A Filter-and-Fan Approach to the 2D Lattice Model of the Protein Folding Problem

César Rego[a*], Haitao Li[b], Fred Glover[c]

a School of Business Administration, University of Mississippi, University, MS 38677, USA. crego@bus.olemiss.edu

b Sorrell College of Business, Troy University-Montgomery, Montgomery Campus, 136 Catoma Street, 2n, Montgomery AL 36103-4419, USA. hli@troy.edu

c Leeds School of Business, University of Colorado, Boulder, CO 80309-0419, USA. fred.glover@colorado.edu

Latest Revision: January 7, 2006

Abstract — We examine a prominent and widely-studied model of the protein folding problem, the two-dimensional (2D) HP model, by means of a filter-and-fan (F&F) solution approach. Our method is designed to generate compound moves that explore the solution space in a dynamic and adaptive fashion. Computational results for a standard set of benchmark problems show that the F&F algorithm performs more robustly and efficiently than the current leading algorithms, requiring only a single solution trial to obtain best known solutions to all but two of the benchmark problems, in contrast to a hundred or more trials required in the typical case by the best of the alternative methods. On the remaining problems a single trial of our method obtains a solution one unit away from the best known solution.

Keywords: metaheuristics, tabu search, compound neighborhoods, filter-and-fan, protein folding, bioinformatics.

* Corresponding author.

1. Introduction

A protein is a linear polymer molecule that may consist of thousands of monomer units. The monomers are the 20 known amino acids. The typical length of a protein chain ranges from 50 to 1000 amino acids. In the natural environment a protein adopts a specific *tertiary structure* called a *conformation*. There are essentially three types of proteins existing in nature: fibrous, membrane and globular. Most studies on protein structure prediction have focused on globular proteins due to the fact that, unlike the other two, a globular protein exhibits an extremely compact and unique conformation in its native state (Chan & Dill 1993). The native state of a protein is often thermodynamic stable, which means that it is associated with a global minimum energy. Also, it is well established that the amino acid sequence is a major factor in determining the folding conformation of a protein (Anfinsen et al. 1961). Other factors such as enzymes catalyze the folding process but do not play a determinant role as the amino acid sequence does in yielding the structure ultimately attained (Richards 1991).

The Protein Folding Problem (PFP) is the problem of predicting the three-dimensional (3D) structure of a protein given only the protein's sequence of amino acids. This is a fundamental yet open problem in the fields of biological chemistry and protein science, and has recently attracted attention in bioinformatics and computational biology. Understanding and predicting the native conformation of a protein is of both theoretical and practical importance. A protein's function is closely related to its 3D structure, and therefore to determine how a protein functions one must know its 3D conformation. Now that the Human Genome Project has mapped the complete human genome sequence (represented by over 3 billion DNA subunits), advanced bio-computing models and techniques are crucial for interpreting human gene functions and so proteins. The PFP is central in a number of practical applications including the designing of new proteins having desirable functions in pharmaceutical, food, and agriculture industry (Lengauer 1993). We refer to Richards (1991) and Chan and Dill (1993) for an overview of the PFP and its applications.

The PFP is a notoriously difficult combinatorial problem due to the combinatorial explosion of valid conformations as the number of amino acids in the chain increases. It has been estimated that using current computer technologies, finding all possible conformations for a 100-aminoacid protein would require 10^{27} years of computer processing (Krasnogor et al. 1998), a length of time that is more than a thousand billion times the age of the universe. Notably the problem has been identified as a National Grand Challenge in biochemistry (Committee on Physical, Mathematical and Engineering Science 1992) in the U.S. Even its simplified version, the lattice HP model (Dill 1985) has been proved to be NP-Complete (Crescenzi et al. 1998) and therefore no polynomial algorithm is available for its solution (and theoretically none can exist unless P=NP).

Two experimental methods of determining the protein structure are X-ray crystallography and Nuclear Magnetic Resonance (NMR). Most of the structural information in the Protein Data Bank (PDB) is obtained through these two approaches (Berman et al. 2000). Although these methods are highly effective they are also exceedingly expensive in their demands on equipment and processing time. Bioinformatics supported by advanced optimization models and techniques provides an alternative approach with the potential to predict the native structure of proteins in a much more cost-effective manner. Since current exact solution methods are unable to solve even the smallest instances of the PDB, effective heuristic (approximation) algorithms are required to find optimal or near-optimal conformations. Also, due to the complex nature of the PFP, the so-called HP lattice model proposed by Dill (1985) constitutes a well established simplification for algorithm assessment.

This paper considers the two-dimensional (2D) version of the HP lattice model and proposes a new algorithm for the solution of the associated PFP. The key contribution is the development of an effective mechanism for adapting simple neighborhood structures to enable them to explore the solution space more effectively. The approach is based on a filter-and-fan procedure that augments a simple neighborhood in a dynamic and adaptive fashion using a truncated form of tree search. Computational testing carried out on a standard testbed demonstrates the superiority of the proposed filter-and-fan algorithm relative to alternative approaches in the literature. The remainder of the paper is organized as follows. Section 2 introduces the 2D HP lattice model and Section 3 develops the filter-and-fan algorithm. Computational results are presented in Section 4, followed by a summary of conclusions and a discussion of promising avenues for future research in Section 5.

2. The 2D HP Model

The HP lattice model (Dill 1985) is the most widely studied protein folding model. Although relatively simple the model captures many global aspects of protein structure (Chan and Dill 1993). Under the assumption that the hydrophobic interaction is the dominant force in protein folding, the model considers two types of amino acids: hydrophobic monomers (denoted by H), which are oil-like and interact poorly with water, and hydrophilic (or *polar*, denoted by P), which interact favorably with water. A sequence of H and P amino acids is configured as a path on a two-dimensional (2D) lattice to define a valid *conformation*. The path designation implies that the conformation is both connected and self-avoiding, i.e., no amino acids can collide in the same cell of the lattice. (In graph theory terminology, such a path is called *node simple*.) The energy function is defined by the number of pairs of H nodes that are *adjacent* in the lattice and not *consecutive* in the chain. Each of these pairs, generally called an H-H contact, decreases the energy value by one unit. The objective is to find a conformation that minimizes the total energy of the given amino acid sequence, which therefore corresponds to maximizing the number of H-H contacts.

Figure 1 illustrates a protein with nine hydrophobic (H) and two hydrophilic (P) monomers represented by black and white nodes respectively. Nodes 1 and 9 denote the two extreme nodes in the sequence.

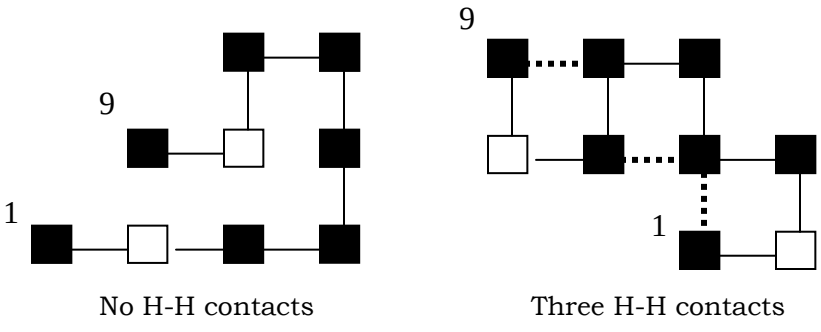

Figure 1. An HP lattice model of protein with 9 hydrophobic (H) and 2 hydrophilic (P) monomers.

In the native state of globular proteins, the nonpolar nature of H monomers pulls them in a manner that causes them to squeeze into the core of the conformation attached to

each other. Conversely, P monomers tend to reside on the surface attached to water molecules.

Various computational approaches have been proposed to tackle this simplified yet NP-complete combinatorial problem. Genetic Algorithms (GA) have been proposed by Unger and Moult (1993), Dandekar and Argos (1994), and Konig and Dandekar (1999). Classical Monte Carlo Simulation (MC) approaches were developed by Covell and Jernigan (1990) and Skolnick and Kolinski (1990). More recently, variants of the MC approach have been introduced in the Pruned Enriched Rosenbluth Method (PERM) by Grassberger (1997), Hsu et al. (2003a, 2003b), the dynamic Monte Carlo algorithm by Ramakrishnan, Ramachandran and Pekny (1997), the evolutionary Monte Carlo (EMC) algorithm by Liang and Wong (2001) and the Multi-Self-Overlap-Ensemble (MSOE) by Chikenji, Kiduchi and Iba (1993). Lesh, Mitzenmacher and Whitesides (2003) apply a randomized multi-start Tabu Search algorithm that obtains several best results for the standard 2D HP benchmark problems. Other approaches include Ant Colony Optimization (ACO) by Shmygelska and Hoos (2002, 2003, 2005), Chain Growth approximation algorithms by Bornberg-Bauer (1997) and Hart and Istrail (1996), and a Constraint Programming (CP) approach by Backofen (2001).

3. The Filter-and-Fan Algorithm

Generally speaking, the success of a metaheuristic algorithm is driven by two fundamental components, the neighborhood structure used for the local search and the strategy to guide the search beyond local optimality. Depending on the size and difficulty of the problem, different levels of sophistication may be required for each of these components. In general, large scale problems require highly effective neighborhood structures that are capable of exploring large neighborhood spaces. A special class of such neighborhoods is represented by the so-called ejection chain methods, which are chiefly designed to explore exponential neighborhoods in polynomial time (see Glover 1992, 1996, and Rego and Glover 2002). Although very powerful, ejection chain methods are typically more complex than other more traditional neighborhood designs. In cases where ejection chain approaches are not available or the problem size is not exceedingly large, simpler neighborhoods accompanied by appropriate search guidance may be adequate. We pursue the approach of seeking an effective guidance strategy within a simpler neighborhood by extending the so-called pull-move neighborhood (Lesh, Mitzenmacher and Whitesides 2003) by means of a filter-and-fan method. To elaborate the algorithm we first describe the pull-move neighborhood structure.

3.1 The Pull-Move Neighborhood

A filter and fan algorithm requires the definition of component moves used to generate trial solutions throughout the search process. Component moves are characteristically simple moves serving as building blocks for the construction of an extended filter and fan neighborhood. We make use of component moves defined by the pull-move neighborhood introduced in the tabu search implementation of Lesh, Mitzenmacher and Whitesides (2003). A pull-move is initiated by moving one node of the current conformation to one of its empty diagonal adjacent positions in the square induced by the node and one of its adjacent neighbors in the sequence. Depending on the structure of the conformation the displacement of the initiating node may require other nodes to change their current positions in order to preserve connectivity. In a pull-move, displaced nodes are only allowed to occupy vacant adjacent positions in the lattice. Consequently, the preservation of connectivity also results in a self-avoiding path. We

differentiate only three types of pull-moves designated by *filling*, *single-pull* and *multiple-pull,* according to the number of nodes that are pulled by the first displaced node. Figures 2, 3 and 4 provide an illustration of each type of these moves and associated reference structures. To simplify the notation in the following, we allow labels be attached to the nodes that denote both the name of the node and its position in the lattice.

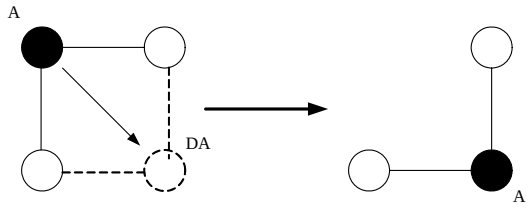

Figure 2. Pull move – *filling*

The *filling* move is the simplest pull-move, displacing a single node in the structure. As shown in Figure 2, a valid conformation is obtained by simply moving node A to its diagonal adjacent position (DA).

A *single-pull*, on the other hand, requires another node to change position after the initiating node takes a new position. Consider the conformation given in the left-diagram of Figure 3. Moving node A to its adjacent diagonal position DA disconnects node B from node A as they are no longer adjacent nodes in the lattice. However, a connected conformation can be obtained by moving node B to position C, the remaining position in the square identified by the dotted lines.

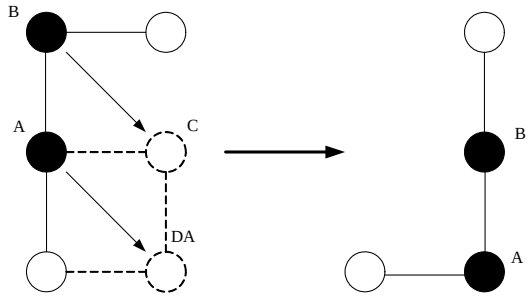

Figure 3. Pull move – *single-pull*

The *multiple-pull* move extends the pull-move to achieve connectivity in more complex structures that become disconnected upon performing a single-pull move. An illustration of such type of structure is provided in Figure 4.

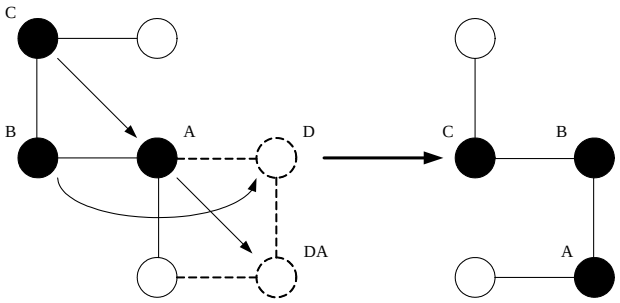

Figure 4. Pull move – *multiple-pull*

The single-pull move represented by moving A to position DA and B to position D disconnects B from C. However, connectivity can be established by moving C to fill the position vacated by A. If the latter move brings about a new disconnected structure the process is repeated until connectivity is achieved. A universal rule to implement a multiple-pull consists of shifting successive nodes two positions ahead along the original chain until connectivity is reached. In the example, node C (i.e. the successor of B) is shifted through position C and then A in the original chain. Since at this point the structure is connected the process stops and the move is completed.

It should be noted that the P nodes (shown in white) in the preceding illustrations could just as well be H nodes. It may also be observed that the single-pull move of Figure 3 can be performed as a succession of two filling moves. However, a single-pull move is not a special case of a multiple-pull move. As it turns out, generalizations of these three types of moves, and of combinations of them, are automatically generated by the method we describe next.

3.2 The Filter and Fan Model and Application

The filter and fan (F&F) method was initially proposed in Glover (1998) as a method for refining solutions obtained by scatter search, and was further extended in Rego and Glover (2002). In the latter, the method is proposed as an alternative to ejection chain methods and as a means for creating combined neighborhood search strategies. Conceptually, it integrates the *filtration* and the *sequential fan* candidate list strategies used in tabu search (Glover and Laguna 1997), and can be viewed as a restrictive form of tabu search that generates multiple paths in a breadth search strategy. From a neighborhood search perspective, the method generates compound moves as a sequence of more elementary *component moves* (or submoves).

Graphically, the F&F model can be illustrated by means of a neighborhood tree where branches represent submoves and nodes identify solutions produced by these moves. An exception is made for the root node, which represents the starting solution to which compound moves are to be applied. The maximum number of levels L permitted in a single sequence of moves defines the depth of the tree. The neighborhood tree is explored breadthfirst, level by level. Each level is governed by the *filter candidate list* strategy that selects a subset of moves induced by the *fan candidate list* strategy. The method incorporates two fundamental components: a *local search* to identify a local optimum and a *filter and fan search* to explore larger neighborhoods in order to overcome local optimality. Any time a new local optimum is found in one search strategy the method switches to the other strategy and keeps alternating this way until the filter and fan search fails to improve the current best solution.

More advanced versions allow for the combination of different types of neighborhood structures and the use of adaptive memory programming as introduced in tabu search. For a detailed description of the method and the use of advanced strategies we refer to Glover (1998) and Rego and Glover (2002). The filter and fan approach used in this study employs component moves identified by the pull-move neighborhood described in the previous section. This neighborhood is likewise used in the local search phase. The algorithm is equipped with a tabu search short-term memory aimed at enhancing the local search and providing the tree search with appropriate legitimacy restrictions to drive the search and keep the method from generating duplicated conformations.

The Filter-and-Fan Construction

Our F&F search can be sketched as follows. Once a locally optimal solution X_o is found (in the local search phase) the best η_1 currently available moves (among the moves evaluated to establish local optimality) are used to create the level 1 of the F&F neighborhood tree. As a basis for creating the next levels, for a given level indexed by k, η_1 denotes the number of solutions that are chosen from all solutions available at level k, as a foundation for generating solutions at level $k + 1$. (For $k = 1$, there are just η_1 solutions available, so all are chosen.) For each of these η_1 solutions, denoted $X_i(k)$ (i=1,..., η_1), we apply η_2 moves to generate η_2 descendant solutions, thereby generating a total of $\eta=\eta_1.\eta_2$ trial solutions for level k+1. At this stage, η_1 of the resulting η solutions are chosen to launch the process for the next level. The values η_1 and η_2 are input parameters, e.g. $\eta_1=2\eta_2$. (In more general versions of the approach, which we do not consider here, these parameters can vary adaptively from one level of search to the next.) If an improved solution (better than the local optimum X_o) is found among the trial solutions, then the method stops branching and switches back to the local search phase, taking this newly improved solution as a starting point. Otherwise, another selection takes place over the set of moves available.

The process of selecting η_2 moves has to obey a set of legitimacy restrictions that assure compatibility of the component moves used for the construction of a valid compound move. The *fan candidate list strategy* is embedded in the generation of the η trial solutions, whereas the selection of the η_1 solutions from this collection constitutes the *filter candidate list strategy.*

A filter and fan neighborhood extends and generalizes the pull move neighborhood in multiple senses. Since the outcome of a move chosen at one node of the tree is transmitted to subsequent nodes of the same branch, the method is *adaptive* in the sense that each pull-move is chosen according to the current state of the search. Also, the method is *dynamic* since the number of pull-moves used to compose a compound move depends on the level of the tree where the best trial move was found, which usually varies from one iteration to another, again depending on the state of the search. This produces a generalized form of a so-called variable-depth neighborhood. Another dynamic feature of the filter and fan method is its flexibility in varying the tree width and branch width throughout the search in the case where the values for η_1 and η_2 are changed from level to level. (In additional variants of the procedure, as when making use of constructive or destructive neighborhoods, a solution can refer to a partial solution, having some components undetermined. Local optimality is then defined in a special sense relative to the determined components, or by employing a default trial completion that fills in the values of the undetermined components.)

Figure 5 shows an example of the filter and fan neighborhood for a 2D HP model with 20 amino acids, where $\eta_1=\eta_2=4$ and $L = 5$.

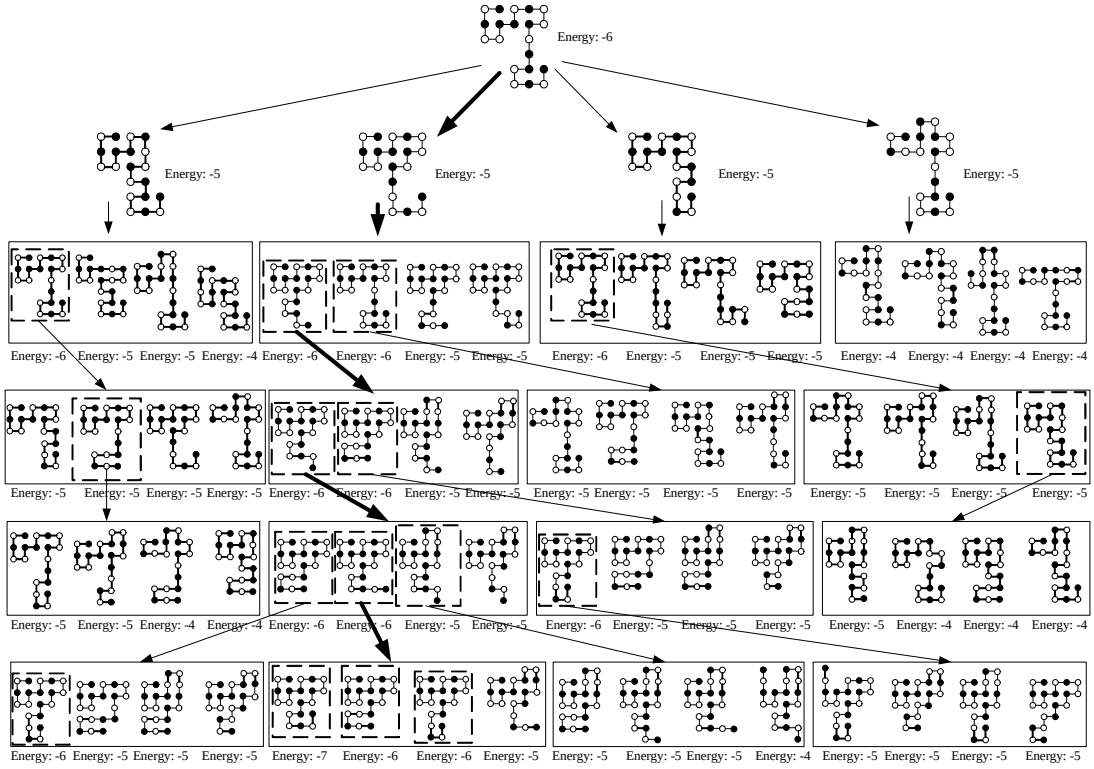

Figure 5. A Filter & Fan neighborhood for 2D HP model

A conformation of energy -6 (represented by the root node) denotes a local optimum determined by the local search phase. The first level of the filter and fan neighborhood is then generated by applying the $\eta_1=4$ best moves to the root conformation. (Note that at this stage, these best moves have already been identified by the local search method during the evaluation of the neighborhood that established the local optimum.) Due to local optimality, none of the conformations obtained in the first level improves the local conformation. The next level is created by applying the $\eta_2=4$ best pull-moves to each of the conformations in the current level, thus generating $\eta_1.\eta_2=16$ trial conformation from which a new set of $\eta_1=4$ best conformations is chosen to initiate the next level. In the figure, the η_2 different conformations derived from the same parent conformation are contained within the rectangles delimited by solid lines whereas the η_1 best conformations selected at each level are contained within "interior rectangles" delimited by dotted lines. The method continues expanding the neighborhood until the improved conformation of energy -7 is found in level 5 of the filter and fan tree. The compound move leading to the improved conformation is then identified by the path indicated by the dark arrows.

3.3 The Filter-and-Fan Implementation

Basic Data Structures

The design of appropriate data structures is crucial for the efficiency of the algorithm. In our approach, an *adjacency structure* is created to capture the relevant attributes of

an amino acid in the conformation such as its ordering position (or index), its type (H or P), its Cartesian coordinates, and its neighboring amino acids. From this we create an *array of adjacency structures* to model the entire conformation.

Specifically, let an array A denote a protein consisting of a chain of amino acids $A[i]$ (i = 1 to $|C|$). We then model $A[i]$ by an *adjacency structure* consisting of a set of components that are relevant for representing the conformation of a protein, writing $A[i]$ as $A[i]$ = {*index*(i), *type*(i), *coordinates*(i), *east*(i), *south*(i), *west*(i), *north*(i)}, where *index* provides the sequential order of an amino acid in a chain, *type* records the type of the corresponding amino acid with 1 referring to H and 0 referring to P, *coordinates* is the structure holding the Cartesian coordinates (x_i, y_i) indicating the position of an amino acid in a 2D lattice, and the remaining components record the indexes of the four neighboring amino acids.

We refer to the array A of adjacency structures as an *adjacency table*. Consider for illustration a protein consisting of 14 HP amino acids shown in Figure 6 that has the following conformation.

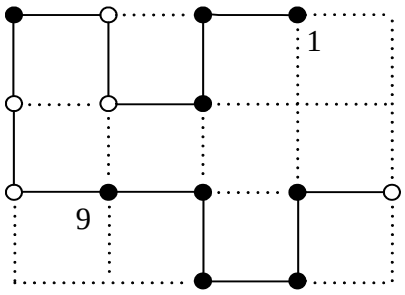

Figure 6. Conformation of 14 amino acids with black nodes denoting 1 (H) and white nodes denoting 0 (P).

The *adjacency table* representing the above conformation is shown in Table 1 below. Each row of the adjacency table is an *adjacency structure* representing one amino acid in the protein.

Node			*Neighboring Node Indexes*			
index	*type*	*coordinates*	*east*	*south*	*west*	*north*
1	1	(0, 0)	0	0	2	0
2	1	(-1, 0)	1	3	5	0
3	1	(-1, -1)	0	10	4	2
4	0	(-2, -1)	3	9	7	5
5	0	(-2, 0)	2	4	6	0
6	1	(-3, 0)	5	7	0	0
7	0	(-3, -1)	4	8	0	6
8	0	(-3, -2)	9	0	0	7
9	1	(-2, -2)	10	0	8	4
10	1	(-1, -2)	13	11	9	3
11	1	(-1, -3)	12	0	0	10
12	1	(0, -3)	0	0	11	13
13	1	(0, -2)	14	12	10	0
14	0	(1, -2)	0	0	13	0

Table 1. The adjacency table representing the 14-node conformation in Figure 6

A zero index in the neighboring list indicates that the corresponding neighboring position is empty. The origin of the Cartesian coordinate system is fixed at the position of the first amino acid in the initial conformation. As new conformations are generated the adjacency structure is updated with the coordinates of the amino acids in the new conformation. It is important to notice that an absolute coordinate system is used so that only points that change their position need to be updated in the structure. However, to prevent the coordinate values from becoming exceedingly large, we set upper and lower bounds for x and y components that are given by the size of an integer data type for the computer in use. Hence, any time a pull-move results in taking an amino acid to a position that is out of the pre-defined range the entire conformation is shifted back to relocate node 1 at the origin point (0, 0). In addition to specifying the coordinates of an amino acid it is important to identify its adjacent amino acids in the lattice. As explained later such adjacency information is fundamental for detecting valid pull moves as well as for their execution. Furthermore, the fact that the adjacency list refers to indexes in the adjacency table associated with amino acids, the coordinates of displaced nodes and the adjacency information can be simultaneously updated.

Another basic data structure is the one for representing a simple pull move. A pull move can be completely defined by reference to three basic components, the pull move initiating node, the location associated with the initial move and the energy value of the new conformation after completing the pull move. The first two components, which define the execution of the move as single- and multiple-pull, are implicitly determined by the pull-move rules as explained in Section 3.1. Since the evaluation of a pull move requires the identification of the resulting conformation, it is useful to record the corresponding energy. Specifically, a pull move structure is defined as PullMove = {*node*, *location*, *energy*}. There are 4 possible locations for a node to move to in the execution of a pull move, defined by the four diagonal positions adjacent to the node, which we denote by NORTHWEST, NORTHEAST, SOUTHEAST, and SOUTHWEST as depicted in Figure 7.

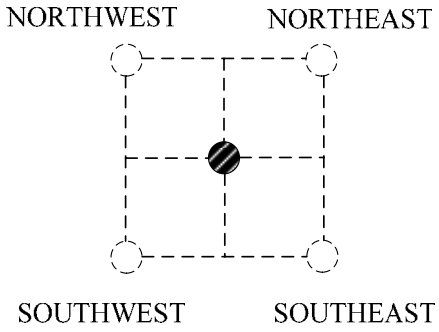

Figure 7. Possible assignments to the *location* data member of a PullMove structure

Filter-and-Fan Data Structures

In order to model the filter-and-fan neighborhood tree presented in Section 3.1, we extend the information associated with the pull move to include a new element, an integer data type named *conformation*, that identifies the conformation on which the move is carried out. Hence the pull move structure becomes PullMove={*conformation*, *node*, *location*, *energy*}. From this point onward we say that a move m is a pull move if it is an instance of the PullMove structure. Also, we refer to x(*node*) and y(*node*) as the x

and *y* coordinates of the pull move initiating *node*. An illustration of the data structures for one level of the filter and fan search is depicted in Figure 8.

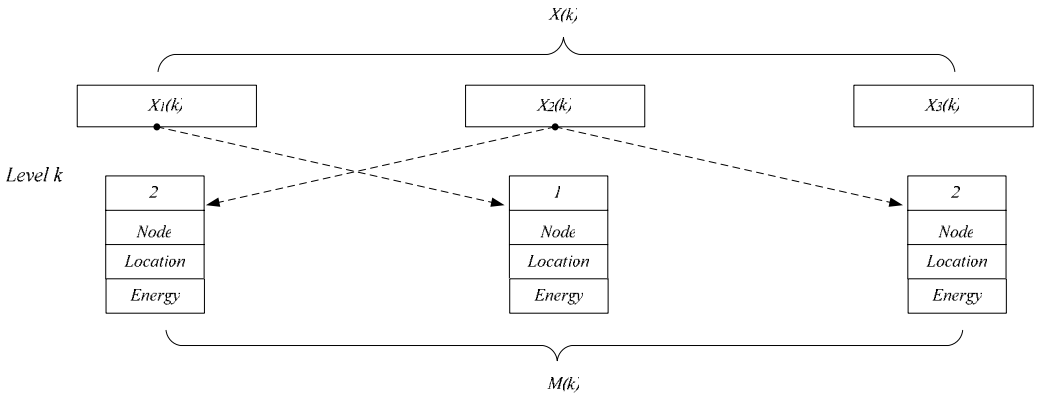

Figure 8. Data structures for the filter and fan neighborhood

Following the conventions introduced in Section 3.2, $X(k)$ denotes the set of best conformations at level k, $X_i(k)$ the i-th best conformation in $X(k)$, $M(k)$ the set of best pull moves associated with $X(k)$ and m_{ik} the i-th best pull move in $M(k)$. The logical relationship between $X_i(k)$ and m_{ik} has been identified by the index of the *conformation* in the generalized pull move structure. From the figure we can see that the best and the third best pull moves were derived from the second best conformation at level k while the second best pull move derived from the first conformation.

We now describe the procedures that are essential for the efficiency of the algorithm.

Procedures for Pull Move Neighborhood

The implementation of a pull move neighborhood requires three basic functions associated respectively with: (a) detecting all valid pull moves available, (b) executing the pull move, and (c) computing the energy value of the resulting conformation.

We define a *local conformation* of node *i*, denoted by *localConformation*(*i*), to be the structure in the current conformation formed by node *i* and its adjacent nodes in the amino acid sequence. Figure 9 illustrates all six possible instances of a local conformation for the stripped black node appearing in the middle of the structure. Indexes 1 to 6 are used to code each of these possibilities.

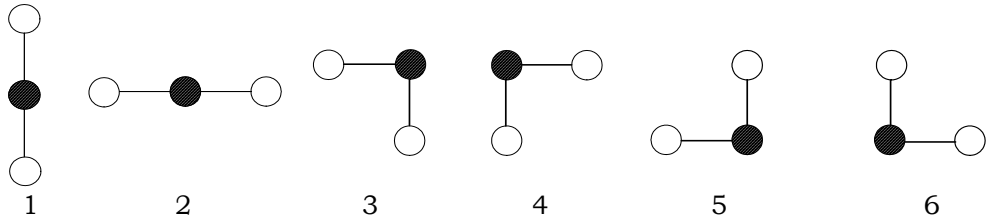

Figure 9. Six possible instances of a *local conformation*

As can be seen, a *local conformation* is either a straight line (type 1) or an L shape (type 2). In a 2D lattice, there are two possibilities for a straight line (indexes 1 and 2) and four possibilities for an L shape (indexes 3 to 6). As shown in Figure 10, for a straight line, all the four moving directions are inspected; for an L shape, only three of them are possible.

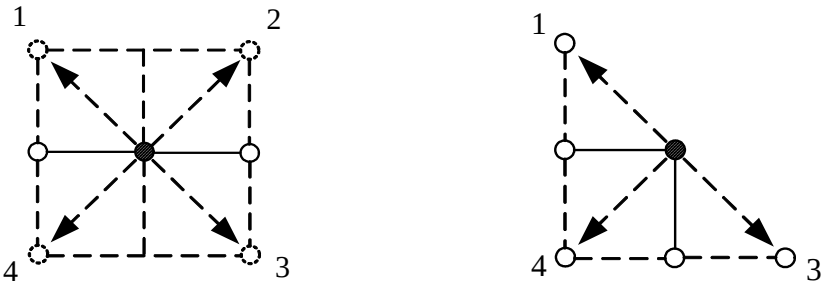

Figure 10. Detecting a valid pull moves for *local conformations* of type 1 (left) and type 2 (right)

We employ the following notation and definitions.

pullMoveList: a list storing all valid pull moves associated with a conformation.

GetLocalConformation (*i, A*): returns the local conformation code associated with the i-th node.

IsLegal (*i, j, A, localConf*(*i*)): returns true if displacing node *i* to location j is a valid pull move, and false otherwise.

AddPullMove (*node, location, pullMoveList*): attaches a valid pull move to the pull move list with initial energy value of zero. (Actual energy values are then evaluated after executing the pull move.)

A straightforward method for detecting all valid pull moves is given by the following procedure:

```
Procedure DetectAllPullMoves (A, pullMoveList)
Begin
    Initialization
        localConformation(index) = 0
        pullMoveList = Φ
    For index = 1 to |C|
        localConformation(index) = GetLocalConformation (index, A)
        For location = 1 to numOfLocations
            If IsLegal (index, location, A, localConformation(node)) then
                AddPullMove (index, location, pullMoveList)
            Endif
        End
    End
End Procedure
```

It is important to realize that the evaluation of a pull move requires the execution of the pull move. Likewise, as the pull move process unfolds, adjacency information of the current active conformation is necessary to detect further valid pull moves and evaluate the corresponding complete moves associated with nodes of the filter-and-fan tree. Consider for implementation purposes an adjacency table A representing the current active conformation and A' be a clone of A on which a pull move m is executed.

GetConnected (*node, location, A, A'*): displaces as many nodes as needed to make the conformation connected after moving the initiating *node* to the specified *location*. Once the move is completed, both the pull-move initiating node and the last displaced node are set *tabu-active* for a predefined number of iterations.

```
Procedure ExecutePullMove (A, m)
Begin
    Initialization Step
       A' = A
       Switch (location)
         Case NORTHWEST: x(node) = x'(node) – 1
                         y(node) = y'(node) + 1
         Case NORTHEAST:  x(node) = x'(node) + 1
                          y(node) = y'(node) + 1
         Case SOUTHEAST:  x(node) = x'(node) + 1
                          y(node) = y'(node) – 1
         Case SOUTHWEST: x(node) =  x'(node) – 1
                          y(node) = y'(node) – 1
    End switch
    GetConnected (node, location, A, A')
End Procedure
```

Let $H(A)$ denote a list of indexes of type H amino acids in the chain and let $N(h)$ denote the indexes of nodes adjacent to node h. Then the energy E of a conformation A in the HP lattice model can be efficiently computed in $O(|C|)$ time using the following procedure.

```
Procedure EvaluateEnergy (A)
Begin
    For h∈H(A)
        For j = N(h)
            If j∈H(A) and h > j then
                E = E – 1
            Endif
        End j
    End h
Return E
End Procedure
```

Procedures for the Filter-and-Fan Neighborhood

The procedures needed for the construction of the filter-and-fan neighborhood tree are as follows. Let $V(k)=\{V_1(k), V_2(k), \ldots, V_{\eta_1}(k)\}$ ($|V_i(k)|=\eta_2$) denote the set of η moves evaluated at the level k from which the η_1 best moves for the (new) collection $M(k)=\{m_{1k}, m_{2k}, \ldots, m_{\eta_1 k}\}$ are selected ($M(k)\subset V(k)$, $k>0$), thus building solutions $X(k+1)$.

EvaluatePullMoveList ($X_i(k)$, $V_i(k)$): evaluates the energy value for all pull moves in $V_i(k)$ associated with the i-th best conformation in $X(k)$ and retains the resultant energy value in the corresponding data member of the PullMove structure. The standard aspiration criterion is used, which overrides the move tabu status when it improves the best conformation found so far.

SortPullMoves ($V(k)$, η): sorts the η valid pull moves in $V(k)$ in ascending order of the associated energy values.

The following procedure is use to set up the first level of the filter-and-fan tree.

```
Procedure InitializeTree (A)
Begin
    DetectAllPullMoves (A, V(k))
    EvaluatePullMoveList (A, V(k))
    SortPullMoves (V(k), η2)
    For i = 1 to η1
        Xi = A
        ExecutePullMove (Xi, mi1)
    End
End Procedure
```

Further levels of the tree are created as follows. Define A^* as the current best conformation associated with the lowest energy value E^*. Consider also the following basic procedures.

AppendSets ($V(k)$, $V_i(k)$, η_2): appends the set $V_i(k)$ of η_2 moves to the set $V(k)$.

UpdateTabuList (T_i): updates the tabu list T_i.

The following procedure is used to generate L levels of a filter-and-fan neighborhood tree.

```
Procedure GenerateTree (X, η1, η2, L)
Begin
    While k < L
        For i = 1 to η1
            DetectAllPullMoves (Xi(k), Vi(k))
            EvaluatePullMoveList (Xi(k), Vi(k))
            SortPullMoves (Vi(k), | Vi(k) |)
            AppendSets (V(k), Vi(k))
        End
        SortPullMoves (V(k), η)
        For i = 1 to η1
            ExecutePullMove (Xi(k), mik)
            UpdateTabuList (Ti)
            E = EvaluateEnergy (Xi(k))
            If E < E* then:
                E* = E
                A* = Xi(k+1)
            Endif
        End
        k = k + 1
    End while
End procedure
```

Now the filter and fan procedure can simply be written as:

```
Procedure Filter-and-Fan (A, η1, η2, L)
Begin
    InitializeTree (A)
    GenerateTree (X, η1, η2, L)
End Procedure
```

The Tabu Search Procedure

The local search phase of the algorithm is carried out by a simple tabu search that utilizes only a short-term memory component. Simple pull moves are used to explore the local search under tabu restrictions embodied in a tabu list *T* and using the classical aspiration criterion explained earlier. The search is performed until encountering a sequence of NMAX iterations that fail to decrease the lowest energy value found so far. The first-improvement strategy is adopted in order to speed-up the search. To simplify the notation we assume that *m** is a global variable that identifies the best pull-move determined by the *EvaluatePullMoveList* function with associated energy *E*.

```
Procedure TabuSearch (A)
Begin
    Initialization
        A* = A
        T = Φ
        NI = 0
    While NI < NMAX
        DetectAllPullMoves (A, pullMoveList)
        EvaluatePullMoveList (A, pullMoveList)
        ExecutePullMove (A, m*)
        UpdateTabuList (T)
        If E < E* then
            A* = A
            E* = E
            NI = 0
        Endif
    End while
    Return A*
End Procedure
```

The main algorithm can now be sketched as

```
Algorithm Filter-and-Fan (A)
Begin
    TabuSearch (A)
    InitializeTree (A*)
    GenerateTree (A*, η1, η2, L)
End Algorithm
```

4. Computational Results

This section analyzes the performance of the proposed Filter-and-Fan algorithm in relation to the performances of the current best alternative approaches. Computations were carried out on the 11 most studied benchmark instances of the 2D HP Protein Folding Problem. Comparisons are established in terms of the *effectiveness* and *efficiency* of the algorithms. We first discuss the effectiveness of the algorithms based upon the results provided in Table 2. The first three columns present the identification number of each problem in the test bank and their characteristics in terms of the number of H and P amino acids in the chain. The fourth column provides the best-known energy value (E^*) for each instance and the remaining columns report the energy values of the best conformations found by the algorithms. These algorithms are: the filter and fan approach proposed here (F&F), a variation of the tabu search approach (GTabu) by Lesh, Mitzenmacher and Whitesides (2003), the genetic algorithm (GA) by Unger and Moult (1993), the evolutionary Monte Carlo (EMC) algorithm by Liang and Wong (2001), two variants of the ant colony algorithm (New ACO and ACO-HPPFP-3)[1] by Shmygelska and Hoos (2003, 2005) and the Pruned Enriched Rosenbluth Method (PERM) by Hsu et al. (2003a, 2003b). (To simplify the notation we will refer to the ACO-HPPFP-3 algorithm simply as ACO-3.) Boldfaced numbers correspond to cases where an algorithm found the best-known energy value and dashed cells indicate that the algorithm has not been tested on the corresponding problem. All results reported in this section are directly taken from Shmygelska and Hoos (2003, 2005), except for the F&F and GTabu results that originate in this study. The last rows of Table 2 provide information on the total deviation in units of energy above the best known energy values.

For an accurate comparative analysis, it is important to consider the conditions under which the different algorithms obtain their best solutions. Solutions for GA and EMC correspond to the best energy values found in 5 independent runs of the algorithms. New ACO performs between 100 and 500 runs on each instance and reports the best energy value over all runs. More specifically, the ACO algorithm performs 500 runs for chains up to 50 amino acids, 300 runs for chains containing more than 50 and up to 64 amino acids, and 100 runs for larger instances. The same run scheme is employed to the ACO-3 algorithm, except for instances larger than 64 on which only 20 runs are executed. Results for PERM are the best energy values obtained in a number of 20 to 200 statistically dependent runs. We report the best energy values found by the F&F and GTabu in a single run of these algorithms using their default parameter settings and discuss the statistical results based on 200 runs of GTabu as reported in the paper (Lesh, Mitzenmacher and Whitesides 2003).

Table 2 shows that under similar running conditions the GTabu algorithm is dominated by the F&F method and presents the worst total energy deviation across the board, except for GA and EMC that only report outcomes for a subset of the testbed, thus leaving the complete assessment by this measure inconclusive. The last two rows of Table 2 provide the deviations for the partial testbed considered by GA and EMC to allow for a direct comparison with the other algorithms on the corresponding subsets. We can see that only F&F and ACO-3 find the best known energy values for the problems tested by GA and EMC algorithms, yielding a zero unit deviation from the best know energy values. However, it should be noticed that for problem 9, the ACO-3 algorithm failed to find the corresponding best known energy in 80% of the runs;

[1] New ACO and ACO-3 are two improved versions of the original Ant Colony algorithm initially proposed in Shmygelska, Hernandez and Hoos (2002).

therefore no other algorithm matches the F&F performance on this partial test set by finding the best known energy values for the problem instances with a 100% success rate. The table shows that GTabu performs better then EMC and PERM on the problems tested by GA, but it is not as good as ACO. As for problems tested by EMC, the GTabu and PERM methods alike perform better than EMC.

In an overall analysis of the complete testbed, the GA and EMC algorithms are clearly not competitive. Even for relatively small instances these algorithms fail to find the best solutions in all 5 runs and no attempt is made by the authors of these methods to solve the larger instances. The New ACO algorithm fails to find 2 of the best solutions found by F&F and 3 of the best solutions found by PERM, in 100 runs of this ACO algorithm (as specified above), and reveals larger total energy deviation from the best values. While F&F performs better in the partial test set, the overall comparative analysis between PERM and F&F is not so straightforward. The PERM algorithm finds one best known solution more than F&F. However, PERM can fail by a significant amount for some arbitrary instances. This result is evidenced in problem 8 to which PERM is unable to find a solution of reasonable quality, providing a solution whose energy is 4 units away from the best known value while F&F finds the best known energy value for this problem. Furthermore, since PERM may be executed up to 200 times on problems for which the best known solution was not reached, the algorithm is not robust when compared with F&F, which finds its best solutions in just one run. An interesting observation stems from the statistical results provided in Lesh, Mitzenmacher and Whitesides (2003) concerning multiple runs of GTabu on some of the larger instances. After giving GTabu 200 trials with different parameter settings chosen at random, the authors report a success ratio of 40%, 12%, and 7.5% for this algorithm to find the best known energy values for problems 9, 10 and 11, respectively, which makes this probabilistic tabu search variant more effective than the two ant colony algorithm variants on this set of problems.

Problem				*Best Known*	*Best Energy Values*						
Number	*|H|*	*|P|*	*|C|*	*E**	*F&F*	*GTabu*	*GA*	*EMC*	*New ACO*	*ACO-3*	*PERM*
1	*10*	*10*	*20*	*-9*	***-9***	***-9***	***-9***	***-9***	***-9***	***-9***	***-9***
2	*10*	*14*	*24*	*-9*	***-9***	***-9***	***-9***	***-9***	***-9***	***-9***	***-9***
3	*9*	*16*	*25*	*-8*	***-8***	***-8***	***-8***	***-8***	***-8***	***-8***	***-8***
4	*16*	*20*	*36*	*-14*	***-14***	***-14***	***-14***	***-14***	***-14***	***-14***	***-14***
5	*25*	*23*	*48*	*-23*	***-23***	*-22*	*-22*	***-23***	***-23***	***-23***	***-23***
6	*24*	*26*	*50*	*-21*	***-21***	***-21***	***-21***	***-21***	***-21***	***-21***	***-21***
7	*43*	*17*	*60*	*-36*	***-36***	*-35*	*-34*	*-35*	***-36***	***-36***	***-36***
8	*42*	*22*	*64*	*-42*	***-42***	***-42***	*-37*	*-39*	***-42***	***-42***	*-38*
9	*59*	*26*	*85*	*-53*	***-53***	*-50*	*-*	*-52*	*-51*	***-53****	***-53***
10	*56*	*44*	*100*	*-48*	*-47*	*-47*	*-*	*-*	*-47*	*-47*	***-48***
11	*55*	*45*	*100*	*-50*	*-49*	*-47*	*-*	*-*	*-47*	*-49*	***-50***
Total Deviation Above Best Known				*0*	*2*	*9*	*8*	*5*	*6*	*2*	*4*
GA				*0*	*0*	*2*	*8*	*4*	*0*	*0*	*4*
EMC				*0*	*0*	*4*	*-*	*5*	*2*	*0*	*4*

* Only 20% of the runs succeeded in finding this best solution

Table 2. Comparison of algorithms' effectiveness

We now assess the relative efficiency of the algorithms for which computational times have been reported in the literature. No running times are available for the GA and EMC algorithms. For the remaining algorithms, times are in seconds on a 1GHz Pentium III computer processor, except for ACO-3 whose test runs were executed on a

2.4GHz Pentium IV. Table 3 summarizes these results, where times for ACO-3 should be multiplied by approximately a factor of 2 (as can be derived from Dongarra, 2006) to reflect the relative speed of the different computer processors. Boldfaced figures in this second table indicate excessive running times for the problems considered, identifying cases in which the associated algorithms encounter significant difficulty in finding the best known solutions.

Problem				*Running Times (seconds)*[1]						
Number	*|H|*	*|P|*	*|C|*	*F&F*	*GTabu*	*GA*	*EMC*	*New ACO*	*ACO-3*	*PERM*
1	*10*	*10*	*20*	*0*	*2*	-	-	*3.33*	*< 1*	*0.01*
2	*10*	*14*	*24*	*2*	*2*	-	-	*2.52*	*< 1*	*0.02*
3	*9*	*16*	*25*	*0.5*	*6*	-	-	*10.62*	*< 1*	***80.01***
4	*16*	*20*	*36*	*4*	*5*	-	-	*11.81*	*4*	*0.05*
5	*25*	*23*	*48*	*10*	*36*	-	-	*405.79*	*60*	***1762.69***
6	*24*	*26*	*50*	*22*	***1005***	-	-	*4952.92*	*15*	*0.48*
7	*43*	*17*	*60*	*56*	*37*	-	-	*62471.24*	*1200*	*0.52*
8	*42*	*22*	*64*	*24*	*137*	-	-	*5844.93*	*5400*	*6.07*
9	*59*	*26*	*85*	*74*	*111*	-	-	*21901.34*	*86400**	*31.95*
10	*56*	*44*	*100*	*408*	*2430*	-	-	*29707.22*	*36000*	*152.71*
11	*55*	*45*	*100*	*808*	*3050*	-	-	*10835.51*	*43200*	*19962*
Average of Running Times				*128.05*	*620.09*			*12377.02*	*15662*	*1999.68*

[1]All algorithms were run on a similar reference machine with 1GHz processor, except for ACO-3 whose times refer to runs on a 2.4GHz computer processor.

*This value corresponds to a cut-off time of 1 day for each run of the algorithm in a 2.4 GHz Pentium IV processor, which would correspond to almost 2 days if it ran on our reference machine.

Table 3. Comparison of algorithms' efficiency

F&F and GTabu report times to find their lowest energy conformation values. ACO reports expected times per run (computed over multiple statistically independent runs) required by the algorithms to find a solution of the quality specified in Table 2. As for PERM, the figures refer to the times needed for the algorithm to reach its best solution for the first time over a sequence of statistically dependent runs. (In this sense, PERM can be seen as a multi-start probabilistic algorithm where statistical information gathered in one run is carried forward to start the search in subsequent runs.)

Taking into consideration the relative speeds of the two computers, New ACO and ACO-3 reveal no significant difference in efficiency when solving relatively small problems 1 to 4. For larger problems the efficiency of these two algorithms varies with no specific pattern. ACO-3 is clearly more efficient on problems 6 and 7 while New ACO performs significantly better on problem 8 and 10. Although the New ACO and ACO-3 algorithms significantly improve the performance of their original version (Shmygelska, Hernadez and Hoos, 2002) these approaches remain extremely inefficient compared to PERM and the approaches based on tabu search such as GTabu and F&F. Both GTabu and F&F provide the smallest average of running times; however F&F reveals a more consistent time performance and is considerably more effective in finding best known solutions (as shown in Table 2). The PERM algorithm is more than 15 times slower on average than F&F and is very inefficient in solving problems 3 and 5. On these latter problems, the PERM algorithm takes more than 80 seconds and 1762 seconds, respectively, compared to approximately 1 second and 10 seconds, respectively, required by the F&F method. In addition, as reported in Shmygelska and Hoos (2003), after running for 48 hours of

wall clock time on a 1GHz Pentium III CPU machine, the PERM algorithm was sill unable to improve its best solution for problem 8 (which was of low quality). In fact, to find a solution having the best known energy for this problem, Shmygelska and Hoos determined that the most effective PERM variant needed 78 hours of wall clock time on a 2.4GHz Pentium IV CPU processor (which translates into more than 6 days on our reference machine). By contrast, the F&F algorithm only requires 24 seconds to obtain such a solution.

In sum, the combined partial and overall performance analysis indicates that the F&F approach has a more efficient and consistently reliable performance than the other state-of-the-art algorithms.

5. Conclusions

The 2D HP lattice model of the Protein Folding Problem (PFP) has brought new challenges to the optimization community. Although the problem has been extensively studied for more than a decade, to the best of our knowledge none of the attempts to create exact solution methods has succeeded in solving even the smallest instance (containing 20 amino acids) of the standard testbed considered here. Therefore, the challenge has been passed to advanced metaheuristic approaches to provide high quality solutions for problems of reasonable size. Classical genetic algorithms, probabilistic multi-start algorithms based on Monte Carlo simulations, hybrid ant colony optimization approaches and tabu search approaches have all entered the fray in an effort to find the best possible solutions to these protein folding models. It is worth noting that although advanced metaheuristics are currently capable of finding optimal and near-optimal solutions for other path-based permutation problems in graphs containing hundreds and sometimes millions of nodes, none of the metaheuristics proposed for the protein folding problem has proved capable of finding a best known solution within a reasonable time span for each of the problems in the testbed of this study, where the largest instance contains only 100 nodes. In fact, as noted earlier, the only algorithm that has found all such best known solutions required hundreds of runs and 78 hours of wall clock time just to find the best solution for a single 64-node instance (though the more robust filter-and-fan algorithm requires approximately 12 seconds on an equivalent computer to find such a solution).

The computational analysis carried out in this study clearly indicates that further research is needed to solve larger instances of the protein folding model more effectively. In pursuit of this goal, it is relevant to observe that the best results for a wide range of permutation-based combinatorial problems have been obtained by tabu search methods and by hybrids of other methods that adopt TS-related strategies for exploiting the search history. This is notably true in the case of the HP PFP problem, where both PERM and the ACO variants have incorporated a number of strategies resembling those proposed earlier in tabu search. For example, the supplementary processes of *enrichment* and *pruning* in PERM are closely related to the *vocabulary building strategy* of tabu search (e.g. Glover 1996, 1999) and also incorporate features of *Probabilistic Tabu Search* (Glover and Laguna, 1993; Glover 1989,1995). The *pool* of high-evaluation partial solutions maintained by the PERM strategies is an instance of the elite solution filtering to create reference sets of solutions as proposed in Glover (1977), which is the foundation of scatter search and is exploited in several *candidate list strategies* used in tabu search.

Similarly, in the New ACO and ACO-3 algorithms, the augmentation of ant colony ideas by introducing local search utilizing intensification and diversification strategies can be

compared to TS proposals, which embody local search intrinsically rather than utilizing a basic design that treats it as something apart. The correspondences are particularly visible in the New ACO and ACO-3 processes for exploring paths of elite ants (i.e. high quality solutions) by local search and for subjecting a larger number of ants to be subjected to local search in order to reach new regions of the solution space. Analogously, intensification and diversification as introduced in tabu search (where the intensification/diversification terminology originated) more generally take the form of decision rules to foster the inclusion of attributes of elite solutions and focus the search in regions around these solutions, joined by complementary rules to induce the search to visit regions not previously visited. (A variety of such rules from the TS literature are summarized in Glover and Laguna, 1997.)

In conclusion, local search that utilizes memory of elite solutions and their attributes (either in direct or statistical form) and that strategically drives the search into new regions plays a critical role in the performance of the leading metaheuristic algorithms for the PFP. In this paper we explore mechanisms for achieving these aspects by proposing a filter-and-fan approach making use of a simple tabu search structure. The algorithm alternates between single-path and multiple-path tabu searches using component moves provided by the so-called pull-move neighborhood, subject to short-term memory controls. The success of the algorithm in performing more efficiently and robustly than alternative state-of-the-art algorithms owes to two fundamental components: (i) the dynamic and adaptive feature of the search method in exploiting the pull-move neighborhood structure; and (ii) the interplay between the tabu search and the tree search phases that creates a strategic oscillation between intensification and diversification. We anticipate that further improvements may result by incorporating longer-term tabu search memory components to achieve higher levels of intensification, and by means of vocabulary building strategies that incorporate ejection chain methods and path-relinking.

References

Anfinsen, C. B., E. Haber, M. Sela and F. H. White (1961). "The kinetics of formation of native ribonuclease during oxidationo of the reduced polypeptide chain," *Proceedings of the National Academy of Sciences* 47(9): 1309-1314.

Backofen, R. (2001). "The protein structure prediction problem: a constraint optimization approach using a new lower bound," *Constraints* 6: 223-255.

Berman, H. M., J. Westbrook, Z. K. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov and P. E. Bourne (2000). "The protein data bank," *Nucleic Acids Research* 28(1): 235-242.

Bornberg-Bauer, E. (1997). "Chain growth algorithms for HP-type lattice proteins," *Proceedings of the First Annual International Conferences on Computational Molecular Biology* (RECOMB97), 47-55, ACM Press.

Chan, H. S. and K. A. Dill (1993). "The protein folding problem," *Physics Today* 46(2): 24-32.

Chikenji, G., M. Kiduchi and Y. Iba (1999). "Multi-self-overlap ensemble for protein folding: ground state search and thermodynamics," *Physical Review Letters* 83(9): 1886-1889.

Covell, D. G. and R. L. Jernigan (1990). "Conformation of folded proteins in restricted spaces," *Biochemistry* 29: 3287-3294.

Crescenzi, P., D. Goldman, C. Papadimitriou, A. Piccolboni and M. Yanakakis (1998). "On the complexity of protein folding," *Proceedings of the 13th Annual ACM Symposium of Theory of Computing* (STOC 98), 597-603.

Dandekar, T. and P. Argos (1994). "Folding the main chain of small proteins with genetic algorithm," *Journal of Molecular Biology* 236: 844-861.

Dill, K. A. (1985). "Theory for the folding and stability of globular proteins," *Biochemistry* 24(6): 1501-1509.

Dongarra, J.J (2006). "Performance of Various Computers Using Standard Linear Equations Software, (Linpack Benchmark Report), University of Tennessee Computer Science Technical Report, CS-89-85.

Glover, F. (1977). "Heuristics for integer programming using surrogate constraints," *Decision Sciences*, 8 (1), 156-166.

Glover, F. (1989). "Tabu search - part I," *ORSA Journal on Computing*, 1(3), 190-206.

Glover, F. (1992). "New Ejection Chain and Alternating Path Methods for Traveling Salesman Problems," *Computer Science and Operations Research*, 449-509.

Glover, F. (1995). "Tabu thresholding: improved search by nonmonotonic trajectories," *ORSA Journal on Computing*, 7(4), 426-442.

Glover, F. (1996). "Ejection Chains, Reference Structures and Alternating Path Methods for Traveling Salesman Problems," *Discrete Applied Mathematics*, 65, 223-253.

Glover, F. (1996). "Tabu search and adaptive memory programming - Advances, applications and challenges," in *Interfaces in Computer Science and Operations Research*, Barr, Helgason and Kennington (eds.), Kluwer Academic Publishers, 1-75.

Glover, F. (1998). "A template for scatter search and path relinking," *Artificial Evolution, Lecture Notes in Computer Science*, 1363. J.-K. Hao, E. Lutton, E. Ronald, M. Schoenauer and D. Snyers (eds.), Springer Verlag, 3-51.

Glover, F. (1999). "Scatter Search and Path Relinking," in *New Ideas in Optimization*, D. Corne, M. Dorigo and F. Glover (eds.), McGraw Hill, 297-316.

Glover, F. and M. Laguna (1993). "Tabu Search," chapter in *Modern Heuristic Techniques for Combinatorial Problems*, C. Reeves, ed., Blackwell Scientific Publishing, 71-140.

Glover, F. and M. Laguna (1997). *Tabu Search*, Kluwer Academic Publishers, Boston.

Grassberger, P. (1997). "Pruned-enriched Rosenbluth method: simulations of theta polymers of chain," *Physical Review*, 56(3): 3682-3693.

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

Hsu, H. P., V. Mehra, W. Nadler and P. Grassberger (2003a). "Growth algorithms for lattice heteropolymers at low temperatures," *Journal of Chemical Physics* 118(1): 444-451.

Hsu, H. P., V. Mehra, W. Nadler and P. Grassberger (2003b). "Growth-based optimization algorithm for lattice heteropolymers," *Physical Review* E 68(2): Article number: 021113.

Konig, R. and T. Dandekar (1999). "Improving genetic algorithms for protein folding simulation by systematic crossover," *BioSystems* 50: 17-25.

Krasnogor, N., D. Pelta, P. M. Lopez, P. Mocciola and E. de la Canal (1998). "Genetic algorithms for the protein folding problem: a critical review," *Proceedings of Engineering of Intelligence Systems*, ICSC Academic Press, 353-360.

Lengauer, T. (1993). "Algorithmic research problems in molecular bioinformatics," *Proceedings of the Second Israel Symposium on Theory of Computing Systems*, ISTCS 1993, Natanya, Israel, 177-192.

Lesh, N., M. Mitzenmacher and S. Whitesides (2003). "A complete and effective move set for simple protein folding," *Proceedings of the 7th Annual International Conference on Research in Computational Molecular Biology (RECOMB)*, ACM Press, NY, 188-195.

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

Ramakrishnan, R., B. Ramachandran and J. F. Pekny (1997). "A dynamic Monte Carlo algorithm for exploration of dense conformational spaces in heteropolymers," *Journal of Chemical Physics* 106(6): 2418-2424.

Rego, C. and F. Glover (2002). "Local search and metaheuristics for the travelling salesman problem," *The Travelling Salesman Problem and its Variations*. G. Gutin and A. Punnen, editors, 309-368, Kluwer Academic Publishers.

Richards, F. M. (1991). "The protein folding problem," *Scientific American* 264(1): 54-7, 60-3.

Shmygelska, A. and H.H. Hoos (2003). "An improved Ant Colony Optimization algorithm for the 2D HP protein folding problem," *Proceedings of Advances in Artificial Intelligence, AI 2003*, *Lecture Notes in Computer Science*, Springer Verlag, 400-417.

Shmygelska, A. and H.H. Hoos (2005) "An ant colony optimization algorithm for the 2D and 3D hydrophobic polar protein folding problem," *BMC Bioinformatics*, 6(1):30.

Shmygelska, A., R. Hernandez and H.H. Hoos (2002) "An Ant Colony Algorithm for the 2D HP Protein Folding Problem," *Proceedings of the 3rd workshop on Ant Algorithms*, *Lecture Notes in Computer Science*, Springer Verlag, 2463:40-52.

Skolnick, J. and A. Kolinski (1990). "Simulations of the folding of globular proteins," *Science* 250: 1121-1125.

Unger, R. and J. Moult (1993). "Genetic algorithms for protein folding simulations," *Journal of Molecular Biology* 231: 75-81.