A Hybrid Local Search for Simplified Protein Structure Prediction

Keywords: Protein Structure Prediction, Discrete Lattices, Local Search, HP Energy Model, FCC Lattice, Segment Search, Heuristics.

Abstract: Protein Structure Prediction based on Hydrophobic-Polar energy model essentially becomes searching for a conformation having a compact hydrophobic core at the center. The hydrophobic core minimizes the interaction energy between the amino acids of the given protein. Local search algorithms can quickly find very good conformations just by repeatedly moving from the current one to one of its "best" neighbor. However, once such a compact hydrophobic core is found, the search stagnates and spends enormous effort in quest of an alternative core. In this paper, we attempt to restructure segments of a conformation that has a very compact core. The segments are conformation segments where an exhaustive search is locally applied. We also apply a mix of heuristics so that one of them could help while another takes into stagnation. We evaluate our algorithm by using Face Centered Cubic (FCC) Lattice on a set of standard benchmark proteins and obtain significantly better results than that of the state-of-the-art methods.

1 INTRODUCTION

Proteins are the most important organisms in a living cell. The function of a protein depends on the three dimensional native structure that it folds into in a particular environment (e.g., solvent and temperature). Protein structure prediction (PSP) can have an enormous impact in drug discovery. Computational methods for PSP are of great interest since the in vitro laboratory methods are very slow, expensive, and error-prone. Ab initio protein structure prediction is a computational method based on the famous Anfinsen’s dogma. The dogma states that the native structure of a globular protein is determined only by its primary amino acid sequence (Anfinsen, 1973).

Ab initio methods perform a search on the conformational space. They do not depend on any matching templates or the knowledge of the protein’s fold while homology and threading approaches do. Ab initio methods encounter a great deal of difficulties with high resolution protein structures and real energy models: protein models with atomic level details require huge amount of computational effort; the contributing factors of different forces to the energy functions are unknown; and the space of possible conformations is very large and complex. As a result there has been a growing interest in sampling a large number of decoys or candidate structures with the help of a guided scoring function or discrimination criteria. The candidate structures could be refined latter to achieve real structures. For such approaches, the simplified models, though lack many details, provide a realistic backbone for the proteins.

Nevertheless, the conformational search space, even with the simplified lattice-based models, contains an astronomically huge number of conformations. This makes complete search methods inapplicable. Local search methods are incomplete, but can produce good quality conformations very quickly. Local search methods repeatedly move from the current conformation to its “best” neighbor. In doing so, they suffer from re-visitisation and stagnation. The nature of the stagnation also depends on the energy function used. In Hydrophobic-Polar (HP) energy model, PSP essentially becomes searching for a conformation having a compact hydrophobic core at the center. The hydrophobic core minimizes the interaction energy between the amino acids of the given protein. Local search algorithms can quickly find such a hydrophobic core. However, once such a core is found, the search stagnates and spends enormous effort in quest of an alternative core.

In this paper, we attempt to restructure segments of a conformation that has a very compact core. The segments are conformation segments where an exhaustive search is locally applied. We select one large segment or a number of small segments. The total number of amino-acid positions affected by the segments selected in an iteration is dynamically adjusted with the stagnation period. We also use a tabu heuristic that prevents recently changed amino-acid positions from being modified again. Moreover, we apply a mix of heuristics so that one of them could help while another takes into stagnation. The heuristics are inspired by domain specific knowledge. We implemented our techniques on top of a local search algo-
rithm for Face Centered Cubic (FCC) lattice. Experimental results show that our approach significantly outperforms the state-of-the-art methods on a set of standard benchmark proteins.

The rest of the paper is organized as follows: Section 2 describes the protein structure prediction problem; Section 3 reviews the related work in the literature; Section 4 presents the detailed description of our approach; Section 5 discusses the experimental results and analyses; and finally, Section 6 presents conclusion and outlines our future work.

2 PROBLEM DEFINITION

A protein is a polymer of amino-acids, which are also called monomers. There are only 20 different amino acids. In the simplified model, each amino acid is represented by the position of its $\alpha$-C atom, a single point in the three dimensional lattice. A simplified function is used in calculating the energy of a conformation. The given amino acid sequence fits into the lattice, where every two consecutive monomers in the sequence are in contact or neighbors on the lattice (called the chain constraint) and two monomers can not occupy the same lattice point (called the self avoiding constraint).

FCC Lattice. Face Centered Cubic (FCC) lattice is preferred over other lattices (see Figure 1). It has the highest packing density for lattices with equal size of a sphere (Cipra, 1998). It has no parity problem (Decatur, 1996). Also, it provides the highest degree of freedom to place an amino acid. Thus, FCC lattice provides a realistic discrete mapping for proteins. An FCC lattice has 12 basis vectors: $\vec{v}_1 = (1,1,0)$, $\vec{v}_2 = (-1,-1,0)$, $\vec{v}_3 = (1,0,1)$, $\vec{v}_4 = (0,1,1)$, $\vec{v}_5 = (1,1,0)$, $\vec{v}_6 = (0,1,1)$, $\vec{v}_7 = (1,1,0)$, $\vec{v}_8 = (1,0,-1)$, $\vec{v}_9 = (0,-1,1)$, $\vec{v}_{10} = (-1,0,1)$, $\vec{v}_{11} = (0,-1,-1)$, $\vec{v}_{12} = (-1,0,-1)$. Two lattice points $p,q \in \mathbb{L}$ are said to be in contact or neighbors of each other, if $q = p + \vec{v}_i$ for some vector $\vec{v}_i$ in the basis of $\mathbb{L}$.

![Figure 1: (a) 3D cubic lattice and (b) 3D face centered cubic lattice](image)

HP Energy Model. The basic Hydrophobic-Polar (HP) model introduced in (Lau and Dill, 1989) divides the amino-acids into two groups: hydrophobic H ($\text{Gly, Ala, Pro, Val, Leu, He, Met, Phe, Tyr, Trp}$); and hydrophilic or polar P ($\text{Ser, Thr, Cys, Asn, Glu, Lys, His, Arg, Asp, Glu}$). The given amino acid sequence of a protein is represented as a string $s$ of the alphabet $\{H,P\}$. The free energy calculation for the HP model, shown in (1) counts only the energy interactions between two non-consecutive amino acid monomers.

$$E = \sum_{i,j \in s \land i < j} c_{ij} \cdot e_{ij} \quad (1)$$

Here, $c_{ij} = 1$ only if two monomers $i$ and $j$ are neighbors (or in contact) on the lattice and 0 otherwise. The other term, $e_{ij}$ is calculated depending on the type of amino acids: $e_{ij} = -1$ if $s_i = s_j = H$ and 0 otherwise. Minimizing the summation in (1) is equivalent to maximizing the number of non-consecutive H-H contacts. Several other variants of HP-model (Bornberg-Bauer, 1997) exist in the literature.

PSP with FCC and HP. Using the HP energy model together with the FCC lattice, the simplified PSP problem is defined as: given a sequence $s$ of length $n$, find a self avoiding walk $p_1 \cdots p_n$ on the lattice that minimizes the energy defined by (1).

3 RELATED WORKS

Protein Structure prediction problem is a hard combinatorial problem (Berger and Leighton, 1998). Due to the complexity, several techniques and their hybridizations have been applied in this problem. The similarity with the thermodynamic nature of the protein folding instigated the researchers to apply simulated annealing (Kawai et al., 1989; Albrecht et al., 2008). Genetic algorithms with Metropolis conditions as an acceptance criteria are found to be better than Monte Carlo simulation (Unger and Moult, 1993b; Unger and Moult, 1993a). Genetic algorithms are subsequently improved by applying various techniques such as tabu-list (Jiang et al., 2003) and twin removal (Hoque et al., 2007; Rashid et al., 2012). A novel set of lattice transformations called pullmoves has been introduced for square lattice along with a tabu meta-heuristics in (Lesh et al., 2003). These techniques are proved to be successful and are extendable to other lattices as well. Most of the benchmark proteins that these methods were evaluated on are small in size (length $\leq 48$).

The Constraint-based Hydrophobic Core construction (CHCC) algorithm (Yue and Dill, 1995) suc-
cessfully produced optimal structures for the famous Tortilla benchmarks (Yue et al., 1995) by using constraint programming techniques. However, it failed to handle degeneracy or non-unique conformations for several proteins. Constraint-Based Protein Structure Prediction (CPSP) tools were developed based on this CHCC algorithm (Mann et al., 2008). CPSP tools depend on pre-calculated hydrophobic cores and do not guarantee convergence for all proteins. Later on, another constraint solver, COLA (Alessandro et al., 2007) was developed using several biologically inspired heuristics and solved the problem with finite domains of the existing SICStus libraries. A two stage optimization method was proposed in (Ullah et al., 2009) by using CPSP tool to generate structures in the first stage, and simulated annealing in the second stage to improve the solutions. Elaborate MJ Matrix energy models were used in (Miyazawa and Jernigan, 1985). Large neighborhood search in (Ullah and Steinhöfel, 2010) when run for a long time is found to be producing better results than simulated annealing. In this work, constraint programming was used as neighborhood generator. This algorithm solved medium-sized benchmark proteins (length ≤ 80) from the Protein Data Bank (PDB).

Tabu search meta-heuristics were applied on the Tortilla benchmarks for the first time on FCC lattice (Cebrían et al., 2008). In their subsequent work, the same authors applied large neighborhood search with constraint programming (Dotu et al., 2008; Dotu et al., 2011) and improved the results achieved by the tabu search. Both of the large neighborhood search techniques reported in the literature used constraint programming as the neighborhood generator for local search. A memory based approach (Shatabda et al., 2012) on top of a local search framework (Dotu et al., 2011) improved the results on all the proteins from standard benchmark and the proteins taken from CASP competition. The memory-based approach produced the state-of-the-art results for the lowest energy level achieved for the benchmark proteins in the FCC lattice using HP energy model.

4 OUR APPROACH

Local search methods produce good results quickly. In HP energy model, they form a compact core of hydrophobic residues at the center of the conformation and search cannot progress unless the core is broken and an alternate core is formed (Shatabda et al., 2012). Even when guided by a good heuristic, the search oscillates within the same region of search space and fails to improve. This obvious nature of the local search algorithms results in stagnation. Large neighborhood techniques are adopted to handle this situation in protein structure prediction (Ullah and Steinhöfel, 2010; Dotu et al., 2011) and in other domains as well (Schrimpf et al., 2000; Bent and Henenryck, 2007). Most of these algorithms depend on constraint programming as neighborhood generation. In this paper, we propose a hybrid local search that can improve the solutions by restructuring a single or multiple segments of selected points, and thus breaking the compact core to create an alternative core. Our algorithm belongs to local search family and do not use constraint programming as neighborhood generator.

The pseudo-code of our method is given below:

```plaintext
Procedure LWS(Conf c)
1. initializeTabu()
2. it = 0
3. nonImp = 0
4. while ++it <= maxIt do
5. selectSegmentType()
6. selectVariables()
7. generateMoves()
8. selectHeuristics()
9. simulateMoves()
10. selectMove()
11. executeMove()
12. updateTabuList()
13. if new globalBest found
14. reset()
15. else
16. nonImp++
17. end if
18. if nonImp >= maxStable then /*stagnation*/
19. nonImp = 0
20. maxStable = maxStable * factor
21. segmentSize++
22. end if
23. end while
24. return globalBest
End Procedure.
```

At each iteration, our algorithm selects a number of variables depending on the segmentSize and segment type. Then, all feasible moves are generated (line 7) using the selected variables. These variables are necessarily Cartesian co-ordinates of amino-acid residues. Once the feasible neighborhood is generated, it then simulates the moves and calculates the changes in the fitness function and heuristics. The simulate function (line 9) updates the fitness functions and the heuristic functions in an incremental fashion. Once a move is selected, the conformation is updated by executing the move (line 11). The execution of the move updates the fitness functions and the heuristics as well. If there is a new globalBest found, all the parameters are reset to the initial condition (line 14). If there is a stagnation, the segment size is increased and the stagnation parameter, maxStable is multiplied.
by a factor. We also maintain a tabu list that prevents from selecting recent variables or amino-acid residue move from re-occurring.

4.1 Algorithm Details

In the rest of this section, we describe different parts of the algorithm in details.

Segment Types. We select one large segment or a number of small segments (see Figure 2). At each iteration a number of variables are selected to fill these segments, which are then used in generating moves. The purpose of the segment search is to locally re-optimize the structure within the segment using exhaustive search. The single large segment allows to re-structure a large subsequence of the protein whereas multiple small subsequences are re-optimized simultaneously using the multiple segment option. We select a segment type randomly at each iteration. The total number of amino-acid positions affected by the segments selected in an iteration is segmentSize.

![Figure 2: Two types of segments, for segmentSize = 6](image)

Variable Selection. We maintain a tabu list to prevent recent moves selected for each of the variables. The tabu tenure is selected randomly using a uniform distribution from the range $[4, sequenceLength/8]$. We select only those variables that are not in the tabu list. We do not use any aspiration criteria since we do not know the effect of selecting a particular move beforehand and the simulation for individual point is not done. In case of the single large segment, we select the variables from the range $[-segmentSize/2, +segmentSize/2]$ around a randomly selected point, if none of them are in the tabu list. Though the tabu record is kept for single points, this mechanism requires all the points in a segment to be out of tabu list and hence a different part of the structure is guaranteed to be selected for restructuring at each iteration. In the case of multiple segments, we select the variables randomly that are not in the tabu list. It creates multiple segments each containing points from different parts of the structure. This multiple segment mechanism helps break the compact core since within a given region of the compact core, not all the amino-acids form a sequential chain. Rather amino-acids from separate parts interacts each other to form a compact core.

Segment Search. Once the variables are selected, the algorithm then generates all the possible moves for those variables, keeping the rest of the chain un-effected. Pseudo-code for the procedure generateMoves() is given below:

```
Procedure generateMoves(Conf c, Segment w)
    1 generator = initialize()
    2 do
    3 for (all points p in w)
    4     newPoint = getPosition(generator)
    5     if(occupied(newPoint))
    6         skip(generator, p)
    7         exit for
    8     end if
    9     add newPoint to currentMove
    10    end for
    11    if(!skip)
    12    add currentMove to moveList
    13    end if
    14    while(next(generator))
End Procedure
```

The algorithm starts with an initial generator string that assigns the same direction vector to all the positions. Direction vector is one of the basis vector between two consecutive points. For each of the points in the segment, a new point is calculated using the generator string (line 4). If that position is already occupied then, the rest of the generator string is ignored by calling the method $skip(generator, p)$. If all the new points are valid and guarantees feasibility, the move is added to the move list. The whole process is enumerated until the $next(generator)$ function produces the last generator string. The procedure $skip(generator, p)$ allows necessary pruning in the segment search. Moreover, this simple iterative implementation has no memory overheads for recursive calls or for the book keeping operations that are needed for constraint programming.

For segment size = 1, the search generates moves that are identical to diagonal moves and for segment size = 2 and single segment type, the generated moves are identical to crankshaft moves. We have also found that generating moves for multiple segments is faster than single moves. Its due to the fact that single segments create contiguous empty space in the lattice and results in an increase in the number of feasible sub-conformations.
4.2 Heuristics

Local search algorithms guided by a single heuristic function often gets trapped in plateau region or local minima. One heuristic can possibly take the search out of the trap of local minima of another different heuristic. In stead of guiding the search by a single fitness function, we maintain three different heuristics and select one of them at each iteration. We explored a number of heuristics in our experiments. The best three are finally used:

1. **Maximize pairwise H-H contacts**: Select a move that minimizes the number of contacts between two non-consecutive amino-acids.

   \[ h_3 = \sum_{i,j, i+1 < j} c(i, j) \times (s_i = H, s_j = H) \]  

   Here, \( c_{ij} = 1 \) only if two monomers \( i \) and \( j \) are neighbors (or in contact) on the lattice and 0 otherwise. This heuristic corresponds to the HP energy function. This heuristic function is not informative when multiple cores exist in a conformation and hydrophobic amino acids from one core is to be taken away to merge with another core. Moreover, a large number of different conformations can have the same heuristic evaluation and thus search often get stuck in the plateaus.

2. **Minimize all pair H-H distance**: Select the move that minimizes the sum of the squared distances between all pairs of non-consecutive hydrophobic amino-acids. The heuristic is defined below:

   \[ h_2 = \sum_{i,j, i+1 < j} \text{dist} (i, j)^2 \times (s_i = H, s_j = H) \]  

   Here, \( \text{dist}(i, j) \) denotes the Euclidean distance of two points in the Cartesian space. This fitness function pulls all the hydrophobic residues towards each other and helps form a compact core quickly. Whenever there is a change in the position of a hydrophobic residue, the heuristic function changes. Thus it has less chance of getting stuck into the plateau region. However, positioning of polar amino-acid residues are not taken into effect and the nature of the search actually oscillates within a search region before it can really improve the structure (Shatahada et al., 2012).

3. **Minimize squared distance to hydrophobic centroid**: Select the move that minimizes the sum of distances of the H-amino acids to the hydrophobic centroid \( (H_c) \). We calculate the coordinates of the hydrophobic centroid from the average of Cartesian co-ordinates of the hydrophobic amino-acids.

   \[
   x_c = \frac{1}{n_H} \sum_{i_H=0}^{n_H} x_{i_H}, \quad y_c = \frac{1}{n_H} \sum_{i_H=0}^{n_H} y_{i_H}, \quad z_c = \frac{1}{n_H} \sum_{i_H=0}^{n_H} z_{i_H}
   \]

   Now the sum of the distances to this hydrophobic centroid \( (H_c) \) is defined below:

   \[
   h_1 = \sum_{i_H=0}^{n_H} (x_c - x_{i_H})^2 + (y_c - y_{i_H})^2 + (z_c - z_{i_H})^2
   \]

   Here, \( n_H \) is the number of hydrophobic amino-acids in the sequence and \( i_H \) is the index of each amino acid. This heuristic function has an interesting local property since, at each iteration it updates the hydrophobic centroid, and thus provides new directions for the residues to move to.

We also explored several other heuristics: minimizing distance from the centroid, where the centroid is defined as \((\frac{1}{4} \sum_{i=1}^{n} x_i, \frac{1}{4} \sum_{i=1}^{n} y_i, \frac{1}{4} \sum_{i=1}^{n} z_i)\); minimizing the distance from the origin defined as \(\sum_{i=1}^{n} x_i^2 + y_i^2 + z_i^2\); and maximizing the sum of neighboring contacts for hydrophobic residues, defined as \(\sum_{i_H} \text{neighbors}(i_H)\). However, these heuristics were not effective and not chosen for the algorithm.

4.3 Implementation

We have implemented our algorithm in C++. All the constraints are incorporated in the representation. The performance of the local segment search mainly depends on the move generation and heuristics calculation at each iteration. Move generation code is written as an iterative routine as described in the pseudo-code. The heuristics are maintained using the invariants provided by Kangaroo (Newton et al., 2011) which is a constraint based local search (CBLIS) system.

**Structure Representation.** We use Cartesian coordinates of the amino-acid residues to represent protein structures. Each amino-acid is a three dimensional point in the Euclidean space. The domain of the individual axes variables are in the range of \([-\text{length, +length}]\). The initial point is set to \((0,0,0)\).

**Self avoiding constraint** Self avoiding constraints are implemented by a hash-map indexed by points themselves. It enables us to check the co-ordinates of the amino-acid residues in \(O(1)\) time. The \texttt{occupied(Point p)} used in line 5 of the procedure \texttt{generateMoves()} uses this hash-map data structure.
Connectivity constraint. The conformations are always valid or feasible, and satisfy the connectivity constraint. Throughout the search, the moves are generated in such a way that only feasible solutions are selected and the connectivity constraint is always maintained automatically.

Invariants. We create invariants in the CBLS system provided by Kangaroo to maintain our heuristics incrementally. Invariants are special constructs that are defined by using simple mathematical operators over the variables. We create invariants for each of the heuristic functions. These invariants enable us to simulate the generated moves and get the difference in heuristic values for each of the moves. The CBLS system Kangaroo, propagates the calculations in a lazy way and maintains the invariants in an efficient way. Moreover, since we select only one heuristics at each iteration, the calculation to simulate moves is kept minimal.

simulateMoves(): For each of the moves from the move-list, we find the change in the invariant for the heuristic selected. Here, only the change in the invariant is calculated, invariant values are not updated and the moves are not performed.

selectMove(): From the simulated values of the heuristics, we select the best move for the heuristics at each iteration. Ties are broken by using a uniform random distribution.

executeMove(): In this procedure, the moves are actually performed. Invariants for all the heuristics are calculated and updated.

5 EXPERIMENTAL RESULTS

We ran experiments in a cluster machine. The cluster has a number of machines each equipped with two 6-core CPUs (AMD Opteron @ 2.8GHz, 3MB L2/6M L3 Cache) and 64GB Memory, running Rocks OS (a Linux variant for cluster). We compared the performance of our algorithm with the tabu search (Dotu et al., 2011) and the memory based search (Shatabda et al., 2012).1 Throughout this section, tabu search is denoted by LS-Tabu and the memory-based search is denoted by LS-Mem. For each of the protein sequences, we ran each of the algorithms for 50 times. Each algorithm was given 5 hours to finish. For our algorithm, we ran memory based algorithms for 20 minutes and the best solutions were used as input to our algorithm. Rest of the time we use our algorithm to improve the solution. In this section our algorithm is denoted by LWS.

Benchmark Set - I. The first benchmark set that we used is taken from Sebastian Will’s PhD thesis (Will, 2005) and publicly available in the website of Peter Clote Laboratory2. These are the R sequences and f180 sequences of length 200 and 180 respectively. The best and average energy levels achieved are reported in the upper part of Table 1. Parameter settings for LS-Tabu and LS-Mem were set as suggested by the authors. We ran our algorithm with segmentSize and maxStable initially set to 1 and 1000 respectively, and multiplying factor was set to 1.2. We could not run large neighborhood search (Dotu et al., 2011) for these benchmarks since the COMET program exited with ‘too much memory needed’ in our system. We report the best energy levels reported in (Dotu et al., 2011) in the ‘LNS’ column. Optimal lower bounds for the minimum energy levels for the proteins are also reported under the column $E_f$ generated by the CPSP tools (Mann et al., 2008). Note that these values are obtained by using exhaustive search methods and are only used to see how far our results are from them. The missing values indicate where no such bound was found and the values marked with are the values for which the algorithm did not converge even after 24 hours of run. Length of the protein sequences and number of hydrophobic residues are also reported in the table. The lower values of average and best energy levels are shown in bold and italic font respectively.

Benchmark Set - II. We also use a second set of benchmarks originally used in (Shatabda et al., 2012) which are derived from the famous Critical Assessment of Techniques for Protein Structure Prediction (CASP) competition3. Six proteins were randomly chosen from the target list that are around the length of 230±50. These sequences were then converted to HP sequences depending on the nature of the sequences. Results for these six proteins are also reported in Table 1 (lower part). The PDB id for each of these proteins is also given and the parameter settings for these six proteins were kept the same. LNS column contains no data for these six proteins since they were not used in (Dotu et al., 2011).

Tortilla Benchmarks. Tortilla benchmarks or "Harvard" benchmarks introduced in (Yue et al.,

1We got the source codes directly from the authors of both of the articles

2http://bioinformatics.bc.edu/clotelab/FCCproteinStructure/

3http://predictioncenter.org/casp9/targetlist.cgi
Table 1: Best and average energy levels achieved for the R, f180 and instances taken from CASP

Table 2: Success rate for Torilla benchmarks of LWS compared to LNS
Table 3: Relative improvements of our approach over the average energy levels achieved by LS-Mem and LS-Tabu

<table>
<thead>
<tr>
<th>Protein</th>
<th>Benchmark Set - I</th>
<th></th>
<th></th>
<th>Benchmark Set - II</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Seq</td>
<td>$E_I$</td>
<td>LS-Mem</td>
<td>LS-Tabu</td>
<td>Seq</td>
<td>$E_I$</td>
</tr>
<tr>
<td>R1</td>
<td>-384</td>
<td>34.48%</td>
<td>42.42%</td>
<td>3no6</td>
<td>-455</td>
<td>27.5%</td>
</tr>
<tr>
<td>R2</td>
<td>-383</td>
<td>30.18%</td>
<td>37.28%</td>
<td>3mr7</td>
<td>-355</td>
<td>20.63%</td>
</tr>
<tr>
<td>R3</td>
<td>-385</td>
<td>34.54%</td>
<td>41.93%</td>
<td>3mse</td>
<td>-323</td>
<td>23.18%</td>
</tr>
<tr>
<td>f180_1</td>
<td>-378</td>
<td>27.27%</td>
<td>37.25%</td>
<td>3mqz</td>
<td>-474</td>
<td>25%</td>
</tr>
<tr>
<td>f180_2</td>
<td>-381</td>
<td>34.14%</td>
<td>42.55%</td>
<td>3on7</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>f180_3</td>
<td>-378</td>
<td>37.14%</td>
<td>43.58%</td>
<td>3no3</td>
<td>-494</td>
<td>24.81%</td>
</tr>
</tbody>
</table>

Search Progress. We show search progress of three algorithms for the protein sequence R1 in Figure 3. Average energy level by each of the algorithms for 50 runs are shown over the timespan of 300 minutes. The tabu search and the memory-based search achieve almost the same level of energy initially but as soon as the search makes progress, fail to overcome stagnation and does not improve after a certain level. But LWS starts from very low level of energy and keeps to improve the solutions, since as the stagnation increases, it adjusts the segmentSize to a higher value, which results in more perturbation and produces better results. It is clearly evident from the graph that our algorithm continues to improve the stagnant situations and thus produces better results.

Figure 3: Search progress for protein sequence R1 with time for three algorithms

Figure 4: Structures produced by (a) LS-Mem and (b) LWS

6 CONCLUSION

In this paper, we presented a hybrid local search that exhaustively explored segments of a conformation and is guided by a mix of heuristic functions. Our algorithm improved on all the benchmark proteins and significantly outperformed state-of-the-art algorithms. We applied single large segments and a number of small segments, and dynamically adjusted the segment size with stagnation period. Moreover, the variable selection was affected by a tabu heuristic. We used several heuristics so that one heuristic can help escape local minima of another. In future, we wish to apply these techniques in other domains such as, satisfiability and traveling salesman problem.

REFERENCES


Bent, R. and Hentenryck, P. V. (2007). Randomized adap-


