# A Genetic Algorithm with Oscillating Simulated Annealing for 2D HP Model

by

Ömer Kırkağaçlıoğlu

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

Master of Science

in

Computational Sciences & Engineering

Koç University

September, 2010

Koç University

Graduate School Engineering

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

Ömer Kırkağaçlıoğlu

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:

_____

Assc. Prof. Ceyda Oğuz

_____

Prof. Atilla Gürsoy

_____

Assc. Prof. Özlem Keskin

Date: _____

*To my wife*

*and*

*My parents*

# ABSTRACT

Protein folding is the process of a linear chain of amino acids folding into a functional 3D native structure. The problem of predicting this 3D native structure given only the amino acids of a protein is one of the most challenging problems in computational biology and is still rigorously investigated. The Hydrophobic-Polar model is a simplified model for the protein folding process. This model is based on the assumption that hydrophobicity is the dominant force that drives the protein folding process.

In this thesis we propose a meta-heuristic algorithm (GAOSA) that combines the well known Genetic Algorithm approach with Oscillating Simulated Annealing to address the protein folding problem in the simplified 2D Hydrophobic-Polar Model. Our approach makes use of the pull move neighborhood for the mutation operator, a brute force 1-point crossover operator, a memory component borrowed from Tabu Search and a problem specific diversification phase. We also provide some insights about the implications of the Hydrophobic-Polar Model and how these implications can be utilized in an algorithm.

# ÖZETÇE

Protein katlanması lineer bir amino asit zincirinin 3 boyutlu işlevsel öz haline katlanması sürecidir. Proteinin 3 boyutlu öz yapısını yalnızca amino asit serisini kullanarak bulmak şu anda hesaplamalı biyoloji dalındaki en zor problemlerden biri olmakla birlikte çok yoğun şekilde araştırılmaya devam edilmektedir. 2 boyutlu HP modeli protein katlanması problemi için basitleştirilmiş bir modeldir. Bu modelde indirgemeler, protein katlanması sürecindeki baskın gücün hidrofobisite olduğu varsayımı üzerine yapılmaktadır.

Bu tezde, protein katlanması problemini 2 boyutlu HP modelde çözmek için Genetik Algoritma ve Benzetilmiş Tavlama algoritmalarının birleşiminden oluşan üstsezgisel bir metod sunuyoruz. Yöntemimiz literatürde tanımlanmış yerel arama metodlarından bazı bölümler kullanmaktadır. Tabu Arama'da kullanılan hafıza bölümü, yeni bir 1-nokta çaprazlama işlemcisi ve probleme özel bir diversifikasyon bölümü algoritmamızda bulunan parçaları oluşturmaktadır. Algoritmamıza ek olarak, HP modelin içerdiği bazı bilgiler ve bu bilgilerin bir algoritmada nasıl kullanılabileceğine dair bazı tartışmalar da tezin son kısmında sunulmaktadır.

# ACKNOWLEDGMENTS

First, I would like to thank my thesis advisor Assc. Prof. Ceyda Oğuz for her invaluable guidance and her never ending patience throughout the writing of this thesis.

I am very grateful to Prof. Atilla Gürsoy and Assc. Prof. Özlem Keskin for taking the time to be on my thesis comitee and providing me with constructive feedback.

I would also like to thank my office mates for all the support they've given me and for the friendly environment they've created during our graduate study, they've made it easier to get through the hard times.

A special thanks to my loving wife for always being there for me with her tender encouragement. She deserves as much credit as I do.

Finally I would like to thank my family for their endless support, none of this would be possible without them.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# NOMENCLATURE

| | |
|---|---|
| PFP | Protein Folding Problem |
| GA | Genetic Algorithm |
| MC | Monte Carlo |
| GAOSA | Genetic Algorithm with Oscillating Simulated Annealing |
| CGM | Chain Growth Method |
| PERM | Pruning and Enriching Rosenbluth Method |
| REMC | Replica Exchange Monte Carlo |
| FandF | Filter and Fan |
| CG | Core Directed Chain Growth |

# Chapter 1

# INTRODUCTION

The Protein folding problem (PFP) is one of the major challenges in computational biology. Due to its importance in understanding the inner mechanisms in organisms and its applications in drug design, this problem is still one of the most rigorously investigated topics.

A protein is a polypeptide that is made up of a linear chain of 20 different types of amino acids. Independent from their type every amino acid contains a carboxyl group, an amino group and a side chain that varies between different types of amino acids. The amino acids in a protein are joined together with the formation of peptide bonds between the carboxyl and amino group of adjacent amino acids. A protein in nature begins its life cycle as a linear chain of amino acids and then folds into a functional 3D native state. A native state is characteristic for a protein however a protein can have different native states under different conditions. An example 3D native structure is given in figure 1.1

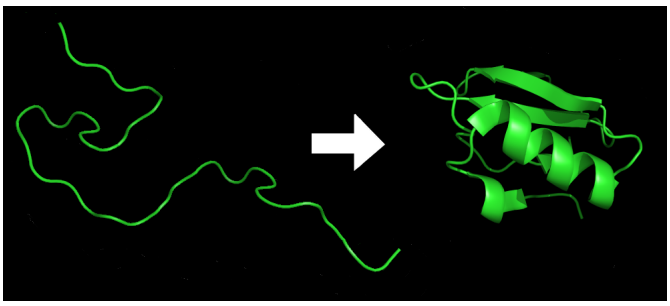

Figure 1.1: Linear chain of amino acids (Left). Folded protein in its native structure (Right)

The PFP is defined as the task of predicting the native 3D structure of a protein given its sequence of amino acids. Each 20 type of amino acid differs from each other by the properties of their side chains. A side chain of an amino acid can be:

1. Positively or negatively charged

2. Uncharged polar

3. Hydrophobic

The side chains of amino acids have different sizes and structures. The structure of a side chain is determined by its rotational angles. These rotational angles are continuous between 0°and 360°. With continuous distribution of rotational angles within amino acids and different forces that act upon the protein based on the aforementioned properties of the side chains creates a very large solution space even for proteins with small sizes. Searching this solution space for the native structure of a protein is an NP-Hard problem [24, 11, 7, 3].

The native state structure of the protein is closely related to its function, and thus to determine the function of an existing protein using only the knowledge of its sequence or to be able to engineer a protein that functions in a certain way, the 3D native state structure of the protein must be known. Current experimental methods that are used to determine the 3D structure of the proteins are X-Ray crystallography and Nuclear Magnetic Resonance, also known as NMR. These methods are used widely to create databases of protein structures with minimal errors. However, the application of these methods is both expensive and time consuming; thus, there is a need for a robust, fast, and cheap computational approach. Due to the failure of exact computational methods in solving the protein folding problem, heuristic methods are widely used to find approximate (close to native) structures.

The challenges that this problem present can be divided into two parts. The first part is to understand the inner forces that act upon the particles during protein folding resulting in the native structure. The second part is devising an efficient

search strategy for finding the conformation with the minimal free energy in the large and complex space of possible configurations. The number of possible conformations of a protein that has 150 amino acids is estimated to be $10^{300}$ [18], thus explaining the size of the solution space.

For the second part of the problem, models that simplify the search and sequence space are developed [16]. These simplified models for the PFP aims to provide a framework that decreases the size of the search space without losing the main properties of the problem. Among these models the most widely studied is the 2D HP Model [8, 16]. The 2D HP Model represents a protein as a linear chain of $n$ amino acids. Each amino acid can be one of two types: H (Hydrophobic) or P (Polar). Within the scope of this model a valid conformation is defined to be a self-avoiding walk on a 2D square lattice. A sample valid conformation for the sequence $PPPPPPPPHHHHPPPP$ is given in Figure 1.2.

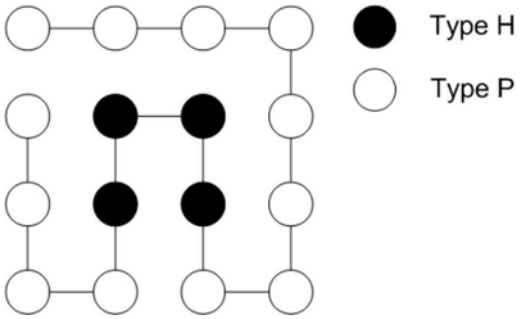


Figure 1.2: Self-avoiding walk on a 2D square lattice

In this thesis, we analyze in depth the best performing methods in the literature that are developed to solve the PFP in the 2D HP Model. We also propose a meta-heuristic algorithm, Genetic Algorithm with Oscillating Simulated Annealing (GAOSA), that combines the Genetic Algorithm (GA) framework with Simulated Annealing (SA). It can be seen from the results that GAOSA is competitive with some of the state of the art algorithms in the literature. The remainder of this thesis is organized as follows. Chapter 2 includes description of the 2D HP Model. Chapter

3 introduces the problem definition and short comings of the current implementations. Chapter 4 analyzes the state of the art methods in the literature by comparison of their individual components and their contributions. Chapter 5 explains the details of the GAOSA algorithm analyzing each component and its effect on the final results. Chapter 6 provides the results for GAOSA together with the results of the best performing methods in the literature.

# Chapter 2

# THE 2D HP MODEL

The HP model [8, 16] is a highly simplified version of the PFP problem and still captures the basic dynamics of the original protein folding problem. The model bases the simplification of the real problem onto the assumption that hydrophobicity is the dominant force that drives the proteins' folding process. In the real problem there are 20 different types of amino acids that can be part of a protein sequence. Each type has different properties but they can be separated into two classes: Hydrophobic and Hydrophilic. The HP model reduces each 20 different types of monomer, to a hydrophobic or polar type monomer. Hydrophobic (H) type monomers interact poorly with water and tends to stay close to other hydrophobic type monomers and form a hydrophobic core to minimize the contact with solvent, whereas the polar (P) type monomers favor interacting with water molecules and thus stay on the surface of the folded protein where interaction with the solvent is maximized. By decreasing the types of monomers from 20 to 2, the HP Model reduces sequential complexity of the problem. Although the reduction of the sequential complexity simplifies the real problem, this simplification doesn't effect the size of the search space which is the real bottleneck for the PFP as stated in Chapter 1.

To be able to reduce the size of the search space, the HP model discretizes the search space with a 2D square lattice. Since each lattice site has $z=4$ neighbors, where $z$ is the coordination number of the lattice, and the distance between two lattice sites is fixed, number of different bond orientations in the sequence becomes $z-1=3$. And thus a valid conformation of H and P type monomers on a 2D lattice becomes a self-avoiding and a connected path on the lattice, meaning no monomer can occupy the same lattice site, and each consecutive monomer should be unit-length apart. The

assumption of hydrophobicity being the dominant force also leads to a very simple energy function to calculate the free energy of a given conformation.

To be able to model the favorable packing of the H type monomers, the energy function for this model rewards a conformation for each of the non-bonded H-H contacts. A single H-H contact contributes -1 to the total energy of the protein and other possible contacts P-P and H-P contributes 0. Therefore, the aim of the problem can be seen as a maximization of the number of H-H contacts or minimization of the total energy of the protein.

The 2D HP Model attracted many researchers and different algorithms are introduced in the literature for solving the model. In the next chapter some of the best performing methods will be investigated in detail.The investigated methods here are the genetic algorithm (GA) [25], the Core-directed chain Growth algorithm (CG) [4], a variant of the ant colony optimization algorithm (ACO-3) [22], the Pruned Enriched Rosenbluth Method (PERM) [14], The Evolutionary Hill Climbing Algorithm (EHC) [5], the Filter and Fan algorithm (FandF) [21] and the Replica Exchange Monte Carlo algorithm (REMC) [23] which is currently the best performing algorithm in both the 2D and 3D HP Model. Also the GTabu algorithm [17] due to the move structure it proposes will be mentioned, however the details will be limited with the move structure utilized. Although some are extensions of previous work, these methods used different representations, move structures and components. In the following Chapter further discussion about the HP Model will be presented with a focus of its implications.

# Chapter 3

# IMPLICIT INFORMATION IN THE HP MODEL

The PFP in HP Model is a course-grained and a highly simplified version of the original PFP. Most the simplifications that is employed by this model uses the underlying assumption that the Hydrophobic interactions are the dominant driving force that guides protein folding process. Using this assumption 20 different types of amino-acids are reduced to 2 simple types Hydrophobic (H) and Polar (P), and a highly complex energy function is reduced to a function where each unbonded H-H contact in the lattice contributes a -1 to the total energy of the protein. These two reduced and explicit properties are the only properties that are used in the state of the art algorithms described above. However given the assumption the model is based on, there are certain principles that are implied by the model. Detailed explanation of these principles is given below.

## *3.1 Force Function*

In the original PFP, there is a force acting on each residue that drives their movements. This force function is very complicated taking into account the charge of the residue, long range interactions with other residues, side chain structure and size, and hydrophobicity of the residue. In the HP Model this force function is not explicitly defined and yet it is implied by the model that the hydrophobicity of the residue is the only driving force for the folding process. Therefore each H type residue has a force a exerted on it that drives it away from the solvent that surrounds the protein. However in none of the methods described in Chapter 4 there are not any mechanisms that incorporates this information into their local search phase.

### *3.2 Bond Formation*

Another principle that is borrowed from the original PFP is the means by which stabilization of a local or a global structure occurs. In the original PFP different types of bond formations with different energy values are formed which stabilizes a structure. In the HP Model the stabilization of a structure depends only on the hydrophobic interactions which is implemented in the given energy function, namely a protein is assumed to be more stable as the energy value decreases or the number of unbonded H-H contacts increases. Although this implies that performing a move that breaks a H-H contact should be harder than a move that doesn't affect or disrupt a H-H contact, none of the algorithms uses this information. In all of the algorithms that uses a Monte-Carlo approach, both the move location and the residue to be moved are selected at random which implies that the algorithm is indifferent between moving a residue away from a local structure.

### *3.3 Structural Implications*

The HP Model also contains information about the structure of a solution which is also inherited from the original problem. This information is implied by the energy function. The energy function in the HP Model can be comprehended in two different ways. The first and mostly used way is each H-H contact contributes -1 to the total energy of a structure thus making the problem a minimization problem. However problem can also be seen as a maximization of the number of H-H contacts. This second point has an implication on the structure of a solution. Which means that given only the structure of a solution we can infer whether it is a promising solution. A good solution is bounded by the energy function to have a high number of H-H contacts which results in compactness and one or multiple cores which contains large number H type residues. This information can be utilized in the diversification phase of the local search algorithm. If a given solution that is prematurely converged can be identified, then it can be diversified by reducing its compactness.

Using the above mentioned information that is contained in the HP Model, it is possible to create more problem specific and thus more efficient guiding strategies for algorithms. In the next chapter influential algorithms created for the HP Model will be investigated . Moreover in the Chapter 5 the specific components of the GAOSA algorithm and how these components make use of the information that is discussed in this chapter are explained in detail.

# Chapter 4

# SOLUTION APPROACHES FOR THE HP MODEL

Different approaches are used to tackle the PFP in the 2D HP Model. These approaches varied in the components they used. These components are: Local search neighborhood, representation of solutions, search guidance strategies, and acceptance of solutions. Although a mutually exclusive separation of these algorithms in all these components is not possible, they can be divided into two classes according to their global structure;

1. The first class is the so called Chain-Growth methods. This class of algorithms start from a subset of the sequence, and grows the chain by adding small parts of the sequence to the initial subset until they fold the complete sequence. Several of the algorithms mentioned in the previous chapter falls into this category.

2. The second class of algorithms starts with the complete chain, and changes it using a specific move structure until a certain stopping criteria is reached. This type of algorithms will be called Monte-Carlo extended algorithms in the rest of this thesis.

Each class of algorithms is explained in detail in the following sections.

## *4.1 Monte Carlo Methods*

In this section algorithms that extends the MC approach are explained in detail in the context of their extension and their improvement over the classical MC approach. The flow of the algorithms, move structures and the problem representations are explained in their dedicated subsections.

A Monte Carlo simulation is done by making very small random perturbations on a solution $C$ changing it to $C'$. After each perturbation the newly created solution $C'$ replaces solution $C$ if it improves it otherwise it is passed through an acceptance criteria called Metropolis criteria which is given below:

$$P(C') = e^{\frac{(E-E')}{\tau}} \tag{4.1}$$

where $P(C')$ is the probability of accepting the newly created solution $C'$ and $\tau$ is the system temperature. After the probability is calculated a random number $r$ between 0 and 1 is selected. If $r$ is smaller than $P(C')$ then $C'$ replaces $C$, otherwise $C$ stays unchanged and enters the MC stage again.

### *4.1.1 Structure of Algorithms*

The main flow is very identical in the all the Monte-Carlo extended algorithms and the Metropolis acceptance criteria is included in all of them, proving the generality and the strength of the MC approach.

Until 1993, simulating protein folding on the 2D lattice with Monte Carlo (MC) simulations alone was the most successful approach to PFP [6]. However in the later years, although algorithms borrowed certain components from the MC approach, the main algorithms varied in a wide range. Most of the algorithms that extended the MC approach borrowed similar components, the Metropolis like criteria in the acceptance of solutions and random changes in the lattice.

This non-deterministic approach to acceptance of solutions, combined with random changes in conformations, theoretically will converge to the global optimum value based on the underlying Markov Chain hypothesis. However, when this MC method is used alone, this convergence may take a large number of iterations, theoretically more than the number of iterations necessary to enumerate all of the search space [25] the MC approach after 1990 has been usually used together with other approaches such as Genetic Algorithms (GA).

The GA by Unger and Moult [25] borrowed some components of the previous MC

approaches through the mutation operator and the acceptance of solutions; however, the main flow of the algorithm was rather dominated by the ideas of evolutionary programming. A population with size $N$ is initialized with extended sequences. Each individual in the population is subject to a number of mutation steps similar to the MC approach and then passed through the Metropolis criteria that is explained above. At the end of the MC stage however, a crossover operator is applied at which point the GA deviates from the general MC flow. The crossover operator is applied to two conformations selected based on a selection strategy *S1, S2* and an offspring $S'$ is created. The newly created $S'$ is accepted if its energy value $E'$ is better than the average energy $E_{ij} = \frac{(E1+E2)}{2}$ of its parents, or if the energy value is worse it is accepted with Metropolis criteria.

The contributions of the GA algorithm was to combine the components from the evolutionary algorithms such as mutation, crossover and parent selection strategy with the MC approach. The improvements of this combination are visible from their results. The GA outperformed the previous MC methods, both in the quality of the solutions and computational time. The method found better solutions and found them $\geq 10$ times faster than the previous MC methods.

Another method that extends the MC approach is the REMC algorithm [23]. This algorithm is currently the best performing algorithm in both the 3D and 2D HP Model in the literature. The main flow of REMC is again very similar to the classical MC approach. Being a population based method REMC algorithm maintains $\chi$ independent replicas of a conformation. However different than both the GA and the classical MC, a different temperature value is kept for each of the replicas. The algorithm performs a MC search for each of the replicas and applies the Metropolis criteria given in equation (4.1) using the replicas' individual temperature. After the MC search stage is completed, a replica exchange stage is started. The replica exchange state consists of exchanging the individual temperature values of adjacent replicas $C$ and $C'$ based on the following measure:

$$P_{C' \to C} = \begin{cases} 1 & \text{if } \Delta \leq 0 \ , \\ e^{-\Delta} & \text{otherwise.} \end{cases} \tag{4.2}$$

The value of $\Delta$ is the product of the energy difference and the inverse temperature difference.

$$\Delta = (\beta_j - \beta_i) \times (E(c_i) - E(c_j)) \tag{4.3}$$

where $\beta_j = \frac{1}{T_j}$ is the inverse of temperature of replica $j$.

This replica exchange criterion aims to assign lower temperatures to promising solutions for better intensification and assign higher temperatures to unfit solutions for diversification.

### *4.1.2 Move Structures*

There are two different types of moves that are used widely by the methods that are developed for the HP model. Local Moves, and Long-ranged or Global Moves. They differ in the amount of modification they cause on a given conformation. In some of the methods these two types are coupled in different operators of the algorithm such as crossover and mutation operator of the GA.

#### *Local Moves*

A local move is a move that is performed on a small number of residues thus causes changes on the local neighborhood of the selected residues. Among the most widely used moves there are VSHD neighborhood [12, 26, 10] which is combined neighborhood from three different papers and the so-called Pull-Move neighborhood proposed by [17].

**VSHD Moves** The VSHD neighborhood contains 3 types of local moves;

1. **End Move**

End Move can only be applied to the first or the last residue in the sequence. The residue is pivoted relative to its connected neighbor to a free position adjacent to that neighbor, if more than one free positions are available than one is chosen at random.

2. **Corner Move**

   Corner Move can be applied to any of the residues except for the start and end residues. To be able to apply the corner move to a given residue, a position to which both the residue's connected neighbors are adjacent to, must be available. See Figure 4.1 (b) for reference.

3. **Crankshaft Move**

   Crankshaft Move requires that a residue i is part of u-shaped bend in the sequence. Referring to Figure 4.1 (c), it can be applied if the positions *i' and i'+1* are available.

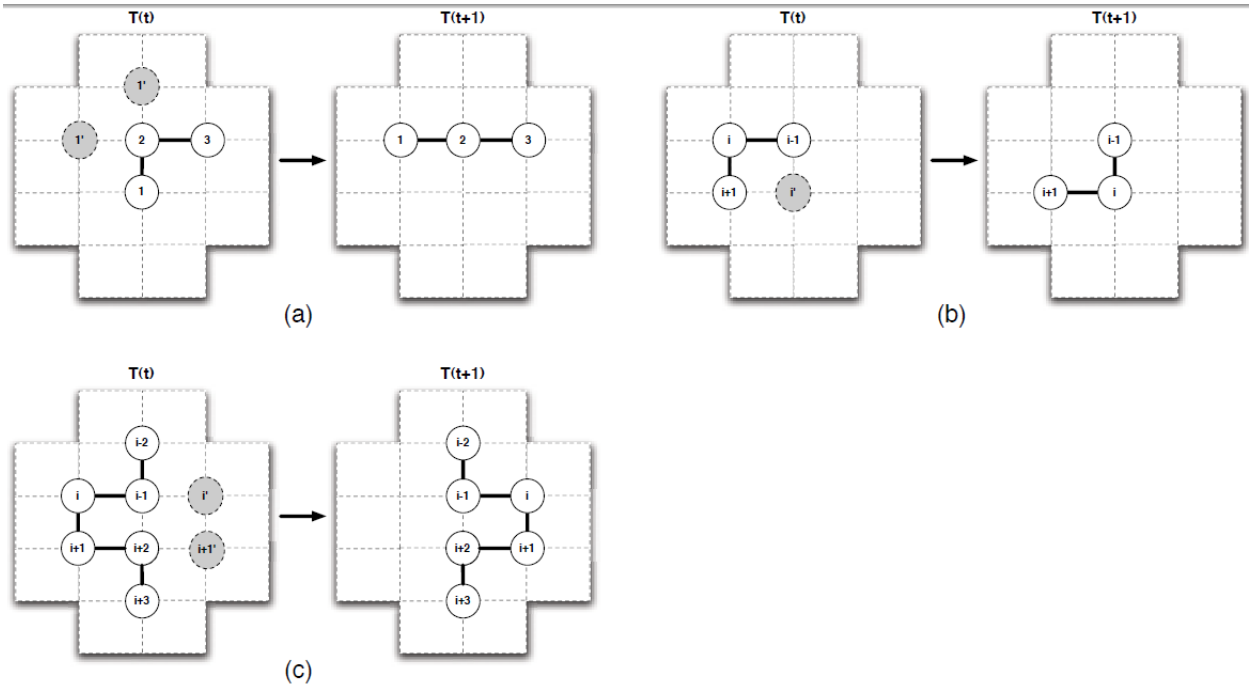


Figure 4.1: VSHD Moves. Residue positions are shown before the move and immediately after a successful move. T(t) denotes the state of the conformation at time t. In 2a there are two possible positions that residue one could be moved to, denoted by 1' in gray circles. Each position is checked in random order for availability. If a position is found to be free, the residue is moved. 2b shows there to be only one potential new position for a corner move. 2c shows the case for a crankshaft move. This figure is borrowed from the paper [23].

**Pull Moves** Among the local move structures that are used in above mentioned methods, the Pull-Move structure [17] is the most effective neighborhood structure which is also used by REMC [23], FandF [21], EHC [5], GAOSA our own method. In the original article the move structure is demonstrated using a simple Tabu Search algorithm called GTabu and gets competitive results.

A pull move consists of an initial single pull event and a following series of corrections if the initial pull breaks the chain. Thus the result of a pull move is always a valid conformation. Another powerful feature of the pull move neighborhood is that it is complete over all of the possible conformations in the lattice. This means that given any two different states of a conformation $P$ and $P'$, there is a set of pull moves

that leads from $P \rightarrow P'$. With this property of the pull moves an algorithm that employs this neighborhood can theoretically reach every solution in solution space.

For a single pull move to happen consider a vertex $i$ at time $t$ in the position $(x_i(t), y_i(t))$. Suppose that a free location $L$ that is adjacent to location $(x_{i+1}(t), y_{i+1}(t))$ and diagonally adjacent to vertex $i$ exist. (This explanation of the pull move refers to Figure 4.2). Then these three locations constitutes the three corners of a square. Let the fourth corner be the location *C*. For a pull move to occur location $C$ must either be free or be occupied by vertex *i-1*. When location $C$ is occupied by vertex *i-1*, then the pull move consists of moving vertex $i$ to location L, otherwise this initial pull move breaks the chain and then a series of correction moves are done. The correction moves starts with moving vertex *i-1* to location *C*. Then the following procedure is applied: starting from vertex $j = i - 2$ down to vertex 1, set $(x_j(t+1), y_j(t+1)) = (x_{j+2}(t), y_{j+2}(t))$. Which means shifting the residues by two up the chain until a valid conformation is reached. The strength of the pull move neighborhood besides the above mentioned completeness, and valid conformation output properties, it forms a square shape namely a u-bend in the sequence which is smallest building block of a folded protein. The moves in the VSHD neighborhood provides only superficial change to the conformation and thus they are dependent on the rest of the conformation to be able create any improvement. However a single pull move can create -1 energy improvement from an extended sequence because it tries to form the u-bend shapes.

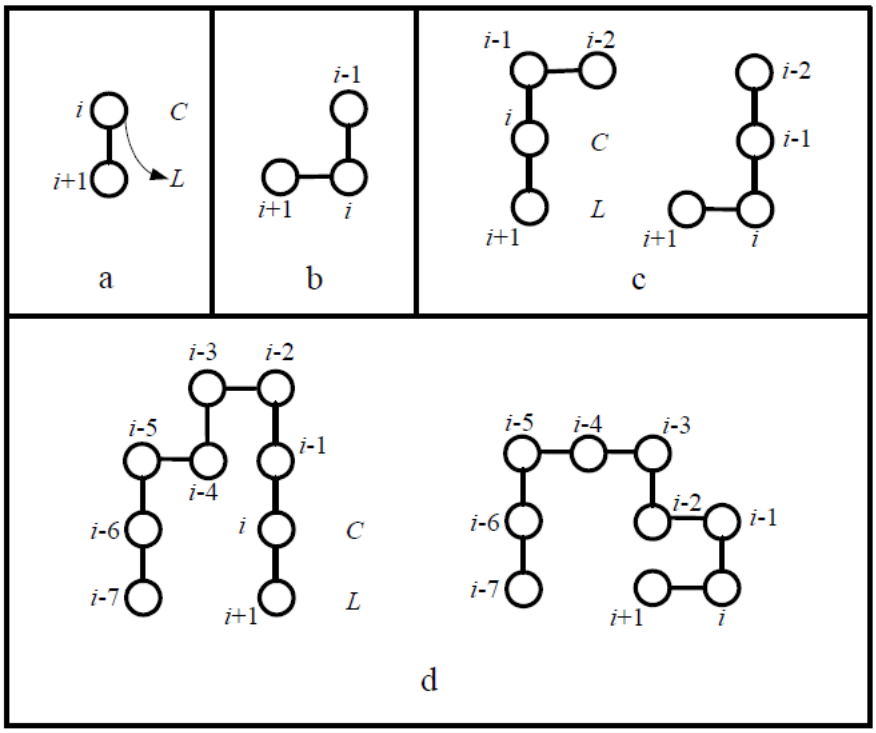


Figure 4.2: In (b), the simplest case where position C is occupied by residue i - 1 is shown. This move is equivalent to a corner move in the VSHD move set. In (c), residue i is moved to L and i - 1 to C. The chain is in a valid conformation and the move is finished. In (d), residues i down to i - 3 must be pulled until a valid conformation is found.

*Global Moves*

Global moves are defined as the moves that cause changes in the overall conformation of the protein instead of changing a small number of monomers. In most genetic algorithms and some population based methods there are two different structures called the crossover operator and the mutation operator.

The so-called crossover operator is used to create new solutions from other promising solutions. The promising solutions selected to create the new solution are called parent solutions and the newly created solutions are called the offspring solutions. A general implementation for the crossover operator is the k-point crossover method. In k-point crossover k random points are chosen for the parent solutions. Each alter-

nating part is switched between parents to create two offspring solutions. A demonstration for a 2-point crossover is given in Figure 4.3.

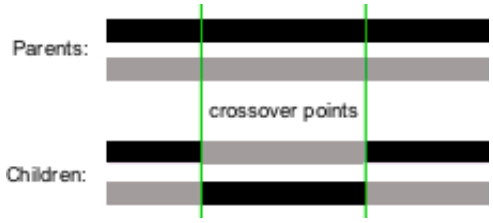


Figure 4.3:

The mutation operator is applied on a single selected solution and is usually implemented to cause smaller changes compared to the crossover operator.

In the above mentioned methods there are moves used in the crossover operator of EMC [19] and GA [25], the mutation operator of GA and the replica exchange of REMC [23] algorithm that can be seen as global moves.

Crossover operators of EMC and the GA are k-point crossover operators explained above. These moves have a large effect on the overall configuration of the protein, thus can be considered as a global move. A demonstration of a 1-point crossover operator of EMC algorithm is given in Figure 4.4.

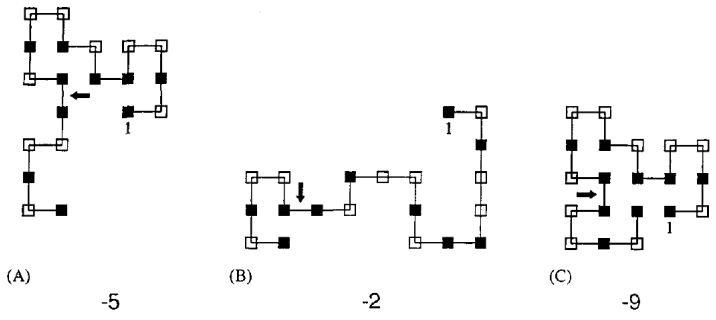


Figure 4.4: (A) and (B) are parent solution from which the offspring will be generated. The random residue is selected to be residue 14. (C) The generated offspring. This figure is borrowed from the paper [19].

Another global move that is mentioned above is in the mutation operator of the GA. A random point on a given conformations is chosen and the rotational angle between the parts that this point divides is changed randomly. Again this changes the overall conformation and thus can be considered a global move. A demonstration is given in Figure 4.5.

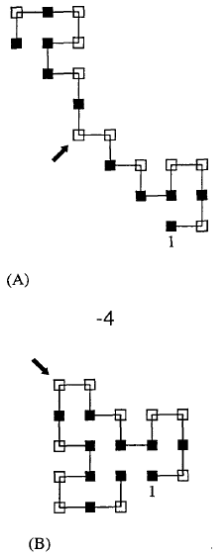


Figure 4.5: The mutation operator of the GA is applied to the conformation in (A). The selected random residue is 11, and the conformational angle is changed from $270^o$ to $90^o$. This figure is borrowed from the paper [25].

Among these global moves the most subtle move is the replica exchange process in the REMC algorithm. This move although does not change the position of any residue, by changing the temperature value of a given replica it either enables diversification or intensification of the replica, which leads to a more extended and a more compact conformation respectively, thus causing a global change in the structure of the conformation.

Another new method that can be categorized as a Global Move, is the diversification method employed in the EHC [5]. In this method at each generation a structural similarity check between the individuals with the same energy values is done. If two

given individuals' similarity is over a threshold, than one of the individuals is changed by applying consequent pull moves until the similarity decreases. The structural similarity check is a very powerful idea to tackle the premature convergence problem that are inherent in combinatorial optimization problems. Structural implications are discussed in Section 3.3.

### *4.1.3 Representations*

Representation of a problem can be both problem specific or implementation specific. In these methods there are 2 different representation schemes. The natural and most straight-forward way of representing a conformation is to assign values to different turns that each monomer can take. These different turns are demonstrated in Figure 4.6.

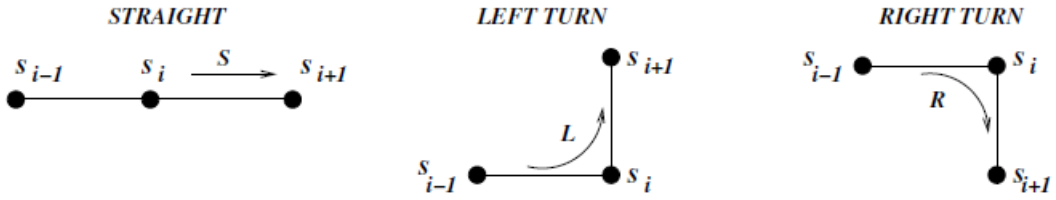


Figure 4.6: Representation of conformations using turns of in the sequence

With this representation the problem is invariant to rotational or translational changes, thus the initial and final turns can be fixed without loss of generality. For instance the representation of the conformation given in Figure 4.7 would be: *LSLL-RRLRLLSLRRLLSL*

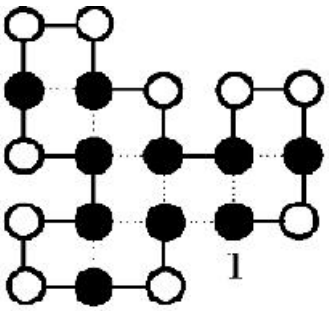


Figure 4.7: Protein conformation for the given relative representation LSLLRRLR-LLSLRRLLSL

The above mentioned algorithms except for REMC along with many in the literature use this representation, due to its simplicity. However in the case of the algorithms that uses the Pull Move structure this representation cannot be used. For the amount of information the Pull Move structure requires this representation is not sufficient. To be able to implement the move structure efficiently, the coordinate system of the 2D lattice must be represented with the 1-level adjacent neighbors for each monomer kept in this representation. An example to this type of representation for the conformation with sequence $HHHPPHPPHHHHHP$ in Figure 4.8 is given in Figure 4.9

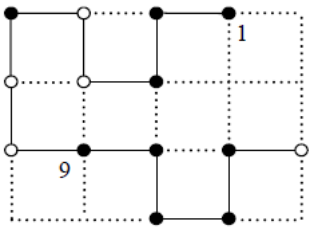


Figure 4.8: Protein conformation for the lattice representation in Figure 4.9

| Node | | | Neighbors | | | |
|---|---|---|---|---|---|---|
| Index | Coordinates | Type | East | South | West | North |
| 1 | (0,0) | H | 0 | 0 | 2 | 0 |
| 2 | (-1,0) | H | 1 | 3 | 5 | 0 |
| 3 | (-1,-1) | H | 0 | 10 | 4 | 2 |
| 4 | (-2,-1) | P | 3 | 9 | 7 | 5 |
| 5 | (-2,0) | P | 2 | 4 | 6 | 0 |
| 6 | (-3,0) | H | 5 | 7 | 0 | 0 |
| 7 | (-3,-1) | P | 4 | 8 | 0 | 6 |
| 8 | (-3,-2) | P | 9 | 0 | 0 | 7 |
| 9 | (-2,-2) | H | 10 | 0 | 8 | 4 |
| 10 | (-1,-2) | H | 13 | 11 | 9 | 3 |
| 11 | (-1,-3) | H | 12 | 0 | 0 | 10 |
| 12 | (0,-3) | H | 0 | 0 | 11 | 13 |
| 13 | (0,-2) | H | 14 | 12 | 10 | 0 |
| 14 | (1,-2) | P | 0 | 0 | 13 | 0 |

Figure 4.9: Lattice representation for the protein in Figure: 4.8

As can be seen from the table in Figure 4.9 each node has seven properties. These properties are *index* specifying where in the chain is the node, *type* specifying "H" or "P" type residue, *coordinates* specifying its place in the lattice and the indexes of the four neighbors of the node. All this information contained in a node is necessary for the implementation of the pull move neighborhood. Representation of the solutions has a large effect on efficiency and simplicity of the implementation, which in turn has a big influence in the performance of the algorithm.

## *4.2 Chain Growth Algorithms*

Besides the MC extended methods other types of approaches are also used to attack the protein folding problem. One of these types of methods is the chain growth method (CGM). The basic idea in CGMs is that each conformation is created starting from a given initial point by adding monomers one by one, instead of starting with an extended chain and modifying it with certain moves. This approach is also coupled with the Metropolis criteria in the literature such as in the case of PERM.

Among the investigated methods three of them falls into this category; ACO-3

[22], PERM [14] and CG [4]. Each of these three algorithms contributed substantial ideas to the literature and they received competitive results.

### *4.2.1 Building the Chain*

In all these three algorithms there is some form of chain building process due to the nature of the algorithms.

In the ACO-3 algorithm the chain construction is done by the so-called ants. Each ant starts from a random point in the chain and starts building the chain in both directions by adding a monomer with a certain relative direction. This is the same relative representation scheme explained in Section 4.1.3.

More specifically each ant adds a monomer at the $i^{th}$ position with a relative direction $d$ with probability $P_{i,d}$ where $d \in (S, L, R)$. The probability function for this construction is based on $\tau_{i,d}$ which is called the pheromone values, and a heuristic function $\eta_{i,d} = e^{-\frac{h_{i,d}}{T}}$ where $h_{i,d}$ is the number of new H-H contacts achieved by placing residue $i$ with direction $d$. The pheromone values $\tau_{i,d}$ are used to calculate a desirability of a certain relative direction for the residue $i$. Then the function for $P_{i,d}$ becomes as follows:

$$P_{i,d} = \frac{[\tau_{i,d}]^{\alpha}[\eta_{i,d}]^{\beta}}{\sum_{e \in (S,L,R)}[\tau_{i,e}]^{\alpha}[\eta_{i,e}]^{\beta}} \tag{4.4}$$

where $\alpha$ and $\beta$ are constants between 0 and 1, that determine the importance of pheromone values and the number of H-H contacts that will be added to the conformation.

During this construction phase of the ACO-3 just like all the CGMs, an infeasible conformation can be encountered. To be able to overcome this problem every CGM implements a certain type of mechanism. ACO-3 deals with this issue by a backtracking mechanism. When an infeasible conformation is encountered, the half of the sequence leading up to the point of infeasibility is unfolded. Then the last unfolded residue is inserted back such that its relative direction value differs from what it had been when the infeasibility occur

Among the CGMs mentioned in this chapter the CG [4] algorithm contributes ideas from the biological properties of the proteins and incorporates these ideas into its chain building process. The CG algorithm starts by estimating a core based on the number of H type monomers in the sequence. After this theoretical core is formed the space inside and outside the core is arranged into shells of layers. A demonstration is given in Figure 4.10.

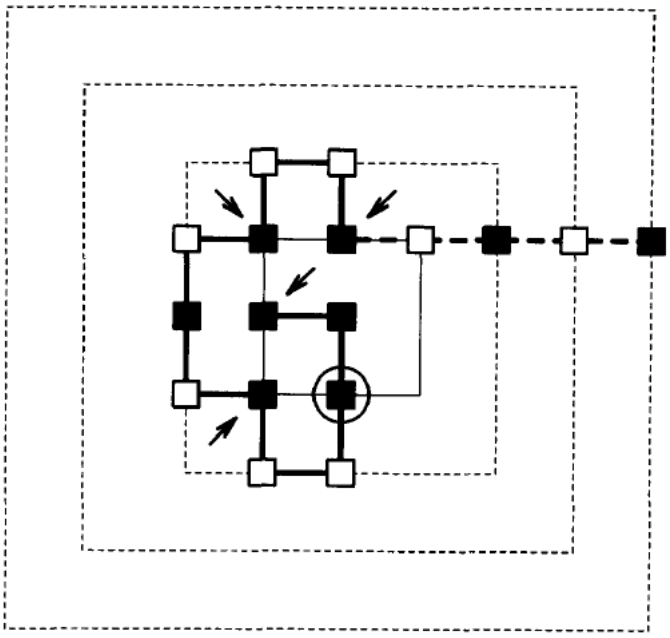

Figure 4.10: A demonstration of the core (thin solid lines) and the surrounding layers (dotted lines). A growing chain is shown, the fixed residues are shown with bold solid bonds. This figure is borrowed from the paper [4].

The chain building process is initialized by selecting any of the H type monomers that has a connected neighbor which is also of type H. The first residue is placed in the outermost shell of the core at random, and all the subsequent additions to the chain are made in segments not only in monomers. When adding a segment to the chain every possible placement is searched exhaustively, and invalid placements are pruned. One placement of the segment is chosen among the remaining valid conformation according to a heuristic function $f_h$ given below:

$$f_h = \sum_{H \in segments} S_H^{field} + \sum_{P \in segments} S_H^{field} + \sum_{HHcontacts} S_{HH} + \sum_{PPcontacts} S_{PP} + \sum_{HPcontacts} S_{HP} \quad (4.5)$$

The main purpose of the heuristic function is to drive the formation of a hydrophobic core. The first 2 terms in the function are unitary weights for both type of residues that scores their position on the chain layers. The values of the parameters are given in the following Figure 4.11.

| Parameter | Value | Description |
|---|---|---|
| $S_H^{field}$ | 1 | Weight of H in core region. |
| | 0, −1, . . .[a] | Weight of H outside core region. |
| $S_P^{field}$ | 1 | Weight of P outside core region. |
| | 0 | Weight of P in first shell inside core region. |
| | −1 | Weight of P in the deeper shells of the core region. |
| $S_{HH}$ | 0 | Weight of HH contacts while $l < 1/3\ l_{chain}$.[b] |
| | 1 | Weight of HH contacts while $1/3\ l_{chain} \leq l < 3/4\ l_{chain}$. |
| | 2 | Weight of HH contacts while $l \geq 3/4\ l_{chain}$. |
| $S_{PP}$ | 0 | Weight of PP contacts. |
| $S_{HP}$ | −1 | Weight of HP contacts. |
| $l_{segment}^{min}$ | 2 | Minimum segment length in growth step. |
| $l_{segment}^{max}$ | 5 | Maximum segment length in growth step. |
| $l_{nuc}$ | $1/3\ l_{chain}$ | Length of the segments left as nuclei in the second phase. |

[a]Weight is decremented by −1 in each lattice shell surrounding the estimated core region.
[b]$l$, Current length of chain. $l_{chain}$, Length of complete chain.

Figure 4.11: Parameters of the heuristic function. This figure is borrowed from the paper [4].

In the CG algorithm chain is grown in segments instead of single monomers, thus the length of the segment to be added next is also a parameter in the chain growth process. This parameter is set to be the distance to the next H residue due to the H type monomer strong effect on the formation of the core. However if this distance exceeds a certain cutoff value than this cutoff value is set to be the segment length. The idea of estimating a hydrophobic core and constraining the search space is unique to the CG method. Although this method is not applied to the larger sequences its

performance on sequences up to size 64 is very competitive in the literature.

From the above mentioned methods, the best performing algorithm is the PERM algorithm [14]. PERM employs a chain growth process where after $n^{th}$ monomer is added to the chain, a weight $W_n$ is calculated. If this weight value is below a certain threshold the chain is pruned. PERM also uses an enriching process. The enriching process is done by calculating a predicted weight $W_n^{pred}$ before the $n^{th}$ monomer is added. If the calculated weight is larger than a specified threshold the population is enriched with the $k$ different continuations of the chain, where $2 \leq k \leq k_{free}$. PERM algorithm has 2 variations that calculates the $W_n^{pred}$ using different methods: nPERMss and nPERMis. However nPERMis outperforms nPERMss in every test case, thus it will be the only variation investigated here. When calculating the predicted weight two factors are incorporated in the formula: The number of free neighbors and energy values after the addition of the $n^{th}$ monomer. Thus an importance factor $q_\alpha$ for each possible continuation $\alpha$ becomes:

$$q_\alpha = (k_{free}^\alpha + \frac{1}{2})e^{-\beta E_{n,\alpha}} \tag{4.6}$$

Given this importance factor for a given continuation $\alpha$, the predicted weight $W_n^{pred}$ becomes:

$$W_n^{pred} = W_{n-1} \sum_\alpha q_\alpha \tag{4.7}$$

The PERM algorithm although is one of the best performing algorithms in the literature, has some shortcomings on a special subset of the problems as stated by their research [14]. For a CGM to be able to reach the symmetric ground-state with interacting C and N-termini, the algorithm needs the pass through high energy states and keep folding in the same direction until it reaches the ground state, thus creating a challenge for any CGM to find the ground-state energy of this subset of benchmarks.

# Chapter 5

# GAOSA ALGORITHM

The GAOSA algorithm is a combination of the widely studied genetic algorithms [13] and simulated annealing [15]. These major parts are modified to include a memory component borrowed from Tabu search [9] and a problem specific diversification phase.

The main flow of our algorithm is very similar to a generic genetic algorithm. It includes a mutation operator and a crossover operator. The pseudo code for the main flow of the algorithm is given in Algorithm 1.

**Algorithm 1:** Main Flow of GAOSA Algorithm

```
input : χ : Population Size
        E* : Optimum energy for the given sequence
begin
    Population ← Initialize();
    E ← 0;
    while E < E* do
        for i ← 0 to χ do
            C_i ← Population[i];
            Mutate(C_i);
        ParentList ← ParentSelect(Population);
        ParentPairs ← CreatePairs(ParentList);
        foreach Pair ∈ ParentPairs do
            Crossover(Pair);
```

In the following sections a detailed explanation of main components of the algorithm will be given, then the methodology for parameter optimization for this final component set will be reviewed and the strategies that improved the algorithm will be discussed.

## *5.1 Mutation Operator*

In genetic algorithms, the mutation operator is generally a function that makes small changes on a given solution. In our implementation the mutation operator is a complex function that is made up of a local search phase employing simulated annealing, a memory component and a diversification phase. Each of these subcomponents will be explained in detail in the following sections. The pseudo code for the mutation operator is given in Algorithm 2.

**Algorithm 2:** Mutation Operator

**input** : $C$ : Solution to be mutated

$\phi$ : Local search iteration size

**begin**

```
LocalSearch(C, φ);
Diversify(C);
UpdateTemperature(C);
```

### *5.1.1 Local Search*

In the local search phase of the algorithm, there are two different components that attempts to address the issues discussed in Chapter 3. One of these components is borrowed from the Simulated Annealing (SA) algorithm and the other involves a memory component similar to the Tabu List from Tabu Search algorithm [9]. The pseudo code for the local search phase is given in Algorithm 3.

**Algorithm 3:** Local Search Phase

**input** : $N$ : Number of local search steps

$C$ : Conformation to be modified

**begin**

```
Memory ← getMemory(C);
for i ← 1 to N do
    residue ← getRandomFreeResidue(Memory);
    location ← getRandomFreeLocation();
    C' ← PullMove(residue, location);
    ΔE ← E(C') − E(C);
    if ΔE ≤ 0 then
        C ← C';
        UpdateMemory(Memory, residue, ΔE);
        if E(C') ≤ E* then
            E* ← E(C');
    else
        r ← Uniform(0,1);
        T ← getTemperature(C);
        if r < e^(−ΔE / (T × K_b)) then
            C ← C'
        UpdateNotImprovedCount(C);
```

*Simulated Annealing*

Each solution in the population has an internal temperature, and this internal temperature value sets the probability of accepting a worse solution. The temperature component enables the algorithm to simulate a bond formation which refers to the issue defined in 3.1 given in Chapter 3. When a move that breaks one or more H-H contacts is made, the temperature value sets the difficulty of accepting this solution into the population, as the temperature value increases the energy of each residue in-

creases and the energy necessary to break the bonds is obtained, thus the probability of accepting a worse solution increases. The acceptance probability given the temperature value is calculated using the equation given in equation (4.1). The parameters for the temperature values and annealing schedules are explained in the following section.

**Parameters for Simulated Annealing:**

There are three different parameters involved in the SA component:

1. *MaxTemp:* The maximum temperature
2. *MinTemp:* The minimum temperature
3. Annealing Schedule

These three parameters set the lower and the upper bounds for the acceptance probabilities and define the transition from one temperature to the other respectively. The bound probabilities are calculated as the minimum and maximum probabilities that is necessary to accept a solution with +1 energy than the current solution. We also borrow the usage of the Gas Constant $R = 0.0019872$ from the REMC source code that is used to normalize the temperatures for reasonable probabilities. Experimental results and the effects of different values of these temperature values are given in detail in section 5.3. Each solution is initialized with the maximum temperature and an annealing schedule described below is applied after every local search. When a solution's internal temperature drops below the given minimum temperature it is set to the maximum temperature.

Annealing schedule is one of the most important parameters that affects the performance of the algorithm, thus in our implementation we selected two of the most widely used annealing schedules, exponential and linear, and tested them with two sets of different parameters. These four annealing schedules are explained below:

Exponential and linear annealing schedules are widely used [15]. The exponential annealing schedules use a parameter $\alpha$ and update the temperature according to the equation (5.1).

$$T_{i+1} = T_i \times \alpha. \tag{5.1}$$

This annealing schedule balances diversification and intensification of solutions by spending less time on higher temperatures to prevent loss of good solutions, however decreasing the temperature slowly thus preventing premature convergence of solutions.

The linear annealing schedules update the temperature by decreasing it a given fixed value using the equation (5.2). In our parameters, we use the number of steps needed to calculate $\Delta T$

$$T_{i+1} = T_i - \Delta T. \tag{5.2}$$

Linear annealing schedules distribute the temperature values uniformly over each level. The plots of each of the annealing schedules are given in Figure 5.1.

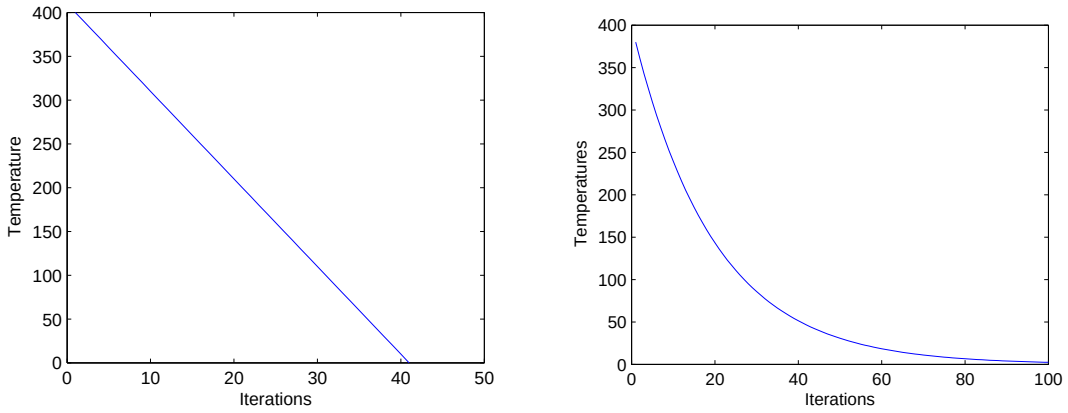


Figure 5.1: Linear annealing schedule with maximum temperature 400 and minimum temperature 0 with 100 steps (Left). Exponential annealing schedule with maximum temperature 400 and minimum temperature 0 with $\alpha = 0.95$

In our implementation simulated annealing temperatures oscillate. The internal temperature of a solution starts from the $MaxTemp$ and decreases to $MinTemp$. When it falls below the $MinTemp$ parameter, it is set to the $MaxTemp$ again.

*Memory Component*

Although the SA component facilitates the simulation of bond formations, it does not directly approach the issue of Bond Formation described in Section 3.2, and it does not take into account the local interactions between residues. To be able to better address this issue, a memory component is implemented. Each solution in the population has a memory that keeps a list of residues that interact locally. This memory component is updated after every pull move in the local search phase. The approach is very similar to the Tabu List in the Tabu Seach algorithm [9]. It contains two lists of movable and non-movable (Tabu) residues for every solution, each with a corresponding time. The amount of time that a residue will be in the non-movable residues list is directly proportional to the amount of energy that the residue contributes. The time that a residue spends in the non-movable list is a parameter of our algorithm $\eta$.

$$t_i = \Delta E_i \times \eta \tag{5.3}$$

A demonstration of how the memory component works is given in Figure 5.2.

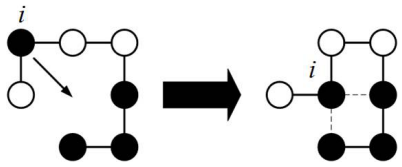


Figure 5.2: The residue i in the figure above caused -2 decrease in the overall energy of the protein. Residue i and the 2 H type neighbors shown in the figure will be put in the memory for $|-2| * \eta$ pull moves

After every move the time values of non-movable residues are decreased by 1. The implemented memory component is used to form a zipper for the structure, fixing the local stable structures and thus reducing the solution space.

### *5.1.2 Diversification Phase*

One of the missing guiding strategy components in the literature is a method to detect and effectively diversify solutions that are stuck in a local optima. Metropolis criteria given in equation (4.1) addresses this problem by enabling acceptance of worse solutions. However it is not specific to a solution that is prematurely converged which implies that when a worse solution is accepted using Metropolis criteria the replaced solution can be a promising solution which can be explored further. To be able to overcome this problem the GAOSA makes use of the structural implications of the HP Model described in Chapter 3 using the method described in the following section.

#### *Detection of Premature Convergence*

As explained above the problem has two independent parts. To be able to effectively diversify solutions that prematurely converged, first a method should be devised to detect the solutions that are stuck in a local optima. For this detection the GAOSA employs a simple approach. A counter is kept within each solution in the population, that keeps record of the number of moves the given solution could not be improved. After the local search phase, these counters are queried. If a solution could not be improved more than 150 moves then the diversification phase is initiated and the internal temperature of the solution is set to the $MaxTemp$ parameter which is shown in detail in the pseudo code Algorithm 4.

#### *Diversification Method*

The diversification phase uses the structural implication of the HP Model to diversify a given solution. A solution that is prematurely converged has a very compact structure as implied by the model. Given this compactness most of the inner residues, number of which is larger than the outer residues, has low mobility. Therefore most pull move attempts are failed due to the compact nature of the structure, which hinders the algorithm's means to diversify the solution. To be able to overcome this problem a diversification method should first decrease compactness of the solution and then fold

it again. The GAOSA has a straightforward approach to this problem described in pseudo code Algortihm 4. For every solution, after every pull move and crossover the gravitational point of the solution is calculated. Using the midpoint coordinates of the solution during the diversification phase, residues are selected at random and a comparison is made between the selected residue's coordinates and the midpoint of the solution. After the comparison is made the selected residues place in the solution relative to the midpoint is detected and the residue is moved away from the midpoint. Given that the pull move neighborhood can cause long range reactions by moving a large number of residues this method is very effective in opening up compact solutions. A demonstration of the diversification phase is given in figure 5.3

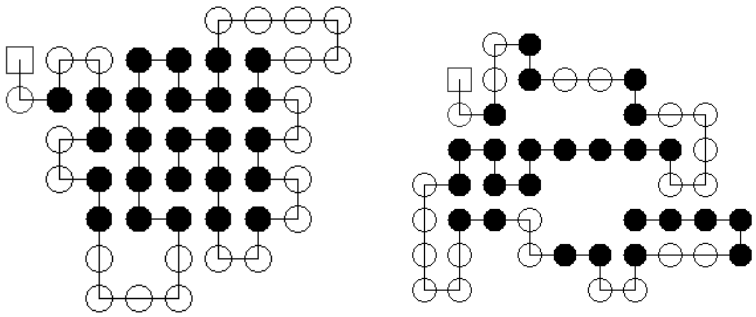

Figure 5.3: (Left) This structure is a local optimum with energy value -22 for the benchmark S1-5. (Right) This structure is reached after 10 steps of diversification applied to the structure on the left.

The pseudo code for the diversification method is given in Algorithm 4.

**Algorithm 4:** Diversification Phase

**input** : $CutOff$ : Upper limit for the not improved count

$C$ : Conformation to be modified

```
begin
    NotImprovedCount ← getNotImprovedCount(C);
    MidPoint ← getMidPoint(C);
    if NotImprovedCount ≥ CutOff then
        for i ← 1 to 10 do
            residue ← getRandomResidue();
            PullResidueAwayFromMidPoint(residue, MidPoint);
        ResetNotImprovedCount();
```

### *5.2 Crossover Operator*

The pull move neighborhood is a very effective local neighborhood, however in the HP Model moves that causes more substansial changes in the overall structure of the protein have been shown to work well in section 4.1.2. The crossover operator can effect the structure of a solution in 1 move whereas a pull move has to be applied multiple times to achieve the same structural change. The crossover operator GAOSA is comprised of the three different strategies:

1. **Selection Strategy**
2. **Creation of Offsprings**
3. **Replacement Strategy**

### *5.2.1 Selection Strategy*

Before the application of the crossover operator certain individuals are selected from the individuals as parent solutions. After every parent solution is selected these solutions are paired at random and the crossover operator is applied. Number of parent solutions to be selected is a parameter of our algorithm is given as percentage $\rho$ of the population size which varies depending on the size of the problem.

For the selection process we implemented three fitness based methods: Roulette Wheel Selection, Tournament Selection [20] with different tournament sizes and Stochastic Universal Sampling [2]. Each of these methods selects the parent solutions according to their fitness function, in the context of PFP this is the energy function. The type of selection strategy is the parameter $\Pi \in \{TOUR, RWS, SUS\}$ in our algorithm

*Stochastic Universal Sampling (SUS)*

This selection strategy is fitness based selection strategy which provides a way of selecting the parent solution without bias and minimal spread compared to the roulette wheel selection. In SUS, selection is made using one random variable. First a random number $r$ is selected between 0 and $\frac{F}{N}$ where $F$ = Total Fitness and $N$ is the number individuals to be selected as parent solutions. After the value of $r$ is selected, a fixed number $\frac{F}{N}$ is added to $r$ $(N-1)$ times. After every addition, a new number is found that corresponds to an individual in the population. Thus using only one random number we can select every parent solution.

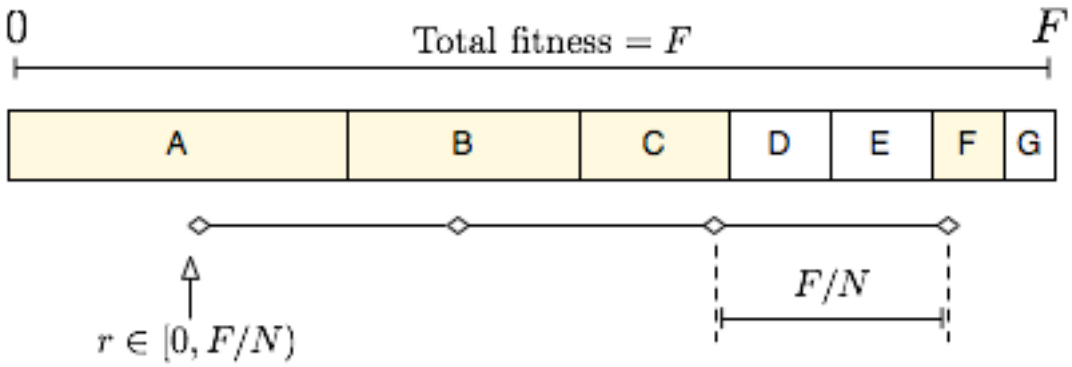


Figure 5.4: Demonstration of Stochastic Universal Sampling

*Roulette Wheel Selection (RWS)*

In RWS for each parent a random number is generated unlike SUS. For each of these random numbers the fitness proportionate buckets are calculated from the start. This is similar to turning a roulette wheel with larger parts of the wheel assigned to individuals with better fitness values. Each time the wheel is rolled there is a higher probability to select the individuals with larger fitness values which creates a bias towards the strong individuals in the selected parent list. This bias causes the domination of the population with high fitness individuals leading to premature convergence.

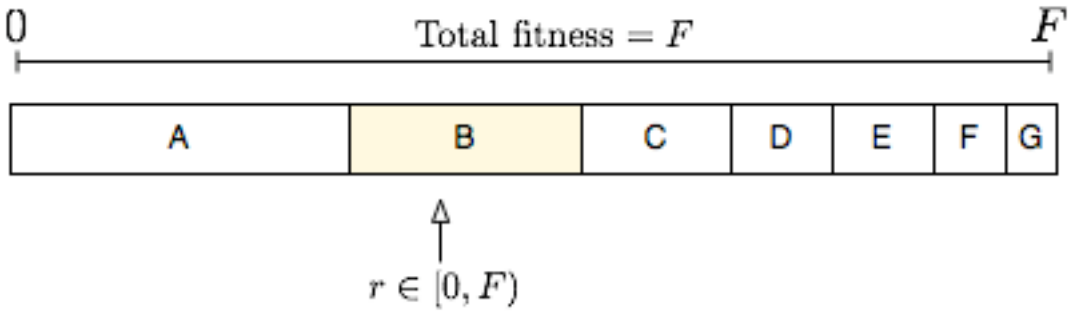


Figure 5.5: Demonstration of Roulette Wheel Selection

*Tournament Selection (TS)*

TS is the simplest selection strategy among the three. This selection strategy selects individuals by generating tournaments between randomly selected individuals and selects the winner of each tournament as a parent. Being very simple this selection strategy also enables controlling the selection pressure with the use of a tournament size parameter $\tau$. As the tournament size increases selection pressure increases thus selection probability for low fitness individuals decreases.

### *5.2.2 Crossover Method*

In GAOSA algorithm a different type of 1-point crossover is implemented. In the standard 1-point crossover described in section 4.1.2 a residue is selected at random and a 90 degree turn over that residue is performed, if collusion occurs than the crossover is rejected. In our implementation given in pseudo code Algorithm 5 a residue is selected at random and 4 fragments are created from 2 parents. However instead of choosing certain angle and a direction the crossover operator exhaustively searches for every possible rotation and symmetric connections between the two fragments. Among the valid combinations the one with lowest energy is selected.

**Algorithm 5:** CrossOver Operator

**input** : *Pair* : Parent Pair to apply the crossover

**begin**

*residue* ← `getRandomResidue()`;

*LeftFragment1* ← `getFragment(`*0,residue, Pair[1]*`)`;

*LeftFragment2* ← `getFragment(`*0,residue, Pair[2]*`)`;

*RightFragment1* ← `getFragment(`*residue+1,size, Pair[1]*`)`;

*RightFragment2* ← `getFragment(`*residue+1,size, Pair[2]*`)`;

*Offspring1* ← `Combine(`*LeftFragment1, RightFragment2*`)`;

*Offspring2* ← `Combine(`*LeftFragment2, RightFragment1*`)`;

**if** $E(Offspring1) \leq E(Pair[1])$ **then**

  `Replace(`$Pair[1]$, *Offspring1*`)`;

**if** $E(Offspring2) \leq E(Pair[2])$ **then**

  `Replace(`$Pair[2]$, *Offspring2*`)`;

### *5.2.3 Replacement Strategy*

After the application of the Crossover Method described above 2 offspring are created from 2 parent solutions. In the HP Model domination of a certain structure can disrupt the diversification of the algorithm. Because the crossover method changes the structure in large chunks the structural similarity between the parents and offspring are conserved. Therefore inserting the offspring in the population without removing the parents will lead to a population of solutions that are structurally similar. To tackle this problem the GAOSA replaces the parent solutions with the offspring only if the offspring's energy is smaller than or equal to the parent's energy value. Using this method a structurally diverse population is conserved.

## *5.3 Parameter Tuning*

To be able to control different components of the GAOSA algorithm each component contains different parameters. In the following sections a list of parameters and their final values are given as well as the methodology through which the parameter tuning is done. Addition to the parameter tuning, a component wise analysis is done. The algorithm is run 20 times with and without the memory and the diversification component and the results are averaged.

### *5.3.1 Parameters of GAOSA*

$\chi$ : Population size( 20 )

$\phi$ : Number of local search iterations( 250 )

$T_{max}$ : Maximum temperature value( 400 )

$T_{min}$ : Minimum temperature value( 160 )

$\eta$ : The factor by which a residue's time in the non-movable list is calculated( 200 )

$\mu$ : The number of iterations the diversification phase is applied( 10 )

$\tau$ : Tournament size( 7 )

$\rho$ : The percentage of parents to be selected( 20% of $\chi$)

$\Pi$ : Type of parent selection strategy $\{"TOUR", "RWS", "SUS"\}$

$pm$ : Mutation Probability

$pc$ : Crossover Probability

To tune the parameters, the problems S1-5 and S1-6 were replicated 20 times and solved with different possible values of every parameter of the algorithm. These results were than averaged. The selection of S1-5 and S1-6 is based on the structural differences between these two structures to tune the parameters without bias. The difference between the native state structures for problem instance S1-5 and S1-6 is given in Figure 5.6.

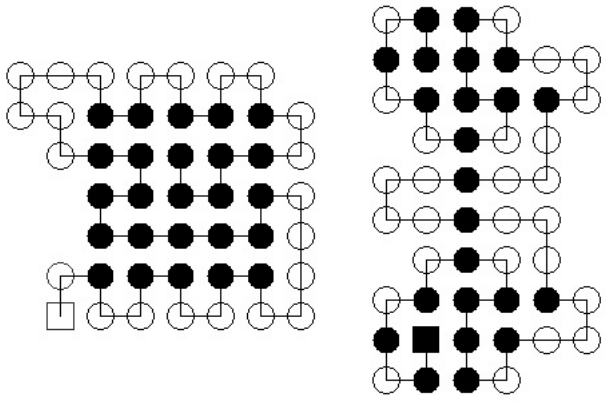

Figure 5.6: (Left) The native state structure for benchmark S1-5. (Right) The native state structure for benchmark S1-6.

### *5.3.2 Experimental Results*

The results in this section are all in terms of seconds of run time unless stated otherwise.

The experiment values for each parameter are given below:

$\chi : [20, 30, 50, 75, 100]$

$\phi : [50, 100, 250, 350, 500]$

$T_{max} : [200, 250, 350, 400]$

$T_{min} : [50, 100, 160, 200]$

$\eta : [10, 50, 150, 200]$

$\tau : [5, 7, 10, 20]$

$\rho : [10\%, 20\%, 30\%, 40\%]$

$\Pi : \{"TOUR", "RWS", "SUS"\}$

$pm : [0.5, 0.75, 0.9, 1]$

$pc : [0.5, 0.75, 0.9, 1]$

Given below are the tables for the averaged run times in seconds for each parameter value.

*Benchmark S1-5*

1. Joint experiment results of the $\chi$ and $\phi$ parameters

$$I(\chi) = [20, 30, 50, 75, 100] \tag{5.4}$$

$$I(\phi) = [50, 100, 250, 350, 500] \tag{5.5}$$

Table 5.1: $\chi$ and $\phi$: Parameter tuning for benchmark problem S 1-5

| $\phi$ \ $\chi$ | 20 | 30 | 50 | 75 | 100 |
|---|---|---|---|---|---|
| 50 | - | - | 1.3 sec | 2.8 sec | 6.2 sec |
| 100 | 2.4 sec | 4.8 sec | - | 5.7 sec | 6.4 sec |
| 250 | 4.2 sec | 6.1 sec | 4.7 sec | 5.8 sec | 11.2 sec |
| 350 | 8.4 sec | 8.2 sec | 22.4 sec | 12.1 sec | 13.6 sec |
| 500 | 15.2 sec | 15.2 sec | 8.3 sec | 14.7 sec | 16.5 sec |

2. Experiments for the $T_{max}$ parameter for benchmark S1-5

$$I(T_{max}) = [200, 250, 350, 400] \tag{5.6}$$

Table 5.2: $T_{max}$ : Parameter tuning for benchmark problem S 1-5

| $T_{max}$ values | 200 | 250 | 350 | 400 |
|---|---|---|---|---|
| Run time ($s$) | 59.8 sec | 6.6 sec | 23.1 sec | 22.6 sec |

3. Experiments for the $T_{min}$ parameter for benchmark S1-5

$$I(T_{min}) = [50, 100, 160, 200] \tag{5.7}$$

Table 5.3: $T_{min}$ : Parameter tuning for benchmark problem S 1-5

| $T_{min}$ values | 50 | 100 | 160 | 200 |
|---|---|---|---|---|
| Run time ($s$) | 16.5 sec | 17.4 sec | 21.9 sec | 32.1 sec |

4. Experiments for the $\eta$ parameter for benchmark S1-5

$$I(\eta) = [10, 50, 150, 200] \tag{5.8}$$

Table 5.4: $\eta$ : Parameter tuning for benchmark problem S 1-5

| $\eta$ values | 10 | 50 | 150 | 200 |
|---|---|---|---|---|
| Run time ($s$) | 14.2 sec | 23.5 sec | 14.1 sec | 28.0 sec |

5. Experiments for the $\Pi$ and $\tau$ parameters for benchmark S1-5

   *The results are set up as:* ["TOUR" [5, 7, 10, 20], "RWS", "SUS"]

Table 5.5: $\Pi$ and $\tau$: Parameter tuning for benchmark problem S 1-5

| $\Pi$ and $\tau$ values | $TOUR : 5$ | $TOUR : 7$ | $TOUR : 10$ | $TOUR : 20$ | $RWS$ | $SUS$ |
|---|---|---|---|---|---|---|
| Run time ($s$) | 20.3 sec | 12.8 sec | 20.6 sec | 19.9 sec | 18.0 sec | 25.7 sec |

6. Experiments for the $\rho$ parameter for benchmark S1-5

$$I(\rho) = [10\%, 20\%, 30\%, 40\%] \tag{5.9}$$

Table 5.6: $\rho$: Parameter tuning for benchmark problem S 1-5

| $\rho$ values | 10% | 20% | 30% | 40% |
|---|---|---|---|---|
| Run time ($s$) | 19.8 sec | 40.3 sec | 44.8 sec | 40.1 sec |

7. Experiments for the $pm$ parameter for benchmark S1-5

$$I(pm) = [0.5, 0.75, 0.9, 1] \tag{5.10}$$

Table 5.7: $pm$: Parameter tuning for benchmark problem S 1-5

| $pm$ values | 0.5 | 0.75 | 0.9 | 1 |
|---|---|---|---|---|
| Run time ($s$) | 38.2 sec | 19.5 sec | 12.3 sec | 22.6 sec |

8. Experiments for the $pc$ parameter for benchmark S1-5

$$I(pc) = [0.5, 0.75, 0.9, 1] \tag{5.11}$$

Table 5.8: $pc$: Parameter tuning for benchmark problem S 1-5

| $pc$ values | 0.5 | 0.75 | 0.9 | 1 |
|---|---|---|---|---|
| Run time ($s$) | 30.1 sec | 24.8 sec | 12.8 sec | 13.1 sec |

9. Experiments for the Exponential and Linear Cooling Schedules for benchmark S1-5

   This experiment was repeated for two different values of the $T_{max}$ parameter: 250 and 400 with $T_{min}$ fixed at 160

   *The results are setup in the following structure:*

   ["ECS" with $\alpha$ = [0.9 , 0.98], "LCS" with number of steps = [5 , 10]]

Table 5.9: Cooling Schedules: Parameter tuning for benchmark problem S 1-5

| | | Cooling Schedules | | | |
|---|---|---|---|---|---|
| | | $ECS : 0.9$ | $ECS : 0.98$ | $LCS : 5$ | $LCS : 10$ |
| $T_{max}$ | 250 | 37.2 sec | 34.0 sec | 73.6 sec | 16.1 sec |
| | 400 | 30.7 sec | 24.0 sec | 66.7 sec | 71.1 sec |

10. Component Experiments for benchmark S1-5

    *The results are structured as:*

    With Memory Component with tabu time 10 steps (WM)

    Without Memory Component (WOM)

    With Diversification of 5 steps (WD)

    Without Diversification (WOD)

Table 5.10: Component analysis for benchmark S1-5. Combinations of different components and run time results for each combination

| Components | $WM : 10$ | $WOM$ | $WD : 5$ | $WOD$ |
|---|---|---|---|---|
| Run time ($s$) | 11.75 sec | 18.15 sec | 66.5 sec | 23.75 sec |

*Benchmark S1-6*

1. Experiments for the $T_{max}$ parameter for benchmark S1-6

$$I(T_{max}) = [200, 250, 350, 400] \tag{5.12}$$

Table 5.11: $T_{max}$ : Parameter tuning for benchmark problem S 1-6

| $T_{max}$ values | 200 | 250 | 350 | 400 |
|---|---|---|---|---|
| Run time ($s$) | 21.0 sec | 9.2 sec | 20.5 sec | 22.3 sec |

2. Experiments for the $T_{min}$ parameter for benchmark S1-6

$$I(T_{min}) = [50, 100, 160, 200] \tag{5.13}$$

Table 5.12: $T_{min}$ : Parameter tuning for benchmark problem S 1-6

| $T_{min}$ values | 50 | 100 | 160 | 200 |
|---|---|---|---|---|
| Run time ($s$) | 14.3 sec | 18.6 sec | 39.6 sec | 72.2 sec |

3. Experiments for the $\eta$ parameter for benchmark S1-6

$$I(\eta) = [10, 50, 150, 200] \tag{5.14}$$

Table 5.13: $\eta$ : Parameter tuning for benchmark problem S 1-6

| $\eta$ values | 10 | 50 | 150 | 200 |
|---|---|---|---|---|
| Run time ($s$) | 18.7 sec | 25.5 sec | 33.7 sec | 17.5 sec |

4. Experiments for the $\Pi$ and $\tau$ parameters for benchmark S1-6

   *The results are set up as:*

   ["TOUR" [5, 7, 10, 20], "RWS", "SUS"]

Table 5.14: $\Pi$ and $\tau$: Parameter tuning for benchmark problem S 1-6

| $\Pi$ and $\tau$ values | $TOUR:5$ | $TOUR:7$ | $TOUR:10$ | $TOUR:20$ | $RWS$ | $SUS$ |
|---|---|---|---|---|---|---|
| Run time ($s$) | 27.5 sec | 30.2 sec | 17.8 sec | 23.8 sec | 46.3 sec | 52.8 sec |

5. Experiments for the $\rho$ parameter for benchmark S1-6

$$I(\rho) = [10\%, 20\%, 30\%, 40\%] \tag{5.15}$$

Table 5.15: $\rho$: Parameter tuning for benchmark problem S 1-6

| $\rho$ values | 10% | 20% | 30% | 40% |
|---|---|---|---|---|
| Run time ($s$) | 8.3 sec | 9.6 sec | 11.4 sec | 11.4 sec |

6. Experiments for the $pm$ parameter for benchmark S1-6

$$I(pm) = [0.5, 0.75, 0.9, 1] \tag{5.16}$$

Table 5.16: $pm$: Parameter tuning for benchmark problem S 1-6

| $pm$ values | 0.5 | 0.75 | 0.9 | 1 |
|---|---|---|---|---|
| Run time ($s$) | 19.9 sec | 19.0 sec | 17.0 sec | 25.3 sec |

7. Experiments for the *pc* parameter for benchmark S1-6

$$I(pc) = [0.5, 0.75, 0.9, 1] \tag{5.17}$$

Table 5.17: *pc*: Parameter tuning for benchmark problem S 1-6

| *pc* values | 0.5 | 0.75 | 0.9 | 1 |
|---|---|---|---|---|
| Run time ($s$) | 60.8 sec | 55.4 sec | 35.9 sec | 43.4 sec |

8. Experiments for the Exponential and Linear Cooling Schedules for benchmark S1-5

   This experiment was repeated for two different values of the $T_{max}$ parameter: 250 and 400 with $T_{min}$ fixed at 160

   *The results are setup in the following structure:*

   ["ECS" with $\alpha$ = [0.9 , 0.98], "LCS" with number of steps = [5 , 10]]

Table 5.18: Cooling Schedules: Parameter tuning for benchmark problem S 1-6

| | Cooling Schedules | | | |
|---|---|---|---|---|
| | $ECS : 0.9$ | $ECS : 0.98$ | $LCS : 5$ | $LCS : 10$ |
| $T_{max}$ 250 | 11.5 sec | 16.8 sec | 12.8 sec | 9.1 sec |
| 400 | 79.4 sec | 51.6 sec | 100.0 sec | 93.3 sec |

9. Component Experiments for benchmark S1-6

   *(*The results are structured as:)

   With Memory Component with tabu time 10 steps (WM)

   Without Memory Component (WOM)

   With Diversification of 5 steps (WD)

   Without Diversification (WOD)

Table 5.19: Component analysis for benchmark S1-6. Combinations of different components and run time results for each combination

| Components | $WM:10$ | $WOM$ | $WD:5$ | $WOD$ |
|---|---|---|---|---|
| Run time ($s$) | 5.2 sec | 9.7 sec | 27.6 sec | 11.3 sec |

*Comparative Graphs*

In this section, graphs are given to show the effects of each parameter value on the run time of the algorithm. The graphs for the two problem instances, S1-5 and S1-6, are given side by side for comparison. Further discussion of these results are given in the next chapter.

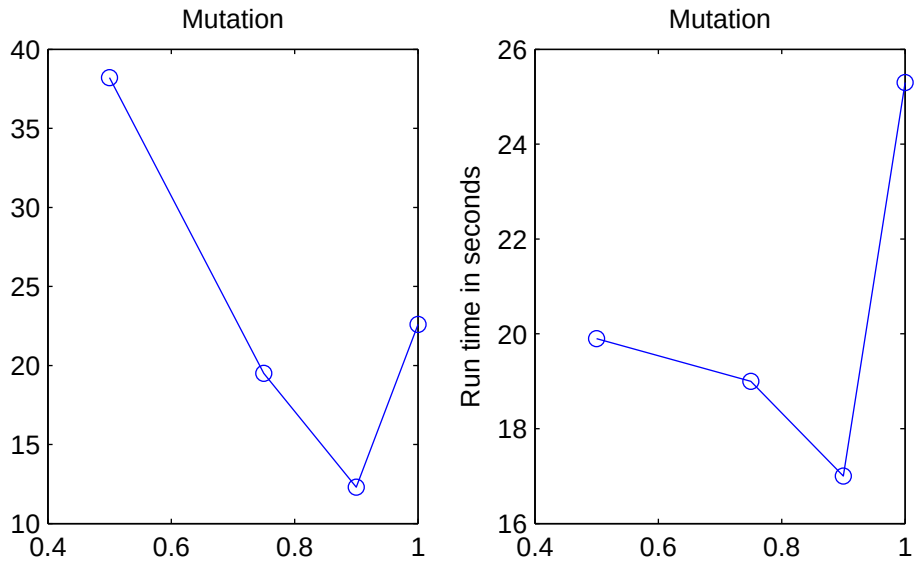


Figure 5.7: Mutation Probability parameters for S1-5 (Left) and S1-6 (Right)

As it can be observed from Figure 5.7, mutation probability has a substantial effect on the run time. For both problems, the value 0.9 performs better than the other values which are experimented with.

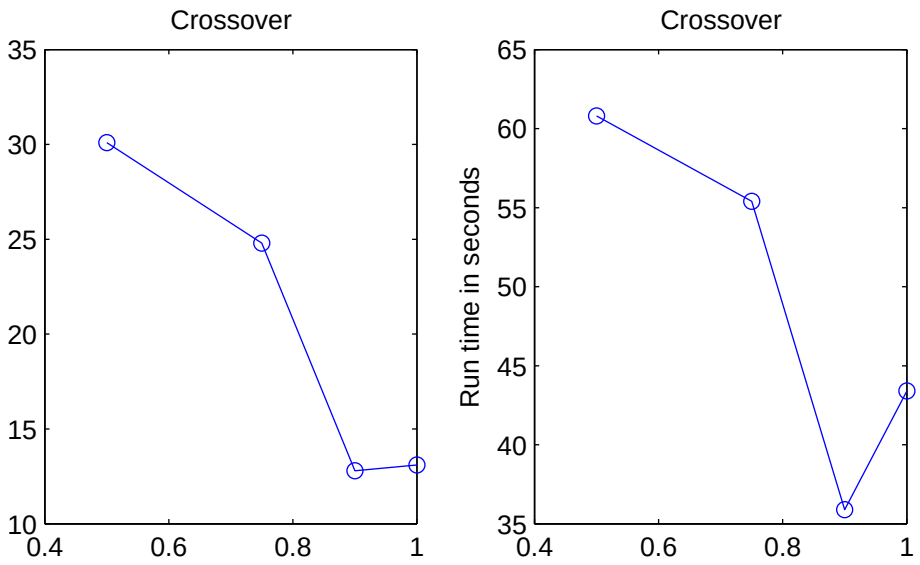


Figure 5.8: Crossover Probability parameters for S1-5 (Left) and S1-6 (Right)

The crossover probability is also a very important factor that affects the run time. This parameter sets the probability with which a new solution will be created from two parent solutions. Among the values that are tested, 0.9 gave the best results for both problem instances (see Figure 5.8).

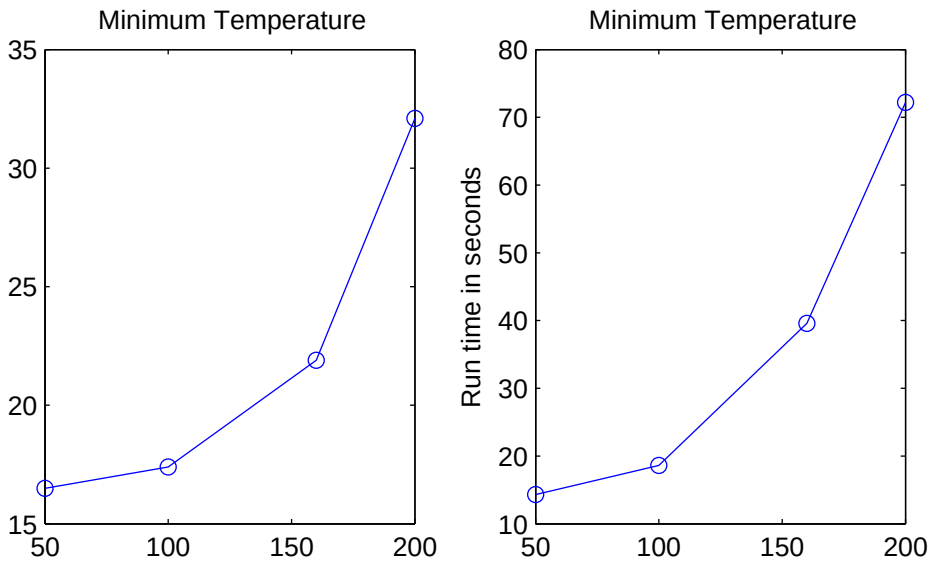


Figure 5.9: Minimum Temperature parameters for S1-5 (Left) and S1-6 (Right)

The minimum temperature parameter identifies the lower bound on the acceptance probability for worsening moves. Therefore high values may result in promising solutions not being further explored and low values may cause premature convergence. In our experiments, a value of 50 performs better for both problem instances although the run time difference between the two parameter values 50 and 100 is very minimal (Figure 5.9). These values translate to the acceptance probabilities of 0.00004257 and 0.00652, respectively, for a -1 energy worsening move using the equation (4.1).

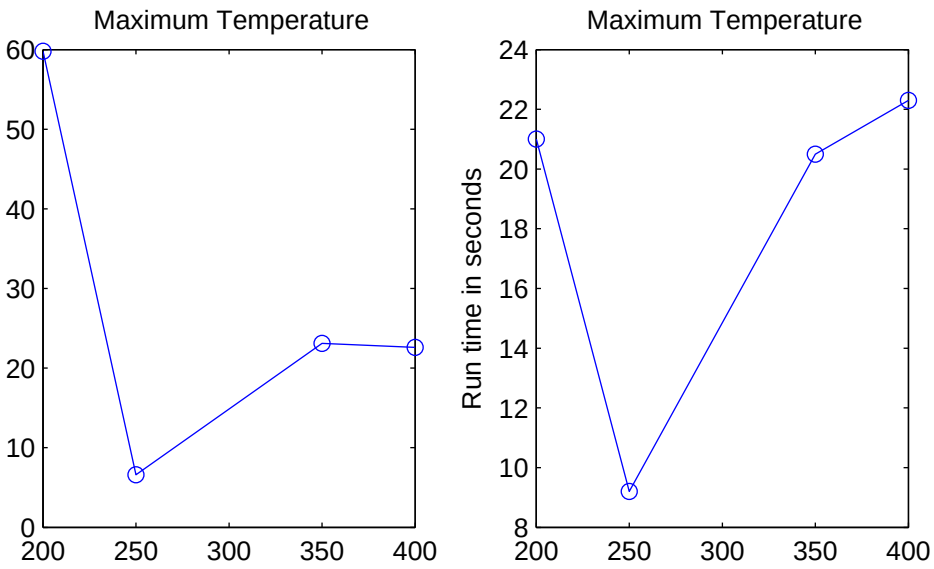


Figure 5.10: Maximum Temperature parameters for S1-5 (Left) and S1-6 (Right)

Similar to the minimum temperature parameter, the maximum temperature parameter is used to set the upper bound on the acceptance probability for worsening moves, thus it is a very important factor for the run time of the algorithm. If the maximum temperature parameter is set too high, the probability of accepting worsening moves increases which can cause the algorithm to lose valueable solutions. Low values can cause premature convergence which consequently increases the run time. Figure 5.10 suggests that for both solutions a value of 250 performs best for both problem instances. A value of 250 translates to an acceptance probability of 0.133603015 for -1 energy worsening move according to equation (4.1).

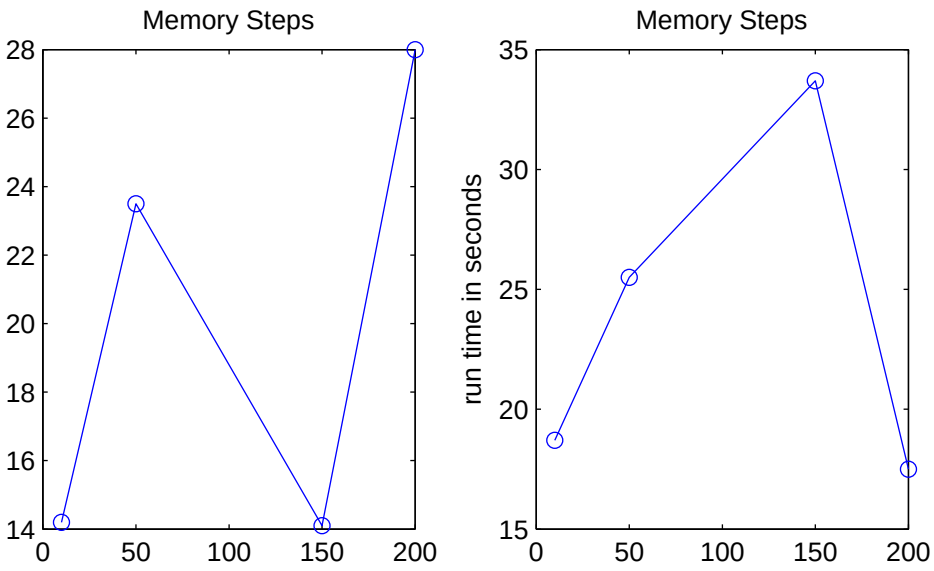


Figure 5.11: Memory Step parameters for S1-5 (Left) and S1-6 (Right)

The memory component aims to preserve local structure by putting the residues that lower the energy value of the structure into a tabu list. The memory step parameter sets the amount of time that these residues stay in the tabu list. High values of this parameter may result in too many residues in the tabu list, which will hinder the ability of the algorithm to change the structure of a given solution. Low values can decrease the effectiveness of the memory component by changing a preserved local structure too quickly. As can be seen from Figure 5.11, a value of 10 steps performs better for both problem instances.

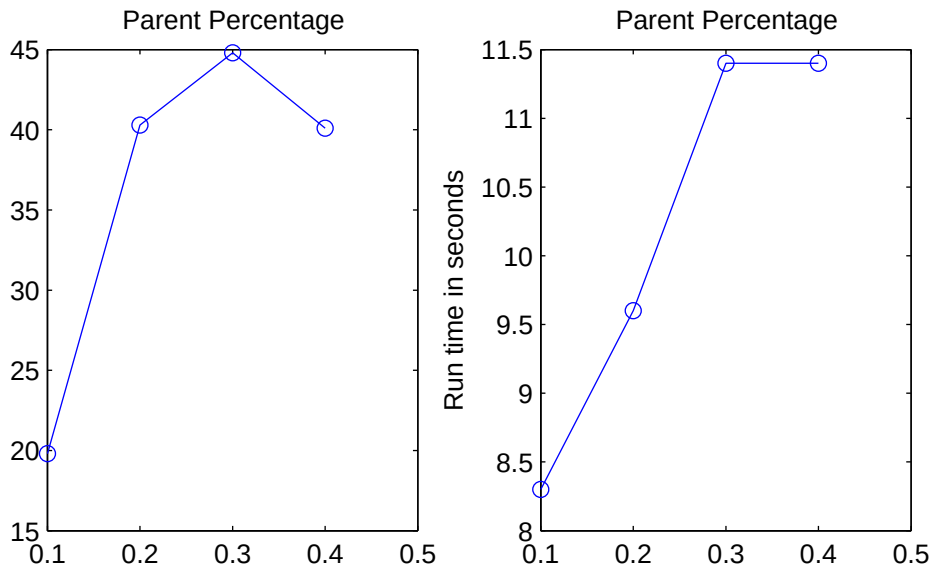


Figure 5.12: Parent Percentage parameters for S1-5 (Left) and S1-6 (Right)

Before the crossover operator can be applied, a set of parents are selected. Parent percentage parameter sets what percentage of the population will be selected as parent solutions. The crossover operator is the most expensive operator in the GAOSA algorithm, therefore high values for this parameter while increasing the quality of the parents that are selected will increase the run time of the overall algorithm. However, lower values may cause the quality of the selected parents to decrease. In the two problem instances, 10% performs substantially better than other tested values (see Figure 5.12).

The summary of the best performing parameter values are given below.

*Tuned Parameters*

From the analysis we performed, the best performing parameters for S1-5 and S1-6 are: $\chi$ : Population size( 20 )

$\phi$ : Number of local search iterations( 250 )

$T_{max}$ : Maximum temperature value( 250 )

$T_{min}$ : Minimum temperature value( 100 )

$\eta$ : The factor by which a residue's time in the non-movable list is calculated( 10 )

$\mu$ : The number of iterations the diversification phase is applied( 5 )

$\tau$ : Tournament size( 7 )

$\rho$ : The percentage of parents to be selected( 10% of $\chi$)

$\Pi$ : Type of parent selection strategy $\{"TOUR", "RWS", "SUS"\}$ ("TOUR")

$pm$ : Mutation Probability (0.9)

$pc$ : Crossover Probability (0.9)

# Chapter 6

# RESULTS

In this chapter the results of the GAOSA on 2 different classes of benchmarks will be discussed and the experimental setup will be explained. Furthermore a discussion of the results will be given in the end of this chapter.

## *6.1 Standard Benchmarks*

There are eleven benchmark sequences for 2D HP Model. These benchmark sequences are widely used and thus provides a good basis of comparison between algorithms created for the 2D HP Model. A table of these sequences and their optimum energies is given in table 6.1

Table 6.1: 2D HP Model standard benchmark sequences. [a] Known optimal energy value.

| ID | Length | $E^{*a}$ | Sequence |
|---|---|---|---|
| S1-1 | 20 | -9 | $(HP)_2PH_2PHP_2HPH_2P_2HPH$ |
| S1-2 | 24 | -9 | $H_2(P_2H)_7H$ |
| S1-3 | 25 | -8 | $P_2HP_2(H_2P_4)_3H_2$ |
| S1-4 | 36 | -14 | $P_3H_2P_2H_2P_5H_7P_2H_2P_4H_2P_2HP_2$ |
| S1-5 | 48 | -23 | $P_2H(P_2H_2)_2P_5H_{10}P_6(H_2P_2)_2HP_2H_5$ |
| S1-6 | 50 | -21 | $H_2(PH)_3PH_4PH(P_3H)_2P_4H(P_3H)_2PH_4(PH)_4H$ |
| S1-7 | 60 | -36 | $P_2H_3PH_8P_3H_{10}PHP_3H_{12}P_4H_6PH_2PHP$ |
| S1-8 | 64 | -42 | $H_{12}(PH)_2(P_2H_2)_2P_2HP_2H_2PPH_2P_2HP_2(H_2P_2)_2(HP)_2H_{12}$ |
| S1-9 | 85 | -53 | $H_4P_4H_{12}P_6(H_{12}P_3)_3HP_2(H_2P_2)_2HPH$ |
| S1-10 | 100 | -50 | $P_3H_2P_2H_4P_2H_3(PH_2)_2PH_4P_8H_6P_2H_6P_9HPH_2PH_{11}P_2H_3$ $PH_2PHP_2HPH_3P_6H_3$ |
| S1-11 | 100 | -48 | $P_6HPH_2P_5H_3PH_5PH_2P_4H_2P_2H_2PH_5PH_{10}PH_2PH_7$ $P_{11}H_7P_2HPH_3P_6HPH_2$ |

To be able to compare GAOSA with respect to the algorithms REMC, ACO-HPPFP-3, PERM the following experimental protocol is used. For sequences of $size <= 30$, 100 independent runs, for sequences with $30 < size <= 60$, 50 independent runs and for $size >= 60$, 20 independent runs were performed and the results were averaged. A comparison is given in table 6.1.

Table 6.2: Results for the 2D Benchmark Problems. [a] Known optimal energy value.

| ID | $E^{*a}$ | $PERM_{texp}$ | ACO-HPPFP-3 | $REMC_{pm}$ | $REMC_m$ | GAOSA |
|---|---|---|---|---|---|---|
| S1-1 | **-9** | **-9**(< 1 sec) | **-9**(< 1 sec) | **-9**(< 1 sec) | **-9**(< 1 sec) | **-9**(< 1 sec) |
| S1-2 | **-9** | **-9**(< 1 sec) | **-9**(< 1 sec) | **-9**(< 1 sec) | **-9**(< 1 sec) | **-9**(< 1 sec) |
| S1-3 | **-8** | **-8**(2 sec) | **-8**(2 sec) | **-8**(< 1 sec) | **-8**(< 1 sec) | **-8**(< 1 sec) |
| S1-4 | **-14** | **-14**(< 1 sec) | **-14**(4 sec) | **-14**(< 1 sec) | **-14**(< 1 sec) | **-14**(3 sec) |
| S1-5 | **-23** | **-23**(2 sec) | **-23**(1 min) | **-23**(< 1 sec) | **-23**(< 1 sec) | **-23**(8 sec) |
| S1-6 | **-21** | **-21**(3 sec) | **-21**(15 sec) | **-21**(< 1 sec) | **-21**(< 1 sec) | **-21**(8 sec) |
| S1-7 | **-36** | **-36**(4 sec) | **-36**(20 min) | **-36**(10 sec) | **-36**(13 sec) | **-36**(70 sec) |
| S1-8 | **-42** | **-42**(78 hrs) | **-42**(1.5 hrs) | **-42**(5 sec) | **-42**(6 sec) | **-42**(390 sec) |
| S1-9 | **-53** | **-53**(1 min) | **-53**(%20 of runs 1 day) | **-53**(2 min) | **-53**(38 sec) | **-53**(20 min) |
| S1-10 | **-50** | **-50**(20 min) | -49(12 hrs) | **-50**(3.5 min) | **-50**(8 min) | **-50**(40 min) |
| S1-11 | **-48** | **-48**(8 min) | -47(10 hrs) | **-48**(1 min) | **-48**(1.2 min) | **-48**(3 hrs) |

### *6.2 Z-Structure Benchmarks*

GAOSA performance is also tested on an artificial benchmark. Z-Structures, proposed by [1], have the property of having a unique ground-state structure. The Z-Structures are generated given the following formula:

$$Z_k = (HP)^u(PH)^d \tag{6.1}$$

where $u = \lceil k/2 \rceil$ and $d = \lfloor k/2 \rfloor$. These structures are shown to have a unique ground-state structure when k is even [1]. A sample unique ground-state of $Z_8$ is given in figure 6.1.

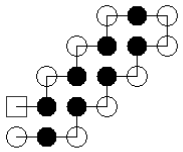

Figure 6.1: Unique Ground State Structure of the Z-8 Protein.

The comparison between PERM, REMC and GAOSA algorithms' performance on the Z-structures is given in table 6.2.

Table 6.3: Results for the Z Structures. ($^a$ Known optimal energy value)

| ID | $E^{*a}$ | $PERM_{texp}$ | $REMC_{pm}$ | $REMC_m$ | GAOSA |
|---|---|---|---|---|---|
| Z-4 | **-3** | **-3**($<$ 1 sec) | **-3**($<$ 1 sec) | **-3**($<$ 1 sec) | **-3**($<$ 1 sec) |
| Z-8 | **-7** | **-7**($<$ 1 sec) | **-7**($<$ 1 sec) | **-7**($<$ 1 sec) | **-7**($<$ 1 sec) |
| Z-12 | **-11** | **-11**($<$ 1 sec) | **-11**($<$ 1 sec) | **-11**($<$ 1 sec) | **-11**($<$ 1 sec) |
| Z-16 | **-15** | **-15**(3 sec) | **-15**($<$ 1 sec) | **-15**($<$ 1 sec) | **-15**($<$ 1 sec) |
| Z-20 | **-19** | **-19**(51 min) | **-19**($<$ 1 sec) | **-19**($<$ 1 sec) | **-19**(3 sec) |
| Z-24 | **-23** | **-23**(49 hrs) | **-23**($<$ 1 sec) | **-23**($<$ 1 sec) | **-23**(1.7 sec) |
| Z-28 | **-27** | -26 | **-27**($<$ 1 sec) | **-27**($<$ 1 sec) | **-27**(2 sec) |
| Z-32 | **-31** | -29 | **-31**($<$ 1 sec) | **-31**($<$ 1 sec) | **-31**(20 sec) |
| Z-36 | **-35** | -31 | **-35**($<$ 1 sec) | **-35**($<$ 1 sec) | **-35**(32 sec) |
| Z-40 | **-39** | -34 | **-39**($<$ 1 sec) | **-39**($<$ 1 sec) | **-39**(2 min) |

### *6.3 Discussion of Results*

In the given benchmark sets, GAOSA has a competitive performance and it correctly identifies every ground-state energy for each sequence. In Section 5.3 effects of different parameters are analyzed. The results provide insights on the effectiveness of each component of the GAOSA algorithms. The mutation operator of GAOSA consists

of a local search phase that is based on simulated annealing. Although there are multiple parameters in the experiments that are related to the mutation operator, the most indicative parameter is the mutation probability ($pm$). The mutation probability sets the probability with which the mutation operator is applied to a given solution. High values of this parameter increase the influence of the mutation operator. Given the results in Tables 5.7 and 5.16, GAOSA performs best when $pm = 0.9$. The performance difference between the lower and higher values of the $pm$ parameter is significant, demonstarting the positive effect of our mutation operator on the performance of GAOSA. However, an interesting result can also be observed when the influence of the mutation operator is set to higher values. With a value of 1 the $pm$ parameter causes deterioration of the performance. The reason for the decrease in the performance, is the preservation of solutions. In the mutation operator, the Monte Carlo criterion given in equation (4.1) is used to accept moves. According to this equation, worsening moves are accepted with a certain probability. When the mutation operator is applied with probability 1, at higher temperature values where it is more likely to accept worsening moves, the algorithm loses promising solutions which in turn deteriorates performance. The value of 0.9 provides a balance for this trade-off. The $pm$ parameter is a good indicator for the influence of the mutation operator, however our memory component, which is utilized in the mutation operator, and its influence on the performance cannot be observed from the $pm$ parameter.

To better understand the effect of our memory component to the overall performance of GAOSA, a component wise experiment is performed. The results of this experiment for the two problem instances are given in Tables 5.10 and 5.19. The results suggest that the GAOSA performs better with the memory component. The memory component's purpose is to preserve local structures that form throughout the folding process. The amount of time that a local structure is preserved is set by the tabu time parameter ($\eta$). From Figure 5.11 it can be observed that this parameter has a significant effect on the performance. The reason for this effect is that after

local structure is fixed by the memory component, it creates a zipper effect and reduces the size of the solution space. However if the tabu time parameter is kept too high then the solution space cannot be explored completely which causes longer run times. On the other hand if the tabu time parameter is kept too low, local structures that are identified by the algorithm are not preserved long enough to affect the size of the solution space. The parameter value that balances this trade-off is found to be 10 from our experiments, and at this value, GAOSA performed significantly better.

Another parameter that sets the influence of a component is the crossover probability ($pc$). Similar to the mutation probability, this parameter is more indicative of the overall effect of the crossover operator to the performance of the algorithm. As can be seen from Tables 5.8 and 5.17, in both problem instances the value 0.9 rendered the best results. This high probability value implies that crossover operator works well in our implementation.

The crossover operator is used to introduce new individuals to the population using the properties of the fit solutions in a generation. Thus parent selection is a crucial part of the crossover operator. During parameter tuning, three different parent selection strategies were tested: Tournament Selection, Roulette Wheel Selection (RWS) and Stochastic Universal Sampling (SUS). The outcome for these tests are given in Tables 5.5 and 5.14. The results suggest that Tournament Selection performs better among the three parent selection strategies. Tournament Selection provides complete control over the selection pressure of the parents. This means that by choosing the tournament size large, the probability of selecting an unfit parent can be decreased and inversely by choosing the tournament size small, the probability of selecting an unfit parent increases. The simplicity and control that Tournament Selection provides cause a more diverse set of parents to be selected, which in turn increase diversification of solutions. On the other hand RWS and SUS methods increase the similarity between the individuals in the population which causes premature convergence to a

structure. Through the usage of Tournament Selection method, GAOSA algorithm attempts to tackle the problem of local optima, however the crossover operator is not enough to solve this problem. For this reason, we introduced a diversification phase in this thesis.

Premature convergence to local optima decreases the performance of the algorithm drastically. The effect of premature convergence can be seen on the convergence graph for benchmark S1-5 and S1-7 given in figure 6.2.

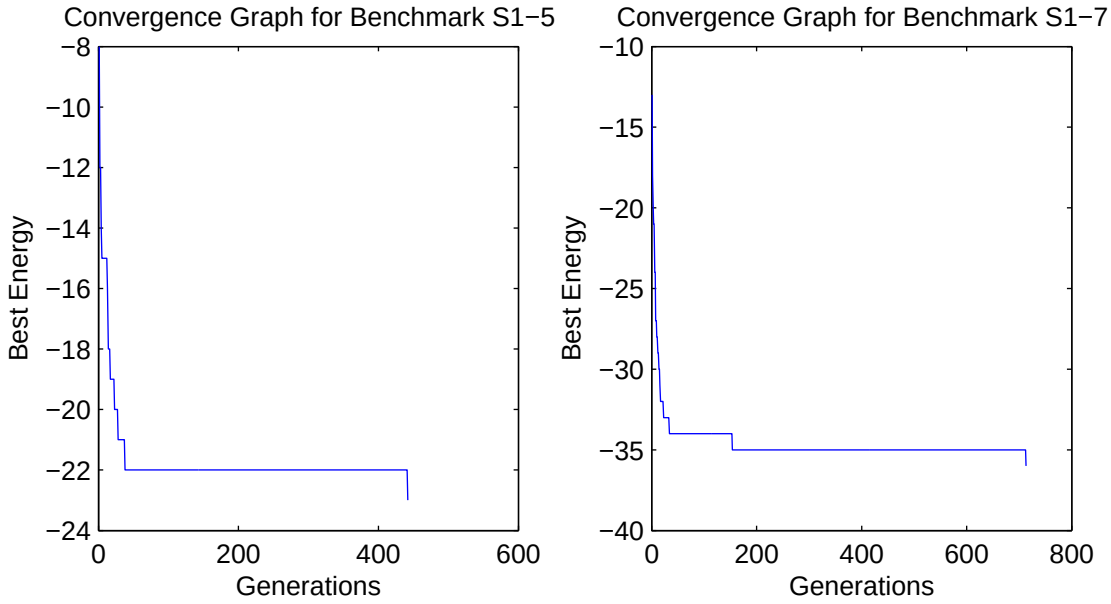


Figure 6.2: Convergence graphs for problem instances S1-5 (Left) and S1-7 (Right)

The long stabilized lines in the figures above are the stages that our algorithm cannot improve the best solution found up to that time. To address this problem, first a solution that has converged to local optima must be detected, diversified and intensified again. To be able to understand the introduced diversification component's effectiveness, a component wise analysis is done and the results are given in Tables 5.10 and 5.19. The aim of the diversification component was to make use of the structural implications of the HP Model given in Section 3.3 to solve the premature

convergence problem structurally. However the result of the experiments showed that the diversification component did not perform well. In further analysis, the reason for the poor performance of our diversification component is found to be that during the detection of prematurely converged solution, our algorithm diversifies the solution even if the solution's structure is close to the global optima. Since there are currently no effective methods to understand whether a given solution is structurally similar to the global optimum structue or not, our method cannot differentiate the promising and the prematurely converged solutions.

# Chapter 7

# CONCLUSION

In this thesis we propose a new meta-heuristic algortihm GAOSA that is based on the Genetic Algortihm and Simulated Annealing for the 2D HP Model. Compared to the state of the art, GAOSA manages to get competitive results on a standard set of benchmarks and gets good results on a second set of artificial benchmarks proposed in [1]. The GAOSA is outperformed by REMC and PERM which are currently the best algorithms in the literature in both benchmarks with large sequences. However it outperforms PERM on S1-8 sequence from the first benchmark and every instance in the second benchmark problems. The GAOSA currently is able to find the ground-state conformation of every sequence in the mentioned benchmark sets, however as sequence sizes grow the reported times increase which implies problems in the diversification strategies employed by the algorithm as mentioned in Section 6.3. The introduced memory component shows promising results and is still open to further improvement in both the detection of the local structures and the sophistication of the tabu time. In our study we provide an analysis of the different components found in the literature and point to certain shortcomings. We also propose new and problem specific methods to address these shortcomings. The methods we propose are inspired from the original PFP to address the issues of intensification and diversification of solutions that are common to any heuristic approach. We show in the research that some of these methods are simple yet very effective.

## *7.1 Future Research*

The HP Model is a relatively new research area, and there are still unexplored areas in this Model. One of these areas is local structure detection. Stable local structures

play an important role in Protein Folding by collapsing early in the process and thus reducing the complexity of the solution space. However currently to our knowledge no algorithm has proposed a way to effectively and efficiently detect and use possible local structures in the HP Model. When analyzed the optimal solution of the first problem instance in the standard benchmark, it can be observed that the structure is made up of 3 local structures shown in the figure 7.1

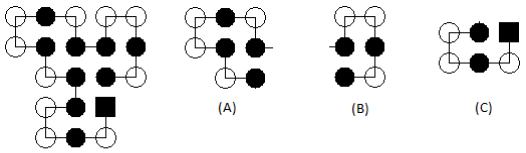


Figure 7.1: Different local structures founded in the ground-state conformation of the benchmark S1-1

These local structures forms in the early iteration of the algorithm however currently there is no method implemented to detect these structures. If a stable local structure could be detected successfully, this information can be used to fix these local structures and confine the solution space. For future work a way of detecting local structures will be pursued. A possible strategy can keep a count of specific H-H contacts in a global memory and with the assumption of the most frequently found contacts are part of a local structure, any move that attempts to break these contacts could be rejected as a strategy to preserve the local structure. Although this idea is in its early stages it can be developed in a working model for future work. A demonstration of the basis of this idea is provided in figure 7.2.

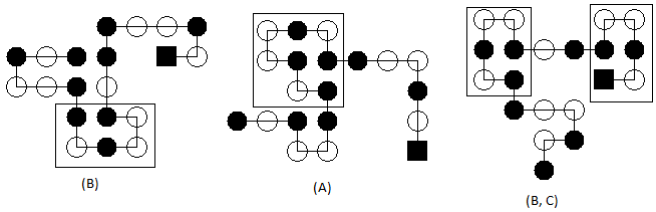


Figure 7.2: A snapshot of the population in the early iterations for the benchmark S1-1.

After the GAOSA algorithm is further improved, a possible direction for future work would be to work on a 3D version of the algorithm which is straightforward after the adaptation of the Pull Move function [23] and the Crossover operator.

# BIBLIOGRAPHY

[1] Oswin Aichholzer, David Bremner, Erik D. Demaine, Henk Meijer, Vera Sacristán, and Michael Soss. Long proteins with unique optimal foldings in the h-p model. *Comput. Geom. Theory Appl.*, 25(1-2):139–159, 2003.

[2] James E. Baker. Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pages 14–21, Hillsdale, NJ, USA, 1987. L. Erlbaum Associates Inc.

[3] B Berger and T Leighton. Protein folding in the hydrophobic-hydrophilic (hp) is np-complete. *Proceedings of the second annual international conference on Computational molecular biology*, 5(1):27–40, 1998.

[4] T. Beutler and K. Dill. A fast conformational search strategy for finding low energy structures of model proteins. *Protein Sci.*, 5:2037–2043, 1996.

[5] C. Chira, D. Horvath, and D. Dumitrescu. An Evolutionary Model Based on Hill-Climbing Search Operators for Protein Structure Prediction. *Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics*, pages 38–49, 2010.

[6] D. G. Covell and R. L. Jernigan. Conformations of folded proteins in restricted spaces. *Biochemistry*, 29(13):3287–3294, Apr 1990.

[7] P Crescenzi, D Goldman, C Papadimitriou, A Piccolboni, and M Yannakakis. On the complexity of protein folding. *Proceedings of the second annual international conference on Computational molecular biology*, pages 61–62, 1998.

[8] K. Dill. Theory for the folding and stability of globular proteins. *Biochemistry*, 24(6):1501–1509, Mar 1985.

[9] F. Glover et al. Tabu search-part I. *ORSA journal on Computing*, 1(3):190–206, 1989.

[10] MT Gurler, CC Crabb, DM Dahlin, and J Kovac. Effect of bead movement rules on the relaxation of cubic lattice models of polymer chains. *Macromolecules*, 16(3):398–403, 1983.

[11] W Hart and S Istrail. Robust proofs of np-hardness for protein folding: general lattices and energy potentials. *Journal of Computational Biology*, 4:1–22, 1997.

[12] HJ Hilhorst and JM Deutch. Analysis of monte carlo results on the kinetics of lattice polymer chains with excluded volume. *The Journal of Chemical Physics*, 63(12):5153–5161, 1975.

[13] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. The MIT Press, April 1992.

[14] Hsiao-Ping Hsu, Vishal Mehra, Walter Nadler, and Peter Grassberger. Growth-based optimization algorithm for lattice heteropolymers. *Phys Rev E Stat Nonlin Soft Matter Phys*, 68(2 Pt 1):021113, Aug 2003.

[15] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[16] Kit Fun Lau and Ken A. Dill. A lattice statistical mechanics model of the conformational and sequence spaces of proteins. *Macromolecules*, 22(10):3986–3997, October 1989.

[17] N Lesh, M Mitzenmacher, and S Whitesides. A complete and effective move set for simplified protein folding. *RECOMB '03: Proceedings of the seventh annual international conference on Research in computational molecular biology*, pages 188–195, 2003.

[18] C. Levinthal. How to fold graciously. *University of Illinois Press*, pages 22–24, 1969.

[19] F. Liang and W. H. Wong. Evolutionary Monte Carlo for protein folding simulations. *Journal Of Computational Physics*, 115:3374–3380, August 2001.

[20] Brad L. Miller and David E. Goldberg. Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, 9:193–212, 1995.

[21] C. Rego, H. Li, and F. Glover. A filter-and-fan approach to the 2d lattice model of the protein folding problem. 2006.

[22] A Shmygelska and H Hoos. An ant colony optimisation algorithm for the 2d and 3d hydrophobic polar protein folding problem. *BMC Bioinformatics*, 6:30, 2005.

[23] Chris Thachuk, Alena Shmygelska, and Holger Hoos. A replica exchange monte carlo algorithm for protein folding in the hp model. *BMC Bioinformatics*, 8(1):342, 2007.

[24] 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.

[25] R. Unger and J. Moult. Genetic algorithms for protein folding simulations. *J. Mol. Biol.*, 231:75–81, 1993.

[26] PH Verdier and WH Stockmayer. Monte carlo calculations on the dynamics of polymers in dilute solution. *The Journal of Chemical Physics*, 36:227–235, 1962.