A SCATTER SEARCH APPROACH FOR PROTEIN FOLDING PROBLEM IN 2D HP-MODEL

by

Sibel Bilge Sonuç

A Thesis Submitted to the
Graduate School of Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in

Industrial Engineering

Koç University

March 2008

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

Sibel Bilge Sonuç

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

Assoc. Prof. Ceyda Oğuz (Advisor)

Assist. Prof. Metin Türkay

Assist. Prof. Deniz Aksen

Date:

ABSTRACT

Proteins are complex molecules performing vital cellular activities. Each protein is composed of an amino acid chain. These amino acid chains are found as folded in their native state in the nature. The activity of a protein is mostly defined by its native 3 dimensional (3D) conformation. Determination of the native 3D conformation of an amino acid chain is called the *protein folding problem (PFP)*. On the other hand, determination of the native state of a protein is not easy. In particular, protein folding problem is proven to be NP-hard.

Determination of native states of protein molecules is vital since most of the medical and genetic studies depend on knowing these native conformations. Therefore many studies and investments are made on this topic. Researchers in various fields such as biochemistry, biophysics, computer science, operations research, bioinformatics, genetics and medical sciences are interested in solving this problem. Even, there are sometimes collaborations between different research areas to merge their techniques, knowledge and experience on this problem. One of these applications is the application of metaheuristics to *protein folding problem* in *minimum free energy conformation* with *HP-model* based on a *lattice structure*, as a cooperation of all these sciences.

This study is on developing a metaheuristic algorithm based on *scatter search* and *path relinking* for the 2D lattice based PFP utilizing HP-model according to minimum energy conformation theory. There are applications of well-known metaheuristics as well as *scatter search* to the PFP in the literature. We have introduced new moves and operators for the elements of *scatter search* for the PFP on 2D lattice HP-model. We have also proposed several application options for these element of the *scatter*

search application, as well as new measurements for the values of the solutions. We introduced and experimented on some combinations of these elements for an effective search algorithm and give the test results in this study. Although the results are not as good as the results in the literature, we have tested a wide set of heuristic elements based on *scatter search* and *path relinking*. The results of these experiments are promising for further improvement on this topic.

ÖZET

Proteinler hücre aktivitelerinden sorumlu karmaşık moleküllerdir. Her bir protein bir amino asit zincirinden meydana gelir. Bu amino asit zincirleri doğada doğal şekillerinde katlanmış halde bulunurlar. Bir proteinin aktivitesi büyük oranda doğal üç boyutlu yapısı tarafından belirlenir. Bir proteinin en az serbest enerji yapısının bulunmasına *protein katlanma problemi* denir. Öte yandan, bir proteinin doğal şeklinin belirlenmesi kolay değildir. Özellikle, protein katlanma probleminin NP-zor olduğu kanıtlanmıştır.

Tıbbi ve genetik çalışmalar bu doğal yapıların bilinmesine dayandığından dolayı protein moleküllerinin doğal şekillerinin bulunması önemlidir. Bu nedenle, bu konu üzerine bir çok çalışma ve yatırım yapılmıştır. Bilgisayar bilimi, yöneylem araştırması, bioinformatik, genetik ve tıbbi bilimler gibi birçok daldaki araştırmacılar bu problemin çüzülmesi ile ilgilidir. Hatta, farklı alanlar arasında yöntemlerin, bilginin ve deneyimin birleştirilmesi için işbirligi yapılmaktadır. Bu alanlar arası uygulamalardan biri de sezgisel üstü yöntemlerin *HP-model*indeki *en az serbest enerji yapısı* için *kafes model*inde *protein katlanma problem*ine uygulanmasıdır.

Bizim çalışmamız HP-modelini kullanan 2 boyutlu protein katlanma problemi için *dağınık arama* ve *path relinking* üzerine dayalı sezgisel üstü bir algoritmanın geliştirilmesidir. Literatürde, protein katlanma problemine sezgisel üstü yöntemlerin ve *dağınık arama*nın çok bilinen uygulamaları bulunmaktadır. Biz HP-modelini kullanan 2 boyutlu protein katlanma problemine dağınık aramanın elemanları için yeni adımlar ve işlemler tanımladık. Ayrıca, dağınık aramanın elemanları ve çözümlerin değerlendirmeleri için yeni ölçümler gibi birçok uygulama seçeneği önerdik. Bu çalışmada verimli bir arama algoritması için bu elemanların farklı eşleşmelerini tanımladık ve test

edip, test sonuçlarını bildirdik. Bu sonuçlar literatürdeki sonuçlar kadar iyi olmasa da, dağınık arama ve path relinking üzerine dayalı sezgisel üstü algoritma elemanlarının geniş bir k ümesini test ettik. Bu sayısal deneylerin ile gelecek araştırmalar için yararlı olacak sonuçlar elde ettik.

ACKNOWLEDGMENTS

I would like to thank my advisor, Ceyda Oğuz, for her guidance and support during my study and research. I would like to thank members of my thesis committee Metin Türkay and Deniz Aksen for their valuable comments, suggestions and their patience.

I should also thank my friends Pelin Armutlu, Aslıhan Kırcalı, Pınar Kahraman, Zehra Bilgintürk-Yalçın, Seda Tepe, Dilek Günneç, Kenan Arifoğlu, Uğur Kaplan and Fadime Üney-Yüksektepe. I am grateful for their support and encouragement.

Finally and foremost, I cannot thank enough my mother. She has never lost her belief and faith in me and has been the greatest source of support. I could not have achieved what I have now without her.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

PFP	Protein Folding Problem
SS	Scatter Search
GA	Genetic Algorithm
TS	Tabu Search
ACO	Ant Colony Optimization
RefSet	Reference Set

Chapter 1

INTRODUCTION

1.1 Problem Definition

1.1.1 Protein Folding Problem

Proteins are complex molecules made of amino acid chains. They are constructive components of a cell and also, they are responsible for vital activities in the cells. These activities require them to interact with other proteins in their environment such as involving in some biological reactions and transportation. The proteins communicate with their environment via a specialized part of their structure, which is called the *active part* of the protein. The shape of the active part of a protein depends on its 3 dimensional (3D) conformation. This is why finding 3D conformation of proteins in human cells have important impacts on medicine as well as other medical and genetic sciences. Thus, the *protein folding problem (PFP)* arises with the need for determination of the conformations (i.e. *native states*) of the proteins in the nature, especially the proteins in the human body.

1.1.2 The structure of a protein

The sequence on an amino chain chain corresponding to a protein is called its *primary structure*. Figure 1.1 represents primary structure of one amino acid chain. However, in its native state an amino acid chain is found as folded in a unique 3D conformation. This conformation is called the *tertiary structure* of the protein. The researches show that the *secondary structure* (2D conformation) defines the local folds and motifs of the protein which are called helices, sheets, loops, etc. The combination of these local

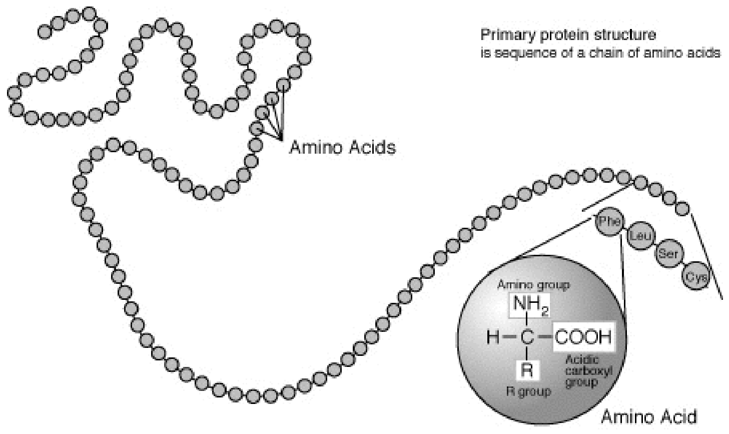

Figure 1.1: Primary structure of a protein [29]

motifs in the 3D environment defines the tertiary structure of a protein. Hence, tertiary structure of an amino acid chain depends on its secondary structure. Moreover, the secondary structure of a protein also depends on its primary structure [9, 10]. Therefore, the model used in this study to predict conformations of proteins is based on their primary structure (*HP-model)*.

Some proteins may consist of more than one amino acid chain. In this case, 3D conformations of these chains come together to form the *quaternary structure* of the protein. That is, quaternary structure of a protein is simply combination of tertiary structures of separate amino acids chains of a protein. Figure 1.2 shows the relationship between four basic structures of a protein.

Our study is focused on prediction of 2 dimensional (2D) conformation (secondary structure) of a given amino acid chain. There are two main reasons for focusing on 2D model, instead of 3D model. First, secondary structure of a protein gives us important clues about its tertiary structure, and thus native state of the protein. Second, the secondary structure is computationally easier to solve compared to the tertiary structure.

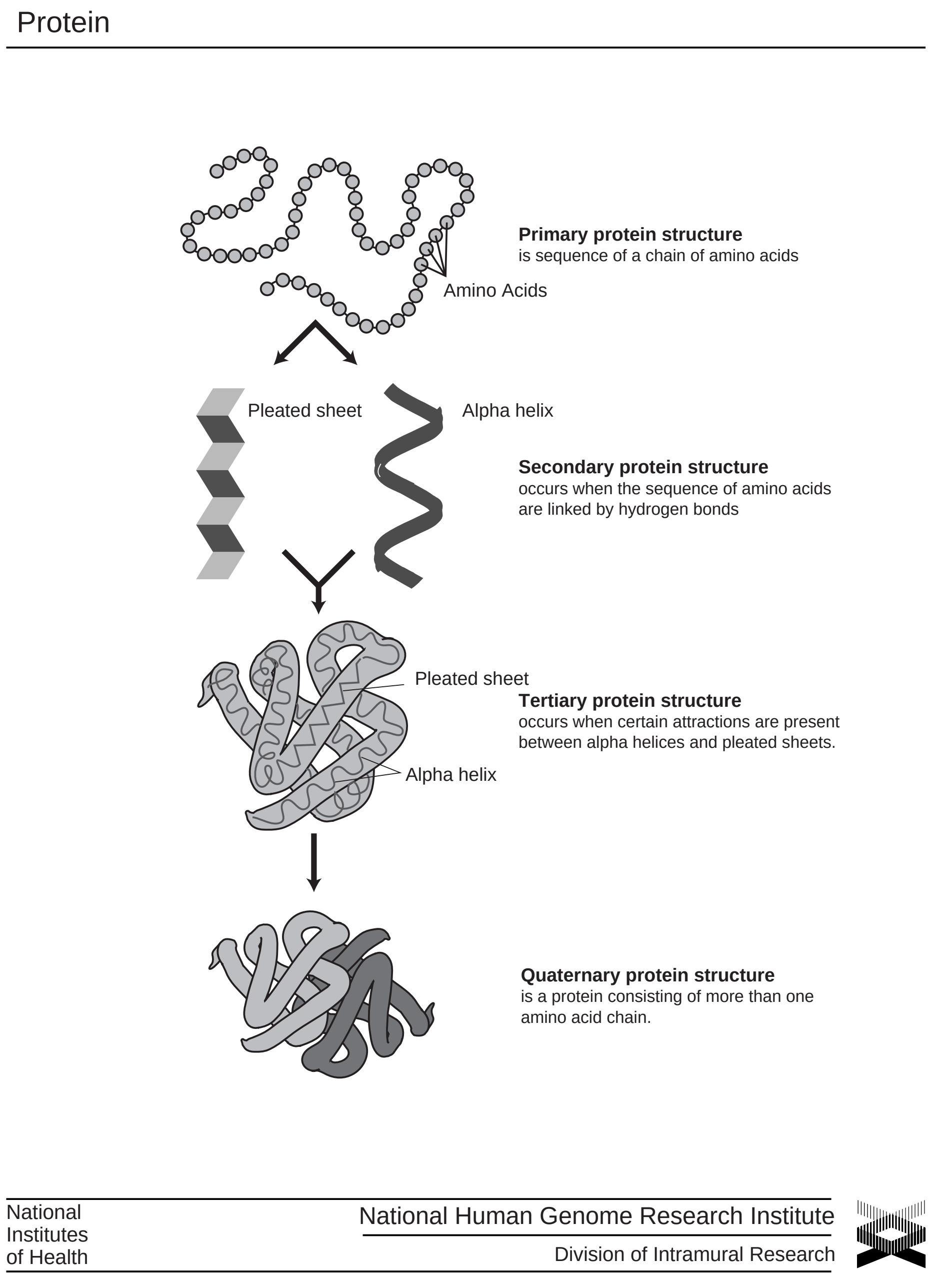
Protein
Primary protein structure
is sequence of a chain of amino acids
Amino Acids
Pleated sheet
Alpha helix
Secondary protein structure
occurs when the sequence of amino acids
are linked by hydrogen bonds
Pleated sheet
Tertiary protein structure
occurs when certain attractions are present
between alpha helices and pleated sheets.
Alpha helix
Quaternary protein structure
is a protein consisting of more than one
amino acid chain.
National
Institutes
of Health
National Human Genome Research Institute
Division of Intramural Research

Figure 1.2: Basic structures of protein conformations [16]

1.1.3 Minimum Energy Conformation

The amino acid chain of a protein folds to form a favorable and stable 3D structure. The *minimum energy conformation* theory states that molecules are in the most stable form in nature when they have the least amount of free energy. The folded protein structure is stabilized with intramolecular (e.g. *covalent*) bonds, connecting the non-consecutive amino acids on the chain. Thus, intramolecular bonds are the key factor keeping a folded molecule in its most stable form. Therefore, minimizing the free energy in a protein molecule is equivalent to maximizing the number of its intramolecular bonds. In this study, our approach is based on the idea of maximizing the number of amino acid pairs in contact which have the properties to create an intramolecular bond together.

1.1.4 The HP Model

There are 20 types of amino acids found in the nature. They all have different characteristics such as size, charge and polarity. In terms of polarity, we can classify all 20 amino acids in two groups: *polar (P)* and *apolar (H)*. Table 1.1 shows the classification of amino acids according to their polarity. Polar amino acids can form bonds with water molecules which are also polar molecules. Thus, polar amino acids tend to be closer to water molecules in the environment. For this reason, they are also called *hydrophilic* amino acids. Consequently, apolar molecules are repelled by the water molecules and tend to move towards inner parts of the protein conformation. These amino acids are called *hydrophobic* amino acids. Therefore, the polarity of amino acids play significant role in the resulting conformation of the protein.

Therefore in this study, instead of considering 20 different types of amino acids, we use a simplified model based on the polarity of the amino acids. *HP-model* proposed by Dill [8] is one of the simplest models used for the PFP. It is based on the polarity the amino acids constructing the primary structure of the protein. In HP-model, the basic idea is that type H amino acids are located at inner parts of the 3D conformation and create bonds (*HH bonds*) within each other to keep their relevant locations.

Table 1.1: Classification of amino acids according to their polarity [29]

Amino acid name	3-letter symbol	1-letter symbol	Polarity
Alanine	Ala	A	apolar
Arginine	Arg	R	polar
Asparagine	Asn	N	polar
Aspartic acid	Asp	D	polar
Cysteine	Cys	C	polar
Glutamic acid	Glu	E	polar
Glutamine	Gln	Q	polar
Glycine	Gly	G	apolar
Histidine	His	H	polar
Isoleucine	Ile	I	apolar
Leucine	Leu	L	apolar
Lysine	Lys	K	polar
Methionine	Met	M	apolar
Phenylalanine	Phe	F	apolar
Proline	Pro	P	apolar
Serine	Ser	S	polar
Threonine	Thr	T	polar
Tryptophan	Trp	W	apolar
Tyrosine	Tyr	Y	apolar
Valine	Val	V	apolar

These HH bonds are the key intramolecular bonds keeping the structure folded. With combination of HP-model and minimum energy conformation theory, it is not wrong to say that the most stable and favorable conformation of a protein corresponds to the conformation with the highest number of HH bonds.

1.1.5 Lattice model

The PFP can be considered either on or off *lattice*. For the models studied off lattice, any value for angle value between three consecutive amino acids is possible. On the other hand, only orthogonal ($\pm 90^o$) or linear (180^o) angle values are allowed. Moreover, on lattice models assume that each amino acid is equal sized and shaped and also, each pair of consecutive amino acids are equidistant. On a square lattice based model, consecutive amino acids must occupy adjacent lattice nodes. Therefore, any two amino acids which are even number of amino acids apart can never be adjacent on the lattice. Consequently, they cannot create intramolecular bonds in between even they qualify. However, this is not the case in nature. For this reason, different types of lattice models (e.g. triangle) may be applied to differ the angle values on amino acid chain. However, to reduce the effort in application, square lattice models are popular in literature to estimate the conformations of proteins.

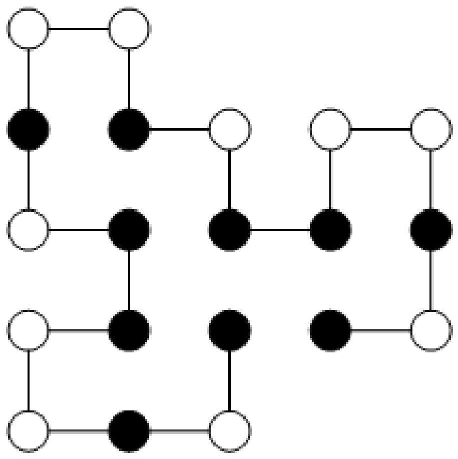

Figure 1.3: Optimal conformation for problem HP20

In Figure 1.3, the optimal 2D conformation of problem HP20 (one of the benchmark problems) on lattice model based on minimum free energy theory is given. In the figure, black beads and white beads represent type H amino acids and type P amino acids, respectively. The lines between pairs of beads represent the primary structure of the protein. That is, the bead sequence (primary structure) on the line represents the amino acid chain defining the protein. Each pair of beads on the chain is at unit distance with respect to each other. Any two H (black) beads with unit distance in between but not adjacent on the amino acid chain (i.e. with no singe line connecting these two beads) make a HH bond.

1.2 Contributions of the thesis

In this thesis, we studied application of scatter search based heuristic algorithm on lattice based PFP in 2D HP-model according to minimum energy conformation theory. We have tested different methods for five elements of scatter search in combination with path relinking. For *diversification generator*, we exploited the local motifs for the initial solution generation including random solution generation. For *improvement method*, the main focus is again on the local motifs. The method tries to embed the favorable local motifs into the solutions whenever the objective function gets to improve as a result of the move. We have also tested how dense the improvement must be applied to trial solutions visited by the search procedure. For the diversity measurement, we have introduced a new diversity measurement which also works on the mirror image solutions. For the *solution combination method*, we have exploited a *path relinking* approach with both single *guiding solutions* and two *guiding solutions* with their objective function values as their weights. We have tested parameters for our algorithm including *RefSet size*, improvement density, *elite solution* inclusion, *trial solutions set* size and number of parents joining *solution combination method*.

We have also defined new moves for PFP on lattice model. `Turn move` is a well performing move which turns infeasible solutions to feasible solutions in finite number of moves, keeping most of the local motifs. `Push move` is based on the *pull move* used in the literature. The difference of `push move` is its ability to perform on the infeasible solutions. In addition, it is more loyal to the local motifs already constructed compared to *pull move*. `Shift move` is a slight modification of `push move` that can perform on both feasible and infeasible solutions and still keeping the local motifs if they are not on the active beads of the move.

1.3 Outline of the thesis

In Chapter 2, we give a literature review on both the PFP and heuristics developed to solve it, including *scatter search* (SS) and its implementations. First, we start with the literature review of the PFP. We give a survey for the main metaheuristic approaches to the PFP in the minimum free energy theory based HP-model on lattice. Then, we introduce scatter search and *path relinking* (PR).

In Chapter 3, we give details about SS and PR methods. We explain five elements of SS in detail with examples. We also explain the relation between SS and PR structures. A general template for the SS elements is also provided in Chapter 3.

Chapter 4 presents solution generation moves and neighborhood structures for the 2D PFP in HP-model used in the literature. We explain these moves in detail in Section 4.1 and give their similarities and differences with respect to each other. In Section 4.2, we introduce the move operators we have proposed for this study: `push move`, `shift move` and `turn move`.

In Chapter 5, we introduce our algorithm and the proposed elements for SS structure. We explain the components of the algorithm proposed and tested for each of the element. The settings for the computational experiments are also explained in this chapter. In addition, we provide pseudo codes for some of the proposed algorithmic elements as well as the general structure of the whole search algorithm we have applied.

Chapter 6 gives the results and tables for the computational experiments. Finally, in Chapter 7, we give give our concluding remarks.

Chapter 2

LITERATURE REVIEW

2.1 Protein Folding Problem

There are various methods applied to the PFP in HP-model in the literature. Some of these methods are exact methods, based on biological and chemical knowledge and involving high-tech equipment, such as X-ray and Nuclear Magnetic Resonance (NMR). However, these techniques are experimental techniques that require too much time, effort and resources. This is caused by the difficulty of the problem and large number of possible conformations, i.e.,m size of the solution space. A typical solution for a given PFP is represented by a sequence of angles. In addition, HP-model is based on the lattice model, that is the angles defining a conformation (or a solution to the PFP) are either 180^o or $\pm 90^o$. Since the number of possible solutions for a problem of size N (where N is the number of amino acids on the primary structure) is 3^N, the solution space is very large. This leads to the conclusion that the PFP is NP-hard [27]. Therefore, many heuristic approaches are used in the literature to solve the PFP.

Pardalos et. al. [21] study a *tabu search* (TS) approach by adopting a lattice based model. On a lattice, a conformation corresponds to a Hamiltonian path with N number of vertices, where N is the size of the problem. Local search move is similar to a 2-Opt move for the well-known *traveling salesman problem* (TSP). The edges between two pairs of consecutive nodes are deleted and new edges are introduced to keep the Hamiltonian path properties. Therefore, the pairs must be topological neighbors on the lattice to each other. Their implementation is on the 3D PFP.

Blazewicz et al. [2] also study a TS approach for the PFP in HP-model. They use a simple neighborhood structure which is constructed by a change in one or more consecutive angles. Thus, angles on each conformation define the attributes of the corresponding solution. Moreover, the change in the values of angles define the tabu status of that solution. Blazewicz et al. [2] were able to find the optimal solutions to the same benchmark problems as we used up to problem size 64 with in some reasonable CPU times. For larger problems, they found better solutions than previous studies. Thus, their results are much better than [4, 17, 26, 28]. Although we also defined the neighborhood structure based on change in consecutive angles, we defined the moves to choose the set of angles to be changed under a set of rules. The rules we have defined for the move operator are based on the local conformations rather than random selection.

Unger and Moult [28] study a *genetic algorithm* (GA) application. They use *1-point crossover* neighborhood structure where at the *crossover point* the angle may get any possible value in $\{-1, 0, 1\}$. They picked the angle value resulting in a *valid conformation.* The crossover point is also randomly selected in their application. However, for some pairs of solutions, none of the possible crossover points with any one of the three angle choices can result in a feasible solution. In such cases, the selected pair is ignored and another pair is selected instead. The offspring is accepted only if its fitness value is larger than the average of the fitness values of its parents. They could not find optimal solutions for problems with size larger than 50 and also for the problem HP48. However, their results outperformed the results found by *Monte Carlo simulation.* The *solution combination method* we have adopted is similar to the 1-point crossover of Unger and Moult [28], with only a slight difference. In our application, we have considered every crossover point possible to produce a new solution, and among the ones generated we pick the solution with the best fitness value. In addition, the infeasible solutions are not eliminated in our application. Instead, we *mutate* them so that the outputs become feasible with the least change on the angle values.

Another GA application to the PFP is by Cox et al. [4]. Like Unger and Moult [28], they use 1-point crossover neighborhood structure. As a slight difference in their application of crossover operator, they copy parents as offsprings if no valid conformation can be found by the combination of this pair instead of picking another pair. They have reported the results for problems with up to size 50. They were able to find the optimal for the problem HP48, which Unger and Moult [28] could not find. However, Cox. et al. make more than twice as many number of evaluations as Unger and Moult. In their study, Cox et al [4]. also define new moves and they use them as mutation operators in GA application. These moves are defined so that they classify moves according to whether they change angles or locations of beads on the conformation. Some of them have resemblance to other moves in the literature. These moves are studied in detail in Section 4.1.

Another heuristic application to the PFP is *ant colony optimization* (ACO) application by Shmygelska and Hoos [26]. In their application of ACO, each ant is responsible for producing a folded conformation. The starting folding point is chosen uniformly on the sequence for each ant. After the solutions are constructed, pheromone values are updated and then local search is applied. In local search, mutation and *Monte Carlo* moves are used, such as *crankshaft move* also defined in Cox et al. [4]. Shmygelska and Hoos [26] could solve all problems up to HP85 to optimality. Thus their results are much better than the results of previous studies.

Lesh at el. [17] propose so-called *pull move* for HP-model. The main property of the pull move is that it generates new solutions from a given feasible solution ensuring feasibility of the output. Because of this property of the pull move and also since it is not hard to implement, it is adopted in other approaches in the literature. For example, the pull move is also used by Rego et al. [22] to define neighborhoods of SS application to 2D lattice model of the PFP. They exploit *filter-and-fan* neighborhood structure using pull moves as component moves. In addition, they classify pull moves in a simpler manner similar to the way Cox et al. [4] does to classify mutation operators. Their search procedure also detects all valid pull moves available and then the *pullMoveList* generated in this way is used to define solutions in the filter-and-fan neighborhood. We do not determine all the moves available in advance in our research but we apply all the possible moves when they are available, to enhance the use of local conformations. The results of Rego et. al. [22] outperform those of Shmygelska and Hoos [26] (ACO application) not only in terms of best objective function values but also in CPU times.

2.2 Scatter Search and Path Relinking

Scatter search (SS) is a metaheuristic method proposed by Fred Glover [12]. It generally embeds the adaptive memory idea of TS into evolutionary search concept. SS is an evolutionary algorithm working on a set of solutions. It constructs new solutions by strategically combining other solutions and searching the local neighborhood of the constructed solutions for elite solutions. The pairs to be combined are selected by a predetermined rule due to the characteristics of the problem. This step is generally a random process in other heuristic methods. Moreover, local search is applied to all possible solutions even if the constructed solutions themselves may not be qualified to join the rest of the process. The consequence of this idea is that not only the constructed solutions are eligible for the search process but also those in their close neighborhoods. By this approach, we are able to search through a wider range of the solution space as well as we are more in control of intensification. This solution combination methodology tries to capture high quality information found separately in the solutions. In addition to all these, SS also handles diversification in a much different way than most other heuristic methods. It exploits a diversification measure accepted for the whole search process, and calculates diversity values of each solution according to other solutions concurrently available at that period. Thus, it enables to search process to choose the most diverse set of solutions to use for the rest of the process. In [13, 14, 19, 20], the main principles of SS are explained.

Scatter search and path relinking also have applications on other optimization problems. There are SS applications on scheduling problems such as project scheduling [6] and project scheduling with resource availability costs [30]. Some other applications of scatter search are on the vehicle routing problem [23], the arc routing problem [3], the knapsack problem [5], software testing [24], warehouse layout problem [31], artificial neural networks [11], clustering problem [25], p-median problem [7] and subset selection problem [18]. There are also some studies for application of the SS to continuous optimization problems [15]. Parallelization strategies of SS are also studied for various problems [1].

2.3 Contributions of the thesis

Our approach to PFP in lattice based 2D HP-model is both combination of the some elements in the literature and new ideas we have developed utilizing the characteristics of the problem. We see that there are various kinds of heuristics approaches in the literature to our problem as well as scatter search. One of these heuristics GA and crossover operator for solution combination method. We have embedded the idea of the crossover operator into the path relinking to extend the space covering the solutions which can be generated by a set of parent solutions. However, for the use of offspring, our approach differentiates from the approach used in GA applications that include mutation as well as improvement. Our aim is to keep the inherited attributes of the offspring as much as possible while improving it by adding the attributes which it misses out. This methodology helps us to increase intensification while eliminating the randomness in the search process. Therefore, given the first initial set of solutions, our algorithm always give the same output solutions and thus eliminating chance effect in the search process.

We have defined new moves to enhance the trial solutions which the search procedure visits. Since our algorithmic approach allows infeasible solutions to occur during search process and even let then involve in the procedure, we need fast but effective moves giving feasible and improved results, but able to work both infeasible and feasible solutions. Therefore, by exploiting the mutation moves defined in the literature, we have defined `turn move`, `push move` and `shift move`. The advantage of these moves is that they can be used in any heuristic approach although they have been defined for characteristics needs of *scatter search.*

Chapter 3

SCATTER SEARCH AND PATH RELINKING

3.1 Elements of Scatter Search

Basically, *scatter search* has five main defined components [13, 14, 19, 20]:

1. Diversification Generator
2. Improvement Method
3. Reference Set Update Method
4. Subset Generation Method
5. Solution Combination Method

Diversification Generator: It is a tool to initialize the search procedure. Since the main idea of SS is to combine diverse solutions so that the offspring inherits the most eligible characteristics of its parents, the initial set of solutions must consist of diverse solutions. In addition, the solutions must be of as high quality as possible. Although it is subject to alter due to the requirements of the problem of interest, it generally starts with one solution constructed either randomly or by a construction heuristic. This solution is called the *seed solution*. Then, by using the same diversity measure described above, it generates new diverse solutions by some *diversification operator* [12, 13]. This operator may generate infeasible solutions. Infeasibility at initial step is not a problem for this application because the solutions for the next step, only the feasible offsprings will be transfered even though the parent solutions might be infeasible. On the other hand, using a *diversification operator* is not mandatory. Depending on the structure of the problem and the solution representation defined for the algorithm, all initial solutions may be generated independently. In this case, we must still ensure that they obey the diversity requirement under the diversification measure defined.

Improvement Method: This is the procedure where local search procedure is defined. Although it is not mandatory to use *improvement method*, it is highly recommended for most of the problems. It handles intensification during the process, and from the SS point of view, it is also used to search the neighborhood of the newly generated offsprings. It may either be used to improve the solution quality of the initial solution set generated via *diversification generator*, or it may be used to search the neighborhood of the solutions generated via *solution combination method*.

Reference Set: The SS procedure works on the *reference set (RefSet)*. *RefSet* is a set of elite solutions of size b. The following three components (*reference set update method, subset generation method*) and *solution combination method* of SS performs on the elements of *RefSet*. Thus, it is the core of the whole SS process. The *RefSet* is generally composed of two main parts: *RefSet1* (of size b_1) and *RefSet2* (of size b_2) so that $b_1 + b_2 = b$. *RefSet1* contains the high quality solutions in terms of the fitness value (or the objective function value), where *RefSet2* contains diverse solutions in terms of the diversification measure defined. In some problems, a *RefSet3* (of size b_3) may be used to introduce other qualities of measurement for the solutions. In this case, we would have $b_1 + b_2 + b_3 = b$

Reference Set Update Method: This is the procedure by which the elements of the *RefSet* are chosen. The selection of elements in *RefSet1* is generally straight forward: We pick b_1 solutions with the highest fitness value from the *Pool*. The *Pool* is the set of all solutions we deal with at that iteration of the algorithm. Initially, *Pool* is equivalent to the set of initial solutions generated by the *diversification generator*, and then may be improved by *improvement method*. In the later steps of the search procedure, *Pool* is the set of all offsprings generated combined with the set of parent solutions (which is *RefSet* in terms of SS). The essential idea while selecting the elements in *RefSet2* is to compute the diversities of the solutions according to the solutions already selected for *RefSet1*. Then the b_2 solutions from the set *Pool*$\setminus$*RefSet1* with the highest diversity values are selected as the elements of *RefSet2*.

Subset Generation Method: This method declares the rules to select the pairs of parents that will be combined later to generate new solutions. The subsets may be of size 2 or more. If the size is of 2, then we have a pair of parents which is similar to the idea in GA. However, SS lets us combine 3 or more solutions to generate an offspring. In this case, the common choice is to use weighted combinations of the parent solutions. The weights can be any measure of the parent solutions as well as their objective function value.

Solution Combination Method: This is the method that is used to generate new solutions using the existing solutions. The procedure for *solution combination* must be decided concurrently with the *subset generation method*. This is because the *solution combination method* must be applicable to the subset of the size returned by the *subset generation method*. Moreover, *solution combination method* must be constructed so that it enhances the good characteristics of the solutions and these characteristic are inherited to the offspring in an efficient way. These characteristics may be either eliteness in terms of fitness value or diversity value.

The general diagram of scatter search is given in Figure 3.1. The first step is the *diversification generation* where the *initial solution set (Pool)* of size P is generated.

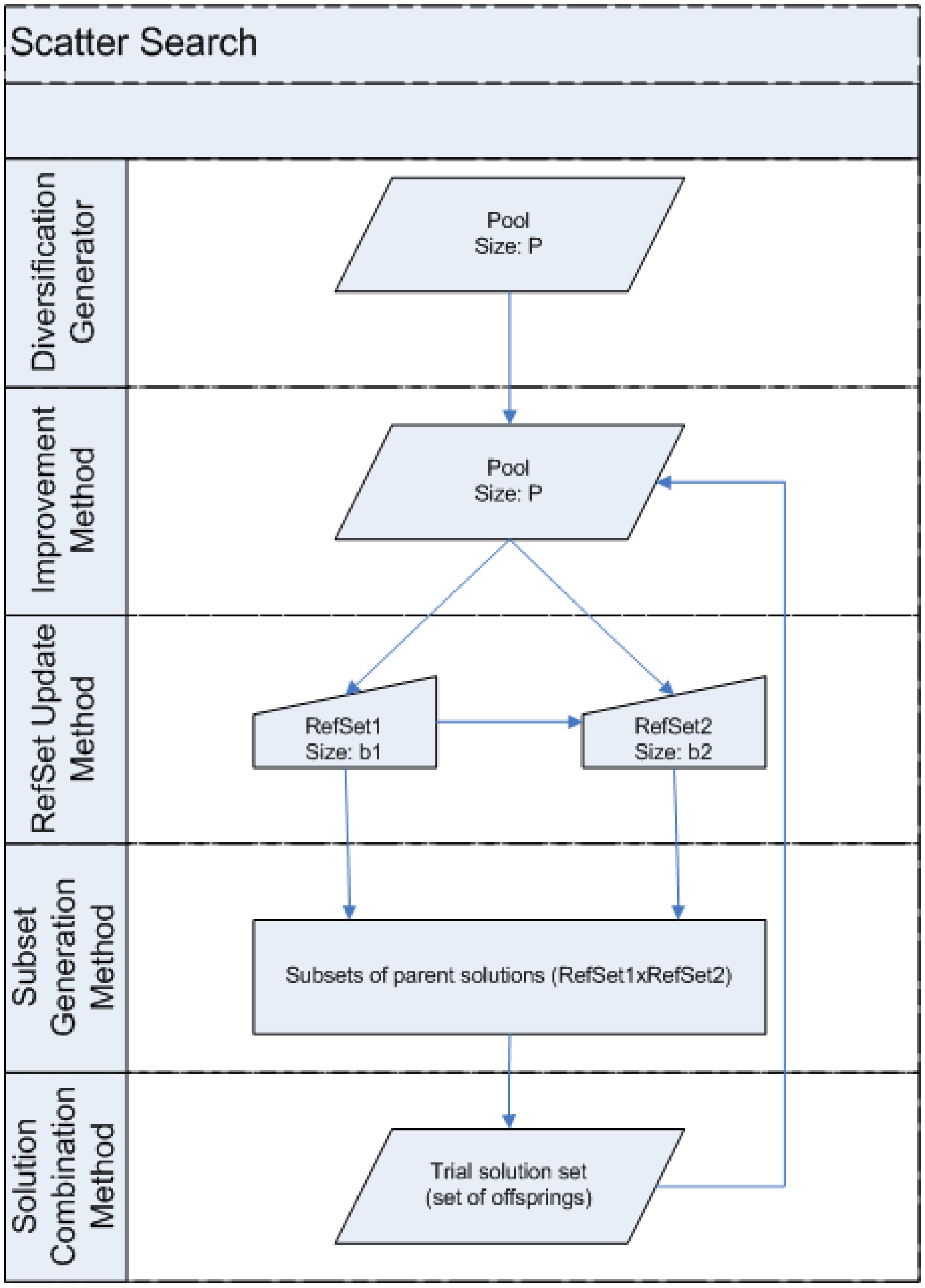
Scatter Search
Diversification Generator
Improvement Method
RefSet Update Method
Subset Generation Method
Solution Combination Method
Pool Size: P
Pool Size: P
RefSet1 Size: b1
RefSet2 Size: b2
Subsets of parent solutions (RefSet1xRefSet2)
Trial solution set (set of offsprings)

Figure 3.1: Scatter search diagram

Pool is a set of diverse solution with respect to a `diversification measurement` of choice. This `diversification measurement` depends on the problem as well as on the algorithm. In the original structure of SS proposed by Fred Glover [12, 13], a *seed solution* is used to generate other solutions in *Pool*. The *seed solution* may be constructed either randomly or by a construction heuristic.

To increase the chance of hitting the *global optimum* with as least computational effort as possible, we may choose to start the algorithm with high quality solutions in terms of their objective function value. For this purpose, we may improve the solutions in *Pool* via an *improvement method*. This is the second step depicted in Figure 3.1. At this step, *improvement method* is applied to all solutions in *Pool*.

The third step shown in Figure 3.1 is the *RefSet update method*. At this step, the algorithm searches in the *Pool* for high quality solutions in terms of both their objective function value and their diversity value. These high quality solutions will be used in the next steps of SS. The high quality solutions in terms of objective function values become the elements of *RefSet1*. Similarly, the solutions picked because of their high diversity values compose the set *RefSet2*. The elements of these two sets play different roles in the general structure of SS.

The elements of *RefSet1* and *RefSet2* are paired as parents in the *subset generation method*. This method is the fourth step in Figure 3.1. In this step, the solution groups that will be combined are determined as subsets of *RefSet*. In our application, these subsets contain at least one solution from *RefSet1* and one solution from *RefSet2*.

The last step in Figure 3.1 is the *solution combination method*. In this step, the parents determined by the *subset generation method* in the previous step are combined to generate a new set of solutions, called *trial solutions set*. The *trial solutions set* acts as the new *Pool* for the next iteration of SS, starting from the second step (*improvement method*).

The procedure depicted in Figure 3.1 continues until the stopping condition is met. The stopping requirement for SS is divergence of *RefSet*. Therefore, the procedure continues until no change (i.e. improvement) is observed in *RefSet1* for $MaxIter$

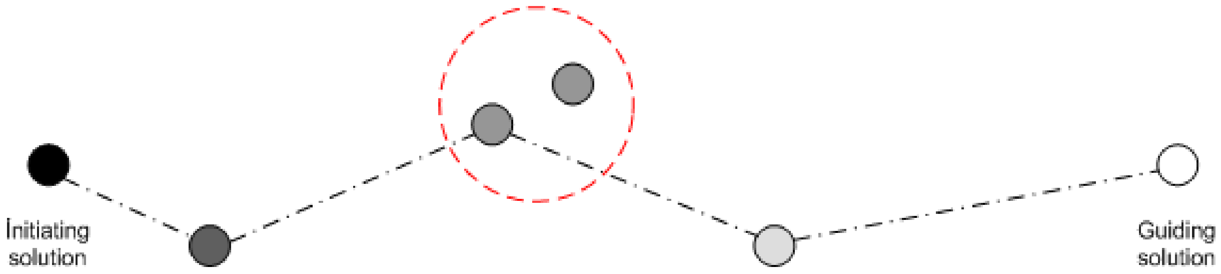

Figure 3.2: path relinking

number of iterations. $MaxIter$ is a positive integer large enough to ensure that the algorithm has diverged to a solution and no more improvement on the high quality solutions can be observed.

3.2 Path Relinking

Path relinking (PR) is a generalization of SS. It also works on a set of solutions to generate a new solution (or solution set) that has inherited attributes from the parent set of solutions. On the other hand, PR has some unique characteristics. In particular, we name solutions as either a *guiding solution* or a *initiating solution*. As the names clearly state, the *initiating solution* is the initiating point of the procedure, and the procedure is guided by the *guiding solution(s)*. *Path relinking* procedure is depicted in Figure 3.2. Initially, we have the *initiating solution*, which is depicted as a black bead in Figure 3.2. The *guiding solution* is shown as a white bead. The attributes of the *initiating solution* which are not shared by any of the *guiding solutions* form the attribute set *from-attribute*, and the attributes of the *guiding solutions* which the *initiating solution* does not have form the attribute set *to-attribute* [14]. Although we may have as many *guiding solutions* as we like, the number of *initiating solutions* is limited to one. This is because the PR procedure introduces only one solution at each step.

At the first step of PR, we drop one of the attributes in the *from-attribute* set from the *initiating solution* and add corresponding attribute from the *to-attribute* set. Thus, we get a new solution, which is the first point on the *path* that *path relinking* procedure is generating. We continue dropping the attributes in the *from-attribute* set one-by-one and adding the corresponding attribute from the *to-attribute* set until we cover all the attributes in these sets. Hence, we generate a path of solutions from *initiating solution* toward *guiding solutions*. The solutions on this path are depicted as gray beads in Figure 3.2 since these solutions have a mixture of the attributes from both solutions. The solutions closer to the *initiating solution* (black bead) are dark gray and the solutions closer to the *guiding solution* are light gray. In some cases, two guiding solutions may have different candidates in the *to-attribute* set to be added to the new solution. Then, we may apply *weighted-combination* for these attributes where the weights are determined by some fitness measurement of choice.

It is clear that the *initiating solution* of PR corresponds to *RefSet1* of SS, and the only difference is that the number of *initiating solutions* is limited to one. Similarly, the equivalent of *guiding solutions* of PR is *RefSet2* in SS. Moreover, *solution combination method* is outlined in PR by *to-attribute* vs. *from-attribute* sets. Therefore, PR may be integrated with SS and may be used as the *solution combination method* in SS. PR also allows us to search the neighborhood of the *path* generated. Thus, PR also has the *improvement method* component of SS. In addition to these, *reference set update method* can be applied similarly. We pick the best solution in terms of fitness value of the problem from the neighborhood defined by the path generated by PR, and set of *guiding solutions* are updated based on a diversity measure of choice. On the other hand, we do not need *reference set update method* or *improvement method* when we use PR as the *solution combination method* in SS. In this case, the *path* generated or the best solution on path is the output of PR procedure.

In our study, we used SS approach to guide *path relinking* that is working in a similar way to the crossover operators used in the literature [4, 28]. Since path relinking also lets us explore the neighborhood of the solution generated on the *path*, we have defined new moves based on the *pull move* idea to guide the local search.

In Figure 3.2, the neighborhood of one of the solutions on the path is shown with a circle, and a solution in this neighborhood is also shown. That is, the output of *path relinking* method may be a solution not on the path, but in the neighborhood of the same path.

Chapter 4

MOVES TO GENERATE NEW VALID CONFORMATIONS

Before defining the moves and the neighborhood structure, we must define some terms related to a *valid* conformation for some protein sequence. A *valid* conformation corresponds to a feasible solution for our problem. A feasible solution means a *self-avoiding* protein structure. In a *self-avoiding* conformation, no two different beads share the same location on the plane (or in the space if we consider a 3D model). Thus, while defining the moves, we must ensure that the resulting conformation is self-avoiding. That is, the neighborhood of a solution must consist of feasible solutions. In addition, we must keep *connectivity* along the moves. If two consecutive beads on the sequence are at unit distance from each other also on the conformation, then we say that it is *connected*.

4.1 Moves defined in the literature

In this study, our aim was to define the moves that would generate new solutions such that each move would introduce a favorable local motif to the conformation. The move that we define may involve displacement of more than one bead when it is necessary. It is based on the *pull move* defined by Lesh et al [17]. *Pull moves* are defined to create feasible solutions when a feasible solution is given as input. It simply searches for possible locations at unit distance to the conformation, to relocate a bead of choice. Generally this choice is random. However, since *pull move* requires the chosen bead to be replaced to a diagonal point according to its current location, the selection is made among the beads with such availability. After the relocation of the chosen bead, one or more beads may be required to be *pulled* so that the conformation is still connected, thus resulting in a new feasible solution. *Pull moves* are also used in [22] as component moves for the filter-and-fan neighborhood. In [22], the authors also identify three types of *pull moves* that we may encounter: *1. filling move, 2. single-pull move,* and *3. multiple-pull move.* In *filling move*, we do not need to relocate beads other than the chosen bead. In *single-pull move*, if bead j is the chosen one, then we also need to replace either bead $j+1$ or bead $j-1$ depending on the direction of the replacement. As it can be understood from its name, *multiple-pull move* requires replacement of multiple beads to keep connectivity. Figure 4.1 depicts examples of these moves.

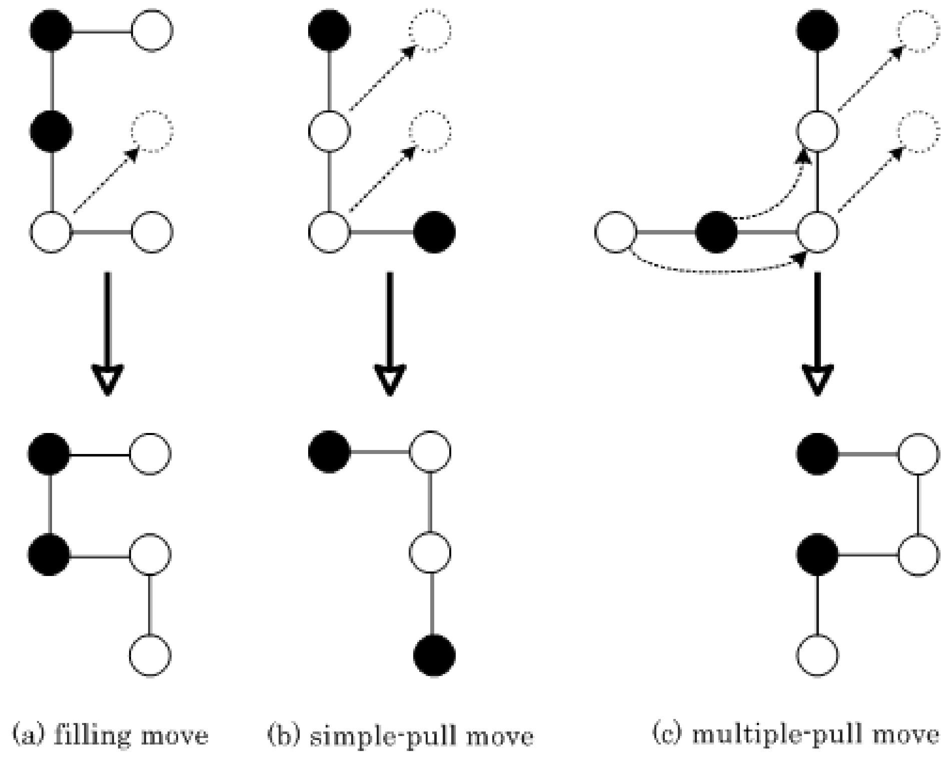

Figure 4.1: three types of pull move

Multiple-pull move resembles a mutation operator defined in [4], which is called *snake move* by the authors. In *snake move*, the beads are pulled along the trace of the chain, each one replacing a previous bead on the sequence, imitating a snake's walk on the ground. Other mutation operators defined in [4] are *in-plane move*, *out-of-plane move*, *crank-shaft move* and *kink* move. An *in-plane move* is simply mutation of an angle on the sequence. In *out-of-plane move*, a bead is picked at random on the sequence and then from this selected bead to any of the two ends, the sub-chain is rotated along either x-axis or y-axis. *Crank-shaft* move is similar to *out-of-plane move*, but only a portion of the chain is rotated. That is, this portion may be a local motif of the conformation. *Kink move* is exactly the same as *filling move* defined in [22].

In [4, 28], GA approach is used for the PFP and 1-point crossover neighborhood structure is used. However, the crossover point is selected so that the offspring is valid. If no feasible offsprings can be found, then in [4] parents are returned as offsprings, and in [28] another parent pair is picked. In either case, in the worst case scenario, both algorithms may converge too early since we may not be able to update the generation. If the offspring becomes *non-self-avoiding* after the mutation operator, *backtracking* is performed. In *backtracking* procedure, starting form the collision point, we change some angles by going one bead back at each step until we get a *self-avoiding* conformation. However, in the worst case, we may need to *backtrack* all the way back, and thus reconstructing almost the whole conformation. In this case, inheritance from the parents will be severely lost.

TS strategy is applied in [2] where generating a new solution from a given solution is done by changing one or more consecutive angles on the sequence so that the output is feasible. It is a fast procedure since the move operator does not have requirements other than feasibility. Besides, generation of move is random. However, random moves cannot be used if we want to add some conditions on the output of the move operator.

4.2 Moves defined in this thesis

In our study, we exploited the favorable motifs found in the optimal conformations. These motifs correspond to subsequences of size 4 *HPPH* and *PHHP*, and the motif is a square where each of the 4 beads is at a corner. However, all moves in the literature are defined on feasible solutions to generate new feasible solutions. However, sometimes it is hard to find a properly defined move on some conformations. In our application, where we exploit motifs in the neighborhood definition, it is even harder. Moreover, sometimes there is no feasible move that constructs a desired motif on a given conformation. Therefore, we need to define new moves that will also work in case of infeasibility.

4.2.1 Push move

When we need to move a bead to a certain location which is already occupied, we might also relocate that bead occupying that location, namely we may *push* that bead away. So, the subsequence including the bead that need to be *pushed* away is *pushed.* In figure 4.2, the *push move* is depicted.

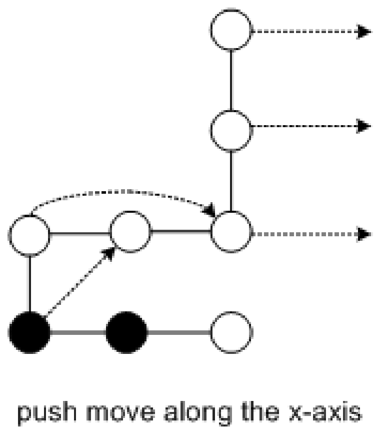

Figure 4.2: push move

The direction of the *push move* can be either along the x-axis or y-axis. The direction satisfying *self-avoiding* conformation requirements may be selected. However, since a segment of the chain is pushed away from the other segment, there is a very small possibility that both directions will lead to an invalid conformation. A collision may occur if one of the segments is surrounded by the other segment.

4.2.2 Turn move

As explained before, there may occur some infeasible solutions in the search process because the moves that we have defined are based on the favorable motifs. Hence, we need an efficient procedure to turn infeasible solutions into feasible solutions. In the literature, *backtracking* is used as explained in Section 4.1. Though, it may omit inherited attributes. Thus, we define *turn moves* (Figure 4.3).

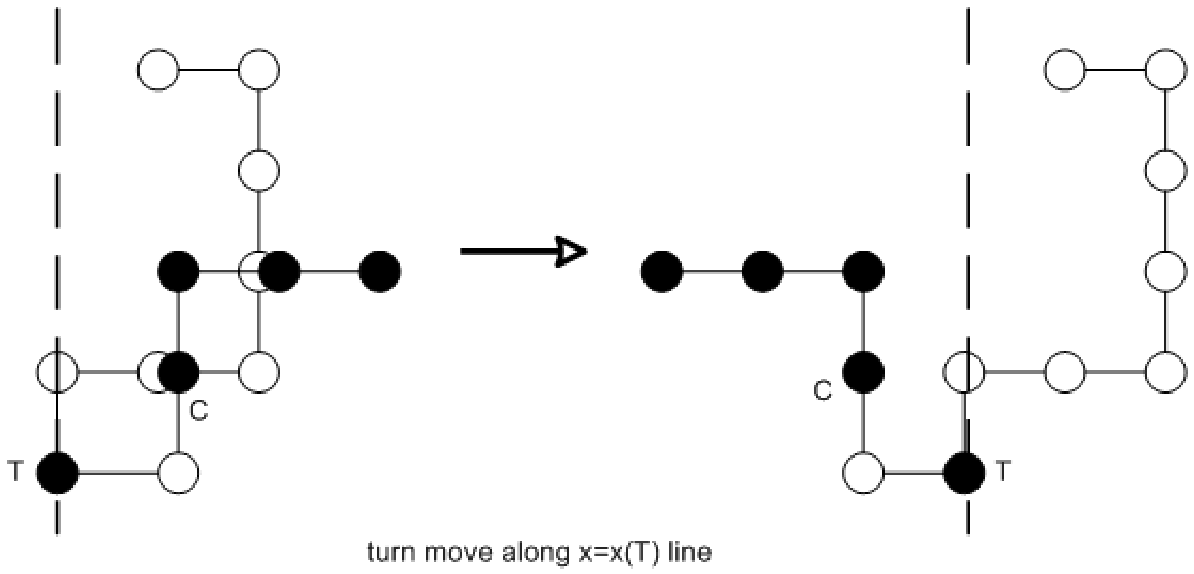

Figure 4.3: turn move

First, the *turn move* detects the point of collision, and then it carries away the whole subsection causing it. The carrying procedure of *turn move* resembles *out-of-plane move* defined in [4]. After detection of the collision point, *turn move* detects the bead closest to the collision point and has the highest (or lowest) x-value and the highest (or lowest) y-value. This bead is defined as the *turn point* and the subsequence will be rotated along the axis defined by this point. So, the *turn point* is on the outmost line on the sub-conformation, and there is a low probability that there will be collision after a *turn move.* If we got a *non-self-avoiding* conformation again, after a *turn move*, we may repeat the procedure on the new conformation until we get a valid one. It is obvious that we need to apply the *turn move* at most N times, where N is the problem size (size of the sequence). On the other hand, we have observed that we are able to get a *self-avoiding* conformation after 2 or 3 moves, depending on the problem size. It is clearly much faster than *backtracking* since *backtracking* requires determining the available positions for beads and reconstructing the conformation, whereas *turn move* just multiplies the angle subsequence to be rotated by -1. Moreover, *turn move* keeps most of the inherited attributes unchanged.

4.2.3 Shift move

When we apply *pull move* to a given conformation, and if we need to apply a *multiple-pull move*, then because of the relocation of some beads previously formed motifs will be deformed. This is not preferred since the precious steps of the improvement method will be undone. On the other hand, *push move* is defined for situations where infeasibility is the main issue. Moreover, *push move* relocates the subsegment not defining the local motif but the subsegment that is in conjunction with the local motif that we are interested in to build. Then we need to define a move where it does not disturb the motifs on the other parts of the conformation while it is also defined for *self-avoiding walks*.

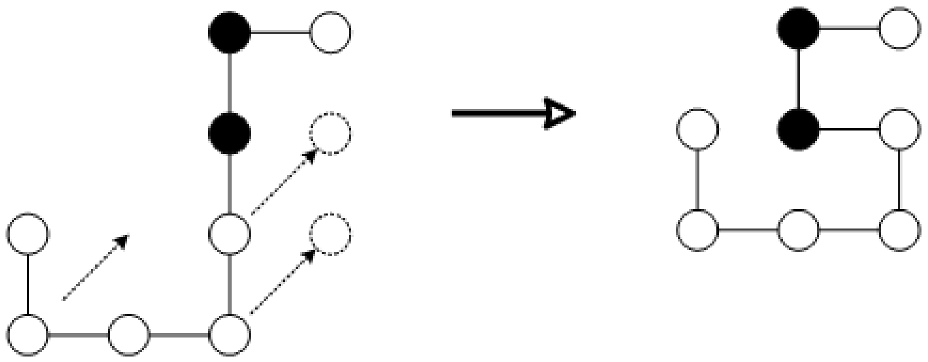
Figure 4.4: shift move

We can integrate the conditions required for *pull move* and the relocation idea of *push move.* That is, we may detect the beads that defined *pull move*, but instead of *sneak move* like pulling the remaining chain, we may shift that subconformation without disturbing the angles of the corresponding subsequences. Figure 4.4 depicts an example of *shift move.* The *shift* values for the x and y coordinates are calculated for only the bead chosen for the move, and the conformation corresponding the subsequences up to this chosen bead is *shifted* by the same values. Hence, not only a local motif is formed, but also the local motifs formed by the corresponding subsequence are saved.

Chapter 5

THE ALGORITHM AND COMPUTATIONAL EXPERIMENTS

5.1 Diversification Generator: Initial Solution Set

5.1.1 Random solution generation

Scatter search requires diverse solutions to initialize the algorithm [20, 14, 13]. The most basic condition for the *initial solution set* is that it should contain diverse solutions so that the algorithm would not converge to a local optima. It is easy to generate random diverse solutions for the PFP since each angle value is ± 1 or 0 and each angle is independent of the others. Thus, we can randomly generate each angle independently. On the other hand, for metaheuristic algorithms, it is always recommended to start with some high quality solutions. Hence, we want the initial solution set to be composed of both high quality solutions and also diverse solutions. These conditions are required since these are also the characteristics that we want *RefSet* to carry for good performance of the search procedure. The characteristics of RefSet are explained in more detail in Section 3.1.

The pseudo code for random solution generation is given in Figure 5.1. This procedure gives a random solution of size N, since a conformation of size N is defined by $N * 2$ angles. The procedure is repeated for each solution to be generated. In our application, we also kept track of each three consecutive angles and made sure that an angle value of 1 or -1 will not be repeated three times, consecutively. This is because in this case, conjunction among beads and thus infeasibility will occur. This requirement is satisfied by changing the third angle value to value of opposite sign (i.e. $1 \leftrightarrow -1$) whenever there occurs such repetition of angle values.

```
for $i \leftarrow 0$ to $N - 2$
    $r \leftarrow (3 * rand(\ )) \ / \ (RAND\_MAX + 1)$
    if $r < 1$
        then $a(i) \leftarrow 1$
    else
        if $r < 2$
            then $a(i) \leftarrow 0$
        else
            then $a(i) \leftarrow -1$
```

Figure 5.1: Diversification Generator: random solution generation

5.1.2 Elite solution generation

The high quality solutions are generated by introducing favorable motifs to the angle sequence whenever there is a corresponding subsequence of amino acid types by `elite solution generation` method. The two basic favorable motifs are $HPPH$ and $PHHP$. The angle sequence for these motifs generally is either $\ldots 1\ 1 \ldots$ or $\ldots -1\ -1 \ldots$. The occurance of these favorable motifs and the corresponding 2D shapes are depicted in Figure 5.1.2 on the optimal conformation of HP20 problem. The motifs are marked with semi-lines.

The `elite solution generation` procedure locates all the motif sequences on a given HP sequence. Then, starting from the first motif located, the angle values corresponding to the current motif are set to 1. Then angle sequence of the constructed subconformation is multiplied by -1 so that we get the mirror image of it. Then, the next motif is added to both conformations with angles value equal to 1. After the addition of the next motif to all the subconformations, the angle sequences of these conformations are again multiplied by -1. If there is space in between motifs, then these angles values are simply assigned to 0, not to increase the possibility of

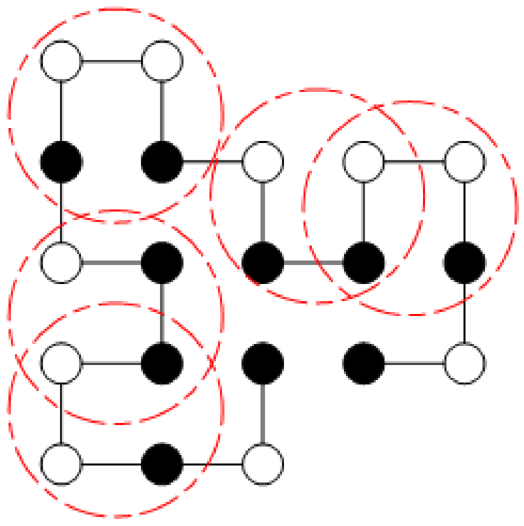

Figure 5.2: Favorable motifs on an optimal conformation

conjunction among beads. This procedure is repeated until all motifs are inserted and all angle values are defined. Hence, we get a set of solutions where all possible conformations and combinations of motifs are considered. For example, for a solution with 5 favorable motifs, there are $2^5 = 32$ such solutions. However, only the *self-avoiding* conformation will be eligible for the *initial solution set*. In our approach, we have added all such constructed solutions directly to *RefSet1* to have a representation of motifs in the initial *Refset*.

There is an example of *elite solution generation* in Figure 5.3 for an artificial problem of size 14. There are 12 angles to determine for this problem, since the first and the last beads do not form an angle. There are three favorable motifs for this problem in the example given in Figure 5.3.

The feasible elite solutions generated are directly passed to *RefSet1* for the first iteration of SS procedure. The remaining slots (if there is any) in *RefSet1* and all of the solutions in *RefSet2* is filled by *RefSet update method*. This is because, the procedure behaves as if the *initial solution set* is actually the `initial trial solution set` (i.e. *Pool*).

Example: H P P H H P H P H H P ($N = 14$)

We do not have an angle value for the first and the last beads.

We put 1's for the first two angles (to construct the first motif: HPPH):

1 1 - - - - - - - , then we multiply all by -1:

-1-1 - - - - - - -

We put 1's for the second motif, PHHP to all solutions:

1 1 1 1 - - - - -

-1-1 1 1 - - - - -

Then we multiply all by -1:

-1-1-1-1 - - - - -

1 1-1-1 - - - - -

We do not have any local motifs for the next three angles (PHP):

1 1 1 1 0 0 0 - -

-1-1 1 1 0 0 0 - -

-1-1-1-1 0 0 0 - -

1 1-1-1 0 0 0 - -

We put 1's for the last motif, PHHP:

1 1 1 1 0 0 0 1 1

-1-1 1 1 0 0 0 1 1

-1-1-1-1 0 0 0 1 1

1 1-1-1 0 0 0 1 1

Then we multiply all by -1:

-1-1-1-1 0 0 0-1-1

1 1-1-1 0 0 0-1-1

1 1 1 1 0 0 0-1-1

-1-1 1 1 0 0 0-1-1

There are four feasible solutions:

-1-1 1 1 0 0 0 1 1

1 1-1-1 0 0 0 1 1

1 1-1-1 0 0 0-1-1

-1-1 1 1 0 0 0-1-1

Figure 5.3: Elite Solution Generation: solution generation using local motifs

5.2 Subset Generation and Solution Combination Methods: All Pairs-Path Relinking

5.2.1 Path relinking using a 1-point crossover approach

We have applied *path relinking* in three different ways as *solution combination method.* Both of the *path relinking* applications we have applied are similar to *crossover* operator of GA. Thus, the offsprings are the combinations of motifs of its parents. The first application is similar to the idea of *1-point crossover* operator of GA. In this *solution combination method*, first the *initiating* and *guiding* solutions are defined. Then starting, from the first angle of the *initiating solution*, the angles are replaced one by one with the values of the angle sequence defining *guiding solution.* In other words, the parent solutions are combined at every possible point on the sequence. Figure 5.4 depicts an example of path relinking for two given solutions. In part *a* of Figure 5.4, *initiating solution* is given in normal font and *guiding solution* is given in *italic* font. After completing the path of solutions by *path relinking* procedure, we exchange the roles of the solutions as *initiating* and *guiding solutions.* The second path created after exchange of roles is given in part *b* of Figure 5.4.

This application of *path relinking* is similarly applied by Unger and Moult [28] in GA application to the PFP, although they did not name it so. In their application, all possible crossover points were tried and the offspring with the best fitness value was picked. However, they had additional conditions to accept the offspring as a *trial solution.* First, it has to be a *self-avoiding conformation.* Second, its fitness value has to be better than the average of its parents. The purpose of first condition is clear since GA only works on feasible solutions. The purpose of the second condition is to increase intensification.

Another difference of Unger and Moults's application of *path relinking* is about selecting the pairs to be combined. In SS terms, they have adopted a different *subset generation method.* In their approach, the parents were selected randomly and a pair of parents who cannot generate a *valid* conformation is neglected and a new pair of

Example:

a)

1 0 0-1-1 0 1 1 0-1 1-1 : initiating solution

0 0 0-1-1 0 1 1 0-1 1-1

0-1 0-1-1 0 1 1 0-1 1-1

0-1 1-1-1 0 1 1 0-1 1-1

0-1 1 1-1 0 1 1 0-1 1-1

0-1 1 1-1 1 1 1 0-1 1-1

0-1 1 1-1 1 0 1 0-1 1-1

0-1 1 1-1 1 0-1 0-1 1-1

0-1 1 1-1 1 0-1-1-1 1-1

0-1 1 1-1 1 0-1-1 1 1-1

0-1 1 1-1 1 0-1-1 1 1 0 : guiding solution

b) after exchange of roles of initiating and guiding solutions:

0-1 1 1-1 1 0-1-1 1 1 0 : initiating solution

1-*1 1 1-1 1 0-1-1 1 1 0*

1 0 *1 1-1 1 0-1-1 1 1 0*

1 0 0 *1-1 1 0-1-1 1 1 0*

1 0 0-1-*1 1 0-1-1 1 1 0*

1 0 0-1-1 *1 0-1-1 1 1 0*

1 0 0-1-1 0 *0-1-1 1 1 0*

1 0 0-1-1 0 1-*1-1 1 1 0*

1 0 0-1-1 0 1 1-*1 1 1 0*

1 0 0-1-1 0 1 1 0 *1 1 0*

1 0 0-1-1 0 1 1 0-1 1 *0*

1 0 0-1-1 0 1 1 0-1 1-1 : guiding solution

Figure 5.4: Path relinking: 1-point crossover

parents is selected to replace them. This procedure continues until they get a predetermined number of offsprings to form the *trial solution set*. Conversely, Cox et. al. [4] did not compare the result of every available *crossover point*. The *crossover operator* they have adopted returned the first *self-avoiding* offspring it finds. That is, if a *crossover operation* does not return a feasible solution, then a *1-point crossover* is applied at a different point on the same parents. Otherwise, the resulting solution is returned as the offspring of these parents without trying other possible *crossover points*. Unlike Unger and Moult, if no *self-avoiding* offsprings can be found from these parents, then one of the parents is returned as the offspring.

One of the main differences of our approach from the other crossover applications is that, every pair of parents returns an offspring as a *trial solution* for the next iteration of SS procedure. Moreover, the offsprings returned are always different than both of the parents, but yet carrying their favorable characteristics. That is, the *subset generation method* that we have adopted for the first kind of *path relinking* application is *all-pairs combination*. That is, each solution in *RefSet1* is combined with every solution in *RefSet2*. In this way, we get $b1 * b2$ different combinations and thus, $b1 * b2$ new solutions. In addition, for *path relinking* if we exchange the roles of *initiating solution* and *guiding solution* that is, if we use the current *initiating solution* to guide the next *path relinking* execution and the current *guiding solution* to initiate it, then we get a completely different offspring. If we apply this exchange of roles approach to all the pairs selected by the *subset generation method*, then the number of offsprings get doubled. For our approach, we get $b1 * b2 * 2$ different trial solution as candidates for *RefSet* of the next iteration of SS procedure. In our approach, every pair of parents has two offspring they are all considered for the rest of the search procedure. If the offspring is infeasible, then by `turn move` we have defined in Section 4.2, a feasible solution is generated with few changes on the angle sequence of the offspring.

In addition to all these, there are two possible ways of choosing the candidate `trial solutions`. Figure 5.5 shows these options for choosing the `trial solutions`.

```
$offspring1 = \mathrm{PR}(parent1 \rightarrow parent2)$
$offspring2 = \mathrm{PR}(parent2 \rightarrow parent1)$
a) 2 offsprings
  trial solution set = $\{offspring1\} \cup$ trial solution set
  trial solution set = $\{offspring2\} \cup$ trial solution set
b) 1 offspring
  if obj.fnc.value(offspring1) > obj.fnc.value(offspring2)
    then trial solution set = $\{offspring1\} \cup$ trial solution set
  else trial solution set = $\{offspring2\} \cup$ trial solution set
```

Figure 5.5: Choosing trial solutions

The first way is the most obvious method where all generated offsprings are added to the *trial solution set*. This is option *a* in Figure 5.5. In this case, *trial solution set* has the most number of elements because it contains the results of all 2-pair combination possibilities. In this case, the *refset update method* has to deal with larger number of solutions. The second option is to determine a method to eliminate some of these offsprings constructed even before they enter to the *trial solution set*. This is shown in part *b* in Figure 5.5. The methodology we have adopted and found quite successful during this study is to pick the one with better objective function value among two offsprings of a pair of parents. Therefore, every pair of parent solutions will give one *trial solution* as output of *solution combination method*.

It is hard to comment on this case (whether giving the best of or both of offsprings as output) about how intensification and diversification aspects of the algorithm will be affected. For intensification, assume that two parents have the attributes of a local optima. Then both offsprings generated from these parents will also have the attributes of this local optima and thus they will have high potential to be in the neighborhood of the local optima. Then, there will be high possibility that both of these solutions will be chosen by the *RefSet update method*. This selection will result in intensification toward local minimum. On the other hand, with the same logic, eliminating one of these two offsprings may result in increase in the number of iterations of whole search procedure (since then intensification will decrease) if they were actually in the neighborhood of the global optima. It follows that allowing both offsprings to the *trial solution set* will increase intensification. This is the first consequent result about intensification. To discuss about diversification, it is obvious that we should examine the change in number of solutions. Eliminating some of the offsprings before they enter to the *trial solution set* means selection of only some eligible solutions. Thus, the solutions with only favorable attributes will be selected, and this directly implies intensification. In other words, decrease in the number of candidate solutions will also result in decrease in diversification because only some high quality solutions will be kept to enter *trial solution set*. As a result, elimination of some solution will increase intensification. This is the second result about intensification. However, this result contradicts with the previous result. Consequently, the impact of *trial solution set* construction method on intensification and diversification aspects of the algorithm highly depends on the characteristics of the solutions at that step.

5.2.2 Path relinking using a 2-point crossover approach

The second application of path relinking results in solutions in a similar way with *2-point crossover* operator of GA. A middle segment of the offspring is from the *guiding solution*, where to edges of the offspring solution is from the *initiating solution*. This middle segment is selected according to the similarity comparison of two parents. They are compared starting from the first angle until a common angle value for the same bead if found. This bead becomes the similarity point, and the middle segment is defined according to this similarity point. *Path relinking* starts from this similarity point and extends the middle segment toward two ends by adding the angle values from the *guiding solution* instead of the values from the *initiating solution*. If there is no similarity point or the similarity point is at the first bead, then *path relinking* performs as in the first application described above and acts as an *1-point crossover* operator.

At the *crossover point*, we adopt the same procedure for both applications of *path relinking*. The angle value at the *crossover point* can get any of the three possible value: ± 1 or 0. All three values are tried for any crossover point, and the solution with highest objective function value is picked. The advantage of this combination approach is that, we do not only try one solution generated by the crossover but also try its two other neighbors. Thus, we extend the solution space that the search is done. Unger and Moult also use the same approach for their *1-point crossover operator*.

Cox et. al. [4] report that *1-point crossover* performs much better than *2-point crossover*. Our computational experiments gave the same conclusion about comparison of the first type of application of *path relinking* and the second type of its application. The possible explanation for this difference is that *1-point crossover* carries local motifs from the parents to the offsprings slightly less changed compared to *2-point crossover operator*. The further details about computational results are given in Chapter 6.

5.2.3 Two guiding solutions and their weighted combination

Another *path relinking approach* we have studied works on subsets of three, where we have two *guiding solutions* from *RefSet2*. These solutions are given weights based on their normalized diversity values. The diversity values are measured according to the *initiating solution* picked from *RefSet1*. Similarly, we adopt an all-pairs resembling *subset generation method*. That is, each solution in *RefSet1* is combined with every pair of solutions from *RefSet2*. Hence, we get $b1 * C(b2, 2)$ number of subsets. As a difference from the previous two applications of *path relinking* given above, here we cannot exchange the roles of *initiating* and *guiding solutions* and we get only one offspring from each subset defined by the *subset generation method*.

Figure 5.6 shows how the path of solutions behave in the case of multiple *guiding solutions*. There is a basic difference of PR with multiple *guiding solutions* from PR

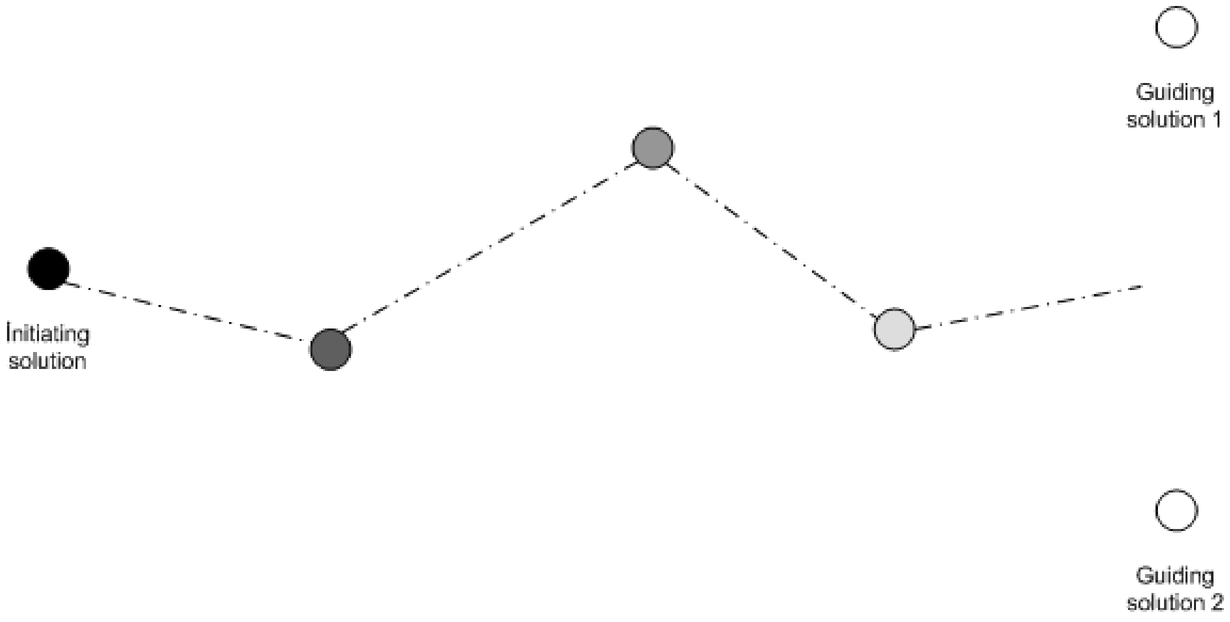

Figure 5.6: Path relinking: 2 guiding solutions

with single *guiding solution*. When we have more than one *guiding solution*, the path of solutions do not end at one of the guiding solutions. The reason is that other *guiding solutions* will cause the path to variate from any of them, so that the path will not touch any of the *guiding solutions*. It is obvious that this statement holds only for positive weight. For example, in a case with two *guiding solutions*, if the weight of one of these *guiding solutions* is zero then it will be equivalent to a single *guiding solution* case, and the solution with non-zero weight will be the only *guiding solution*.

5.3 Diversity Measurement

We have tested two different diversity measurement methods. The first one is a simple comparison method where each angle value is compared one by one. The second method defines a measurement by adding and then subtracting the solutions to/from each other.

5.3.1 Comparison measurement

When we have two solutions, we can compare these solutions by comparing each of their angle values one by one. Then the diversity values will increase as the number of angles with different values on each solution increases. However, this measurement method cannot detect the mirror images. When we have two solutions, one being the mirror image of the other, the comparison method will only give penalty for the angle values of zero. It cannot detect that they are actually the same conformation since for angle value of 1 in one of them, the corresponding angle value is -1 in the other one, and vice versa. Consequently, the diversity value measured will be nonzero although it should have been zero since they are the same conformation.

```
Example (a):
1 0 0-1-1 0 1 1 0-1 1-1 : solution 1
0-1 1 1-1 1 0-1-1 1 1 0 : solution 2
0 0 0 0 1 0 0 0 0 0 1 0 : similarity vector
similarity value = 2
Example (b):
0 1 1-1-1 0 1 1 0-1 1-1 : solution 1
0-1-1 1 1 0-1-1 0 1-1 1 : solution 2
1 0 0 0 0 1 0 0 1 0 0 0 : similarity vector
similarity value = 3
Example (c):
0 1 1-1-1 0 1 1 0-1 1-1 : solution 1
0 1 1-1-1 0 1 1 0-1 1-1 : solution 2
1 1 1 1 1 1 1 1 1 1 1 1 : similarity vector
similarity value = 12
```

Figure 5.7: Diversity measurement: comparison

In Figure 5.7, three examples for `diversity measurement` with comparison are given. In example a, two random solutions are compared and their similarity value is found as 2. This means that they are only similar at two points. In example b, two mirror images are compared and the similarity value here is 3. This is because `diversity measurement` with comparison can only detect 0 valued angles as common angles on mirror images. Thus, these mirror images are found to be different solutions although the corresponding protein structures have the same conformation in the space. In example c, two solutions with exactly same angle sequences are compared. In this case, `diversity measurement` with comparison is able to catch their similarity on all angles on the sequence.

In the algorithmic sense, introduction of mirror images may increase the diversity. However, for SS, we want *RefSet* to converge but we cannot evaluate the convergence of *RefSet* properly because of the mirror images and methodology of comparison measurement.

5.3.2 Adding and subtracting the solutions

When we have two solutions as mirror images of each other, we may add the angles on each bead of two solutions instead of comparing them. Since $0 + 0 = 0$, the zero angle values are not the problem again. However, in this case, for non-zero angles we have $-1 + 1 = 0$. Hence, when we add two mirror images, we get a sequence of zeros identifying zero diversity value.

On the other hand, when we have two solutions with exactly the same angle values, we need to subtract them from each other. Actually, this is the same as taking the mirror image of one of them by multiplying its angle values by -1 and then adding them together as described above.

Therefore, for any two solutions given to be compared, we will have two values: the sum and the difference. If we accept the minimum of these two values as their diversity measurement, we have an accurate measurement for diversity of the solutions.

Figure 5.8 shows three examples for the `diversity measurement` by adding and

Example (a):

1 0 0-1-1 0 1 1 0-1 1-1 : solution 1

0-1 1 1-1 1 0-1-1 1 1 0 : solution 2

1 1 1 0 2 1 1 0 1 0 2 1 : absolute values of sums

1 1 1 2 0 1 1 2 1 2 0 1 : absolute values of differences

diversity value = min(total sum, total difference)= min (11,13) = 11

Example (b):

0 1 1-1-1 0 1 1 0-1 1-1 : solution 1

0-1-1 1 1 0-1-1 0 1-1 1 : solution 2

0 0 0 0 0 0 0 0 0 0 0 0 : absolute values of sums

0 2 2 2 2 0 2 2 0 2 2 2 : absolute values of differences

diversity value = min(total sum, total difference)= min (0,18) = 0

Example (c):

0 1 1-1-1 0 1 1 0-1 1-1 : solution 1

0 1 1-1-1 0 1 1 0-1 1-1 : solution 2

0 2 2 2 2 0 2 2 0 2 2 2 : absolute values of sums

0 0 0 0 0 0 0 0 0 0 0 0 : absolute values of differences

diversity value = min(total sum, total difference)= min (18,0) = 0

Figure 5.8: Diversity measurement: Addition and subtraction

subtracting the solutions. In example *a*, we have two random solutions to be compared. We add each angle pair on the sequences to get a sums sequence but we keep the absolute values for each sum value. When we add the values on this sums sequence, we get a diversity value, which is 11. Then we repeat the same procedure but taking the differences of angles pairs on two solutions. Thus, we get a second diversity value, which is 13. The minimum of these two diversity values is passed to the algorithm as the real `diversity measurement` result for this two solutions ($min(11, 13) = 11$). In example *b* of Figure 5.8, we have two mirror images. We see that the addendum of the angle values results in 0 diversity value. So, we are able to catch the mirror images as the same solutions with this method of `diversity measurement`, which was not the case with the measurement by comparison. In example *c*, we have two exactly same solutions. In this case, the difference vector gives 0 `diversity measurement` value, meaning exact match. Thus, we are able to catch similar solutions with `diversity measurement` by adding and subtracting the solutions.

In particular, the second diversity measurement where we add and subtract the solutions gives better results than the comparison method in all of the experimental runs. The details of this comparison is given in Chapter 6.

5.4 Improvement method: Shift move + Turn move

The procedure of *improvement method* is based on the favorability of moves defined in this study. `Shift move` is the final state of the moves we have studied during our research study. The other moves that have been applied *pull move* introduced by Lesh et. al. [17] and *push move* defined in Section 4.2.

First of all, starting from the first bead of the problem, the procedure checks whether there is already a constructed motif. If not, then the algorithm applies the move we have chosen on this subsequence. This checking requires constant time since all we need to check is to see whether the first and the forth beads are topologically neighbors. If so, then we do not need a local improvement procedure for this motif.

If the motif is not already in the conformation, then the movement of the beads that are actually defining these motifs are similar to *pull move*. On the other hand, the movement of the subconformation is more like a *push move* approach where we can keep almost all of the motifs constructed on the conformation. Thus, computational studies are done comparing *pull move*, `push move` and `shift move`. These moves have the difference that comes from the idea keeping the connectivity of beads. They are all explained in more details in Section 4.2.

5.5 Feasibility: Backtracking vs. Turn move

For TS application, feasibility problem is not hard to handle. In TS, infeasible solutions are not even considered to be in the search space. In [2], they do not even mention about any feasibility issues, yet about how they handle it. Since they have used a very simple move operator, which is change of one or more consecutive angles, it is easy to find a move resulting in a feasible solution.

For the other application where solution combination is considered and thus infeasible solutions are inevitable, the general application if there is a *non-self avoiding* conformation is to first determine the point of conjunction, j. The conjunction point, j is the first point where a common location is shared with a bead with a lower rank value. After determination of this point, the bead j leading to the conjunction is tried to be replaced. If there is no place for bead j where we can have a both feasible and connected solution, then we also need to replace $j-1$. All conditions are the same for bead j and bead $j-1$. So we keep *backtracking* until we can finally have feasible and connected conformation. Cox et. al. [4] clearly note they have used *backtracking* procedure to handle infeasible solutions. In the worst case, we will need to *backtrack* the sequence all the way back to the first node. Then any information carried on this subsegment of the solution will be completely lost since it will be reconstructed. Not only it is very time consuming since it involves reconstruction of the whole solution in the worst case, but also it loses any inherited information on the solution.

On the other hand, although Unger and Moult [28] also studied GA, they have not adopted and methodology to handle infeasible solutions. Instead, they have decided to ignore any infeasible solutions from the search procedure. So, it is pretty much the same approach that Blazewicz et. al. [2] used for TS application. The only exception is that in GA, the solutions are generated but they are not considered as candidates for next population. However in TS, the infeasible solutions are never generated. Shmygelska and Hoos [26] also try to avoid generating infeasible solutions. They use random mutation moves the diversify the solutions, but yet they do not implement any move resulting in an infeasible solutions.

So, if we sum up, there are two basic methods in the literature to handle infeasibility: 1 - ignoring the infeasible solution, and 2 - *backtracking*. In this thesis study, we proposed a new move called `turn move` that turns infeasible solutions into feasible solutions. Its details are explained in Section 4.2. It can be easily seen that `turn move` is faster since there is no reconstruction or search for optional conformations. It only consists of taking a mirror image of a subsequence so that it will be turned over an axes defined. One of the other most important characteristics of `turn move` is that, it does not disturb any of the local motifs constructed. This is because in practice, it is just a change (180^o rotation) on only one angle. So, the inheritance from the parents after *combination method* is secured as well as we can keep the improvement we have achieved on solutions enhanced by the *improvement method*.

Consequently, we got an open question: When should be apply `turn move`? We may apply it both after *solution combination method* and *improvement method*, as well as we may only apply is after *improvement method*. In addition, we may integrate `turn move` into `shift move`. That is, after each *shift move*, if the resulting solution is infeasible `turn move` can be applied. This will not lead to unnecessary computational method because according to our observations during the experiments, after at most 2-3 `turn moves`, we get a feasible solution. That is, it is acceptable to use `turn move` after each single *shift move*.

There was another decision that we should decide on about `turn move`: the `turn point, T`. The `turn point T` may be either the first bead that cannot go any further from the conjunction point (without going back first) or the second bead that cannot go any further on the other axes value. That is, starting from the conjunction point, we keep following the chain, and x and y values on the coordinate plain for the beads. While moving along the chain, at first, these x and y values will either be always decreasing or increasing. So we keep following the chain as the change in the x and y values remains in the same direction. In this way, we got two different points: one is the bead at which x value will be changing in the reverse direction and the second bead is for the y value. Selection on either points change the axes of rotation.

However, the results indicate that the `turn point T` at the second bead that change is redirected dominates the results with `turn point T` at the first such point.

5.6 Scatter search application on the protein folding problem

The algorithmic approach we have studied in this thesis for the *protein folding problem (PFP)* is based on the general structure of *scatter search (SS)*. In Figure 5.9, we can see the pseudo code for the general structure of our algorithm.

In Figure 5.9, lines 1-8 are the initialization steps for main SS procedure. However, it also includes one of the main elements of SS: *diversification generator*. As common to the general SS, we start with *diversification generator* operator to create set of initial solutions, *Pool*. These solutions are generated completely randomly and independent of each other as described in Section 5.1. However, different from the general template of SS, we have `elite solution generator` operator to create a set `elite solutions` with some desired attribute. The desired attributes for the PFP are favorable motifs. Therefore, `elite solution generator` generates feasible solutions with favorable motifs. The details for this procedure is given in Section 5.1. Different than *Pool*, all solutions in `elite solution set` (*Elite* in Figure 5.9), directly enter to *RefSet1* for the first iteration of SS. If the number of elements in *Elite* exceeds $b1$, then we pick the first $b1$ elements in *Elite*. If the number of elements in *Elite* is less than $b1$, then the rest of the open slots in *RefSet1* are filled by *RefSet update method*.

The procedure for *RefSet update method* is basic, and we use the same idea as in the general template (Section 3.1). We simply pick best elements (in terms of objective function value) in *Pool* to enter *RefSet1* until we get $b1$ solutions in *RefSet1*. Then *RefSet2* elements are picked by the `RefSet2 update method`. This method calculates the diversity values according to the chosen `diversity measurement method` of all the elements remained in *Pool* with respect to elements in *RefSet1*. The solution with the highest diversity value enters to *RefSet2*. Then the diversity values are updated with respect to all the solutions in *RefSet* ($RefSet = RefSet1 \cup RefSet2$). We keep adding solutions to *RefSet2* with the same methodology until we get $b2$ diverse

```
1 .   $Pool$ = DiversificationGenerator();
2 .   $Elite$ = EliteSolutionGenerator("$motif\ set$");
3 .   $maxE \leftarrow$ size($Elite$);
4 .   for $i \leftarrow 0$ to min($maxE$,$b1$)
5 .       $RefSet1(i) \leftarrow Elite(i)$;
6 .   $RefSet1$ = RefSet1Update($Pool$,$b1-$min($maxE$,$b1$));
7 .   $RefSet2$ = RefSet2Update($Pool$,$RefSet1$);
8 .   $NumIter \leftarrow 0$;
9 .   do the following (steps 10-21) while $NumIter < MaxIter$:
10.       for $i \leftarrow 0$ to $b1$
11.           for $j \leftarrow 0$ to $b2$
12.               $offspring1$ = PR($RefSet1(i) \rightarrow RefSet2(j)$);
13.               $offspring2$ = PR($RefSet2(j) \rightarrow RefSet1(i)$);
14.               if obj.fnc.value($offspring1$) $>$ obj.fnc.value($offspring2$)
15.                   then $Trial \leftarrow \{offspring1\} \cup Trial$;
16.               else $Trial \leftarrow \{offspring2\} \cup Trial$;
17.       $RefSet1$ = RefSet1Update($Trial$,$b1$));
18.       $RefSet2$ = RefSet2Update($Trial$,$RefSet1$);
19.       if "there are new solutions in $RefSet1$"
20.           then $NumIter \leftarrow 0$;
21.       else $NumIter \leftarrow NumIter + 1$;
22.   OUTPUT: $RefSet1$
```

Figure 5.9: Pseudocode for SS application to the PFP

solutions in *RefSet2*.

The *subset generation method* is controlled by the commands in lines 10-11, that let the algorithms traverse each solution in *RefSet2* (by incrementing j from 0 to $b2$) for each solution in *RefSet1* (by incrementing i from 0 to $b1$). Thus, the algorithm performs an `all-pair solution combination method` on $RefSet1$x$RefSet2$. The details are explained in Section 5.2.

The lines 12-16 says that two different offsprings are generated by interchanging the roles of parent solutions as *initiating* and *guiding solutions* in PR (as *solution combination method*). However, only the best one of these two offsprings can enter *trial solution set*. As described in Section 5.2 in details, there are other variants of PR we have applied in this thesis such as adding both of the offspring to `trial solution set` and using weight combination with multiple *guiding solutions*. Moreover, we have discussed in Section 3.2 that we are allowed to search the neighborhood of the path of solutions generated. Therefore, for each solution on the path, we apply *improvement method* as explained in Section 5.4. The output solutions are feasible since *improvement method* also handles feasibility of the solutions by application of `turn move` (Section 4.2) during or after the local search moves.

Finally, *RefSet* is updated by *RefSet update method* with the solutions in *trial solution set* ($Trial$). Then *solution combination method* and *RefSet update method*(lines 10-21) are repeated until the stopping condition is met, which is divergence to a set of solutions.

In Figure 5.9, $MaxIter$ value on line 9 refers to the maximum number of iterations of SS allowed without any improvement in *RefSet1*. That is, we increment $NumIter$ value by one, whenever there is no new solution in *RefSet1*. In our approach, we do not renew *RefSet1* completely. Any solution in *RefSet1* is replaced by a new `trial solution` if and only if there is a better solution in `trial solution set`. We set $NumIter$ to 0 whenever a *trial solution* achieves to enter to *RefSet1*. This control procedure is done on lines 19-21 in Figure 5.9.

The output of the whole search procedure is *RefSet1* of the last iteration. It is also the set of solutions converged by the algorithm. All of the solutions in this output set are feasible because all solutions in *trial solution set* are feasible by construction, and elements of *RefSet* are picked from `trial solution set` by *RefSet update method* (line 17). The solutions in `trial solution set` are feasible since they are originally the offspring generated by PR procedure. `Turn moves` that handle infeasible solutions are embedded in the local search procedure (*improvement method*) for all solutions generated during PR (Section 5.4). Hence, the solutions in the output set of our algorithm are feasible.

Chapter 6

COMPUTATIONAL RESULTS

The computational experiments are done to test the different parameters and components of the proposed algorithm in Chapter 5: diversity measurement, solution combination method, `turn point`, improvement method, diversity control, and *RefSet* size. The test runs are done on nine benchmark problems found in the literature. The algorithm is coded in C++ using Visual Studio 6.0. The experimantal runs are done on KUMPEM server (Intel Xeon 3.20 GHz with 2 GB RAM) in Koc University.

6.1 Algorithmic elements

The first 8 columns in the Tables 6.1, 6.2, 6.3 and 6.4 give the parameters for the algorithm. The first column is the `problem name` depicting also the problem size. For example, HP20 is a problem of size 20 and HP24 is of size 24. Thus, HP20 is a PFP of an amino acid chain with 20 amino acids on the *primary structure*. The next value is *max_e* and it is the total number of feasible elite solutions generated by the `elite solution generation` method. By definition, it is also the maximum number of elite solutions we can use as input to the search procedure. $e1$ is the actual number of elite solutions given as input to the algorithm. It is set to $\min\{max\_e, b1\}$ as default since we want to utilize the favorable motifs. Then we have $b1$ and $b2$ on the tables, which are the sizes of *RefSet1* and *RefSet2*, respectively. $MaxIter$ is the limit for the number of consecutive iterations without any improvement. That is, the algorithm aborts when the procedure cannot improve *ResfSet1* for $MaxIter$ number of iterations. M gives the total number of favorable motifs sequences found on the *primary structure* of the HP chain. Finally, *opt* is the optimal or best known objective function value for the corresponding problem.

Table 6.1: turn: 2nd node, combination: 2 offsprings

$b1 = b2 = N/2$ **and other settings as default**								M*M (improve)		M*1 (improve)	
problem	max_e	$e1$	$b1$	$b2$	$MaxIter$	M	opt	best	CPU(sec.)[1]	best	CPU(sec.)[1]
hp20	8	8	10	10	5	9	9[28]	9	3.313	9	2.188
hp24	2	2	12	12	5	7	9[28]	7	29.630	7	9.218
hp25	2	2	13	13	5	4	8[28]	7	4.500	7	5.921
hp36	4	4	18	18	5	7	14[28]	12	106.391	12	69.657
hp48	4	4	24	24	5	9	23[26]	21	679.000	21	418.656
hp50	400	25	25	25	5	10	21[26]	17	84.297	17	155.625
hp60	16	16	30	30	5	5	35[28]	31	339.084	31	551.578
hp64	8	8	32	32	5	19	42[17]	34	9828.440	33	3317.760
hp85	2	2	43	43	5	6	53[22]	48	6750.880	48	5200.770

Tables 6.1, 6.2, 6.3 and 6.4 compare the effect of the change in the *improvement method*. The $M * M$ column gives the results when the `shift move` along the chain is repeated M times. The $M * 1$ column gives the results when for each local motif `shift move` is applied only once for each motif in the order of their appearance on the sequence. Thus, we make M `shift moves`. The difference is that for the case of $M * M$ column, after the *shift move* for the last motif is completed, we restart the procedure for the whole sequence again starting from the first motif. The advantage of repeating the procedure is that the algorithm may catch a *shift move* on a motif which it was not able to make before due to the conformation. In addition, it repairs the motifs decomposed because of the `turn moves` performed in between the repeats in case of infeasibility.

Table 6.1 depicts the results when the `turn point` is the second node as described in Section 4.2, and *solution combination method* gives both of the offsprings as outputs as elements of the *trial solution set*. We can see that $M * 1$ case of the *improvement method* performs better in terms of computation time. This is an expected result

Table 6.2: turn: 1st node, combination: 2 offsprings

$b1 = b2 = N/2$ **and other settings as default**								M*M (improve)		M*1 (improve)	
problem	max_e	$e1$	$b1$	$b2$	$MaxIter$	M	opt	best	CPU(sec.)[1]	best	CPU(sec.)[1]
hp20	8	8	10	10	5	9	9	9	20.818	9	6.953
hp24	2	2	12	12	5	7	9	7	57.859	7	22.250
hp25	2	2	13	13	5	4	8	7	41.562	7	26.187
hp36	4	4	18	18	5	7	14	12	1177.560	11	325.015
hp48	4	4	24	24	5	9	23			20	2140.030
hp50	400	25	25	25	5	10	21				
hp60	16	16	30	30	5	5	35				
hp64	8	8	32	32	5	19	42				
hp85	2	2	43	43	5	6	53				

because it performs less number of `shift moves` compared to the $M * M$ case. On the other hand, $M * M$ case finds a better result for the problem HP64.

Table 6.2 depicts the case when the *solution combination method* gives two offsprings as output and the *turn point* is the first node. When we compare Table 6.1 and Table 6.2, we can say that *turn point* at the second node performs much better both in terms of computation time and objective function value. In Table 6.2, the CPU times are extremely large compared to the Table 6.1. Moreover, it performs worse even for the medium size problems, such as HP36 and HP48.

Table 6.3 gives the results for the case when the `turn point` is at the second node and the *solution combination method* gives only the best one over two offsprings as output. If we compare Table 6.1 and Table 6.2, we see that the CPU times are almost the same for all problems except for the problem HP85. The computation times are smaller for the case when we have both of the offsprings as *trial solutions* for the next iteration of SS. However, it is not straight forward to compare both cases in terms of the objective function value. For small and medium sized problems up to HP60, the

Table 6.3: turn: 2nd node, combination: 1 offspring

$b1 = b2 = N/2$ **and other settings as default**								M*M (improve)		M*1 (improve)	
problem	max_e	$e1$	$b1$	$b2$	$MaxIter$	M	opt	best	CPU(sec.)[1]	best	CPU (sec.)[1]
hp20	8	8	10	10	5	9	9	9	3.984	9	2.031
hp24	2	2	12	12	5	7	9	7	32.359	7	9.796
hp25	2	2	13	13	5	4	8	7	6.891	7	5.547
hp36	4	4	18	18	5	7	14	13	124.172	13	74.109
hp48	4	4	24	24	5	9	23	23	675.610	21	362.656
hp50	400	25	25	25	5	10	21	18	145.515	18	145.906
hp60	16	16	30	30	5	5	35	31	561.781	31	530.641
hp64	8	8	32	32	5	19	42	33	5711.610	32	3095.420
hp85	2	2	43	43	5	6	53	45	5044.090	33	11199.200

objective function values in Table 6.2 are better. In particular, the optimal solution is found for problem HP48 with $M * M$ as the *improvement method*. But, for the two largest problems the objective function values are better in Table 6.1.

The graph in the Figure 6.1 gives the relation between the problem size N and corresponding CPU time (in seconds) to solve that problem for the results in Table 6.3 with $M * M$ case for *improvement method*. We see that there is no obvious correlation between the problem size and the time required to solve it. This is because the CPU time also depends on the HP pattern of the amino acid chain i.e., the *primary stucture* of the protein.

Table 6.4 depicts the case of `turn point` at the first node and the *solution combination method* giving one offspring as a new `trial solution`. The results in this table are worse than the other three tables, both in terms of computation time and objective function value.

In all cases, we were able to find the optimum for the smallest problem HP20. In

[1]The runs are done on KUMPEM (Koç University) server (Intel Xeon 3.20 GHz with 2 GB RAM)

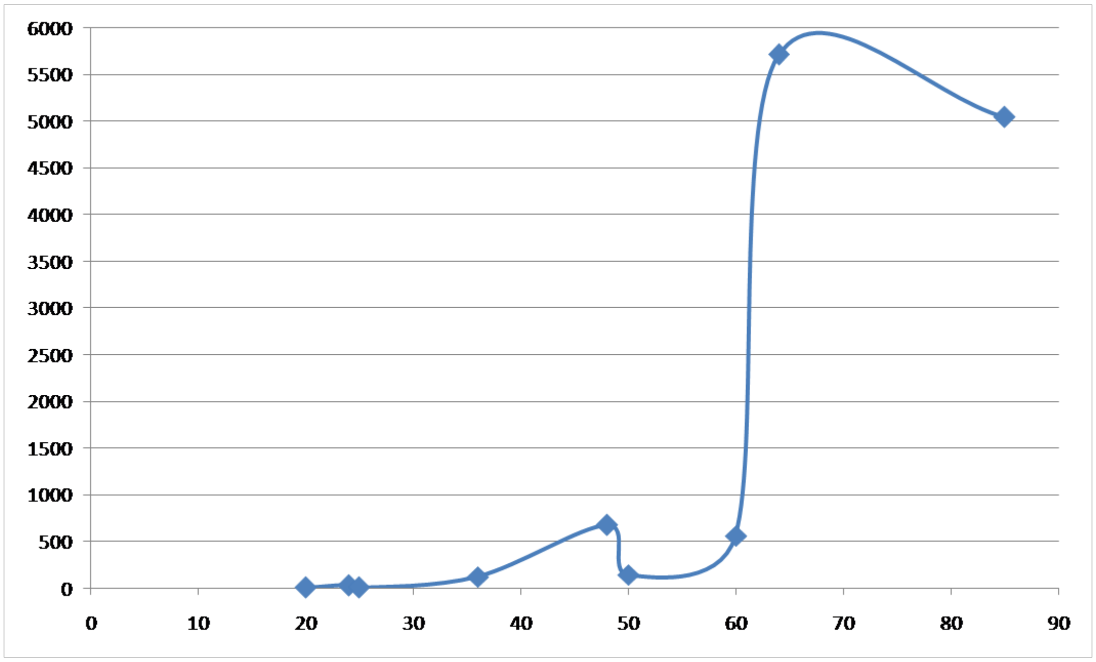

Figure 6.1: N vs CPU

Table 6.4: turn: 1st node, combination: 1 offspring

$b1 = b2 = N/2$ **and other settings as default**								M*M (improve)		M*1 (improve)	
problem	max_e	$e1$	$b1$	$b2$	$MaxIter$	M	opt	best	CPU(sec.)^1	best	CPU (sec.)^1
hp20	8	8	10	10	5	9	9	9	14.093	9	6.281
hp24	2	2	12	12	5	7	9	7	35.344	7	22.375
hp25	2	2	13	13	5	4	8	7	31.406	7	28.719
hp36	4	4	18	18	5	7	14	11	983.703	11	282.625
hp48	4	4	24	24	5	9	23	21	5765.800	16	2134.140
hp50	400	25	25	25	5	10	21				
hp60	16	16	30	30	5	5	35				
hp64	8	8	32	32	5	19	42				
hp85	2	2	43	43	5	6	53				

Table 6.5: test of diversity measures

M*M, T:2nd node, 1 offspring							RefSet: kept the same		RefSet* -1	
problem	$e1$	$b1$	$b2$	$MaxIter$	M	opt	min(+,-)	comparison	min(+,-)	comparison
hp20	8	10	10	5	9	9	9	9	9	9
hp24	2	12	12	5	7	9	7	7	7	7
hp25	2	13	13	5	4	8	7	7	7	7
hp36	4	18	18	5	7	14	13	10	13	10
hp48	4	24	24	5	9	23	23	20	23	20
hp50	25	25	25	5	10	21	18	16	18	16
hp60	16	30	30	5	5	35	31	28	31	28
hp64	8	32	32	5	19	42	33	30	33	30
hp85	2	43	43	5	6	53	45	41	45	41

addition, for all cases, the *improvement method* with $M*M$ shift moves finds solutions with better objective function values than $M*1$ moves. For the location of the `turn point`, it is clear that `turn point` on the second node performs much better than `turn point` on the first node. This may be explained by a simple observation: If we choose the second point to be the `turn point`, then the conformation is expanded on a larger plane (or lattice). This is because the second node refers to a point with a larger distance to the conjunction point. When we compare the cases for one offspring output or two offsprings output for the *solution combination method*, we may say that for problems of size larger than 60, the procedure taking both of the offsprings as `trial solutions` performs better. Since in real life cases, the protein molecules are of size larger than 60, we may also conclude that this procedure is also preferable for the real life cases.

Table 6.5 gives the results for the diversity measurement procedures. The first pair of columns giving test results which is named as "RefSet: kept the same" correspond to the case where the RefSet1 was passed to the next iteration as is. On the

Table 6.6: RefSet size (Test 1)

M*M, T:2nd node, 1 offspring, div=min(+,-)							
problem	$e1$	$b1$	$b2$	$MaxIter$	M	opt	best
hp20	8	20	20	5	9	9	9
hp24	2	20	20	5	7	9	7
hp25	2	20	20	5	4	8	7
hp36	4	20	20	5	7	14	11
hp48	4	20	20	5	9	23	20
hp50	25	20	20	5	10	21	16
hp60	16	20	20	5	5	35	28
hp64	8	20	20	5	19	42	31
hp85	2	20	20	5	6	53	42

other hand, the next pair of columns gives the case where each solution in RefSet1 is multiplied by -1 while they are passed to the next iteration whenever there is no improvement in $RefSet1$. The aim in this application was to increase the diversity since once they enter the *solution combination method*, i.e., *path relinking*, the direction of the guidance would change. However, we see that it has no effect on the solutions. When we compare two `diversity measurement methods` in Table 6.5, we see that `addition and subtracting method` either gives better or same results `comparison method` in all problems. This is probably because the `addition and subtracting method` let the algorithm avoid similarities more than the `comparison method`.

6.2 Size of the Reference Sets

Table 6.6 gives the results for the case where both $b1$ and $b2$ are set to 20 for all problems. We have chosen 20 as the test value for both $b1$ and $b2$ because this is the recommended value in [13, 20]. The other components of the algorithm are fixed to

Table 6.7: RefSet size (Test 2)

M*M, T:2nd node, 1 offspring, div=min(+,-)							
problem	$e1$	$b1$	$b2$	$MaxIter$	M	opt	best
hp20	8	30	20	5	9	9	9
hp24	2	30	20	5	7	9	7
hp25	2	30	20	5	4	8	7
hp36	4	30	20	5	7	14	12
hp48	4	30	20	5	9	23	20
hp50	25	30	20	5	10	21	18
hp60	16	30	20	5	5	35	31
hp64	8	30	20	5	19	42	33
hp85	2	30	20	5	6	53	43

the best working elements defined and tested during this study (Section 6.1). The results in this table shows that the performance of the algorithm depends on the size of the *RefSet*. For small sized problems such as HP20, HP24 and HP25, the results are not affected. On the other hand, the results for the larger problems has got worse. The reason can be connected to the decrease in the size of the *trial solution set*. Consequently, *diversification* and the area of the search space has decreased. This change led the procedure to be pulled to the local optima.

In Tables 6.12 and 6.8, we have increased the values of $b1$ and $b2$ to 30 independently. Both of the tables show better results compared to the results in Table 6.6. In addition, the results in Table 6.8 are slightly better for large problems. On the other hand, when we compare these results with the results in Section 6.1, we see that increasing both $b1$ and $b2$ to a very large number (it is $N/2$ for Table 6.3) does not improve the solutions.

The following results are with the best algorithmic elements we have found in

Table 6.8: RefSet size (Test 3)

M*M, T:2nd node, 1 offspring, div=min(+,-)							
problem	$e1$	$b1$	$b2$	$MaxIter$	M	opt	best
hp20	8	20	30	5	9	9	9
hp24	2	20	30	5	7	9	7
hp25	2	20	30	5	4	8	7
hp36	4	20	30	5	7	14	12
hp48	4	20	30	5	9	23	21
hp50	25	20	30	5	10	21	18
hp60	16	20	30	5	5	35	31
hp64	8	20	30	5	19	42	35
hp85	2	20	30	5	6	53	45

the previous computational experiments. These elements are $M * M$ moves in the *improvement method*, the `addition and subtraction method` for the `diversity measurement`, `turn move` on the second node and 1 offspring utilization from the *solution combination method*. In addition, $MaxIter$ value is set to 5.

In Tables 6.9 through 6.18, we have tested the $RefSet1$ and $RefSet2$ sizes which are $b1$ and $b2$ respectively. the values for both $b1$ and $b2$ ranges in $\{14, 20, 24, 30, 34\}$. These tests are also done with respect to inclusion of `elite solutions` in the initial *RefSet1*. In Tables 6.9, 6.10, 6.11, 6.12 and 6.13, the number of `elite solutions` included into process is equal to the maximum number of solutions generated by `elite solution generation method` but still bounded by $b1$. In Tables 6.14, 6.15, 6.16, 6.17 and 6.18 the number of `elite solutions` used in initialization is set to 0.

When we compare the results given in Tables 6.9, 6.10, 6.11, 6.12 and 6.13, we see that in Table 6.11, the results are either as good as or better than those in the other tables. Therefore, we should say that $b1 = 24$ is the best coice for $RefSet1$ size

Table 6.9: RefSet size (Test 4)

$b1 = 14$, $e1 = \min\{b1, max\_e\}$								
problem	$e1$	M	opt	$b2 = 14$	$b2 = 20$	$b2 = 24$	$b2 = 30$	$b2 = 34$
hp20	8	9	9	8	9	9	9	9
hp24	2	7	9	7	7	7	7	7
hp25	2	4	8	5	7	7	7	7
hp36	4	7	14	10	11	11	12	11
hp48	4	9	23	20	20	20	21	20
hp50	25	10	21	16	16	18	16	16
hp60	16	5	35	28	28	28	28	28
hp64	8	19	42	31	31	31	31	31
hp85	2	6	53	41	41	41	41	41

we have found. When we compare the same tables column-wise, we see that the best results are found in either columnn $b2 = 24$. However, in neither of these tables, best known solution in the literature was achived except for problems HP20 and HP48.

Among Tables 6.9, 6.10, 6.11, 6.12 and 6.13, Table 6.9 gives the worst result ehich has the smallest $RefSet1$ size value, thus smallest number of high quality solutions. However, we see that objective function value of the output solutions is not positively correlated with $b1$ value. If we look at Table 6.12, we see that most of the results are not as good as for the results with $b1 = 20$ or $b1 = 24$.

The results in Tables 6.14, 6.15, 6.16, 6.17 and 6.18 show the results with no use of `elite solutions`. We see that the results in these tables is not as good as the results in Tables 6.9, 6.10, 6.11, 6.12 and 6.13 except for 2 or 3 results for each table. For example if we compare none of the results in Table 6.16 is better than those in Table 6.11 (both with $b1 = 24$), and even some of them are worse. This is true also for the results in the other tables. Therefore, we should say that introduction of `elite`

Table 6.10: RefSet size (Test 5)

$b1 = 20,\ e1 = \min\{b1, max\_e\}$								
problem	$e1$	M	opt	$b2 = 14$	$b2 = 20$	$b2 = 24$	$b2 = 30$	$b2 = 34$
hp20	8	9	9	8	9	9	9	9
hp24	2	7	9	7	7	7	7	7
hp25	2	4	8	5	7	7	7	7
hp36	4	7	14	10	11	11	12	11
hp48	4	9	23	20	20	20	21	21
hp50	25	10	21	16	16	18	18	18
hp60	16	5	35	28	28	28	31	28
hp64	8	19	42	31	31	33	35	35
hp85	2	6	53	41	42	43	45	45

Table 6.11: RefSet size (Test 6)

$b1 = 24,\ e1 = \min\{b1, max\_e\}e$								
problem	$e1$	M	opt	$b2 = 14$	$b2 = 20$	$b2 = 24$	$b2 = 30$	$b2 = 34$
hp20	8	9	9	8	9	9	9	9
hp24	2	7	9	7	7	7	7	7
hp25	2	4	8	5	7	7	7	7
hp36	4	7	14	10	11	11	11	11
hp48	4	9	23	20	21	23	21	20
hp50	25	10	21	16	18	18	18	16
hp60	16	5	35	28	28	28	31	28
hp64	8	19	42	31	33	33	35	31
hp85	2	6	53	41	42	43	45	42

Table 6.12: RefSet size (Test 7)

$b1 = 30,\ e1 = \min\{b1, max\_e\}$								
problem	$e1$	M	opt	$b2 = 14$	$b2 = 20$	$b2 = 24$	$b2 = 30$	$b2 = 34$
hp20	8	9	9	8	9	9	9	9
hp24	2	7	9	7	7	7	7	7
hp25	2	4	8	5	7	7	7	7
hp36	4	7	14	10	12	11	11	11
hp48	4	9	23	20	20	23	20	20
hp50	25	10	21	16	18	18	18	16
hp60	16	5	35	31	31	31	31	28
hp64	8	19	42	30	33	33	33	31
hp85	2	6	53	41	43	43	43	42

Table 6.13: RefSet size (Test 8)

$b1 = 34,\ e1 = \min\{b1, max\_e\}$								
problem	$e1$	M	opt	$b2 = 14$	$b2 = 20$	$b2 = 24$	$b2 = 30$	$b2 = 34$
hp20	8	9	9	8	9	9	9	9
hp24	2	7	9	7	7	7	7	7
hp25	2	4	8	5	7	7	7	7
hp36	4	7	14	10	12	12	11	11
hp48	4	9	23	20	20	21	20	20
hp50	25	10	21	16	18	18	18	16
hp60	16	5	35	28	31	31	31	28
hp64	8	19	42	31	33	33	35	31
hp85	2	6	53	41	42	43	45	42

Table 6.14: RefSet size (Test 9)

$b1 = 14,\ e1 = 0$								
problem	$e1$	M	opt	$b2 = 14$	$b2 = 20$	$b2 = 24$	$b2 = 30$	$b2 = 34$
hp20	0	9	9	8	9	9	9	9
hp24	0	7	9	7	7	7	7	7
hp25	0	4	8	5	5	7	7	5
hp36	0	7	14	10	10	11	12	11
hp48	0	9	23	20	20	21	20	20
hp50	0	10	21	16	16	18	18	16
hp60	0	5	35	28	28	28	28	28
hp64	0	19	42	31	31	31	31	31
hp85	0	6	53	41	41	41	42	41

Table 6.15: RefSet size (Test 10)

$b1 = 20,\ e1 = 0$								
problem	$e1$	M	opt	$b2 = 14$	$b2 = 20$	$b2 = 24$	$b2 = 30$	$b2 = 34$
hp20	0	9	9	8	9	9	9	9
hp24	0	7	9	7	7	7	7	7
hp25	0	4	8	5	7	7	7	7
hp36	0	7	14	10	11	11	12	11
hp48	0	9	23	20	20	20	21	21
hp50	0	10	21	16	16	18	18	18
hp60	0	5	35	28	28	28	31	28
hp64	0	19	42	31	31	33	33	33
hp85	0	6	53	41	41	43	43	43

Table 6.16: RefSet size (Test 11)

$b1 = 24,\ e1 = 0$								
problem	$e1$	M	opt	$b2 = 14$	$b2 = 20$	$b2 = 24$	$b2 = 30$	$b2 = 34$
hp20	0	9	9	8	9	9	9	9
hp24	0	7	9	7	7	7	7	7
hp25	0	4	8	5	7	7	7	7
hp36	0	7	14	10	11	11	11	11
hp48	0	9	23	20	21	21	21	20
hp50	0	10	21	16	18	18	16	16
hp60	0	5	35	28	28	28	28	28
hp64	0	19	42	31	31	33	33	31
hp85	0	6	53	41	41	42	42	42

Table 6.17: RefSet size (Test 12)

$b1 = 30,\ e1 = 0$								
problem	$e1$	M	opt	$b2 = 14$	$b2 = 20$	$b2 = 24$	$b2 = 30$	$b2 = 34$
hp20	0	9	9	8	9	9	9	9
hp24	0	7	9	7	7	7	7	7
hp25	0	4	8	7	7	7	7	7
hp36	0	7	14	10	11	11	11	11
hp48	0	9	23	20	20	21	21	20
hp50	0	10	21	16	16	18	18	16
hp60	0	5	35	28	28	28	28	28
hp64	0	19	42	30	31	33	33	30
hp85	0	6	53	41	41	42	42	41

Table 6.18: RefSet size (Test 13)

$b1 = 34,\ e1 = 0$								
problem	$e1$	M	opt	$b2 = 14$	$b2 = 20$	$b2 = 24$	$b2 = 30$	$b2 = 34$
hp20	0	9	9	8	9	9	9	9
hp24	0	7	9	7	7	7	7	7
hp25	0	4	8	7	7	7	7	7
hp36	0	7	14	10	11	11	11	11
hp48	0	9	23	20	20	21	20	20
hp50	0	10	21	16	16	16	18	16
hp60	0	5	35	28	28	28	31	28
hp64	0	19	42	31	31	33	31	31
hp85	0	6	53	41	41	41	42	42

`solutions` to the initial set of solutions improves the output of the algorithm.

To see the effect of number of randomly generated initial solutions, we have performed experiments with different sizes for initial *Pool*. The algorithmic elements are set to the best performing elements in the other tests. The *RefSet* size is set as $b1 = 24$ and $b2 = 24$ (with $e1 = \min\{b1, max\_e\}$). The results are given in Table 6.19.

As we see in Table 6.19, $|Pool| = 100$ is sufficiently large. Moreover, since *improvement method* is applied to all the solutions in *Pool*, intensification increases too much because we get more solutions converging to a local optimum. By the characteristic of *RefSet1*, the best $b1$ solutions are picked from `improved Pool`. When we already have a convergent *Pool* to pick from, the probability to converge to a local optimum increases. A similar argument also holds for *trial solution set* and it is discussed in Section 5.2.

Table 6.19: *Pool* size

problem	$e1$	M	opt	$\|Pool\| = 100$	$\|Pool\| = 200$	$\|Pool\| = 500$	$\|Pool\| = 1000$
hp20	0	9	9	8	9	9	9
hp24	0	7	9	7	7	7	7
hp25	0	4	8	7	7	7	7
hp36	0	7	14	11	11	11	10
hp48	0	9	23	23	23	21	20
hp50	0	10	21	18	18	18	16
hp60	0	5	35	28	28	28	28
hp64	0	19	42	33	33	33	30
hp85	0	6	53	43	43	43	41

6.3 Local optimality of the solutions found

It is easy to see that even the best found solution over all tests is a local optimum. For example, if we look at Figure 6.2 depicting a solution for HP36 problem and Figure 6.3 showing an optimal conformation for the same problem, we can see that we cannot define a proper `shift move` on conformation for HP36 in Figure 6.2 which will improve objective function value. Therefore, we should say that this is the *local optimum* for HP36 instance of the PFP.

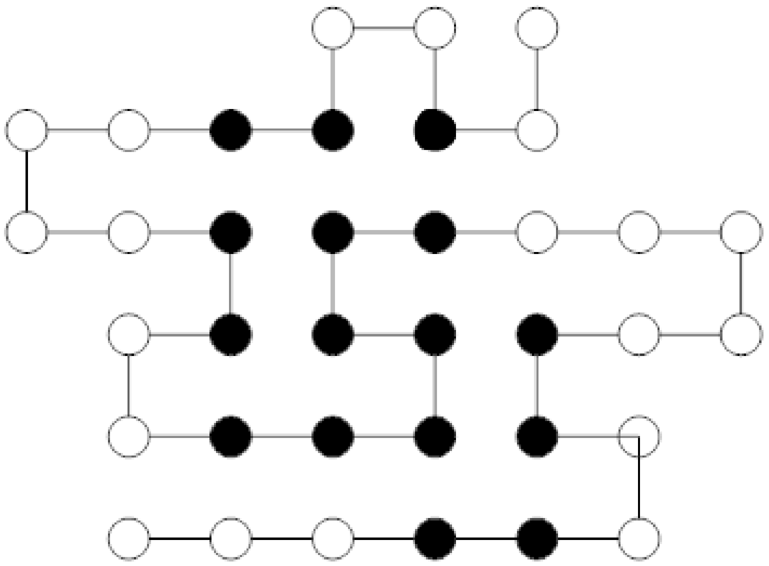
Figure 6.2: A conformation of HP36 with fitness value 13

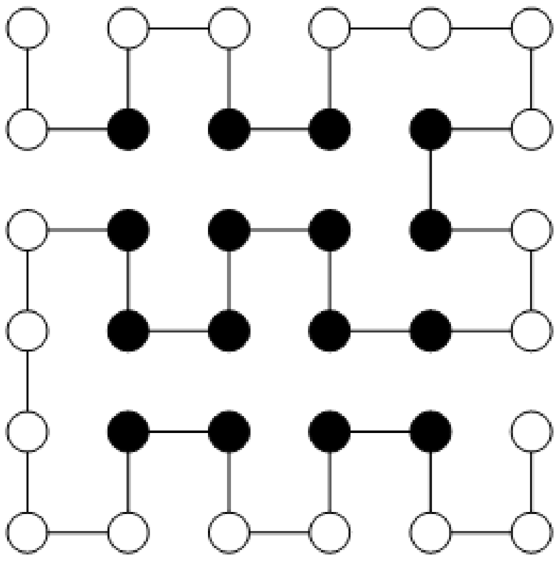
Figure 6.3: An optimal conformation of HP36 with fitness value 14

Chapter 7

CONCLUSION

The protein folding problem is a crucial and vital problem for almost all of the biological sciences. Therefore, it is an open challenge for other sciences such as computer science and operations research as an application area. Since the PFP is also proved to be NP-hard, it is an open area for application of metaheuristic methods. Our motivation for this study was developing a metaheuristic method for the PFP. Our approach was based on scatter search procedure. Thus, we have proposed several options for each of the elements of scatter search. Although we could not find an efficient combination of these elements to be used together, we were able to compare various kinds of application options.

For *diversification generator*, we have proposed a method for construction of some elite solutions that have some favorable motifs already configured. This idea aims to introduce the favorable motifs from the beginning of the search procedure. For the *subset generation method*, the classic all-pairs combination strategy is adopted. Three new moves are proposed for the PFP: `push move`, `shift move` and `turn move`, where `push move` and `shift move` are used as *improvement method* components, `turn move` is proposed to handle infeasible solutions. We have also proposed a new measurement for the diversification values of the solutions. This measurement is based adding ans subtracting the corresponding angles of the solutions to be compared. This new `diversification measurement method` is able to catch mirror images in contrast to measurement by angle comparison. There are also several options for handling *trial solution set* and *RefSet update method* since there are large number of solutions to choose from for the algorithm and a number of measurement values to differentiate the solutions.

In our study, we could not find any improved results compared to the literature. On the other hand, during the study, we have tried a number of possibilities for algorithmic components and parameters. For the scatter search, we see that $RefSet$ size plays crucial role in the performance of the search procedure. In addition, the balance between the subsets of the $RefSet$ is equally important to balance intensification and diversification on the search space. Another key element influencing this balance is the size of the $Pool$ which collects the candidate solutions for the next iteration of the search algorithm. Increasing the $Pool$ size may seem to increase diversification. However, when we apply *improvement method* for a local search on the solutions in the *trial solution set*, they may or may not converge to the same local optimum. This behavior depends on the attributes of the problem instance. Likewise, the density of the *improvement procedure* is an important factor keeping this balance at a desirable level to reach to global optimum. Applying the *improvement method* intensly may result in getting the algorithm stuck in some local optimum. However, we need an efficient *improvement method* to be able to converge to the global optimum.

The future research should focus on identifying such elements affecting intensification and diversification balance and study their behavior with respect to the output since this is one of the key ideas in the metaheuristic methods. In addition, moves defining the local neighborhood for the problem instances should be defined well so that these moves should respect both the method requirements and problem attributes.

BIBLIOGRAPHY

[1] B. Adenso-Diaz, S. Garcia-Carbajal, and S. Lozano. An emprical investigation on a parallelization strategies for scatter search. *European Journal of Operational Research*, 169:490–507, 2006.

[2] J. Blazewicz, K. Dill, P. Lukasiak, and M. Milostan. A tabu search strategy for finding low energy structures of proteins in hp-model. *Computational Methods in Science and Technology*, 10:7–19, 2004.

[3] F. Chu, N. Labadi, and C. Prins. A scatter search for the periodic capacitated arc routing problem. *European Journal of Operational Research*, 169:586–605, 2006.

[4] G. A. Cox, T. V. Mortimer-Jones, R. P. Taylor, and R. L. Johnston. Development and optimisation of a novel genetic algorithm for studying model protein folding. *Theoretical Chemistry Accounts*, 112:163–178, 2004.

[5] C.G. da Silva, J. Climaco, and J. Figueira. A scatter serach method for bi-criteria 0,1-knapsack problems. *European Journal of Operational Research*, 169:373–391, 2006.

[6] D. Debels, B. de Reyck, R. Leus, and M. Vanhoucke. A hybrid scatter search/electromagnetism meta-heuristic for project scheduling. *European Journal of Operational Research*, 169:638–653, 2006.

[7] J. A. Diaz and E. Fernandez. Hybrid scatter search and path relinking for the capacitated p-median problem. *European Journal of Operational Research*, 169:570–585, 2006.

[8] K. A. Dill. Theory for the folding and stability of globular proteins. *Bichemistry*, 24:1501–1509, 1985.

[9] K.A. Dill. Polymer principles and protein folding. *Protein Science*, 8:1166–1180, 1999.

[10] K.A. Dill, S. Bromberg, K. Yue, K.M. Fiebig, D.P. Yee, P.D. Thomas, and H.S. Chan. Principles of protein folding: A perspective from simple exact methods. *Protein Science*, 4:561–602, 1995.

[11] A. El-Fallahi, R. Marti, and L. Lasdon. Path relinking and grg for artificial neural networks. *European Journal of Operational Research*, 169:508–519, 2006.

[12] F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8:156–166, 1977.

[13] F. Glover. A template for scatter search and path relinking. In J.K. Hao, E. Lutton, E. Ronald, M. Schoenauer, and D. Snyers, editors, *Lecture Notes in Computer Science*, number 1363, pages 13–54. ., 1997.

[14] F. Glover, M. Laguna, and R. Marti. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 29(3):653–682, 2000.

[15] F. Herrera, M. Lozano, and D. Molina. Continuous scatter search: An analysis of the integration of some combination methods and improvement strategies. *European Journal of Operational Research*, 169:450–476, 2006.

[16] National Human Genome Resarch Institute. *Protein.* World Wide Web, http://www.genome.gov/Pages/Hyperion/DIR/VIP/Glossary/Illustration/Pdf/protein.pdf, 2007.

[17] N. Lesh, M. Mitzenmacher, and S. Whitesides. A complete and effective move set for simplified protein folding. In *RECOMB*, pages 188–195, Berlin, Germany, April 10-13 2003.

[18] F.G. Lopez, M.G. Torres, B.M. Batista, J.A.M. Perez, and J.M. Moreno-Vega. Solving feature subset selection problem by a parallel scatter search. *European Journal of Operational Research*, 169:477–489, 2006.

[19] R. Marti. Scatter search - wellsprings and challenges. *European Journal of Operational Research*, 169:351–358, 2006.

[20] R. Marti, M. Laguna, and F. Glover. Principles of scatter search. *European Journal of Operational Research*, 169:359–372, 2006.

[21] P. M. Pardalos, Xin Liu, and G.L. Xue. Protein conformation of a lattice model using tabu search. *Journal of Global Optimization*, 11:55–68, 1997.

[22] C. Rego, H. Li, and F. Glover. A filter-and-fan approach to the 2d lattice model of the protein folding problem. Master's thesis, School of Business Administration, University of Mississippi, 2006.

[23] R. A. Russell and W.C. Chiang. Scatter search for the vehicle routing problem with time windows. *European Journal of Operational Research*, 169:606–622, 2006.

[24] R. Sagarna and J.A. Lozano. Scatter search in software testing, comparison and collaboration with estimation of distribution algorithms. *European Journal of Operational Research*, 169:392–412, 2006.

[25] S. Scheuerer and R. Wendolsky. A scatter search heuristic for the capacitated clustering problem. *European Journal of Operational Research*, 169:533–547, 2006.

[26] A. Shmygelska and H.H. Hoos. An ant colony optimization algorithm for the 2d and 3d hydrophobic polar protein folding problem. *BMC Informatics*, 6(30), 2005.

[27] R. Unger and J. Moult. Finding the lowest free energy conformation of a protein is an np-hard problem: proof and implications. *Bulletin of Mathematical Biology*, 55(6):1183–1198, 1993.

[28] R. Unger and J. Moult. Genetic algorithms for protein folding simulations. *Journal of Molecular Biology*, 231:75–81, 1993.

[29] Wikipedia. *Amino acid.* World Wide Web, http://en.wikipedia.org/wiki/Amino_acid, 2007.

[30] D.S. Yamashita, V.A. Armentano, and M. Laguna. Scatter search for project scheduling with resource availability cost. *European Journal of Operational Research*, 169:623–637, 2006.

[31] G.Q. Zhang and K.K. Lai. Combining path relinling and genetic algorthms for the multiple-level warehouse layout problem. *European Journal of Operational Research*, 169:413–425, 2006.

VITA

Sibel Bilge Sonuç was born in İstanbul, Turkey on March 27, 1982. She graduated from Kultur Fen Lisesi, İstanbul, Turkey in 2000. She received her B.S. degrees in Industrial Engineering and Mathematics from Koç University, İstanbul, Turkey in 2005. She has started her M.S. study in Koç University in September 2005, joining Industrial Engineering department as a research and teaching assistant. Since August 2008, she is a Ph.D. student in University of Florida, Gainesville, FL, U.S.A.