

Iterative Monte Carlo Protein Design

Camilo Rostoker
University of British Columbia
Department of Computer Science
rostokec@cs.ubc.ca

Reza Lotun
University of British Columbia
Department of Computer Science
rlotun@cs.ubc.ca

January 19, 2005

Abstract

A modification to the iterative design of proteins in the 2D HP model is made, employing local search of sequence space and Monte Carlo sampling of compact conformation space. Our algorithm finds supersets of the solution set, and by varying parameters can minimize the size of this superset to any desired accuracy.

1 Introduction

The so-called *central dogma* of molecular biology states that the process of decoding genetic data goes from DNA to RNA to proteins. A protein is composed of a chain of amino acids called its *polypeptide chain*. The three dimensional structure of a protein play vital roles in determining its function, thus understanding the relationship between its amino acid chain and structure are of vital importance. The process of determining the structure or *conformation* of a protein from its amino acid sequence is known as *protein folding*. The inverse problem – determining an amino acid sequence that folds into a given conformation – is the problem of *protein design*.

The protein design problem is highly dependent on the choice of model for protein structure representation and amino acid interactions, and the energy evaluation of conformations in the model. Models can vary from atom-based quantum mechanical simulations, to two dimensional lattice models with a reduced amino acid alphabet. One simple useful model is the 2D-HP model [3] first proposed by Lau and Dill in 1989, where the twenty naturally occurring amino acids are categorized as either hydrophobic (H) or polar (P) on a two dimensional lattice or grid, and where a conformation of a polypeptide chain is represented as a self avoiding walk on the lattice. The energy of a conformation in this model usually involves a function of the number of non-adjacent HH contacts in the resulting structure.

This simple classification is motivated by the observation that the hydrophobic force is the dominant force in protein folding, and that native conformations of globular proteins

are compact with dense hydrophobic cores surrounded by a shell of polar amino acids exposed to the solvent. The HP-lattice simplification is also attractive in that exact enumerations of sequences and structures (compact ones) are possible, providing a means of testing correctness of a proposed design method, though exact enumeration eventually becomes infeasible for large sequence lengths. Self avoiding walks and lattice problems are also well studied and reasonably well understood, allowing for the application of many diverse tools to the problem domain. It is important to realize however that the HP model, especially applied to a two-dimensional lattice, does not give any direct indication of underlying protein structure, but allows for the study of other properties related to a protein's structure, such as general patterns of hydrophobic interactions, and values such as the protein's surface-to-volume ratio.

In this paper we introduce the Iterative Design approach to protein sequence design, originally developed by Rossi et al, and then present a modification of this algorithm along with implementation details and analysis. The remainder of the paper is structured as follows: Section 2 reviews some of the existing work in the field of protein sequence design. Section 3 reviews the original Iterative Design algorithm, and Section 4 presents our modified version of this algorithm with detailed discussion on rationale for the changes. We conclude Section 4 by discussing various implementation details such as optimization using simulated annealing, reduction of sequence space using basic laws of symmetry, as well the methods we use to generate self-avoiding walks of varying compactness. In Section 5 we describe a series of computational experiments designed to investigate our algorithm, and Section 6 presents concluding remarks as well as avenues of exploration for future work.

2 Existing Work

As noted by Deutsch and Kurosky in [1], irrespective of the model used, three criteria are essential in determining the success of the determined sequence.

1. The sequence should have the desired conformation when it is in its ground state (positive design).
2. There should be no ground state degeneracy – that is, the sequence should always fold into the same conformation (negative design).
3. There should be a large gap in energy of the ground state and the energies of the higher energy states – the sequence is stable.

From a thermodynamic perspective, given some energy function E which depends on the interactions between H and P amino acids, the conditional probability of a sequence σ folding into conformation Γ_0 at temperature β is denoted by

$$P(\Gamma_0 | \sigma) = \frac{1}{Z(\sigma)} \exp[-E(\Gamma_0, \sigma) / k_B T]$$

where the *partition function* Z is defined by

$$Z(\sigma) = \sum_{\Gamma} \exp[-E(\Gamma, \sigma) / k_B T]$$

An often used expression called the “ensemble free energy” is given by

$$F(\sigma) = -T \ln Z(\sigma)$$

where k_B is Boltzmann’s constant (since our energy function for HP is the number of HH contacts and this is not physically meaningful, we usually combine the temperature and Boltzmann’s constant term in to $\beta = 1/k_B T$). Maximizing the above conditional probability with respect to the sequence is difficult because it entails the exploration of both sequence and conformational space (as a sequence is fixed in the numerator, the denominator must then range over all conformations for that sequence).

Early work in protein design can be viewed as a maximization of the conditional probability, and can be seen to approach the problem in three ways (see [3] and [7]):

1. The free energy is ignored or set to a constant, and the conditional probability maximization is cast as a minimization of $E(\Gamma_0, \sigma)$. This is usually a fast approach, but may produce spurious sequences (all H’s), thus some constraints or parameters are added to enforce the desired ratio of H’s to P’s in the sequence. This is theoretically unsatisfactory and fails on a number of examples.
2. Approximation of the free energy is made via a high-temperature Taylor expansion [1] of F , taking lower order terms and ignoring constants. While more accurate than energy minimization, the assumption of high temperature is inaccurate since folding takes place at low temperature.
3. Using Monte Carlo (MC) approaches to estimate F , either in the form of nested MC approaches by fixing the sequence, or using multisequence MC approaches where both the sequence and conformation are updated simultaneously [2]. Such methods are more advantageous than the former two since they make no temperature assumptions and take the free energy into account, but can be computationally intensive, since large parts of conformation space are needlessly taken into account when calculating the free energy F .

In this work we present an algorithm which attempts to intelligently search through sequence and structure space, retaining the accuracy of a method which takes the ensemble free energy into account, but one which does not “waste” time on spurious solutions. The following section describes one such solution approach, and Section 4 describes our improvements to it.

3 Iterative Design Approach

A novel approach incorporating an iterative design strategy is presented in [5] and [6]. The idea is to incorporate negative design by maintaining a set of *decoy* structures which compete with the target native state to “weed out” spurious sequence solutions. The approach is independent of the specific function used to calculate amino acid interaction potentials and also works for three dimensional lattice approaches.

Let $P_\beta(\Gamma | \sigma)$ be the probability of the conformation given the sequence at temperature β^{-1} . For a sequence s with native state Γ , the *folding temperature* β_F^{-1} is defined so that

$$P_\beta(\Gamma | \sigma) > \frac{1}{2}$$

for all $\beta > \beta_F$. Given some conformation Γ , all sequences σ with a probability of folding into Γ larger than 1/2 will have their unique ground state in Γ and folding temperature greater than β^{-1} . Thus

$$\begin{aligned} \frac{\exp(-E(\Gamma_0, \sigma)\beta)}{\sum_{\Gamma} \exp(-E(\Gamma, \sigma)\beta)} &> \frac{1}{2} \\ -E(\Gamma_0, \sigma)\beta - \ln\left(\sum_{\Gamma} \exp(-E(\Gamma, \sigma)\beta)\right) &> -\ln(2) \\ -E(\Gamma_0, \sigma)\beta + \beta F(\sigma) &> -\ln(2) \\ E(\Gamma_0, \sigma)\beta - \beta F(\sigma) &< \ln(2) \\ \beta[E(\Gamma_0, \sigma) - F(\sigma)] &< \ln(2) \quad (*) \end{aligned}$$

provides a check to isolate sequences that *could* fold into the given conformation Γ_0 . The problem with this approach is that the calculation of $F(\sigma)$ ranges over the entire space of structures, and must be performed each time anew with the given σ .

The iterative design strategy attempts to restrict the sum in F to a subset of conformation space. We define the *decoy set* D as this subset, which is initially empty or set to the target conformation itself. At each iteration we exhaustively check conformation space (or more realistically all *compact conformations*) and test our set of possible solutions produced by filtering sequence space by (*), and discard those sequences that have lower energies on conformations other than the target. We discard those sequences that do have lower energies on other conformations, but add those conformations to D. At the next iteration (*) ranges over the new D, and the process continues until we have a sufficient number of solutions or we reach some maximum iteration limit.

1. Initialization:
 - a. TARGET = the given target structure
 - b. SEQUENCES = set of all sequences amongst which solutions will be exhaustively searched
2. Initialize Decoys:
 - a. DECOYS = set of decoys, is initially empty or set to target structure
3. ITERATE until STOP criterion is met:
 - a. Exhaustively search out sequence space to find a sequence S such that:
 - i. S has minimum energy on TARGET (where S is not already in the solution set)
 - ii. S has lowest energy on all structures in DECOYS
 - b. Explore the structure space of compact structures to find the lowest energy state(s) LES of S and compare it to the energy obtained on TARGET.
 - c. Compare lowest energy states obtained in 3(a) with TARGET:

```

    FOREACH L in LES
        isSolution = true
        If L ≠ TARGET and energy(S,L) ≤ energy(S,TARGET)
            isSolution = false
            L is added to DECOYS

        if (isSolution)
            add S to the set of solution sequences

STOP criterion is:
- A sufficient number of solutions has been found
- No more sequences can be found satisfying equation in step 3(a)

```

Listing 3.1: Pseudo-code for Iterative Design Algorithm

Given a target conformation and energy function, an exhaustive enumeration of sequence space is performed to obtain a sequence of minimum energy for that target. This sequence is then fixed and an exhaustive search over conformational space is made to find conformations of equal or lower energy. If none are found, the sequence is added to the list of known solutions, and is removed from sequence space, and the procedure continues. If there *are* conformations other than the target for the sequence, then those conformations are added to our set of decoys. Another search of sequence space is performed, but this time the potential sequence solution S must have minimum energy E on our target structure such that E is greater than any energy obtained by S on all structures from the decoy set.

The above iterative procedure takes into account the free energy and also has complexity comparable to fast energy minimization approaches [5]. However, exhaustive searches of sequence space (for fixed conformation) and compact conformation space (for fixed sequences) are made which could present a hindrance for the algorithm to scale to large conformations. Also, increasing the amino acid alphabet would adversely affect running time. The choice of the number and type of decoy structures affects the accuracy of the method as well.

4 Our Modification to the Iterative Design Algorithm

The main idea behind our proposed modifications is to utilize a combination of the iterative design strategy and two approximation methods, namely Monte Carlo sampling and simulated annealing. Our work takes a novel approach to the Protein Design problem by providing a flexible means to specify the degree of desired accuracy of the solution set. We do this by eliminating the exhaustive steps of searching sequence and conformation space, and replace them with approximation methods that can be adjusted to achieve different results with respect to the tradeoff between accuracy and efficiency. We also introduce two new ideas into the iterative design: pre-populating the decoy set, and a refinement step in which the set of solution sequences is “refined” in order to detect

degenerate sequences that have been falsely accepted. The figure below shows the pseudo-code for our algorithm.

```
1. Initialization:
   a. TARGET = the given target structure
   b. SEQUENCES = set of all relevant sequences amongst which solutions will
      be searched

2. Initialize the collection of decoy structures
   a. DECOYS = batch_of_pivots(targetStructure)

3. ITERATE until STOP criterion are met:

   a. S = search sequence space using Simulated Annealing and an Approximate
      Free Energy Sum calculation

   b. nativeStates = search conformation space using Iterative Growth Walk
      Monte Carlo sampling

   c. Determine if S has unique ground state on TARGET:

       if isempty(nativeStates)
           add S to set of solution sequences
           numSolutions = numSolutions + 1
       else
           S is not a solution
           add nativeStates to DECOYS

   d. Refine solutions

4. STOP criterion is:
   a. A sufficient number of solutions has been found
   b. No more sequences can be found satisfying equation in step 3(a)
   c. After step 3, if no solutions found and refining yields no degenerate
      solutions, then consider the search "stale" and quit.
```

Listing 4.1: Pseudo-code for our new algorithm

4.1 Representation of Conformations

Self avoiding walks (SAWs) on a 2D lattice are represented in three ways:

- 1 **Coordinate Representation:** Beginning at the origin (0,0) a list of coordinates occupied by the SAW is given in an Nx2 matrix, where (N-1) is the length of the Self-avoiding walk (since we start at the origin). This is the principal representation used by most of the algorithms in our codebase.
- 2 **Quad Representation:** The standard representation in the literature for 2D SAWs represents them as strings of letters from {U, R, D, L} denoting the directions of "Up", "Right", "Down", and "Left" on the lattice respectively. This representation is also useful as an intermediate representation between the coordinate representation and ternary representation described below.
- 3 **Ternary Representation (Balanced Ternary):** A self-avoiding walk is described by a series of local moves and is represented by a string of letters

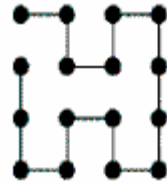
from {F, R, L} denoting the moves “Forward”, “Right” and “Left” respectively. Such strings can be represented by ternary (base 3) strings {L=1, F=2, R=3} or even more conveniently as *balanced ternary* {L=-1, F=0, R=1}. The primary use of ternary representation is in checking for symmetrical SAWs (walks that have been rotated, or reflected along the horizontal, vertical and diagonal axes). Adopting a convention of representing the first move of a SAW by “Forward” in balanced ternary, all symmetrical walks will be represented in either the same balanced ternary string, or some string where R and L are interchanged (this can be conveniently achieved by multiplying one walk by -1).

4.2 Pre-populating the decoy set

One of the keys to the iterative design is the concept of decoy structures. As discussed by Rossi et al. in [5], the most significant contribution to the conformational free energy of a given sequence summed over all possible conformations comes from those structures that have a similar set of H-H contacts as the native structure [5]. Thus, it is clear that the set of decoys contribute to the accuracy of the free energy approximation, which in turn directly affects the quality of the solutions and hence the overall efficiency of the algorithm. Moreover, these “competing” structures are used to minimize the sequence space that must be searched by enforcing an additional integrity check on the sequence before the costly step of searching for low energy states through the vast regions of the conformation space.

The original Iterative Design algorithm starts with an empty decoy set, and in a follow-up paper[6], the authors initialize the decoy set to include the target structure as well as one other compact structure. In our algorithm, we have incorporated a method that, given a target native structure, can construct a relatively good set of decoy structures. As we derived previously, the constraint set by the original authors, $\beta[E(\Gamma_0, \sigma) - F(\sigma)] < \ln(2)$, simply suggests that the approximation must imply that the sequence has a *more than likely chance* of folding to the given conformation at temperature β . Thus, the decoy set plays an increasingly dominant role as the algorithm proceeds. Initially, when there are only a few decoys, the free energy sum is rough and imprecise, but as the number of decoys increase, the approximation becomes more accurate. Our algorithm therefore attempts to overcome this problem by populating the decoy set *a priori*, resulting in an increased performance when searching both sequence and conformation space. In order to accomplish this, we use a modification of the Pivot Method [10,13]. This works by randomly choosing a pivot site in the self avoiding walk, and randomly applying one of seven fundamental group transformations of the dihedral group of order 4 (i.e. rotation by 90, 180, 270, reflection across vertical, horizontal and two diagonals) to the structure at the chosen pivot point. We do this iteratively until we have a set of decoys which are all highly correlated to the given target structure, and then proceed with the algorithm as usual.

We wanted to see exactly what kind of affect decoys have with respect to the free energy approximation of a sequence. We therefore created a test using the standard benchmark conformation (shown in figure 4.2.1 on the right) for sequence length $N=16$, and used four sequences: one which is a designing sequence (a solution sequence), the all H and all P sequences, and another chosen at random. We then calculated the free energy approximations for an increasing number of decoys, obtained using the modified pivot method described above to produce conformation highly similar to the target structure. Figure 4.2.2 below shows the result:



N = 16

Figure 4.2.1

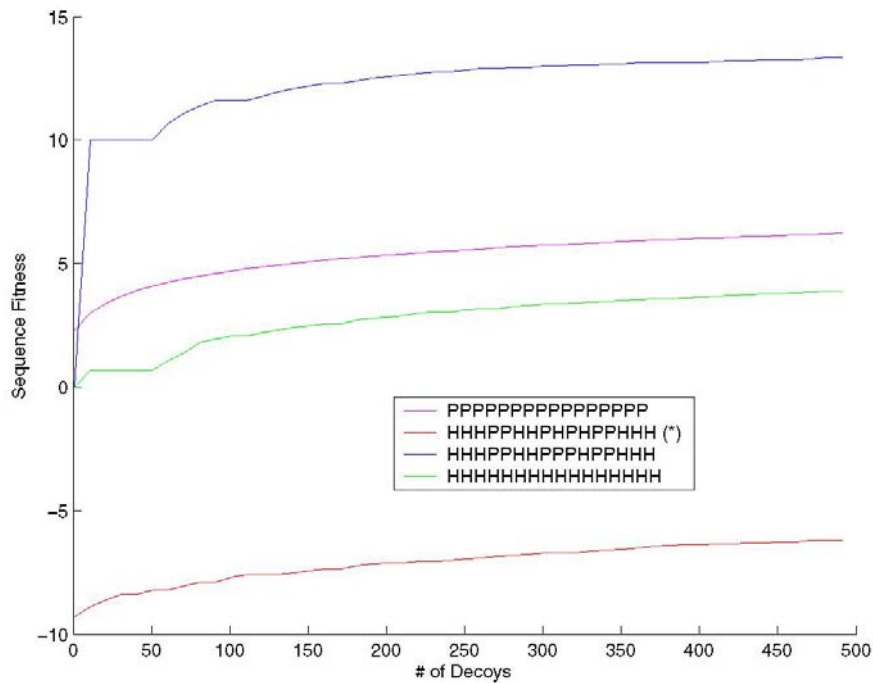


Figure 4.2.2: minimum free energy approximation vs. # of decoys (fitness is free energy)

As shown by the plot, when there are only a few decoys the free energy approximation of incorrect sequences is relatively low, and thus it is possible one may be falsely accepted as a solution. However, as the number of decoys increase, the approximation becomes evidently more accurate and the incorrect solutions are quickly assigned large free energy values.

4.3 Searching Conformation Space

In step 3(b), the current algorithm determines all the lowest energy states of s (chosen as a possible solution sequence) using a Hamiltonian function, which assigns a score based on the number of H-H contacts. For short amino-acid sequences, this exhaustive

technique is satisfactory, but for realistic protein sequences with lengths in the hundreds and thousands, exhaustive enumeration is impractical.

Our proposed strategy replaces this bottleneck step with a Monte Carlo-style approach that performs stochastic sampling of the associated structure space.

The Interacting Growth Walk (IGW) algorithm of Narasimhan et al. described in [17] is a Monte Carlo approach to sample the space of compact self avoiding walks. Let the initial coordinate of the SAW $r_0 = (0,0)$ and r_1 be chosen at random from amongst the neighbours of $(0,0)$. Let the set $\{r_j^m \mid m = 0, \dots, z_j \leq 3\}$ be the possible choices for the j^{th} step (that is, the subset of the nearest neighbours of r_{j-1} which do not contain coordinates already occupied by the SAW). Let $n_{NN}^m(j)$ be the number of non-bonded nearest neighbours of r_j^m . Then the probability that r_j^m is chosen at the j^{th} step of the walk is given by

$$p_m(r_j) = \frac{\exp(-\beta n_{NN}^m(j) \varepsilon_0)}{\sum_{m=1}^{z_j} \exp(-\beta n_{NN}^m(j) \varepsilon_0)}$$

where $\beta = 1/k_B T$ and ε_0 is defined so that if n_{NN} are the number of non-bonded nearest neighbour contacts in a self avoiding walk, then the energy for that SAW is $E = n_{NN} \varepsilon_0$ (for the 2D HP case we set it to 1 since the energy function for a 2D HP SAW is the number of HH contacts – this will slightly overestimate the actual energy according to the model).

```
function nativeStates = foldSequence(SEQ,eTarget,perConf)
// SEQ: potential solution sequence obtained in step 3(a)
// eTarget: energy of SEQ on TARGET conformation
nativeStates = {}
numSamples = {perConf fraction of upperbound on size of conformation space}
for i = 1 to numSamples
    betaTemp = {calculate beta temperature parameter}
    SAW = makeSAW_IGW(n,betaTemp)
    eNew = energy(SEQ,SAW)
    if eNew <= eTarget AND notMember(SAW,nativeStates)
        nativeStates = nativeStates U SAW
return nativeStates
```

Listing 4.3.1: Pseudo-code for foldSequence algorithm

Although the conformation space becomes exponentially large as N increases, the number of compact conformations remains relatively small in comparison. The Iterative Growth Walk algorithm generates nearly maximally compact self-avoiding walks. This allows our algorithm to both exploit the fact that most ground state conformations are compact, but also allows for the option of exploring conformation that are not maximally compact. The IGW algorithm takes a parameter called beta (temperature beta) that controls the level of compactness of the SAW's that it generates. A second parameter, eps, is also used for controlling compactness, however it remains constant for all 2D lattices. Figure 5.4.1 plots the average density of a large batch of SAW's produced with

IGW with varying beta and eps parameters. We define the density as the ratio of N to the perimeter of the bounding box of the SAW in the plane.

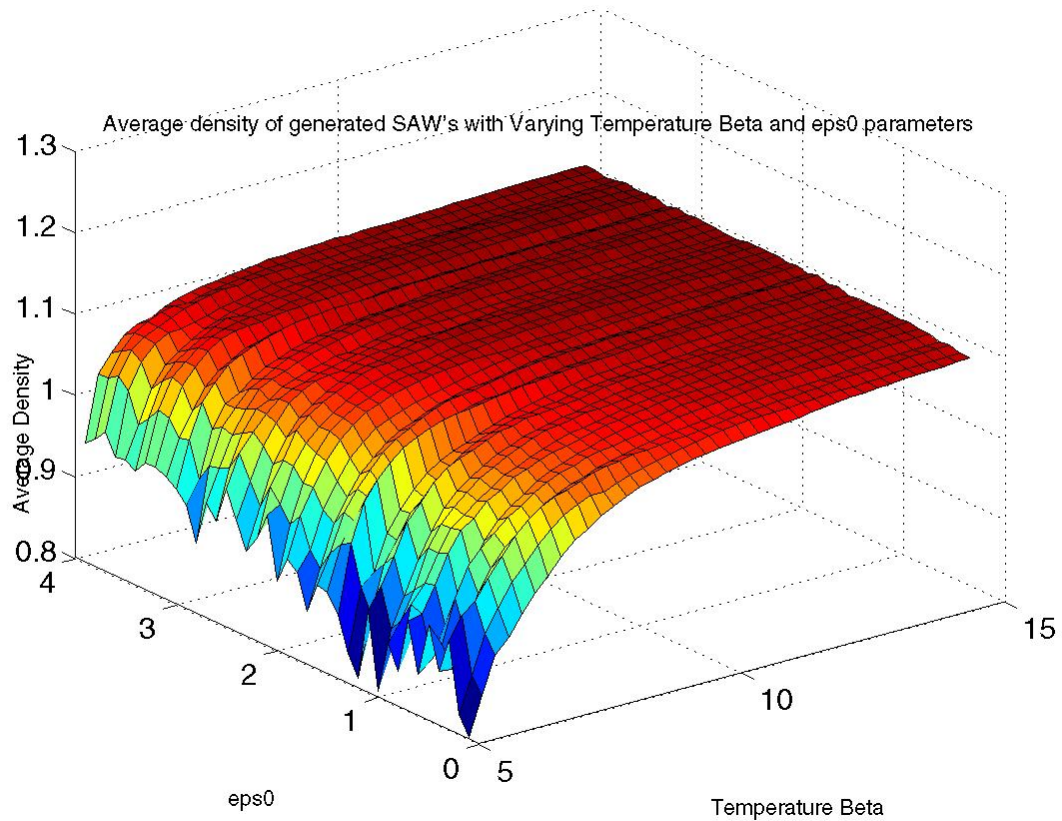


Fig. 4.3.1 – How average density behaves by changing parameters in IGW algorithm

In order to use IGW to sample both compact and less compact SAW's, we dynamically update beta during the sampling process. Figure 5.4.2 shows the calculation of beta as a monotonically decreasing function, providing a majority of the samples with a large beta (maximally compact) and a small portion of the samples with a lower beta (less compact). The function used resembles a cooling schedule of the form:

$$T(i) = (0.5 * (T_o - T_f) * (1 - \tanh(10 * i / T_c - 5)) + T_f)$$

where T_o is the initial temperature, T_f is the final temperature, T_c is the current temperature.

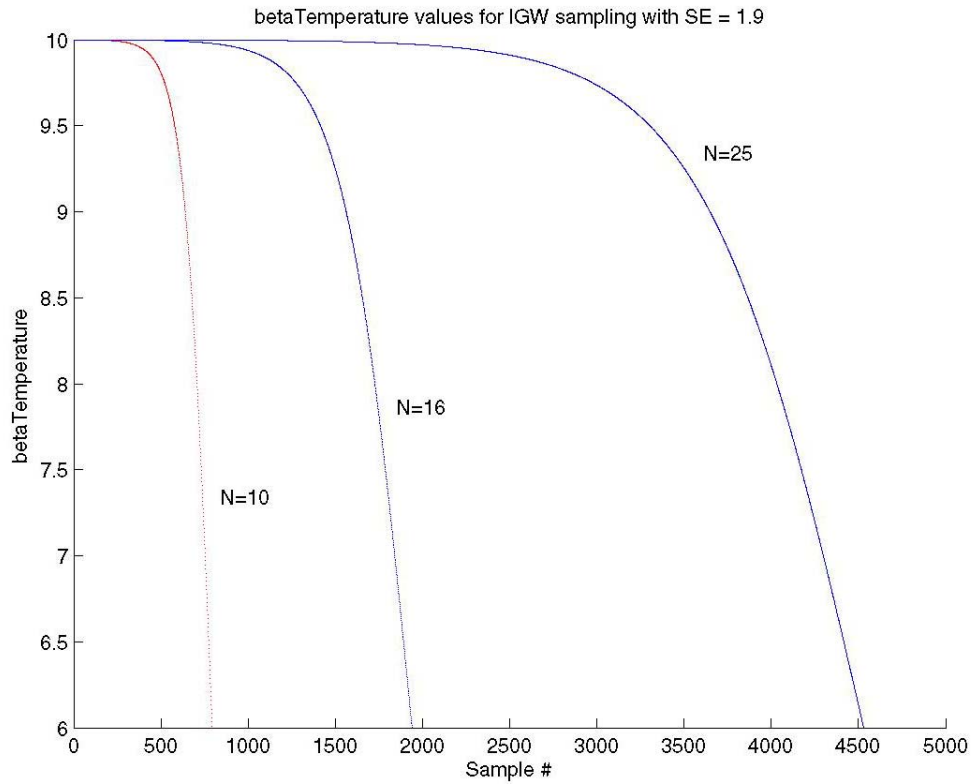


Fig. 4.3.2 – Decrease in temperature as samples increase

Fig. 4.3.3 shows typical examples of self-avoiding walks of length $N=26$ generated by the IGW algorithm for beta values 1, 5 and 10 respectively.

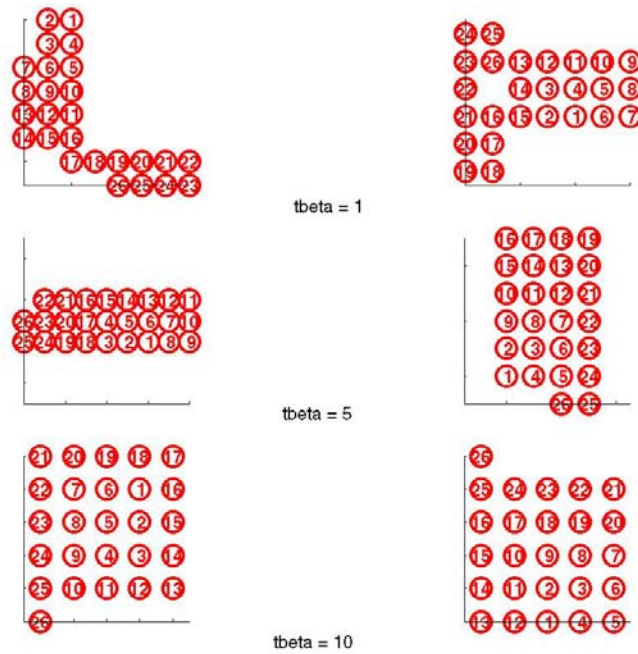


Fig. 4.3.3 – Various SAWs of length 26 at different temperatures as produced by IGW

4.4 Solution Refinement Step

At the beginning of the algorithm, when there are a minimal number of decoys, the optimization procedure of searching for putative solutions has a higher chance of finding sub-optimal solutions. If the sampling rate is insufficient to capture low-energy states of the sub-optimal solution, then it can be falsely accepted as a true solution. However, as the algorithm proceeds, more and more decoys are added to the decoy set. Due to the fact that all decoys are unique, each new decoy added has the potential to be a low energy state for a sub-optimal solution. We therefore perform occasional refinements by checking if any of the sequences in the solution set have low-energy states amongst the decoys – and thus signaling a degenerate sequence that can be removed from the solution set. Fig. 4.3.1 shows how the solution count of a typical run can fluctuate during runtime due to the refinement process.

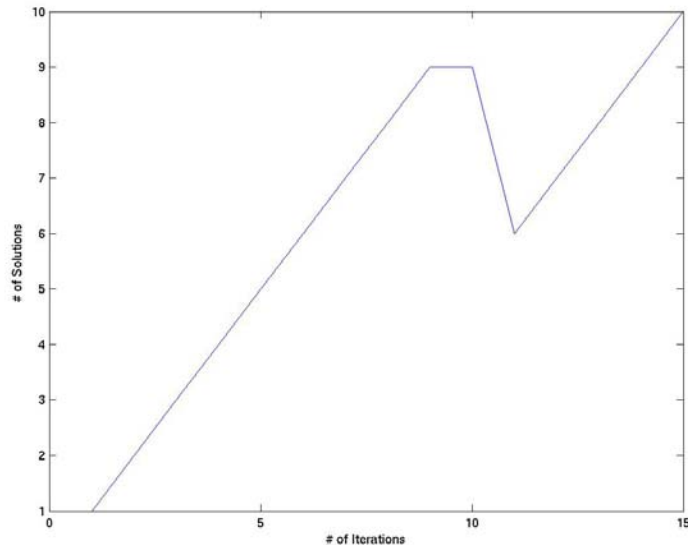


Figure 4.4.1: A fluctuating solution count for N=10

Through all our testing there were only a few instances when solutions were removed from the solution set due to the refinement process. However, we believe that this is partly related to the small amount of sampling as well as the small number of solutions available for the sequence sizes were tested. Moreover, the computational overhead is negligible for performing the refinement step, and considering it has the potential to weed out spurious solutions, we consider the refinement process a worthy addition to our algorithm.

4.5 Implementation Details

In this section we provide additional details concerning the implementation of our algorithm. In particular we discuss our approach to reducing the initial sequence space, the simulated annealing approach to searching sequence space, and details on how we generate self-avoiding walks in order to sample conformation space. We do not discuss any additional details concerning hardware or software resources, but we will mention that our algorithm was written in MATLAB 6.5 and executed on a shared Sun Fire V880 with 8x1.2 GHz UltraSPARC III processors, 32GB RAM, running SunOS 5.9. Since computations on lattices and self-avoiding walks are at the core of the algorithms involved in the approach described, MATLAB was chosen as the environment in which to implement our routines. Its fast matrix handling abilities, high-level interface, data-visualization capabilities and portability make it a convenient choice.

4.5.1 Reducing Initial Sequence Space

Conveniently, HP sequences can be encoded as binary numbers. In our project we use the notation H=1 and P=0. For this reason, storing, saving and accessing the actual sequences is of little computational effort. For an HP sequence of length N, there are 2^N possible sequences. Fortunately, due to symmetries in the sequences, it is not necessary to include all these combinations. For example, consider the sequence HHPH and HPHH. Clearly, these two sequences are identical with respect to the HP model, and so we only need to keep one of them. In [9], an efficient method for enumerating this subset of sequences was proposed, which takes binary numbers (HP sequences) and splits them in the middle in order to create a doublebit number. Considering an even length N sequence, starting from the bits on each side of the split, bits are paired with each other, called “doublebits”, and each of these doublebits is assigned a value. The values used for each HP combination are HH=3, HP=2, PH=1, PP=0. To enumerate all sequences then, we would simply generate a set of quaternary numbers, skipping doublebits of HP=2 unless there exists a doublebit of higher significance (greater q) that is equal to PH=1 (symmetry).

Next we provide a brief example of the conversion to and from a doublebit number:

Converting from binary to doublebit		Converting from a doublebit to binary
HP Sequence	P P H P H P	012
Binary Sequence	0 0 1 0 1 0	012XXX (the sequence is double the length)
Doublebit index	2 1 0 0 1 2	P12XXX (since PP=0)
Doublebit Sequence	0 1 2	PP2XHP (since PH=1)
		PPHPHP (since HP = 2)

As shown, the doublebit number corresponding to the above sequence is 012. This means doublebits allow us to effectively store only half the binary sequences, taking symmetry into account. To extend this method to odd-length sequences, we realize they are simply even length sequences with either an H or a P in the middle position of the sequence.

4.5.2 Simulated Annealing

Simulated annealing was used to optimize searching through sequence space. The standard algorithm was applied using a proportional cooling schedule. Candidate sequences are generated by performing one of two different local moves: randomly switching up to N randomly chosen sequence positions, or replacing an entire segment from another randomly chosen sequence. A Metropolis acceptance criterion is used to decide if the new sequence should be accepted. Numerous tests were performed to find optimal settings for the algorithm's parameters such as number of iterations and number of alterations. To calculate the initial and final temperatures, as well as the cooling rate (i.e. the cooling schedule), we use the following formulas obtained from [19]:

$$kT_i = -dE_i / \ln(P_i)$$

kT_i is the initial temperature, $-dE_i$ is the change in the cost function that is expected at the start, and P_i is the probability we accept that change. The energies we are calculating are between 0 and $\log(2) = 0.6391$. At the start we are willing to accept an increase in energy of 0.32 (approximately 50% of the expected energy range) with probably 0.25. The initial temperature is then:

$$kT_i = -dE_i / \ln(P_i) = kT_i = -0.32 / \ln(0.25) = 0.23$$

We calculate the final temperature similarly, choosing to allow an increase of 0.07 (about 10% of energy range) with a probability of 0.05, resulting in the final temperature of:

$$kT_f = -dE_f / \ln(P_f) = kT_f = -0.07 / \ln(0.05) = 0.0234$$

Finally, the proportional cooling factor C_{prop} is calculated as follows:

$$C_{prop} = \exp(\ln[(kT_f) / (kT_i)] / N)$$

where N_{kT} is the # of iterations. Using our initial and final temperature, and setting $N_{kT}=50$, we obtain:

$$C_{prop} = \exp(\ln[0.0234/0.23]/50) = 0.9553$$

This means after each iteration, we decrease the temperature by approximately 5%, resulting in a slow, stable cooling process.

5 Experiments & Analysis

We tested our algorithm with three different structures with 8, 10 and 12 residues. From the analysis in Section 4.2, we concluded that having a small initial decoy set of 5 is sufficient to help the free-energy approximation overcome the initial underestimates. The parameters for the Simulated Annealing remained constant (as specified in Section 4) throughout these experiments. For each of the three test sequences, we use small and

large number of samples in order to determine how the quality of the solution is affected. Sample sizes were calculated using $\text{numSamples} = N^X * C$ where $1.5 \leq X \leq 2.5$ and C is some scaling constant. In our tests, small samples sizes use $X=1.5$ and large sample sizes use $X=2.5$.

Because conformation space grows exponentially with N , the number of samples we chose did not grow exponentially with N , but were instead chosen heuristically to be as large as possible while remaining computationally feasible.

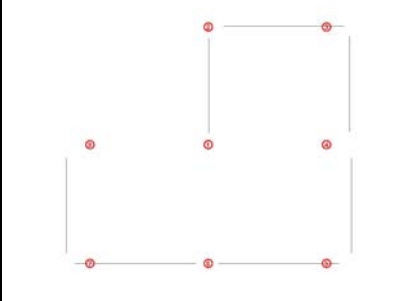
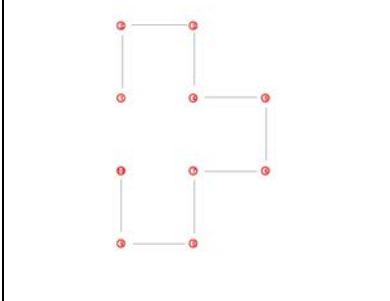
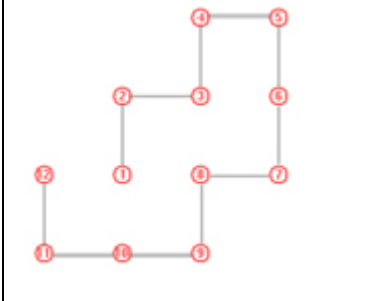
Test Case #1:	Test Case #2:	Test Case #3:
N = 8 URDDLLU 2 solutions	N = 10 URDRDLDLU 3 solutions	N = 12 URURDDLDDLU 4 solutions
		
Solutions/energy: HPPHPHPH 3 HHPHPHPH 3	Solutions/energy: HPPHPHPHPH 4 HHPHPHPHPH 4 HPPHPHPHPH 4	Solutions/energy: HPHPHPHPHPHPH 5 HHHPHPHPHPHPH 5 HPHPHPHPHPHPH 5 HHHPHPHPHPHPH 5

Table 5.1 – The main test cases

Tables 5.2, 5.3 and 5.4 display results for multiple experiments using the same sequence length of 8, 10 and 12 respectively.

The column abbreviations are as follows.

sampling exponent (SE)	The exponent value used to calculate how much we sample
# of Samples taken	The actual number of values taken, calculated using the following formula: $\text{numSamples} = N^{\text{SE}} * 10$ where N = sequence length
# iterations to finish	How many iterations before reaching one of the stopping criteria
max iterations	Simulated Annealing parameter: maximum iterations for each run
max alterations	Simulated Annealing parameter: Maximum alterations of a candidate sequence at a given temperature
actual # of solutions	The true number of solutions (obtained from benchmark results [16])
# solutions returned	Total # of solutions returned (correct and incorrect)
# of correct solutions	# of correct solutions

N	sampling exponent	# of samples taken	# iterations to finish	max iterations	max alterations	Actual # of solutions	# solutions returned	# of correct solutions
---	-------------------	--------------------	------------------------	----------------	-----------------	-----------------------	----------------------	------------------------

8	1.7	342	6	20	100	2	5	2
8	1.7	342	7	50	30	2	4	2
8	2.3	1194	5	20	100	2	3	2
8	2.3	1194	7	50	30	2	5	2

Table 5.2: Results for sequence length 8

Analysis: Both correct solutions were found each time, along with a few other incorrect solutions. As a rule of thumb, we always requested the algorithm to find N solutions. In general, when we requested less solutions (i.e. the exact number of solutions), the results were less accurate. This is because of the tradeoff between accuracy and performance; that is, the cost versus benefit of stochastically searching a larger or smaller subset of conformation space.

N	Sampling Exponent	# of samples taken	# iterations to finish	MI	MA	Actual # of solutions	# solutions returned	# of correct solutions
10	1.7	501	11	20	100	3	10	2
10	1.7	501	11	50	30	3	10	3
10	2.3	1995	13	20	100	3	10	3
10	2.3	1995	12	50	30	3	10	3

Table 5.3: Results for sequence length 10

Analysis: All three solutions were found each time. The algorithm was requested to find ten solutions, so it returned the three correct solutions and another 7.

N	Sampling Exponent	# of samples taken	# iterations to finish	Max Iterations	Max Alterations	Actual # of solutions	# solutions returned	# of correct solutions
12	1.7	683	14	20	100	4	12	4
12	1.7	683	14	50	30	4	12	4
12	2.3	3034	15	20	100	4	12	4
12	2.3	3034	14	50	30	4	12	4

Table 5.4: Results for sequence length 12

Analysis: Again, all the correct solutions were found but mixed incorrect solutions. The algorithm was asked to return 12 solutions, so it continues finding solutions even after it has obtained the correct ones.

Additional tests with longer sequence lengths:

Test Case #4:	Test Case #5:
N = 12 URULULDLDRD 12 solutions	N = 16 URULURRRDLDRDLL 2 solutions

Solutions/energy: PHPHPHPHPHP 4 HHPPHPHPHPHP 4 PHHHPHPHPHP 4 HHHHPHPHPHP 4 PHPPHHHPHPHP 4 PHPPHPHHPPHP 4 PHHHPHHPHPHP 4 PHPPHPHPHHP 4 PHPPHHHPHHP 4 PHPPHPHPPHH 4 HHPPHPHPPHH 5 PHPPHPHPHHH 4	Solutions/energy: HHHHPHPHPHHHPHH 9 HHHHHHPHPHHHPHH 9

Table 5.5 – Challenging test cases

N	Sampling Exponent	# of samples taken	# iterations to finish	Max Iterations	Max Alterations	Actual # of solutions	# solutions returned	# of correct solutions
12	1.7	683	14	50	30	12	12	1
12	2.3	3034	16	50	30	12	12	2
16	1.9	1114	12	20	100	2	10	1
16	1.9	11143	13	50	30	2	10	1

Table 5.6: Results for challenging test cases, with sequence lengths 12 and 16

Analysis: Here we present two dramatically different results. For N=16, the algorithm performs well and returned one of the two true solution sequences. Considering the sampling exponent of 1.7, which really only samples a very small portion of the actual conformation space, this shows that the algorithm is able to take advantage of the decoys and the free energy approximation, as well as the compact conformation sampling, and find one of the two solutions amongst a large solution space. The test for N=12, is a different story. Unfortunately, it was only able to find 1 or 2 solutions (this experiment was repeated several times with the same result). However, we believe this is due to the characteristic of the solution sequences. As shown above, the set of solution sequences contains one sequence with energy -5, and the rest all have energy -4. Interestingly, the algorithm returned all solution sequences with energy -5. These sequences are incorrect, and should have been weeded out during the sampling phase by finding the low energy states, but with such a large conformation space and so many putative solution sequences with energy -5, our algorithm continually converged to this same result. Moreover, these sequences should have been weeded out in the simulated annealing step by the free energy approximation constraint $\beta[E(\Gamma_0, \sigma) - F(\sigma)] < \ln(2)$, but we can now see that the decoy set was insufficiently large to produce a reasonable approximation. Therefore we

can conclude that this problem can most likely be remedied by performing a more thorough sampling of conformation space, which should result in more putative solutions being labeled as degenerate, as well as a larger set of decoy conformations, which in turn will help to improve the free energy sum approximation.

6 Conclusions and Future Work

In this work we present a novel approach to the Protein Design problem, our principal contribution being the ability to specify the degree of accuracy desired in the solution set. We achieve this by replacing the exhaustive search steps of the original Iterative Design [6] algorithm with Monte Carlo sampling of conformation space and a Simulated Annealing search of sequence space. There is an obvious theme that can be interpreted from this work: designing long protein sequences will inevitably require non-exhaustive search techniques in order to solve the problem in a realistic amount of time. We believe our approach is a step in this direction, which will ultimately allow for designing relevant and realistic proteins. Our modifications to the original Iterative Design algorithm provide glimpses of a non-exhaustive protein sequence design algorithm utilizing Monte Carlo sampling and simulated annealing optimization, as well as incorporating several other concepts that result in additional integrity constraints and improved approximations to the free energy sum calculation. While doing so, we also provide a flexible framework for manipulating the relative degree of accuracy in tradeoff for runtime efficiency. This level of control allows the algorithm to explore various levels of solution quality, and thus adjust the settings to suit the task at hand. For example, if only the exact solution sequences are desired, we can increase the sampling rate and sequence search parameters to move closer to the global optimum. Now consider the case where a researcher may only need a set of designed sequences which may only contain a few optimal solutions, but requires them in a very short amount of time. In this case the sampling rate and sequence search parameters can be adjusted to return a best approximation within a reasonable amount of time.

While our algorithm shows some promise, there remain many avenues of exploration for future work. First of all, an obvious step to demonstrating the power of our algorithm would be in tackling larger sequence sizes. The main reason for our limited testing sizes was due to insufficient resources. Theoretically, because our algorithm used relative sampling ratios, it has the ability to scale to large sequence sizes, although we acknowledge that as the sequence sizes increase, so does the margin for error. To solve this, rigorous research into the fine-detailed characteristics of the problem space may result in finding ways to optimize the sampling parameters such that the sampling rate is sufficiently likely to find the low energy states in a large conformation space. We believe that our algorithm will be well-suited to be ported into the 3-dimensional domain, where the search spaces become even larger. There is also ample room for research into how our approximation method handles proteins with more than two amino-acid classes or alternative interaction matrices. Lastly, we admit that Simulated Annealing might not be the best strategy for searching sequence space, and perhaps there exists another optimization procedure more suited for the task. One particular method we have already

considered is Particle Swarm Optimization, a relatively new stochastic optimization technique, which uses the power of collective intelligence within a population to efficiently search a rugged landscape[18]. PSO shares many similarities with other evolutionary computation models, such as Genetic Algorithms (GA). The main difference is that PSO represents solutions as particles in an N-dimensional search, and these particles “fly” through the search space looking for the optimal solution. A variation of the PSO is the Binary PSO (BPSO), where each bit is a dimension of the solution vector. For example, if we have a sequence PPHHPP we can represent it as 110011 and perform a BPSO by defining an appropriate fitness function. We therefore believe that binary PSO is a perfect candidate for searching the sequence space for the lowest free energy sequence given the target structure and some decoy structures.

Searching conformation space can be improved as well. Since many self-avoiding walks correspond to one contact-map, Monte Carlo algorithms to sample contact-map space could provide a significant speedup in the sampling phase of the algorithm. Though the Interacting Growth Walk produces compact walks, it does not produce a set of them. Batch methods analogous to the pivot [13] and perm [14] algorithms could be employed in such a way that compact conformation space is searched instead. In addition, more experiments to show the relationship between sampling and sequence size should be shown to provide a conceptual framework for how to change sampling based on a given conformation length.

Acknowledgements: We would like to thank Cristian Micheletti for fruitful email correspondence.

References

- [1] J. Deutsch, and T. Kurosky: New Algorithm for Protein Design. *Physical Review Letters*, Vol 76, Num 2, pp. 323-326, 1996.
- [2] A. Irback, C. Peterson, F. Pottharst, and E. Sandelin: Monte Carlo procedure for protein design. *Physical Review E*, Vol 58. Num 5, pp. 5249-5252, 1998.
- [3] A. Irback, C. Peterson, F. Pottharst, and E. Sandelin: Design of Sequences with Good Folding Properties in Coarse-grained Protein Models. *Structure with Folding & Design*, Vol 7, 347, 1999.
- [4] K. F. Lau and K. A. Dill., A lattice statistical mechanics model of the conformation and sequence spaces of proteins. *Macromolecules*, 22:3986-3997, 1989.
- [5] C. Micheletti, A. Maritan, J. Banavar: A comparative study of existing and new design techniques for protein models. *J. of Chem. Physics*, Vol 110, Num 19, pp. 9730-9738, 1999.
- [6] A. Rossi, A. Mariton, C. Micheletti, A novel iterative strategy for protein design. *J. of Chem. Physics*, Vol 112, Num 4, pp. 2050-2055, 1999.
- [7] E. Shakhnovich: Protein design: a perspective from simple tractable models. *Folding & Design*, 3:R45-R58, June 1, 1998.
- [8] Betancourt, M. R. and D. Thirumalai, "Protein sequence design by energy landscaping." *Journal of Physical Chemistry* 106: 599-609, 2002.
- [9] Reinhard Schiemann: Exact Enumeration of 3D Lattice Proteins. Diploma thesis, Institute for Theoretical Physics, University of Leipzig, Sept. 2003.
- [10] Alan D. Sokal: Monte Carlo Methods for the Self Avoiding Walk. *Monte Carlo and Molecular Dynamics Simulations in Polymer Science*, edited by Kurt Binder, Oxford University Press, 1995. (hep-lat/9405016).
- [11] Balanced Ternary Webpage. <http://perun.hscs.wmin.ac.uk/~jra/ternary/>, Accessed: Oct. 2004.
- [12] Brian Hayes: How To Avoid Yourself. *American Scientist*, Volume 86, Number 4, 1998.
- [13] Tom Kennedy: A Faster Implementation of the Pivot Algorithm for Self-Avoiding Walks. *J. Stat. Phys.* 106, 407-429, 2002.

- [14] Peter Grassberger: Pruned-Enriched Rosenbluth Method: Simulations of Theta Polymers of Chain Length up to 1 000 000. *Phys. Rev. E*, volume 56, number 3, 3682, 1997.
- [15] Peter Grassberger: Sequential Monte Carlo Methods for Protein Folding. NIC Symposium 2004, Proceedings, Jon von Neumann Institute for Computing, NIC Series, Vol. 20, pp.1-10, 2003.
- [16] Anders Irback and Carl Troein: Enumerating Designing Sequences in the HP Model. *Journal of Biological Physics*, 28:1-15, 2002.
- [17] S.L. Narasimhan, P.S.R. Krishna, K.P.N.Murthy and M. Ramanadham: A New Monte Carlo Algorithm for Growing Compact Self Avoiding Walks, (cond-mat/0108097 v4 27 Aug. 2001).
- [18] J. Kennedy, R. Eberhart, and Y. Shi. *Swarm Intelligence*. Morgan Kaufmann Academic Press, 2001.
- [19] B.T. Luke: Simulated Annealing. Learning from the Web, <http://members.aol.com/btluke/featur05.htm/>. Accessed: November 2004.