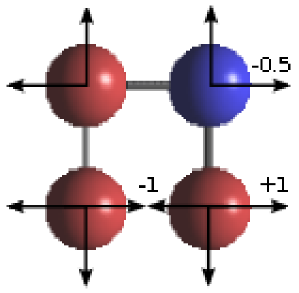


Figure 4: Illustration of checked neighbors and example of assigned score.

This will then be used to generate a certain score for each residue, and then summarized to the total score of the folded residue sequence. The scoring function is configured to store all different connected neighbors in different arrays, so the user of the program can choose what type of score should be presented (i.e. the user can choose to only present the scores for the HH connected neighbors).

3.2 Exhaustive Search

In order to generate all possible conformations for a given sequence, we implemented a recursive solution. The ES algorithm goes through all residues in the sequence from the starting point to the final point. On each recursive level, it will at first attempt to make a leftward step. If this would lead to a collision with a previously placed residue, an attempt is made at an upward movement and if this also fails, it will attempt to make a rightward step.
Once the base case is reached, meaning that the very last residue in the sequence has been placed, the produced conformation is stored and we move back recursively to obtain the next possible conformation.

A rectangular box constraint was implemented for the ES algorithm. Maximum width and height values can be given as arguments to the algorithm, leading to that the generated conformations will all lie within a box of the specified dimension. Since we can often assume that the optimal solution lies within a limited rectangular box when using the basic (HH) scoring function (and always when using the extended scoring function, see Section 2.1), wherein the conformation is forced to be maximally compact, using this method can shorten the execution time needed to find optimal solutions for a given sequence.

The minimum surface (the perimeter) the molecule can have, P_{min}, is defined as [8]:

$$P_{min} = 2\,(n + 1 - t_{max}) \tag{1}$$

where t_{max} = the largest number of topological neighbors and n = sequence length.

If $P = P_{min}$, or in other words that the box is minimal so that only maximally compact conformations are generated, the number of conformations that must be explored is reduced by a factor of $\sim e^{-n}$ [8].

3.3 Random Walk

Similar to ES, Random Walk is implemented using recursion. Traverse the sequence and place residues one by one to gradually define the conformation. In every step taken, the three relative movements (left, right and forward) are randomized for the residue. If the first attempted relative movement succeeds – meaning that it preserves the self-avoiding walk – we move to the next residue recursively. Otherwise, the other movements are tried in the same manner. Once the base case is reached, or in other words when a movement has been set for the last residue, the conformation is returned.

3.4 Monte Carlo Algorithm

This algorithm is basically a self-avoiding random walk, but as mentioned earlier, with the addition of heuristics favoring hydrophobic connections.

In our implementation the first two residues in the sequence receive a given direction and are then placed at a fixed position on the lattice. The algorithm uses the direction of the previous residue to determine which of the adjacent sites on the lattice would be of interest when placing the next. Each residue has four adjacent sites in the 2D-lattice representation, and since the site where the last residue was just placed is not considered, there are three sites of interest. A clarification of this process can be seen in Figure 5.

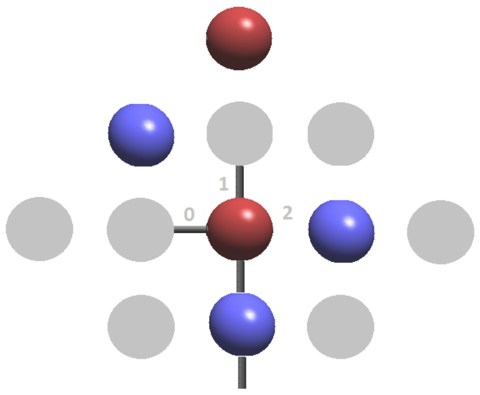


Figure 5: The next H residue should have a large probability of being placed in site 1, a small probability of being placed in site 0 and zero probability of being placed in site 2.

In order to determine the different probabilities of the next H residue being placed at the three adjacent sites, we calculate the number of topological hydrophobic neighbors at each site. Since these are just probabilities, and we use a pseudo-random number generator to make the actual decision, we added a weight, w, for the user to be able to adjust how favorable the HH interactions should be considered. The following, normalized, equation was derived to calculate the probability of the next amino acid being placed in site i:

$$P(i) = \frac{h_i(nrH_i(w+1)+1)}{nrS + \sum_{j=0}^{2} nrH_j(w+1)} \tag{2}$$

where $i = 0, 1, 2$ represents the site in the relative direction (left, forward or right) of the last placed residue, and h_i is 1 if site i is not occupied by a residue, and 0 if site i is occupied. nrS represents the number of adjacent sites that are available. Larger values of w will increase the impact potential HH interactions will have on the the probability. If all of the adjacent sites are occupied this calculation will not be performed. The algorithm will then retreat to the prior step, and move the last residue placed, to the site next in line of actual placements. This procedure of stepping backwards in the placement process will be performed until all residues in the sequence can be placed.

In the case of placing a P residue, no consideration is taken to the surroundings, and the probability is calculated as follows:

$$P(i) = \frac{h_i}{nrS} \tag{3}$$

In order to determine the actual placements, we use a pseudo-random number ($0 \leq rnd \leq 1$). Since the probabilities from (2) sum to one we then make the following choices:

Left	Forward	Right
rnd≤P(0)	P(0)<rnd≤P(0)+P(1)	P(0)+P(1)<rnd≤P(0)+P(1)+P(2)

Table 2: Rules for using the calculated probabilities, when determining actual placements.

3.5 Ant Colony Optimization

The ACO algorithm was written in Java and like the other algorithms uses the relative moving scheme. The implementation mimics the general idea of any ACO algorithm with separate Construction, Local Search and Pheromone Update phases. Internally the algorithm uses the original scoring function based on HH interactions and returns the globally best solution found during the execution.

3.5.1 Construction phase

The Construction phase is performed using a backtracking RW algorithm very similar to the MC algorithm. The folding direction and relative starting point has been shown [11] to be important to the effectiveness of ACO, however this implementation folds exclusively from one end to the other and does not allow arbitrary starting positions.

In addition to the landscape, ants are also influenced by the pheromone trails in their choice of direction during the random walk. Initially, any direction is as probable as another as there are no pheromone trails and no HH interactions. With time, ants will start influencing each other via the pheromone trails per the following equation:
Let step i, direction d, heuristic probability h and pheromone influence w. Then we have that probability p of choosing direction d in step i is

$$p_{i,d} = \frac{w_{i,d}^{\alpha} \cdot h_{i,d}^{\beta}}{\sum_{k \in \{L,S,R\}} w_{i,k}^{\alpha} \cdot h_{i,k}^{\beta}} \tag{4}$$

The balance between heuristics and pheromones can be adjusted using the α and β parameters.

The backtracking mechanism will not exhaustively try all possibilities to create a conformation, like the exhaustive search, rather it will retreat halfway of the distance folded so far before simply trying again. This behavior is important primarily in longer sequences, where self-avoidance is more difficult [12]. This implementation does not remove the option to select a previously failed direction. It is not clear if choice has any real implications.

3.5.2 Local Search Phase

The Local Search phase is an important part of the ACO and aims to improve existing conformations by modifying them according to a mutation scheme. If the resulting conformation c' is better, it will normally replace the original conformation c. The rationale [12] is that it is easier to improve an already good conformation than to construct a new one.

ACO implements the modification operation from [12] in which an arbitrary residue a_n is selected and its direction is mutated uniformly at random. The remaining sequence will then adapt to the new situation in the following way:
Conditioned that it is still feasible, a_{n+1} will use its previous relative direction with probability $P = 0.5$. If not, it will select a different direction with probabilities proportional to the HH heuristic h_{id}.

The *iterative first improvement procedure* implemented here is greedy in the sense that it will be satisfied with the first improvement found. In contrast to the original variant [11], but in line with [12] it only accepts improvements.

In this implementation the top 5% of the current iterations conformations are selected as candidates for improvement. Furthermore it makes as many attempts to improve the conformation as there are residues in the sequence.

3.5.3 Pheromone Phase

The pheromones, which importance increases with the length of the sequence [11], are stored in a matrix T of sequence length i and directions d width to hold the pheromone strength T_{id}.

During each phase, before any ants are allowed to leave trails, the existing pheromones will evaporate from the environment by an inverse persistence factor ($0 < p \leq 1$).

Similar to the local search phase only the top 5% conformations will leave a trail of pheromones for the next swarm of agents to follow. In this phase the table is updated according to $T_{i,d} \leftarrow T_{i,d} \cdot p + \delta T_{i,d}$.

In [11] the weight factor δ is called the relative solution quality and is calculated as $E(c)/E^*$ *"where E^* is the known minimum energy or an approximation based on the number of H residues in the sequence"*. Since no such approximation is to be found, and would require solving the problem in part this implementation will determine a trivial upper bound if no score is provided as argument.

As with all situations including probabilities there is a risk for underflow or infinitesimal values to appear. A measure to prevent this situation and search stagnation [12], is *threshold renormalization*. This measure is currently not implemented.

Figure 6 shows the pheromone matrix development over several iterations where yellow/red colors indicate strong scent.

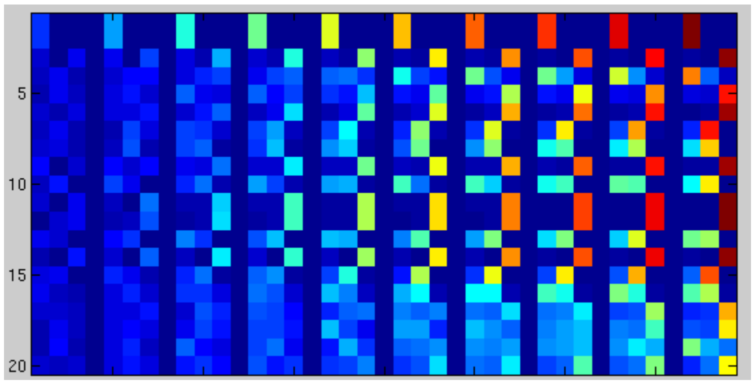


Figure 6: Pheromone matrix of length 20 (vertical, i), developing over 10 iterations (horizontal). Each cell represent the pheromone strength, T_{id}, for the three directions d.

3.6 Zipping and Assembly Mechanism by Dynamic Programming

The ZAMDP algorithm was written in Python. This implementation was initially intended as a prototype to be translated to Java and incorporated into our main framework at later stage. One reason for this was that the early phase of the implementation could benefit from a simple graphical interface in which the algorithm could be executed step by step interactively which is a feature that is not available and never intended to be available in the main Java application. Due, in part, to time constraints as well as additional requirements from the ZAMDP algorithm that the framework simply did not provide, the Python implementation was never translated to Java directly, however we still incorporated it in the main application by calling the Python executable from within a Java module conforming the frameworks interfaces.

One of these additional requirements was that ZAMDP in its initialization phase uses exhaustive search to enumerate all possible conformations of small subsequences which it then assembles into the complete conformation. This use of the exhaustive search algorithm within the main algorithm proved to be difficult to achieve in the framework we had started to create.

As for the implementation itself, the main principle of the algorithm, which is described earlier, in many ways closely mimics that of the CKY algorithm. A parse chart is constructed and traversed in the same way, however the assembly procedure is where ZAMDP differs significantly from CKY. At any given cell in the parse chart except for the initial cells, i.e. the diagonal, the conformations to be stored in that cell are created by assembling every possible pair of smaller conformations that are folded from two adjacent subsequences that together form the larger subsequence that the current cell represents.

This assembly phase constitutes one of the main parts of the algorithm. Two HP-sequence conformations need to be assembled into one larger conformation. This can in some ways be thought of as assembling two pieces of a jigsaw puzzle but with slightly relaxed restrictions. Depending on how the smaller conformations are shaped they can at most be assembled in nine different ways. This number comes from the fact that an end point on one of the conformations to be assembled has at most three unoccupied neighbor cells which can be used as the connection point for the second conformation. For any one of these three starting points, the second protein can at most be rotated in three different directions, the fourth being blocked by the end point of the first conformation. The number is of course often less than nine since some or even all of the three possible connection points can be blocked by other parts of the first conformation. Additionally the conformations may be folded in such a way that collisions occur in other parts of the sequence for some or all possible rotations in any of the connection points.

When all pairs are assembled for one cell in the parse chart, the best solutions are saved and later assembled with other conformations. Eventually the top level is reached, which only contains one cell, representing the entire sequence. The assembly phase is performed as usual on all possible combinations of two folded subsequences that together form the complete sequence. The best solutions found in this cell are the ones that are returned by the algorithm.

3.7 Testing Methods

To test the different algorithms, the user needs to run the main program in headless mode (without GUI) and specify the test case to execute along with what algorithm to use in the test and specify arguments for the given algorithm (unless default values are desired).

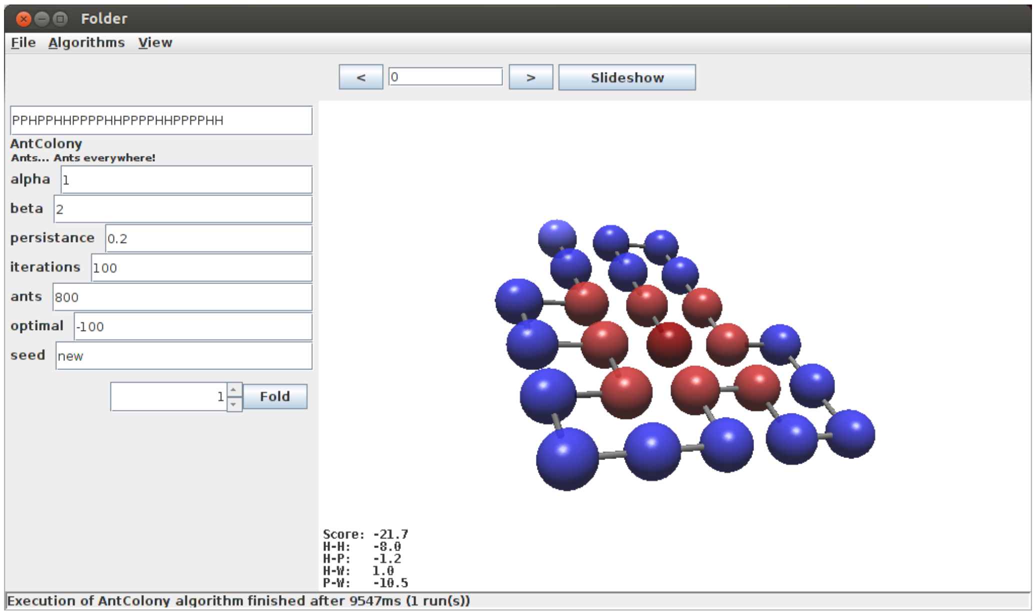
We have defined two different main tests, which are the following:

- **Small test** For this test, the user is required to specify the sequence length in addition to other parameters. If it is the first time that the user runs this test, a test data file will be generated by running all possible amino acid sequences with the given sequence length, using Exhaustive Search to find the optimal solution for each sequence (hence, the sequence length has to be shorter than 11 amino acids). Otherwise, an already existing test data file will be loaded.
- **Medium test** This test executes the benchmark sequences as shown in Table 1.

Once the test is finished, the results are written to a test result file, which will contain the results of each sequence in the test run as well as a summary of all test sequences

3.8 GUI

We developed a GUI in the modular-based and platform-independent Swing framework, together with JOGL [17], to allow for fold visualizations (see Figure 7 for its appearance). The user can easily input any desired sequence of H and P residues and then select the algorithm that should generate a conformation or a list of conformations based on this sequence. After algorithm execution, the result is displayed on a GL canvas as a two-dimensional chain (see Figure 1), that may be rotated and translated in three dimensions.



PPHPPHHPPPPHHPPPPHHPPPPHH

AntColony
Ants... Ants everywhere!

alpha 1
beta 2
persistance 0.2
iterations 100
ants 800
optimal -100
seed new

1

```
Score: -21.7
H-H:   -8.0
H-P:   -1.2
H-W:   1.0
P-W:   -10.5
```

Execution of AntColony algorithm finished after 9547ms (1 run(s))

Figure 7: GUI.

3.8.1 Swing

Operations based on user input are defined through the use of action listeners. We have used the observer pattern thoroughly to allow for communication between the GUI and the rest of the framework. For instance, when the user chooses to execute the selected algorithm with given parameters, this execution is launched as a subprocess to prevent the event dispatching thread from halting. As the GUI view class is added as an observer of the model class (responsible for starting an algorithm execution), the GUI receives the generated conformations once the execution is completed and the GUI view class thus observes a change.

3.8.2 JOGL

JOGL, which integrates with the Swing widget sets, provided us with Java bindings for the OpenGL API [17]. We used drawing routines provided by the OpenGL Utility Library (GLU), with material and lighting properties set, to render the spheres used to resemble hydrophobic and polar residues.

The rendering function builds up the conformation by walking a relative path through the resulting list of residue orientations. If the next residue in the conformation takes a leftward or rightward step, the OpenGL matrix is rotated -90 degrees and 90 degrees respectively relative to previous residues.

Unprojection is used to enable centering of the conformation on the GL canvas. The GL canvas minimum and maximum coordinate bounds are converted to OpenGL coordinate values, where after the width and height of the given conformation is calculated. The size and placement of the rendered conformation is adjusted to the values gained in this process, effectively centering it on the screen.

The possibility exists to save what is currently displayed in the GL canvas and store it in .tga (TARGA) format. This is achieved by reading a block of pixels from the frame buffer after scene flushing and then storing it into a new file with a defined TARGA header structure.

4 Results and Discussion

This chapter presents and provides discussions of the results from performed tests and evaluation methods.

An evaluation of the two scoring functions to investigate which one favors conformations with a hydrophobic core, is performed as well as an evaluation of the execution time of ACO and ZAMDP for sequences of increasing length. We also provide a detailed discussion of the data obtained from these algorithms. The three stochastic algorithms – RW, MC and ACO – are compared using the benchmark sequences. We also study the effects that different native conformations have on the folding rate and running time of ZAMDP. Finally, we discuss what else could have been done in the project, if there had been more time.

4.1 Comparison Between Basic and Extended Scoring Functions

To compare the different scoring functions, a test using Exhaustive Search on the sequence $HPH_3PH_2P_6$ was performed, where two given optimal solutions for each scoring function are shown in Figure 8 and in Figure 9. This comparison shows how the extended scoring function also take into account that the hydrophobic amino acids should avoid water. However, for this particular sequence, it does not generate as many HH interactions as the basic scoring function.

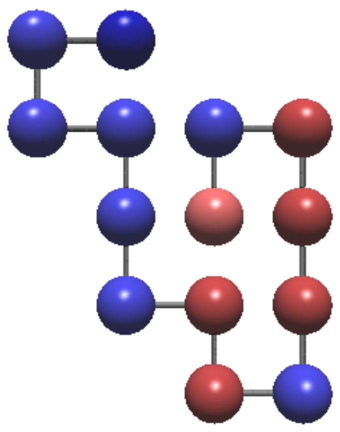


Figure 8: An optimal solution obtained using the basic scoring function, with a HH score of -3.0.

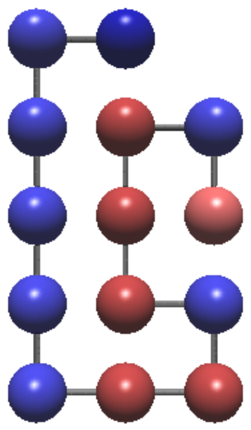


Figure 9: An optimal solution obtained using the extended scoring function, with a total score of -7.0 (and a HH score of -2.0).

4.2 Time Comparison between ACO and ZAMDP

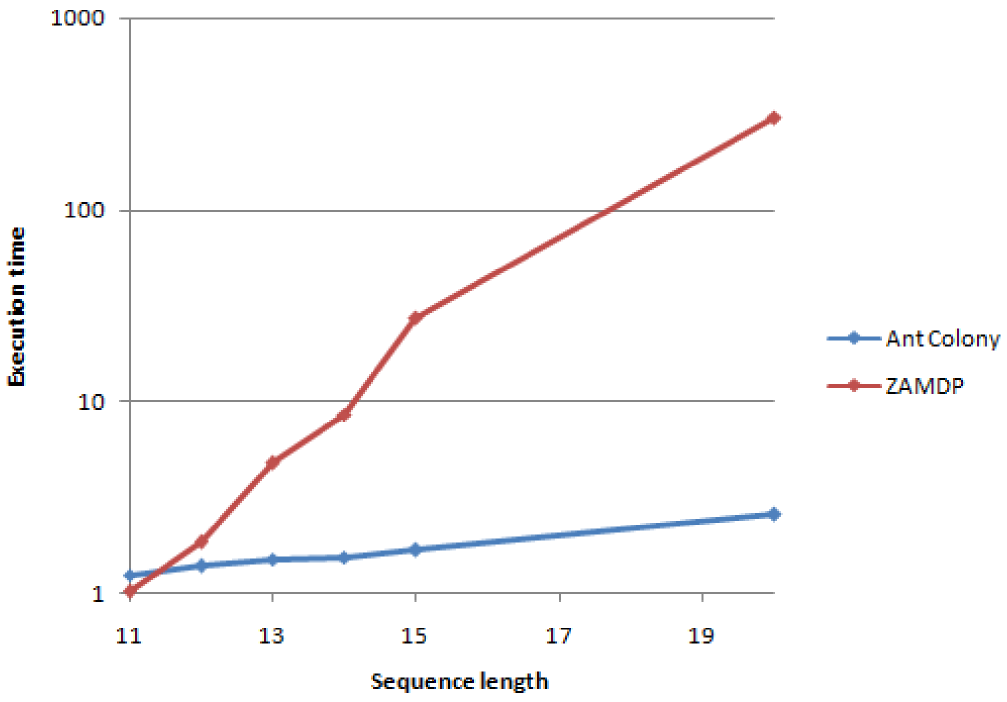


Figure 10: Average execution time for the ACO and ZAMDP algorithms, the execution time scale is logarithmic.

As seen in Figure 10 the execution time of the ACO algorithm grows linearly as the length of the sequences grow, whereas the execution time of ZAMDP algorithm grows exponentially. This makes ZAMDP very time-consuming for longer sequences. For the shorter sequences of length 7 to 10 (referred to as the small test) both ACO and ZAMDP found all the native-like solutions. On these sequences ZAMDP was the faster of the two algorithms, which makes ZAMDP an excellent algorithm for these sequences, even though proteins in nature are never close to being this short.

Due to the deterministic logic of ZAMDP, the execution time for each sequence can vary a lot. The ZAMDP divide the sequence at every hydrophobic residue into subsequences. Therefore, if the sequence has a conformation with many subsequent hydrophobic residues, the ZAMDP algorithm has to test many more different possibilities to match the subsequences. The result of this is a longer execution time.

The stochastic logic of ACO has the benefit of not over-elaborating any sequences. The downside of this logic is an increase in the risk of not finding the optimal solution for a sequence. ACO has the possibilty to take in parameters, in contrast to ZAMDP, which affects the execution time as well as the probability to find the optimal solution. The test result seen in Figure 10 is executed with default parameters. As one may assume, ACO is a much faster algorithm than ZAMDP. On the other hand the correctness is traded for the speed in ACO, and where ZAMDP only takes one run to find its optimal solution, ACO sometimes have to be run numerous times to obtain its optimal solution.

4.3 Random Walk and Monte Carlo

We performed the benchmark sequence test on both the Random Walk and Monte Carlo algorithms between 10^2 and 10^6 times each. The lowest score along with how many times it was found was stored for each sequence and can be seen in Tables 3 and 4. For the MC algorithm the variable w was set to the value 10 which, for example, would yield a 0.85 probability for a H residue to move in the direction of another H residue, if the adjacent sites of the other two directions are empty. See equation (2) for calculations.

Random Walk

Seq	**Length**	**Global Min**	**RW Min** 10^2	**RW Min** 10^3	**RW Min** 10^4	**RW Min** 10^5	**RW Min** 10^6
1	20	-10	-4 (2)	-6 (1)	-7 (1)	-8 (3)	-8 (18)
2	20	-9	-5 (1)	-6 (3)	-7 (1)	-8 (1)	-9 (1)
3	24	-9	-4 (2)	-6 (2)	-7 (1)	-7 (2)	-8 (2)
4	25	-8	-3 (4)	-4 (2)	-6 (1)	-5 (30)	-6 (12)
5	36	-14	-6 (3)	-7 (2)	-9 (2)	-9 (7)	-11 (1)
6	48	-22	-8 (4)	-11 (2)	-13 (1)	-14 (2)	-16 (1)
7	60	-34	-21 (1)	-21 (2)	-23 (2)	-26 (3)	-27 (2)
8	64	-42	-18 (1)	-24 (1)	-23 (1)	-26(1)	-26 (4)

Table 3: Best energy score found for different number of iterations for the benchmark sequences. Number in parentheses indicates conformations found.

Monte Carlo

Seq	**Length**	**Global Min**	**MC Min** 10^2	**MC Min** 10^3	**MC Min** 10^4	**MC Min** 10^5	**MC Min** 10^6
1	20	-10	-8 (1)	-9 (1)	-10 (1)	-10 (12)	-10 (51)
2	20	-9	-6 (1)	-8 (2)	-8 (10)	-9 (7)	-9 (101)
3	24	-9	-5 (7)	-7 (6)	-8 (3)	-9 (1)	-9 (10)
4	25	-8	-4 (1)	-5 (2)	-6 (2)	-7 (4)	-7 (17)
5	36	-14	-9 (1)	-10 (2)	-11 (1)	-12 (1)	-14 (1)
6	48	-22	-13 (1)	-15 (2)	-17 (1)	-17 (9)	-19 (1)
7	60	-34	-26 (2)	-30 (1)	-32 (1)	-32 (3)	-33 (7)
8	64	-42	-26 (2)	-30 (1)	-30 (2)	-32 (2)	-34 (1)

Table 4: Best energy score found for different number of iterations for the benchmark sequences. Number in parentheses indicates conformations found.

The MC algorithm provides a lower score than the RW algorithm in all but two cases. These cases (sequence two and four and iterations 10^6 and 10^4 respectively) have the same score. We can conclude that the probability of finding a low energy conformation increases with the number of iterations, and that these conformations are harder to find with increasing sequence length.

An unexpected result was that the MC algorithm did not find the global minimum for sequence number four, the sequence that holds the highest global minimum of all benchmark sequences. It is also only slightly longer than the the first three sequences, which the MC algorithm did find the global minimum for. This indicates that there are sequences of moves involved that are improbable for the MC algorithm to make.

We can however find the global minimum with the ACO algorithm which lets us take a closer look at the conformation, which can be seen in Figure 11.

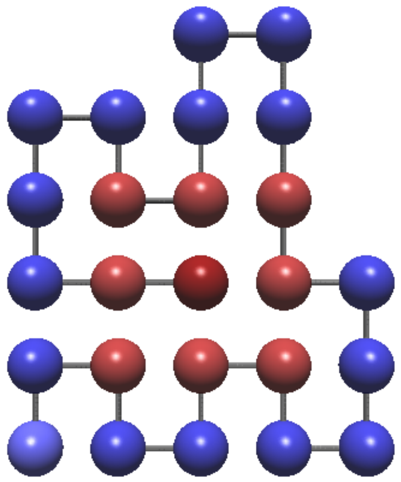


Figure 11: Benchmark sequence 4 global minimum found with ACO algorithm.

The first residue placed is the light blue sphere in lower left corner. We clearly see a hydrophobic core and observe that whenever a H residue can be placed to make an interaction with another H residue, it is. This contradicts with the assumption of improbable moves for the MC algorithm. If we instead look at the H and P composition of the sequence we see that there are three different subsequences containing four subsequent P residues. These subsequences will not have any heuristic and to have them placed in the exact manner as in Figure 11 would correspond to a very small probability.

If we also investigate sequence one, two and three from Table 1 we see that there are no more than two P residues in a row in any of those sequences. This might explain why the MC algorithm and its greedy heuristic folds the three first sequences optimally fairly often for one million iterations. Another result in support of this statement, is that only one occurrence of the optimal solution in sequence five is found - a sequence with two subsequences of P residues, each of length five.

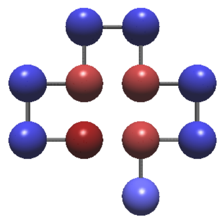
4.4 Zipping and Assembly Mechanism by Dynamic Programming, Results and Discussion

One interesting quirk of the way ZAMDP works is that the rate at which a sequence is folded, and thus the running time of the algorithm, is greatly affected by properties of the solution such as the optimal score, the number of conformations that obtain that score and the way those conformations are folded, i.e. their secondary structure. This comes from the fact that the folding rate is directly affected by the number of partially folded subsequences that need to be combined in order to find the final conformation. Since only the best conformations in every cell are stored and assembled with other conformations in later steps, the running time will be reduced if conformations with a low energy are found early in the algorithm, thus removing conformations with higher energy form the search space.

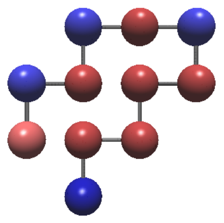
One effect of this is that if a certain sequence has an optimal score close to zero it tends to have many conformations with this score and very large search space to traverse during the execution of the algorithm. This of course leads to a very long running time.

As mentioned earlier, the folding rate is also affected by the final conformation's secondary structure. To explain this we need to establish the term ”local interaction” which we will define as a HH interaction consisting of two hydrophobic residues that are close to each other on the sequence. How close they need to be to be ”local” will depend on the length of the sequence but the details are not very important. The point is that since ZAMDP works by folding and assembling small subsequences first according to the ”local first, global later” principle of dynamic programming, local interactions will be found early in the execution of the algorithm and thus reduce the search space rather than let it grow exponentially as is the case otherwise.

A trivial example of this occurs when folding all of the 60 sequences of length 11 that have a unique optimal conformation. The running times of these 60 executions of our ZAMDP implementation ranges from approximately 0.1 seconds to 3.7 seconds. The optimal conformations from the executions with the shortest and the longest running time are shown in Figures 12a and 12b, respectively.



(a)



(b)

Figure 12: Two conformations with different fold rate.

Both of these conformations have a score of -4, however their interactions are different. For these short sequences we will define local interactions to be only those interactions consisting of two H residues with at most two other residues between them on the chain, which is the closest that two residues in interaction can be to each other. Using this definition we see that for the "fast" conformations, three out of its four interactions are local whereas the "slow" conformation only has one local interaction. How this affects the running time is further evidenced by the fact that the parse tree created during the fast execution has at most 14 folded subsequences in any given cell while as many as 192 conformations are stored in one single cell during the slow execution.

These results are very interesting since they closely mimic the way folding rates for real physical proteins depend on their native structures. Proteins with high occurrence of α-helical structures tend to fold faster than structures dominated by β-sheets[13] and one can easily see the resemblance between α-helices and high occurrence of local interactions. This further strengthens the theory that physical proteins do indeed fold in a hierarchical manner similar to that in which ZAMDP works[14].

Sequence	**Length**	**Global Min**	**ZAMDP solution**	**Execution Time (s)**
1	20	-10	-10	32
2	20	-9	-9	20
3	24	-9	-9	23
4	25	-8	-8	236
5	36	-14	-14	12337
6	48	-22	–	–
7	60	-34	–	–
8	64	-42	–	–

Table 5: ZAMDP test results for the benchmark sequences.

Since the ZAMDP algorithm is a deterministic algorithm that relies on a few basic principles there is not a whole lot that can be tweaked or modified to get better results without changing those fundamental principles. There are however two points where decisions are made that could have been made differently. The first of these points is in how to split the given sequence in the initial subsequences that are to be folded using exhaustive search. The proposed method is to split it in a way such that every subsequence only contains one hydrophobic unit [14]. The reasoning behind this is not explained in the article but our assumption is that by limiting the number of hydrophobic units to one, you prevent the exhaustively enumerated initial conformations from forming any interactions and getting a score other than zero. This in turn, prevents the algorithm from filtering out any of the initial conformations which, although not providing any interactions to the solution, would have been more advantageous building blocks for the later stages of the algorithm.

The second point of decision is exactly that of filtering. As mentioned earlier, only the best conformations in every cell are stored and used in subsequent steps. One could just as well have saved all of them. Only saving the best solutions would seem to put us at risk of losing other of the above mentioned advantageous building block. However, this was the proposed method [14] and as of yet we have not completed any run of ZAMDP for which it did not find the optimal solution. It is worth mentioning that the original developers of the ZAMDP algorithm successfully folded 96.6% of all 24,900 HP sequences of length 20 into their single optimal conformation [13]. We have unfortunately not been able to perform such rigorous tests.

At one point we discussed whether it would be possible to enhance the performance of ZAMDP by storing previously found optimal conformations for small substrings and simply using them in the assembly phase rather than finding them again. However by the same reasoning as the one behind restricting the initial subsequences to only contain one hydrophobic unit in order to not lose any of the inital building blocks that would come to better use unfolded, we decided against it.

Since at least the initial phases of the algorithm are inherently parallel, there could quite possibly be some performance to be gained from distributing at least parts of the executing on multiple processors although we never had the opportunity to test this.

4.5 Monte Carlo and Ant Colony

Given they execute a comparable number of random walks, we expected the ACO algorithm to perform better than a straightforward Monte Carlo algorithm. However as the benchmark results (Table 6) show we have to analyze the situation a bit further.

Sequence	**Length**	**Global Min**	**ACO best solution**	**MC best solution**
1	20	-10	-10	-10
2	20	-9	-9	-9
3	24	-9	-9	-9
4	25	-8	-8	-7
5	36	-14	-13	-14
6	48	-22	-18	-19
7	60	-34	-31	-33
8	64	-42	-32	-34

Table 6: Test results for the ACO algorithm executed with 1000 ants and 1000 iterations and the MC algorithm executed 10^6 times for each benchmark sequence.

For consistent performance in ACO it is important that the sequence is folded from a random starting position[11]. Doing so requires a mechanism to fold in two directions and means to decide which direction to choose at any given point in time. The referenced implementations use a probabilistic method to decide which direction to extend. The probability of a certain direction is the number of residues remaining at the respective end divided by the total remaining residues in the sequence.
Our implementation hence suffers a great disadvantage since it is currently only capable of folding in one direction. Yet another measure to counter search stagnation[12] (threshold renormalization) remains unimplemented. We have not specifically studied the effects of these features, but they are both vital components for a competent ACO implementation.

We wish to show the effects of the Local Search phase and Pheromone matrix by devising another comparable situation. In order to do so, we need to limit the effectiveness of the Ant Colony construction phase, so that it relies more on the other mechanisms. We devised a special sequence targeting the HH heuristic of the algorithms thereby weakening both the MC algorithm and Ant Colony construction phase. The special sequence $S_{spec} = P_3H_3P_8H_3P_6H_3P_8H_3P_3$ has long parts of subsequent P residues between small groups of H residues and an optimal HH score of -8. We then compared Ant Colony and Monte Carlo using $5 \cdot 10^2$ ants and at most 10^2 iterations versus $5 \cdot 10^4$ runs in Monte Carlo.

Score	ACO ratio	MC ratio
-8.0	34%	8%
-7.0	48%	46%
-6.0	18%	46%

Table 7: Test results for ACO versus MC algorithms on sequence S_{spec}.

As can be seen from Table 7, ACO performed better despite the disruption of the HH heuristic, which forced ACO to rely on Local Search and Pheromones to discover the solutions. We cannot tell if the final solution was found during the Construction or Local Search phase, however we observe that an optimal solution (there are several) was found more frequently in ACO then than in MC. Examples of resulting conformation are displayed in Figure 13 and 14.

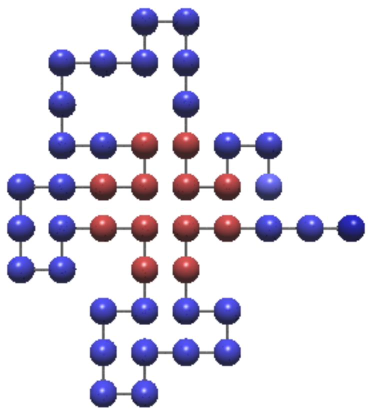


Figure 13: ACO finding an optimal solution in $\leq 5 \cdot 10^4$ constructions.

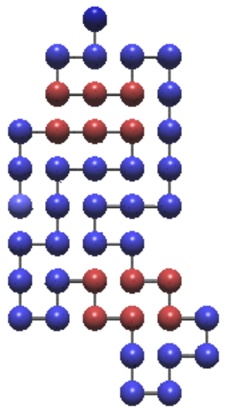


Figure 14: As this picure nicely illustrates, as an effect of being unguided by lengths of P residues, HH heuristic alone often fails to identify a globally optimal solution. Monte Carlo, $\leq 5 \cdot 10^4$ iterations sequence S_{spec}.

4.6 Ant Colony Optimization Development

The colony size is possibly the primary parameter of ACO and determines the number of random walks performed in the construction phase. It is important that the colony explores a large area of the search space, so that many different candidates are found. Failure to do so will prevent an effective Local Search phase. The search space area grows exponentially with the length of the sequence and long sequences require more ants in order to achieve sufficient coverage.

It would be interesting to try this principle and shift focus over time by decreasing the number of exploring ants and increasing the number of improving ants. Thereby assuming that some conformation that is close to the optimal solution has been found and that it is more important to spend time improving it, than to expand the area.

When following this line of thought we also considered to have two different improvement phases, the first performs the large kind of mutation described in this report and the second polish the resulting conformation by performing micro operations on top of it, if such an operation improves the result.

While on the topic of mutations there is another intuitive and safe mutation operation we would want to try. The *pullmove*[10] principle is simple: Select a residue which can be moved to a neighboring location. Place this residue at this location and pull all the previous residues along with it to fill the empty space. We have prepared code for a pull move, but unfortunately we did not have the time to evaluate it.

All sequences have different properties and like the optimal score, there is an optimal number of conformations, e for which improvements should be attempted [11]. This number is important for optimizing the cost and to benefit the ratio of the algorithm. This depends on the colony size as well as the sequence composition and length. The literature[11] is not clear on whether to use a fixed number of ants or a proportion of the colony size. In our implementation e is 5% of the colony size. It would be interesting to use only fixed number of ants regardless of the colony size to see if there is a lower bound to this number.

Since the local search process is stochastic, another factor of interest is the number of attempts made to improve a certain conformation. Naturally it is infeasible to construct and evaluate all neighboring states, and constructing too few weakens the process. A function of the sequence length is a natural candidate and is also used in other implementations.[11]

In an attempt to improve the general performance of our ACO it would be interesting to use a different internal scoring function such as the extended scoring function presented in this report. This would however change the rules of the game.

4.7 Future Directions

We initially intended to expand from the HP model that we are currently using by including a third dimension. However we made the decision to stay in two dimensions and instead focus on further development of the algorithms. One reason for this is that some parts of the algorithms would have been hard to translate to a more complex model.

There are many ways in which this project could have been expanded, given more time. These mainly include improving the model and making it more realistic but there are also other algorithms that would have been interesting to study. Some examples of further directions that we would have liked to pursue are listed below.

4.7.1 Three-dimensional Lattice

The HP model on a cube lattice is not much different from the square lattice. The extra degree of freedom would yield a larger conformational space. This has a maximum size approximately 5^{n-1} where n is the chain length [7]. A larger conformational space would require more computer power for performing the ES, for example. There could be constraints on the ES on the cube lattice as well - instead of implementing a rectangle we could implement a box that lessens the search space. All of our algorithms, except ZAMDP, could be implemented on a cube lattice using straightforward methods. This is because the conformational space for each subsequence that ZAMDP manages becomes larger and thus the increase in total CPU time becomes too large.

4.7.2 Protein Folding Off-lattice

To move closer to the real protein folding problem, we would eventually have to move off-lattice. This has been done previously [18] by gathering known three-dimensional structures of real proteins. These structures are then used as templates for other sequences of residues. By aligning the proposed sequence to the templates and estimate the energy states of each of these structures, an assignment is made. Then the stability of the new structure is tested by using a lattice model and dynamic Monte Carlo simulations.

While we see no direct comparison to the model we have implemented we observe the occurrence of what could resemble α-helices and β-sheets in our two-dimensional HP Model. These are so-called secondary structures of a real protein and the forms adopt naturally during the folding process. Translated to the two-dimensional HP Model they will look like the structures of Figure 15 and Figure 16. These type of structures can be found in many of our larger conformations.

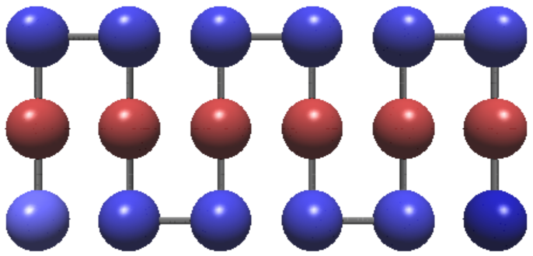


Figure 15: α-helix structure in 2D HP Model.

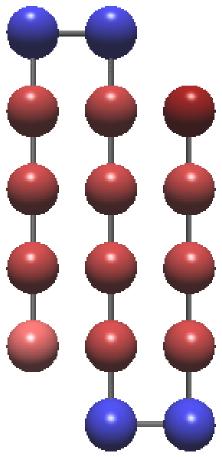


Figure 16: β-sheet structure in 2D HP Model.

4.7.3 Temperature

To make protein folding simulations more like the actual process that occur in nature we could choose to implement a measure of temperature in some of our algorithms. The idea behind introducing temperature is to imitate increasing molecular activity with increasing temperature. We could implement it in the MC algorithm to make the probability *less* likely to move in direction of other H residues when having a high temperature. This would correspond to make a move requiring a high energy which is more likely for high temperatures. In [19] for example, there is a temperature parameter implemented that makes the particular algorithm described greedy for low temperatures.

4.7.4 Improving Algorithm Execution Time by the use of Threading, Distributed Computing and GPU Programming

To lower the execution time of our algorithms (and thus be able to generate solutions for longer sequences within reasonable time), we could have chosen to use threading in the algorithms where this would be possible, making use of the additional cores in multi-core processors. Several worker processes could execute in parallel to solve subproblems in an algorithm and contribute their result to the final result once completed. Likewise, distributed computing could be used (albeit likely more time-consuming to implement), to let large and time-consuming problems be distributed onto many computers, similar to the way Folding@Home works [6].
A third possibility to decrease execution time could be to make use of the computer's GPU(s) in addition to the CPU(s). This would likely involve usage of the OpenCL framework [20] or NVIDIA:s proprietary CUDA platform [21] which both gives an application access to a GPU for non-graphical computing.

4.7.5 Genetic Algorithm

It has been found that a genetic algorithm (GA) can improve the search effectiveness dramatically, when using the HP model [22]. Of this reason, we had in mind to implement a GA together with an auxiliary MC algorithm, similar to the solution described in the article mentioned above. However, our limited time did not allow us to implement a fully working GA.

The basic idea behind GA:s is to mimic the principle of natural genetic evolution, in order to solve computational problems. In a GA, a population of candidate solutions is maintained, which in application to the protein folding problem would be constituted of conformations. Execution of the GA is carried out through several generations, in which three important genetics-inspired operations are carried out. These are the following [23]:

- **Crossover (or recombination)** The exchange of data between two parents, in order to produce new offspring. Selection of data for crossover could occur at a random or set point in the data sequence.
- **Selection** The creation of a new generation according to certain fitness values. Fitness can be either how successful a candidate solution has been in terms of produced offspring or the probability that it will continue its existence, to later on reproduce.
- **Mutation** The replacement of the value at a randomly chosen point with a randomly chosen new value.

Unger and Moult explains in their article *Genetic Algorithms for Protein Folding Simulations* [22] how a simple GA together with the Metropolis Monte Carlo procedure can be used with the HP model on a two-dimensional lattice to find the functional conformations of proteins.

In this approach to the protein folding problem, a population of evolving conformations of size N is maintained.
First in the GA, as explained in the article, each conformation in the population is subject to a set number of MC steps, which gradually alters its structure. This state corresponds to the mutation stage. A mutation is accepted if it meets the MC acceptance criteria (preventing formations that are not self-avoiding).
After this, crossover is performed. Selected conformations get their data switched to generate new conformations, which are accepted, again, if the MC criteria are met. In order to find a valid solution, all three possible rotations when connecting the first structure to the second are checked (0°, 90° and 270°).
The chance, $p(S_i)$, of a structure i being selected for crossover, or in other words the MC acceptance criteria, is equal to its energy value divided by the total sum of energy values in the population [22].

Accepted conformations from the crossover phase continues into the next generation if $E_k \leq E_{ij}$, where E_k is its energy value and E_{ij} equals the averaged energy value of its parents.
The crossover operation is used to get $N-1$ new hybrid structures into the next generation. The N:th added element is the conformation with lowest energy, which always gets copied directly to the next generation.

After a certain number of generations has passed, the solution is finally obtained by returning the conformation(s) with the lowest energy value from the last generation.

5 Conclusions

We set out to gain an understanding of the protein folding problem by implementing and studying various models and algorithms that are used in research of this area. Even though we mainly used the highly simplified 2D HP model rather than something more realistic we feel that we got a great sense of the extreme complexity of the actual problem. By studying algorithms based on such fundamentally different approaches as Monte Carlo, Ant Colony Optimization and Dynamic Programming we really forced ourselves to think about the many aspects that make protein folding such a hard and interesting problem.

The interdisciplinary nature of protein folding and bioinformatics in general is something that really appealed to us as a group with different backgrounds in both cell biology and physics as well as computer science. This was further expanded upon, through the studying of Ant Colony Optimization which uses one biological phenomenon to model another and ZAMDP which is based on something so seemingly unrelated as computational linguistics. We feel that we in most cases accomplished our goals with this project and we have still barely scratched the surface of the fascinating world of computational biology.

References

[1] Kennedy D and Norman C. What Don't We Know? *Science*, 309:78–102, 2005.

[2] Alberts B, Johnson A, Lewis J, Raff M, Robert K, and Walter P. *Molecular Biology of the Cell*, chapter Proteins, pages 125–193. Garland Science, New York, 2008.

[3] Zwanzig R, Szaba A, and Bagchi B. Levinthal's paradox. *Proc. Natl. Acad. Sci. USA*, 89:20–22, 1992.

[4] Anfinsen CB. Principles that Govern the Folding of Protein Chains. *Science*, 181:691–692, 1973.

[5] Berg J, Tymoczko JL, and Stryer L. *Biochemistry*, chapter Protein Composition and Structure, pages 25–64. W. H. Freeman and Company, New York, 2007.

[6] Folding@home. http://folding.stanford.edu/English/FAQ-Diseases, 2013. Online; accessed 14 Feb 2013.

[7] Dill KA. Theory for the Folding and Stability of Globular Proteins. *Biochemistry*, 24:1501–1509, 1985.

[8] Lau KF and Dill KA. A Lattice Statistical Mechanics Model of the Conformational and Sequence Spaces of Proteins. *Macromolecules*, 22:3986–3997, 1989.

[9] Xinchao Z. Advances on Protein Folding Simulations Based on the Lattice HP Models with Natural Computing. *Applied Soft Computing*, 8:1029–1040, 2008.

[10] Thachuk C, Shmygelska A, and Hoos HH. A replica exchange Monte Carlo algorithm for protein folding in the HP model. *BMC Bioinformatics*, 8:342, 2007.

[11] Shmygelska A and Hoos HH. An Improved Ant Colony Optimization Algorithm for the 2D HP Protein Folding Problem. *AI 2003,LNAI 2671*, pages 400–417, 2003.

[12] Shmygelska A and Hoos HH. An Ant Colony Optimisation Algorithm for the 2D and 3D Hydrophobic Polar Protein Folding Problem. *BMC Bioinformatics*, 6:30, 2005.

[13] Dill KA. Computational Linguistics: A New Tool for Exploring Biopolymer Structures and Statistical Mechanics. *Polymer*, 48:4289–4300, 2007.

[14] Dill KA, Hockenmaier J, and Aravind KJ. Routes Are Trees: The Parsing Perspective on Protein Folding. *PROTEINS: Structure, Function, and Bioinformatics*, pages 1–15, 2007.

[15] HP Benchmarks. http://www.cs.sandia.gov/tech_reports/compbio/tortilla-hp-benchmarks.html. Online; accessed April 2013.

[16] GIT. http://git-scm.com/, 2013. Online; accessed 19 May 2013.

[17] JOGL. http://jogamp.org/jogl/www/, 2013. Online; accessed 7 May 2013.

[18] Godzik A, Kolinski A, and Skolnick J. Topology Fingerprint Approach to the Inverse Protein Folding problem. *Journal of Molecular Biology*, 227:227–238, 1992.

[19] Bastolla U, Frauenkron H, Gerstner E, Grassberger P, and Nadler W. Testing a new Monte Carlo Algorithm for Protein Folding. *Proteins*, 32:52–66, 1998.

[20] OpenCL. http://www.khronos.org/opencl/, 2013. Online; accessed 20 May 2013.

[21] CUDA. http://www.nvidia.com/object/cuda_home_new.html, 2013. Online; accessed 20 May 2013.

[22] Unger R and Moult J. Genetic Algorithms for Protein Folding Simulations. *J. Mol. Biol.*, 231:75–81, 1993.

[23] Mitchell M. *An Introduction to Genetic Algorithms*, chapter 1. MIT Press, 1998.