A Complete and Effective Move Set for Simplified Protein Folding

N. Lesh[1], M. Mitzenmacher[2], S. Whitesides[3]
[1] Mitsubishi Electric Research Laboratories, 201 Broadway, Cambridge, MA, 02139
lesh@merl.com
[2] Harvard University, Computer Science Department
michaelm@eecs.harvard.edu*
[3] McGill University, School of Computer Science
sue@cs.mcgill.ca†

Abstract

We present new lowest energy configurations for several large benchmark problems for the two-dimensional hydrophobic-hydrophilic model. We found these solutions with a generic implementation of tabu search using an apparently novel set of transformations that we call *pull moves*. Our experiments show that our algorithm can find these best solutions in 3 to 14 hours, on average. Pull moves appear quite effective and may also be useful for other local search algorithms for the problem. Additionally, we prove that pull moves are complete; that is, any pair of valid configurations are mutually reachable through a sequence of pull moves. Our implementation was developed with the Human-Guided Search (HuGS) middleware, which allows rapid development of interactive optimization systems.

1 Introduction

The two-dimensional hydrophobic-hydrophilic model for protein folding, or 2D HP model, was introduced by Dill [5, 15]. A problem consists of a sequence of amino acids, each labeled as either hydrophobic (H) or hydrophilic (P). The sequence must be placed on a two-dimensional grid without overlapping, so that adjacent amino acids in the sequence remain horizontally or vertically adjacent in the grid. The goal is to minimize the energy, which in the simplest variation corresponds to maximizing the number of adjacent hydrophobic pairs. Although the model is extremely simple, it captures the main features of the protein folding problem. This problem is NP-complete, and hence unlikely to be solvable in polynomial time [3, 4, 10, 22]. Recent theoretical work has focused on approximation algorithms [1, 9, 18], although these have not proven helpful for finding minimum energy configurations. Many heuristic algorithms for finding minimum energy configuration have been explored, such as genetic algorithms in the seminal work of Unger and Moult [21].

In this paper we demonstrate the effectiveness of *pull moves*, a new local move set, with a tabu search algorithm on the 2D HP problem. The pull moves we define recall the classic de Gennes

*Supported in part by NSF CAREER Grant CCR-9983832 and an Alfred P. Sloan Research Fellowship. This work was done while visiting Mitsubishi Electric Research Laboratories.

†Supported in part by NSERC and FCAR. This work was done in part while visiting Mitsubishi Electric Research Laboratories.

reptation model [6, 7] for polymer motion.[1] In our experiments, pull moves appear quite effective, they may be useful in conjunction with other local search techniques that have been applied to the problem. As a theoretical contribution, we prove that pull moves are complete, i.e., any valid configuration can be reached from any other valid configuration by a sequence of pull moves.

Tabu search is a heuristic approach for exploring a large solution space. Our tabu algorithm is implemented using middleware from the ongoing Human Guided Search (HuGS) project [11]. Our implementation uses the generic tabu search algorithm available in the HuGS middleware; the only specialization necessary was to define the local move set. The HuGS middleware offers other significant advantages, including frameworks for visualization and human interaction that we have shown can improve the performance of tabu search for other optimization problems [11]. In this work, however, we focus on results obtained by the automatic algorithm alone. The middleware code, including the 2D HP application, is freely available for research purposes; we hope that the code may be useful for continued research in the area. (Contact lesh@merl.com for the code.)

A highlight of our experimental results is that we have found new lowest energy configurations for the three longest benchmarks we found in the literature. For example, a sequence of length 85 used in [13, 16] was conjectured to have a ground state energy, or minimal energy, of -52; we have found a configuration with energy -53. We have similarly found two new best lowest energy configurations for two sequences of length 100 used in [2, 19]. All of our results can be found in with approximately 3 to 14 hours of CPU time, without fine-tuning the parameters of our algorithm. Besides showing these new lowest energy configurations, we provide an experimental analysis of pull moves.

The remainder of the paper proceeds as follows. In Section 2, we briefly review the 2D HP model. In Section 3, we formally describe pull moves and give a proof of completeness. Section 4 describes our implementation, focusing on relevant aspects of the HuGS system and tabu search. In our discussion of experiments in Section 5, we present our new best solutions and an experimental evaluation of both pull moves and our implementation.

2 The 2D HP model

In graph theoretic terms, a polymer in the 2D HP model is represented combinatorially as a chain of n vertices embedded as a simple (i.e., non-self-intersecting) path P in a unit grid. Vertices that are adjacent in the chain must be placed at adjacent points in the grid. Each vertex is assigned one of two colors (H or P) according to the physical properties of the corresponding amino acid in the polymer. A path is assigned a score that is computed by summing weighted numbers of various color adjacency occurrences. The score models the energy of a configuration of the molecule. Here we focus on the standard case where the only interactions counted are between pairs of vertices that are adjacent in the grid and both labeled H; each such pair decreases the score by 1. Pairs of vertices that are adjacent in the chain are not counted, since they must occur in all valid configurations. The algorithms we present could also be applied to other types of interactions.

More formally, we define the following terms. A *location* is a node of the grid, corresponding to an (x, y) coordinate pair. Locations are said to be *adjacent* if they are adjacent either horizontally or vertically. Similarly, two locations are *diagonally adjacent* if they lie one horizontal and one vertical step from each other. A *vertex* is a node of the chain (molecule), which has a *label*, either H or

[1]According to this model, which is widely used in polymer physics, the motion of a mobile polymer chain moving through a confining environment is governed by slack entering at the ends of the polymer and diffusing along its length.

P. Vertices are numbered consecutively from 1 to n along the chain. A *valid* configuration of the chain lies along a non-self-intersecting grid path P of locations such that adjacent vertices occupy adjacent locations. In describing our move set and the algorithm, we refer to the configuration at different time points: $P(t)$ defines the configuration at time t. We may drop the reference to t when the meaning is clear. For $j > i$, the subpath of $P(t)$ from vertex i to vertex j, inclusive, is denoted $P_{i,j}(t)$. The location occupied by vertex i at time t is denoted by $(x_i(t), y_i(t))$. A location is *free* at time t if there is no vertex there.

The *energy* of a configuration is obtained by counting the number of adjacent pairs of vertices labelled H that are not consecutively numbered, and multiplying by -1. Our goal is to find the configuration with the lowest energy.

3 Pull Moves

We are guided in our selection of possible moves by several desired properties. We begin with some definitions. A *move* is a function that takes as input a valid chain configuration $P(t)$ and produces a valid configuration $P(t+1)$. A set $\mathcal{M}$ of moves is *reversible* if, for any move in $\mathcal{M}$ applied to a configuration $P(t)$ to obtain a configuration $P(t+1)$, there is some move in $\mathcal{M}$ that can accept $P(t') = P(t+1)$ as input and produce $P(t'+1) = P(t)$. A set $\mathcal{M}$ of moves is *complete* if, given any configurations P and P', there is a sequence of moves in $\mathcal{M}$ that relocates P to a configuration that is congruent (after translation and rotation) to P'.

It is clearly beneficial that our move set be complete; otherwise, otherwise an algorithm using these moves will not be able to reach all configurations and thus might miss the optimal solution. Reversibility is useful in proving completeness, as we demonstrate below. Furthermore, it is also desirable from the standpoint of building effective local search algorithms (as well as for simplifying user interaction) if the moves generally avoid drastic changes that destroy the global structure of the current configuration. Hence *local* moves, that is, moves that displace as few vertices as possible and do not displace any vertex very far from its current location, are attractive.

We now describe *pull moves*, a set of moves that is complete, reversible, and local. Of course, any larger move set that contains pull moves will be complete, whether or not the additional moves are local or reversible. Although we describe pull moves on a 2D grid, it should be clear from the discussion below that pull moves and the completeness proof for pull moves can be extended to 3D and higher.

We describe pull moves in terms of how they might be implemented. Consider a vertex i at time t in location $(x_i(t), y_i(t))$. Suppose that a free location L is adjacent to $(x_{i+1}(t), y_{i+1}(t))$ and diagonally adjacent to $(x_i(t), y_i(t))$. Figure 1 shows examples that we use as a reference. The vertices $(x_i(t), y_i(t))$, $(x_{i+1}(t), y_{i+1}(t))$, and the free location L constitute three corners of a square; let the fourth corner be the location C. For a pull move to occur, the location C must either be free or must equal $(x_{i-1}(t), y_{i-1}(t))$.

When $C = (x_{i-1}(t), y_{i-1}(t))$, the entire local pull move consists of moving vertex i to location L. When C is free, first vertex i is moved to location L and vertex $i-1$ is moved to location C. Then, until a valid configuration is reached, the following action is performed: starting with vertex $j = i-2$ and down to vertex 1 set $(x_j(t+1), y_j(t+1)) = (x_{j+2}(t), y_{j+2}(t))$. That is, vertices are pulled two spaces up the chain until a valid configuration is reached. Notice that this ensures that a valid configuration is maintained; vertices i and $i-1$ have moved to free locations, and the lower indexed vertices are repeatedly pulled into vacated locations.

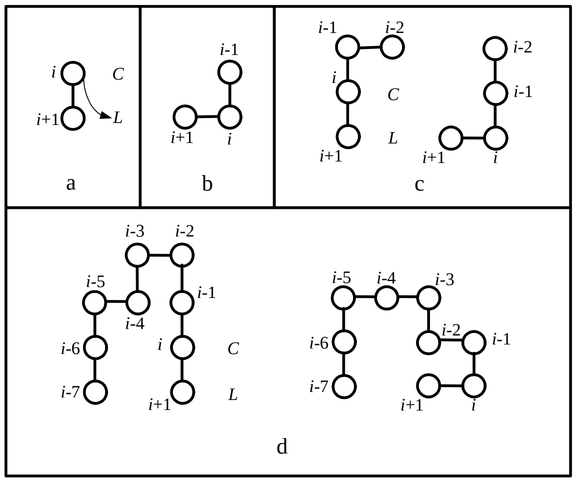

Figure 1: (a) To begin a move, vertex i moves into a free location L. (b) In the case where the fourth corner C holds vertex $i-1$, the move is complete. (c) Otherwise, vertex $i-1$ is moved to C. It is possible that this completes the move. (d) If this does not complete the move, vertex $i-2$ is moved to the position previously held by vertex i, vertex $i-3$ is moved to the position previously held by vertex $i-1$, and so on until a valid configuration is reached. Here vertex $i-3$ is moved, but then the move is complete.

If the pull goes down to vertex 1, a valid configuration is reached. If, however, we can stop the move before then in a valid configuration, then we do so. This improves the *locality* of the move, in that fewer vertices change position. As we observe in our experimental section, in practice most moves displace few vertices. Note that the chain is moved over an even number of places (two), due to the parity issues implicit in working on a unit grid.

We have described the move pulling vertices from vertex i down to vertex 1. Similarly, we could consider a pull in the other direction, starting from a free location L adjacent to $(x_{i-1}(t), y_{i-1}(t))$ and diagonally adjacent to $(x_i(t), y_i(t))$.

Finally, for technical reasons to make our move space reversible, we must add some special pull moves at the end vertices. Consider any path of two free locations, with one of the locations adjacent to vertex n. We may move vertices $n-1$ and n to these two free locations, and then pull the remaining vertices: starting with vertex $j = n-2$ and down to vertex 1, if the chain is not in a valid configuration, set $(x_j(t+1), y_j(t+1)) = (x_{j+2}(t), y_{j+2}(t))$. We similarly introduce such pull moves from vertex 1.

To help think about pull moves that affect the location of three or more vertices, we suggest the following intuition. The pull starts by creating a loop in the form of a *square*; that is, four consecutive vertices are put in a square formation. The vertices are pulled along in one direction until another square loop is reached. At this point, the existing square loop is undone and the move ends.

Theorem 1 *The class of pull moves described above is reversible.*

Proof: This follows by a case analysis. Moves that re-locate just one or two vertices are clearly reversible, so we focus on moves that re-locate three or more vertices.

Without loss of generality, suppose that vertices i to k inclusive are moved, for $k < i$. In the case where $k > 1$, the move has found an early stopping location. The intuition above clarifies this case. To reverse the pull move, we recreate the square by moving vertices k and $k+1$ back to their original positions and pull in the other direction. This pull must go back to the square that was created by moving vertices i and $i-1$. It is impossible for this move to stop before moving vertex i, since the square created by moving vertices i and $i-1$ must be the first square the reverse move reaches. Otherwise, the original move would have found a square before vertex k.

More formally, in this case, at time t vertices j and $j+3$ cannot be adjacent for $k < j < i-2$, or the move would have stopped before reaching k. To undo the move, note that $(x_{k-1}(t), y_{k-1}(t))$, $(x_k(t), y_k(t))$, $(x_{k+1}(t), y_{k+1}(t))$, and $(x_{k+2}(t), y_{k+2}(t))$ must have formed a square. We can therefore reverse the move by moving vertices k and $k+1$ back to their original positions and pull in the other direction. Suppose, for a contradiction, that the last vertex to change position in the reverse move is vertex m, with $k < m < i-2$. Then vertices $m+1$ and $m-2$ were adjacent after the first move; but this implies that vertices $m+3$ and m were adjacent in the original configuration at time t, a contradiction. One can also show that the reverse move cannot stop at vertices $i-2$ or $i-1$ by explicitly checking these cases.

The case for $k = 1$, in which the move pulls all the way to the end of the chain, is entirely similar. Here we need to make use of the fact that, in reversing the move, we can move the end along any adjacent path of length 2. $\square$

To obtain the fact that these moves are complete, we show that some sequence of moves can turn any valid path configuration into a horizontal line. Reversibility then implies that any valid configuration can be turned into any other valid configuration via a sequence of moves, since for any two configuration P and P', we can move from P to the horizontal line, and then from the horizontal line to P'. Initially, both ends of the chain may be buried deep inside the configuration, surrounded, for example, by spirals. Nevertheless, we give a simple argument that shows how pull moves can straighten out one of the tails of the chain from any starting position to yield a horizontal configuration of the tail, which protrudes from the rest of the chain. We then take advantage of this to straighten the rest of the chain.

Theorem 2 *Any configuration P can be straightened to form a horizontal line by some sequence of pull moves.*

Proof: Let $L(t)$ and $R(t)$ denote, respectively, the leftmost and rightmost vertical grid lines containing at least one edge of a chain configuration $P(t)$. Possibly $L(t) = R(t)$. If $P(t)$ contains no vertical edge, then the chain is already straight and horizontal, so from now on, suppose that $P(t)$ contains a vertical edge. The *exterior region* at time t is the set of locations that are neither on, nor between, $L(t)$ and $R(t)$.

First consider the case that an endpoint of $P(t)$ lies on $L(t)$ or $R(t)$ or in the exterior region. For example, suppose vertex 1 lies on $L(t)$ or left of $L(t)$. If 1 is left of $L(t)$, it is joined to a vertex on $L(t)$ by a horizontal subchain. In either case, the locations to the left of 1 are unoccupied, and applying a sequence of pull moves that pull vertex 1 two spaces to the left each time eventually yields a horizontal line. The situation where 1 is on or right of $R(t)$ is handled similarly.

The only remaining case is that vertex 1 lies strictly between $L(t)$ and $R(t)$. Proceeding along the chain from vertex 1, let vertices i and $i+1$ constitute the first edge lying on $L(t)$ or $R(t)$. Suppose, without loss of generality, this edge lies on $L(t)$. Then the two locations immediately to the left of vertices i and $i+1$ are free. Hence we may move vertex i to the left of vertex $i+1$, applying the local pull move that moves vertices i down to (possibly) vertex 1. In the new configuration, the left boundary $L(t+1)$ is one unit to the left of $L(t)$, with vertices i and $i-1$ lying on this boundary. Hence this process can be repeated (moving vertex $i-1$ to the left of i, and so on) until vertex 1 reaches the left boundary, at which time the chain may be straightened as described earlier. □

We note that the pull moves above can be generalized, so that instead of pulling the chain two spaces, the chain is pulled any (even) number of spaces. This will be discussed further in the full version of the paper.

4 HuGS and Tabu Search

We developed and evaluated pull moves for 2D HP using the Human-Guided search (HuGS) toolkit for rapidly developing interactive optimization systems. With HuGS, users can manually modify solutions, backtrack to previous solutions, and invoke, monitor, and halt a variety of search algorithms. Pull moves, in fact, are a refinement of an operation we initially designed for our user interface to help users manually move multiple vertices at a time.

The HuGS toolkit includes general and human-guidable search algorithms, the most powerful of which is GTabu, a variation of tabu search. We have shown GTabu to be effective on a variety of problems including jobshop scheduling, edge-crossing minimization, and the selective traveling salesman problem [11].

To apply GTabu to a problem, each problem instance must be composed of a finite number of *elements*. For 2D HP, the elements are the vertices in the given sequence. In HuGS, each possible move is defined as *operating* on one problem element and *altering* that element and possibly others. For 2D HP, a move operates on vertex i if it begins by moving i to a diagonally adjacent location, or if i is an endpoint that is moved initially. Each move is defined as altering all vertices that are moved to a new location on the grid by the move.

The primary mechanism for guiding the search is to assign *mobilities* to the problem elements. Each element is assigned high, medium, or low mobility. The HuGS algorithms are only allowed to apply moves that operate on a high mobility element and that do not alter any low mobility elements. Such moves are called *legal*. Thus, for example, in 2D HP, a vertex with low mobility will remain in its current location after any legal move. Mobilities can be assigned by the user to guide, or constrain, the search. In our experiments we use GTabu *without guidance*, i.e., all elements are initially assigned a high mobility. However, mobilities are also used by GTabu to control its own search.

In each iteration, GTabu evaluates all legal moves. It then applies the move that yields the configuration with the lowest energy, even if it increases the energy level. GTabu then updates the mobilities in order to prevent cycling and encourage exploration of new regions of the search space. First, it sets all altered elements to medium for *memorySize* iterations, where *memorySize* is one of GTabu's three control parameters. Second, it randomly sets each element to medium for one iteration with probability *noise*. Third, GTabu encourages the algorithm to choose moves that alter elements that have been altered less frequently in the past based on a *minDiv* control parameter. Exact details are given in [11].

An advantage of GTabu using mobilities to control its search is that it can display the state

name	sequence	best published result	best from pull moves
S64	12H 1P 1H 1P 1H 2P 2H 2P 2H 2P 1H 2P 2H 2P 2H 2P 1H 2P 2H 2P 2H 2P 1H 1P 1H 1P 12H	−42 w/ constraints by [16]; −40 w/o constraints by [20];	−42
S85	4H 4P 12H 6P 12H 3P 12H 3P 12H 3P 1H 2P 2H 2P 2H 2P 1H 1P 1H	−52 w/ constraints by [16]; −47 w/o constraints by [13]	−53
S100a	6P 1H 1P 2H 5P 3H 1P 5H 1P 2H 4P 2H 2P 2H 1P 5H 1P 10H 1P 2H 1P 7H 11P 7H 2P 1H 1P 3H 6P 1H 1P 2H	−47 by [2]	−48
S100b	3P 2H 2P 4H 2P 3H 1P 2H 1P 2H 1P 4H 8P 6H 2P 6H 9P 1H 1P 2H 1P 11H 2P 3H 1P 2H 1P 1H 2P 1H 1P 3H 6P 3H	−49 by [2]	−50

Table 1: The four longest test sequences for 2D HP we found in the literature with previous published best results and new best results.

of its search to the user. All HuGS applications provide a color-coded visualization of the current configuration with mobilities. This same mechanism can be used to display GTabu's mobilities if the user opts to visualize the progress of the search algorithm.

Our 2D HP application uses the same implementation of GTabu as all the other HuGS applications. The generic GTabu function calls domain-specific functions for comparing configurations, producing moves, applying moves to configurations to generate new configurations, identifying the altered elements, and producing initial configurations. Additionally, we developed a visualization component for interactive optimization.

For an overview of Tabu, see [8]). For details of GTabu, see [11]. For details of the HuGS toolkit, see [12].

5 Experimental Results

5.1 New Best Results

To begin our discussion of experimental results, we provide new best lowest energy configurations for the three longest benchmark sequences we found in previous papers. Thus, one contribution of this work is demonstrating that previously believed putative ground states are not ground states. These results do not provide a direct comparison to previous approaches, because we might be using significantly more computational resources to find these solutions. We show below, however, that these results can be obtained with reasonable computation time, even with loosely-tuned parameters. We also provide a direct comparison with results reported in [16] using a platform- and implementation-independent metric on one of our benchmark problems.

A particularly compelling example is sequence S85 in Table 1. In [13], it is stated that the optimal ground state has energy −52; it appears that the authors constructed this sequence themselves with an optimal solution in mind to test their algorithm. The genetic algorithms of [13] found a ground state of −47. In [16], an evolutionary Monte Carlo algorithm found a ground state of −52, but only

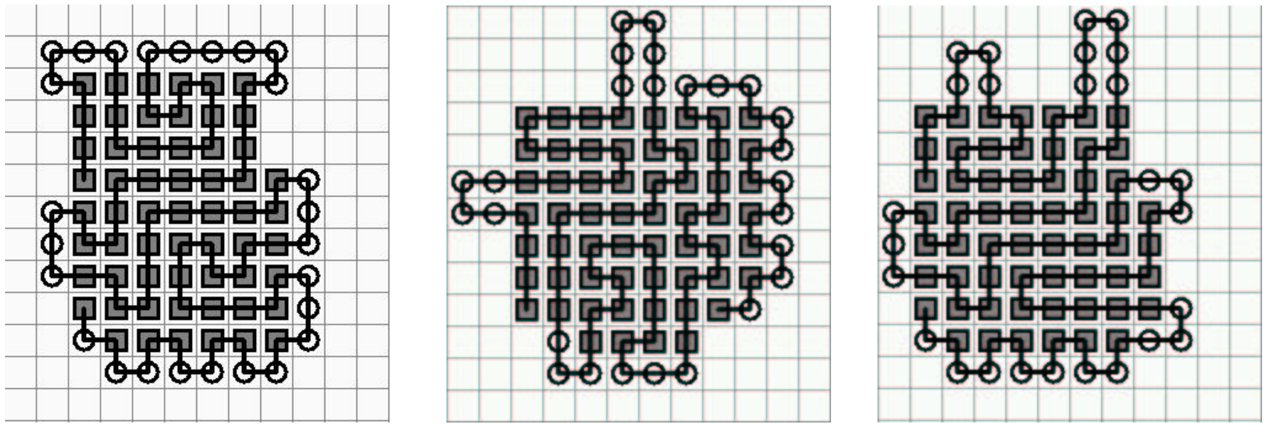

Figure 2: Three -53 solutions for S85

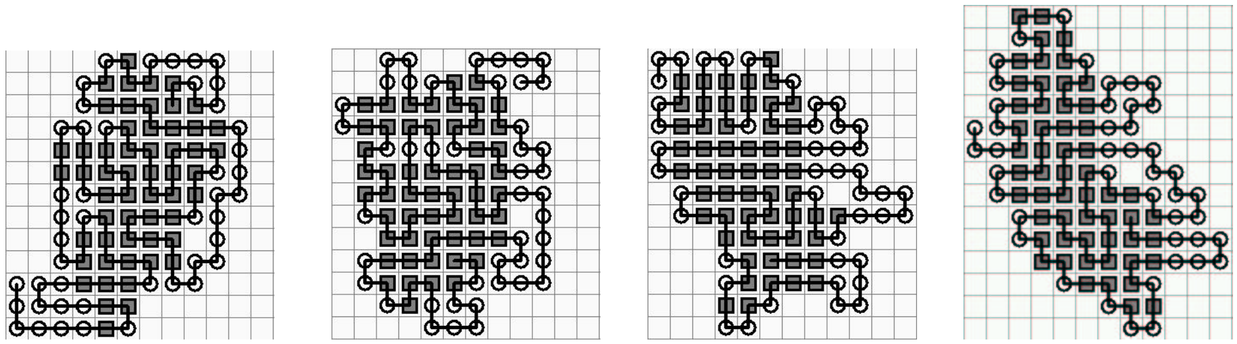

Figure 3: Two -48 solutions for S100a (on left) and two -50 solutions for S100b (on right)

by specifying constraints that significantly cut down the search space. That is, the algorithm is modified to constrain specified subsequences of hydrophobic residues (covering approximately 40% of the sequence) to take one of three forms, shown in Figure 4. Their resulting solutions, such as the one shown in Figure 4, are therefore highly structured.

Our algorithm finds several configurations with energy -53, three of which are shown in Figure 2. Note that these configurations do not have the secondary structures postulated in [16]; indeed, it seems quite unstructured. This demonstrates the potential risk of searching only for structured configurations.

Table 1 summarizes the other new lowest energy configurations found by our algorithms. We found configurations for S100a and S100b that with energies 1 point lower than the previous best solution we could find in the literature. As with S85, we found several configurations for the lowest energy; two configurations for each benchmark are shown in Figure 3. For S64, configurations with energy -42 had been previously known and found using the substructure constraints described above. To our knowledge, previously the lowest energy configuration found algorithmically without using constraints had energy -40 [20].

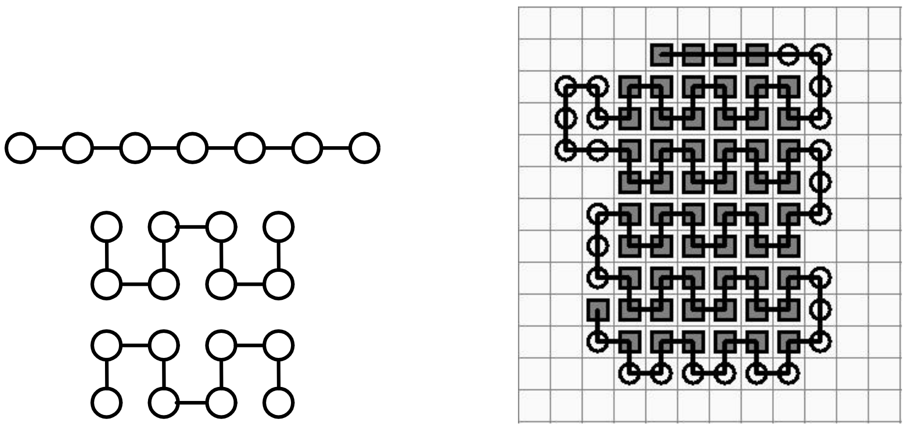

Figure 4: The constrained structures used in [16] are extended sheets and directed helixes shown on the left. These constraints yield a structured solution for S85 with energy$=-52$ shown on the right.

name	results after 30 min.	results after 60 min	number of iterations after 60 min
S64	$100.0\% = -42$	$100.0\% = -42$	2,412,258
S85	$26.5\% = -53$; $65.5\% = -52$; $8.0\% = -51$	$40.0\% = -53$; $56.0\% = -52$; $4.0\% = -51$	1,610,600
S100a	$7.5\% = -48$; $58.5\% = -47$; $34\% \leq -46$	$12.0\% = -48$; $73.0\% = -47$; $15.0\% = -46$	1,144,800
S100b	$3.5\% = -50$; $25.0\% = -49$; $71.5\% \leq -48$;	$7.5\% = -50$; $32.0\% = -49$; $60.5\% \leq -48$;	1,137,800

Table 2: Results of 200 runs of GTabu with restarts using loosely tuned parameters.

5.2 Computational assessment

Our goal in this section is to assess how much computational effort our algorithm required to produce these new best solutions, and to compare our approach to previous approaches.

It is difficult to perform a computational assessment on problems with a small number of (non-trivial) benchmarks both because of the small sample size and because it is difficult to avoid fine-tuning parameters for the target benchmarks. We address these problems by repeatedly running GTabu with randomly chosen parameters. The parameters *noise* and *minDiv* are chosen uniformly at random from the range 0 to 0.5 (the full range for these parameters is 0 to 1). The parameter *memorySize* is chosen uniformly at random from the range 5 to 10; a typical value in the tabu search literature for the corresponding parameter is 7.

We ran GTabu with these conditions 100 times for each problem with different random seeds. Each run is 1 hour on a 1000 MHz Alpha processor, using the Java code from the HuGS toolkit. GTabu found a configuration with energy -53 in 26% for S85 of these runs. It found the lowest known energy configuration for S64 in 97% of these runs, for S100a in 2% of these runs, and for S100b in 1% of these runs.

We found we could improve the performance of GTabu with a simple and well known restart strategy. We repeatedly ran GTabu for between 1,000 and 10,000 iterations (chosen randomly and uniformly from that range). Each time we restarted GTabu, we restarted it from the best

problem	Pulls		Long Pulls		Pivots	
name	adopted	considered	adopted	considered	adopted	considered
S64	3.7	4.0	12.1	36.1	21.3	19.1
S85	3.4	4.4	14.0	47.4	14.0	23.0
S100a	3.0	5.1	14.7	54.9	15.1	26.5
S100b	2.7	4.9	15.2	55.3	20.6	25.8

Table 3: Number of vertices relocated by the moves adopted and considered by GTabu using pull moves, long pull moves, and rotation moves. The numbers are averages from a single 10 minute run on each problem.

configuration found so far with new control parameters selected from the ranges described above.

The results of GTabu with restart, based on 200 runs, are shown in Table 2. These results provide a rough estimate of how long it would take our algorithm to find the best configurations known. For example, suppose we restarted our algorithm every hour (which itself is a tunable parameter). If our experiments indicated that 50% of the runs found the best configuration in 1 hour, then we would expect to find the best solution within two hours, on average. Thus, from the data above, we would expect to find a configuration with the lowest known energy in less than 1/2 an hour for S64, less than 2.5 hours for S85, less than 8.5 hours for S100a, and less than 13.5 hours for S100b. We note that we also ran our algorithm for 60 10-hour runs on each of the large benchmarks and did not find lower energy configurations. Because our algorithm is coded in Java and is currently not highly optimized, we expect these running times could be improved by straightforward engineering by a factor of 2 to 5. Table 2 also indicates the average number of iterations of GTabu search that were used to obtain these configurations.

Finally, although it is difficult to compare algorithms directly, we attempt a comparison with the computational results presented in [16]. There, the authors report the number of valid configurations evaluated to reach the best configuration found by five independent runs of various algorithms (without the of use of additional constraints). For S64, one run found -39 after evaluating about 560,000 configurations. To provide a fair comparison, we multiply their reported number by five to find that their total computational effort required examining roughly 2.8 million configurations. We ran GTabu, with restarts, until it had computed 2.5 million configurations. On average, roughly 50 moves are considered per iteration, and so 2.5 million moves are considered but only 50,000 moves are taken. Based on 175 runs, GTabu finds a configuration with energy -42 53% of the time, and with energy -39 or better about 84% of the time.

5.3 Move Statistics

In other experiments, we examined the behavior of pull moves and gathered statistics on their behavior.

Our intuition was that an important aspect of pull moves, in terms of their effectiveness on these problems, is that they make relatively small adjustments to a given configuration. We modified the algorithm to keep track of the number of vertices that were altered by the moves considered and adopted by the tabu search. (As described above, in each iteration, GTabu considers all legal moves and adopts the one leading to the lowest energy state.) We ran GTabu for 10 minutes on each problem. The results confirmed that pull moves alter only a small number of vertices. As shown in Table 3, the average number of vertices relocated in moves considered by GTabu ranged from 4.0 to 5.1 on the four benchmarks. Interestingly, the move actually adopted by GTabu consistently altered, on average, fewer vertices.

One opportunity suggested by these results is that we could improve the efficiency of our algorithm by computing the energy of the configuration that results from applying a move by computing the change based on vertices that moved.

To further explore our hypothesis about why pull moves are effective, we defined a set of moves called *long pull moves*. Long pull moves are identical to pull moves, except that they always continue to pull vertices until the end of the sequence is reached, rather than stop as soon as a valid configuration is reached. Table 3 shows that, as expected, long pull moves alter a much larger number of vertices, on average, than pull moves. Also, using long pulls dramatically degrades the performance of GTabu. Based on 50 1-hour runs, the lowest energy found by GTabu with long pulls was -38 on S64 in 2% of the runs, -51 on S85 in 2% of the runs, -45 on S100a in 4% of the runs, and -44 on S100b in 20% of the runs.

Finally, we compared pull moves to *pivot moves* [17, 14, 20]. Our implementation of pivot moves essentially swivels a portion of the sequence (from one vertex to an end point) 90 degrees in one direction or another. While pivot moves alter the grid-location of a large number of vertices, they preserve the relative relationship between most pairs of vertices that are adjacent in the sequence. However, GTabu also performed poorly with pivot moves. Based on 50 1-hour runs, the lowest energy found by GTabu with pivot moves was -40 on S64 in 4% of the runs, -49 on S85 in 10% of the runs. -47 on S100a in 4% of the runs, and -48 on S100b in 4% of the runs.

6 Conclusions and Future Work

The pull moves we have developed in this paper appear quite natural: move one amino acid a small distance and then pull the chain along, stopping as soon as possible. It would be interesting to determine if there is any relation between the theoretically designed pull moves and protein folding in the real world.

We would like to expand our prototype to handle more challenging protein folding problems. A relatively simple extension would be to the 3-dimensional HP model. Functioning in more complex geometries appears more difficult but may be possible. A reasonable question is whether pull moves and tabu search can be extended to 3-dimensional space without an underlying unit grid.

Although our algorithm was built using the HuGS toolkit, which is designed to allow human interaction with the optimization algorithms, we have not yet tested whether human guidance can lead to low energy configurations more quickly than the automatic algorithm alone. It would be especially interesting to see if domain-experts could provide better guidance to our algorithm than people who only understand the algorithms well. Furthermore, other advantages of interactive optimization include allowing people to incorporate real-world knowledge they have that is not part of the objective function given to the computer and to help people better understand the problems and solutions they are working with. We hope to use HuGS to explore these benefits for interactive protein folding in future research.

Finally, our results show that previously used benchmarks for this problem were not well understood, in that our algorithm found new lowest energy configurations. Further research on the 2D HP model would benefit from a larger and richer set of benchmarks, preferably with known optimal solutions. Because hardware speeds and algorithms may improve rapidly, any benchmark should also include some significantly longer strings to challenge the next generation of algorithms and hardware applied to the problem.

References

[1] R. Agarwala, S. Batzoglou, V. Dancik, S. Decatur, M. Farach, S. Hannenhali, S. Muthukrishnan, and S. Skiena. Local rules for protein folding on a triangular lattice and generalized hydrophobicity in the HP model. Journal of Computational Biology, 4(2):275-296, 1997.

[2] U. Bastolla, H. Frauenkron, E. Gerstner, P. Grassberger, and W. Nadler. Testing a new Monte Carlo algorithm for protein folding. *Proteins: Structure, Function, and Genetics*, 32:52-66, 1998.

[3] B. Berger and T. Leighton. Protein folding in the hydrophilic-hydrophobic (HP) model is NP-complete. *Journal of Computational Biology*, 5(1):27-40, 1998.

[4] P. Crescenzi, D. Goldman, C. Papadimitriou, A. Piccolboni, and M. Yannakakis. On the complexity of protein folding. *Journal of Computational Biology*, 5(3):409-422, 1998.

[5] K. A. Dill. Theory for the folding and stability of globular proteins. *Biochemistry*, 24:1501, 1985.

[6] P. G. de Gennes. *Journal of Chemical Physics* vol. 55, p. 572, 1971.

[7] P. G. de Gennes. *Scaling Concepts in Polymer Physics.* Cornell University Press, 1979.

[8] F. Glover and M. Laguna. 1997. *Tabu Search.* Kluwer Academic Publishers.

[9] W. E. Hart and S. Istrail. Fast protein folding in the hydrophobic-hydrophilic model within three-eights of optimal. *Journal of Computational Biology*, 3(1):53-96, 1996.

[10] W. E. Hart and S. Istrail. Robust proofs of NP-hardness for protein folding: general lattices and energy potentials. *Journal of Computational Biology*, 4(1):1-22, 1997.

[11] G. Klau, N. Lesh, J. Marks, and M. Mitzenmacher. Human-Guided Tabu Search. In *Proceedings of the 18th National Conference on Artificial Intelligence*, pp. 41-47, 2002.

[12] G. Klau, N. Lesh, J. Marks, M. Mitzenmacher, and G.T. Schafer. The HuGS platform: A toolkit for interactive optimization. In *Proceedings of Advanced Visual Interfaces*, pp. 324-330, 2002.

[13] R. König and T. Dandekar. Improving genetic algorithms for protein folding simulations by systematic crossover. *BioSystems*, 50:17-25, 1999.

[14] M. Lal. Monte Carlo simulations of chain molecules. *Mol. Phy.*, 17:57-64, 1969.

[15] K. F. Lau and K. A. Dill. A lattice statistical mechanics model of the conformational and sequence spaces of proteins. *Macromolecules*, 22:3986, 1989.

[16] F. Liang and W. H. Wong. Evolutionary Monte Carlo for protein folding simulations. *Journal of Chemical Physics*, 115(7):3374-3380, 2001.

[17] N. Madras and A. D. Sokal. The pivot algorithm: a highly efficient Monte Carlo method for the self-avoiding walk. *Journal of Statistical Physics*, 50:109-186, 1988.

[18] A. Newman. A new algorithm for protein folding in the HP model. In Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms, pages 876-884, 2002.

[19] R. Ramakrishnan, B. Ramachandran, and J. F. Pekney. A dynamic Monte Carlo algorithm for exploration of dense conformational space in heteropolymers. *Journal of Chemical Physics*, 106:2418, 1997.

[20] L. Toma and S. Toma. Contact interactions method: A new algorithm for protein folding simulations. *Protein Science*, 5:147-153, 1996.

[21] R. Unger and J. Moult. Genetic algorithms for protein folding simulations. *Journal of Molecular Biology*, 231:75, 1993.

[22] R. Unger and J. Moult. Finding the lowest free energy conformation of a protein is an NP-hard problem: Proof and Implications. *Bull. Math. Biology*, 55(6):1183-1198, 1993.