Neural Computing and Applications (2020) 32:3117–3134
https://doi.org/10.1007/s00521-019-04447-8

ORIGINAL ARTICLE

Optimization of protein folding using chemical reaction optimization in HP cubic lattice model

Md. Rafiqul Islam[1] · Resheta Ahmed Smrity[1] · Sajib Chatterjee[1] · Md. Riaz Mahmud[1]

Received: 15 November 2017 / Accepted: 19 August 2019 / Published online: 5 September 2019

Abstract
Protein folding optimization is a very important and tough problem in computational biology. For solving this problem, a population-based metaheuristic algorithm named chemical reaction optimization (CRO) with HP cubic lattice model has been proposed in this paper. The proposed algorithm is combined with evolution and H&P compliance mechanisms which are responsible for increasing the performance of the algorithm. The evolution mechanism improves the performance of each individual solution. On the other hand, the H&P compliance mechanism tries to place the H monomer close to the center and place the P monomer as far as possible from the center of the related structure. The algorithm also applies four reactant operations of typical CRO algorithm decomposition, on-wall ineffective collision, synthesis and inter-molecular ineffective collision to solve the problem efficiently. The reactants or mechanisms may cause overlapping of the corresponding solutions. The algorithm also includes a repair mechanism which transforms invalid solutions into valid ones by removing overlapping in cubic lattice points. This algorithm has been tested over some sets of sequences and it shows very good performance.

Keywords Protein folding optimization · Chemical reaction optimization algorithm · Hydrophobic-polar model · Cubic lattice · H&P · Evolution mechanism · Repair mechanism

1 Introduction

Protein folding problem is one of the fundamental problems in computational biology. Simplified HP cubic lattice model plays an important role in protein structure prediction (PSP) and protein folding optimization (PFO) problem. The PSP problem represents how to predict the native structure of a protein from its amino acid sequence. The PFO problem represents a computational problem for simulating the protein folding process within the protein structure prediction. The protein structure is the result of the so-called protein folding process where the final structure is obtained from its primary unfolded chain of amino acids [1]. The primary formation consists of a sequence of amino acids with a peptide bond which are interconnected. Secondary structure refers to the coiling or folding of the primary sequence. There are two types of secondary structures observed in proteins: alpha (α) helix and beta (β) pleated sheet. The tertiary structure of a polypeptide or protein is the three-dimensional arrangement of the amino acids in the form of secondary structure within a single polypeptide chain. On the contrary, quaternary structure is the arrangement of the amino acids with multiple polypeptide chains [2]. In general, the length of the sequence can vary from one protein to another. Particularly, from the tertiary structure of a protein all functionalities of that protein can be determined. The functionalities include physical fitness, illness, the aging process of human body, etc. Moreover, protein structure changes over time. This can produce infeasible structure of the protein in human body. Many diseases can take birth

✉ Md. Rafiqul Islam
dmri1978@cseku.ac.bd

Resheta Ahmed Smrity
smrityku@gmail.com

Sajib Chatterjee
sajib.ku.cse12@gmail.com

Md. Riaz Mahmud
riazmahmudcse@gmail.com

[1] Computer Science and Engineering Discipline, Khulna University, Khulna 9208, Bangladesh

from this type of wrong folding [3]. By determining the correct structure, it is possible to understand how protein does all these tasks. It is also possible to identify the diseases and produce antidotes of the diseases. New drugs can be invented experimentally with help of the information associated with the tertiary structure of the protein. The application of protein folding optimization also includes the improvement of protein functionality and the particular modeling of cells [4–6].

The PFO problem can be illustrated with a simple model such as hydrophobic-polar (HP) model which was proposed by Dill [7]. This PFO problem has been authenticated as an NP-complete problem according to this model [8]. In this model, amino acids can be classified into two types, hydrophobic (H) and hydrophilic or polar (P). The main concern of this model is maximum number of hydrophobic–hydrophobic interactions in a unit distance [9, 10]. Because the maximum number of hydrophobic–hydrophobic (H–H) interactions represents the most stable protein structure. If S denotes the most stable structure of a protein, then the objective function (F_s) according to this model can be defined for that protein as follows:

$$F_s = \mathrm{Max}(H{-}H) \tag{1}$$

According to the law of thermodynamics, it is considered that the most stable structure holds the lowest energy value. Now if E_s denotes energy value of the structure, then the energy function can be shown as

$$E_s = (-1) * F_s \tag{2}$$

There are some dissimilarities in the hydrophobic-polar model. The model can be depicted by any of lattice models such as square, face-centered-cubic (FCC), general crystallographic, triangular and cubic lattices. In our proposed method, the cubic lattice model has been used to find the optimal solution for PFO problem. Any protein sequence in the cubic lattice is converted to a binary string of length l which can be represented as a string $s = \{s_1, s_2, \ldots, s_l\}$ where $s_i \in \{H, P\}$ and $i = 1, 2, \ldots, l$ [9]. There are six directions such as left (L), right (R), up (U), down (D), forward (F) and backward (B) in a cubic lattice [5]. Each amino acid of a particular structure is represented by (x, y, z) coordinates in the cubic lattice and also with a set of directions. A conformation is valid only when no lattice point is occupied by more than one amino acid [11]. If any overlapping occurs, the conformation is said to be invalid. If a_m and a_n are two adjacent amino acids and V is a set of all valid structures, then the energy function (F_v) given in [5, 12] is as follows (Fig. 1):

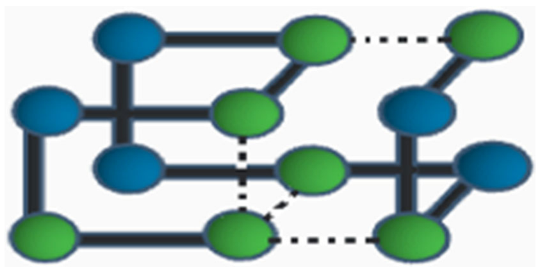


Fig. 1 Energy value calculation. Here, the P monomers are shown in blue and the H monomers in green. The protein energy value $F_v = -4$ for sequence S = {H, H, P, H, H, P, H, P, H, P, H} in the cubic lattice (colour figure online)

$$F_v = \sum_{m=1}^{l} \sum_{n=m+2}^{l} g(a_m, a_n) * h_{m,n} \tag{3}$$

where

$$g(a_m, a_n) = \begin{cases} 1, & \text{if } a_m, a_n \text{ are adjacent and not connected amino acids,} \\ 0, & \text{otherwise.} \end{cases}$$

$$h_{m,n} = \begin{cases} -1, & \text{if } a_m \text{ and } a_n \text{ are hydrophobic amino acids,} \\ 0, & \text{otherwise.} \end{cases}$$

where two hydrophobic amino acids that are adjacent to each other by lattice points are scored as − 1. The summation of all H–H contacts is the energy score for the confirmation [5].

In this paper, a chemical reaction optimization algorithm $(\mathrm{CRO_{PFO}})$ has been applied to HP cubic lattice model to optimize the tertiary protein folding. In real proteins, hydrophobic amino acids tend to be in core portion as well as the hydrophilic amino acids tend to move in the outer portion of protein structure. The most stable structure of protein contains maximum hydrophobic amino acids in core portion. We have redesigned the basic operators of CRO for solving PFO problem. Besides that, we have proposed two extra mechanisms, hydrophobic and polar (H&P) compliance and evolution. The H&P compliance mechanism visualizes a center of each individual structure and determines the distance of each H and P monomer from the center. After determining the distance of the monomers from core position, this mechanism changes the direction of each monomer in such a way that the H monomers tend to the nearest positions of the core and the P monomers are folded to far positions from the core. Basically, this mechanism tries to imitate the behavior of real protein folding. The evolution mechanism tries to move each monomer in a protein using every possible direction. In this mechanism, each monomer folded to the new direction provides better energy value than the existing one. These two advanced mechanisms improve the performance of the $\mathrm{CRO_{PFO}}$ algorithm dramatically. In assistance with these two extra mechanisms, the four operators of CRO algorithm also improve the energy values of the solutions. While applying these operators and mechanisms, many invalid structures can be produced. The invalid structure represents a structure that contains two or more amino acids in the same position of cubic lattice. If any invalid structure is created during the execution process of the operators or mechanisms, then the repair mechanism repairs the invalid structure to a valid one.

In recent years, CRO algorithm has become more familiar among other metaheuristic algorithms. It has been successfully applied to many familiar problems such as population transition problem in peer-to-peer live streaming [13], shortest common supersequence problem [14], standard continuous benchmark functions [15], cognitive radio spectrum allocation problem [16], network coding optimization problem [17], probabilistic select grid scheduling problem [18], stock portfolio selection problem [19] and artificial neural network training [20]. This information has inspired us to use CRO algorithm to solve PFO problem. The particularity of this paper is as follows: (1) the proposed algorithm for the PFO problem, (2) the evolution mechanism improves the energy value of each solution and (3) H&P compliance mechanism places the H monomer in the core positions and P monomers in the outer portions produces a stable structure gradually. The algorithm has been examined over some sets of sequences and also has been compared with the genetic algorithm with advanced mechanisms [5], which is state of the art.

2 Related work

Since the PFO is an NP-hard optimization problem, a small mistake in the prediction process can mislead the process completely. The algorithms that already solved the PFO problem are reviewed below.

2.1 Particle swarm optimization

A discrete particle swarm optimization ($DPSO_{HP}$) algorithm was introduced in [21] to solve protein folding optimization problem in both 2D and 3D lattices. The discrete PSO method was in accordance with the possibility theory from a set-based PSO (S-PSO). First, the authors represented the protein sequence in HP model for avoiding overlapping in the lattice or cubic. In this method, they introduced special velocity and position updating processes which represent the overall framework of $DPSO_{HP}$. At the beginning of the construction phase, each particle chooses two middle amino acids and placed them in the two central cells in the lattice or cubic board. After that, the particles randomly choose amino acids to fold the left part or right part of the protein sequence. In path construction, new amino acids could not be placed on the lattice replacing old ones. For this reason, the protein could not be folded anymore during the construction phase and they called it motionlessness. A path retrieval mechanism was used to build an infeasible solution into a feasible one for solving the problem. The whole sequence was divided into two parts for indicating the two middle amino acids of the amino acid sequence and then checked which side of the sequence was motionless. The results of this algorithm are good for small sequences of the protein, but the results of the longer sequences of the protein are not efficient for the search space [22]. Another PSO algorithm was proposed for the prediction of the protein structure by Mansour et al. [22]. The algorithm predicted the 3D protein structure with low energy. To update the velocity, a function is introduced of the particle and it is the main operator of this algorithm. The task of this operator is to explore new areas of the search space to find optimal solutions.

2.2 Genetic algorithm

Khimasia et al. [23] proposed a genetic algorithm for the prediction of protein structure problem. A method was launched based on the simple lattice for prediction of the tertiary structure of the protein in this paper. The study has two important conclusions: First, they require multi-point crossovers and second, a local perturbation for any GA is fully effective. König et al. [24] used systematic crossover for search strategy by coupling the best individuals. After that, they have checked every possible crossover point of the best individuals and has taken two individuals for the next generation. The difference between two individuals was used as an information for selecting the best individual. In 2004 Unger and Ron proposed another genetic algorithm for the prediction of protein structure problem [25]. In their paper, arrangements were changed through mutation, in the form of conventional Monte Carlo steps and crossovers where parts of the polypeptide chain were exchanged between conformations. Both genetic operators were repeated up to the creation of feasible structure. An upgraded genetic algorithm accompanied by mutation mechanism was proposed in by Lin and Su [12]. Their algorithm was designed in accordance with the particle swarm optimization algorithm. Custódio et al. [26] developed a structure which was phenotype-based crowding structure in order to maintain the useful of diversity within the populations.

2.2.1 Genetic algorithm with advanced mechanism

In 2016 a stochastic, population-based GA was proposed by Bošković and Brest [5]. They divided the total process into crowding, clustering, repair, local search and opposition-based mechanisms. Here, the population P was generated by populating a group of individuals $x_i = 1, 2, 3, \ldots, popSize$ (i is a subscript and $popSize$ is the population size) and each individual was encoded with ultimate encoding. Each solution or individual had a fixed size (which represents the length of the amino acid sequence) absolute directions: left (L), right (R), upper (U), down (D), forward (F) and backward (B). Initial population P is selected uniform randomly and improved with local searches at the beginning of the evolutionary process. Subpopulations from the population P were generated using compare mechanism with the nearest structures, and the nearest individual structure was determined by the Hamming distance between two individuals. For every generated subpopulation, the algorithm applied one point, two points or three points (multi-point) crossover randomly and the segment mutation randomly selected directions of one, two or three consecutive directions. The mutation operation improved the quality of the protein structure and ensured the diversity of the population. The local search was used to improve convergence speed by trying to improve the quality of individual structure by applying local movements within one or two successive monomers through the entire conformation where one of the consecutive monomers was hydrophobic (H). The repair mechanism was used to construct a feasible solution from an infeasible solution using a backtracking algorithm. Within the population P, improved structures were then compared with the nearest structures and the nearest individual structure was determined by the Hamming distance between two individuals. An opposition-based mechanism was used to generate good conformation from both sides (starting and ending) of the sequence and terminated when the number of repair method invocations grew up to a predefined level. The algorithm obtained best results for most of the cases and reduced the time to acquire the best native energy.

2.3 Ant colony optimization

ACO follows the foraging behavior of real ants to solve optimization problems. An improved version of ACO algorithm was proposed by Shmygelska et al. [27]. Long-range moves and selective local search improved the performance of this algorithm. Selective local search used in this algorithm reduced the time complexity of the local search phase. Thilagavathi and Amudha proposed another ant colony optimization (ACO) algorithm [28]. In this paper, the pheromone level of a trail was set to a constant value and after the end of the construction phase, the value was updated. The pheromone value was updated in two stages: local update and global update. In the global update, the pheromone level was reduced and this reduced the possibility. As a result, the search is more diversified. The global update rule was applied in order to deliver a large amount of pheromone on the suitable path of the optimal solution. In the course of the construction phase, the next move was selected based on pheromone matrix value and heuristic information. The ant colony was initialized with five ants, and each ant performed the folding process and gave different folding structures using the energy functions. The initial population was only five ants which could be proved as an obstacle for getting the best solution. The performance of this algorithm was appreciable for only smaller protein sequences, but for real-time proteins or long-length protein sequences its performance was not applicable.

3 Chemical reaction optimization

Human beings are closely associated with the nature which is a very complex system. The complex system operates in its own way without any obstacle. The way in which nature operates can be worthy principle to solve real-world problems. In the field of computer science, the principles are called algorithms or more precisely nature-inspired algorithms [29]. CRO is one of the latest optimization algorithms designed by Lam and Li [30]. Basically, it is a population-based metaheuristic algorithm that imitates the behavior of chemical reactions. Furthermore, CRO is a technique that loosely couples with chemical reactions [29]. The functions of CRO operate on two laws of thermodynamics. The first law declares that energy cannot be produced or destroyed; energy can be transformed from one kind to another and transferred from one entity to another. In a chemical reaction, there are chemical substances and surroundings around these. Every chemical substance has potential energy (P_{E}), kinetic energy (K_{E}), and the energy of surroundings is symbolically represented as the buffer energy. So from the first law of thermodynamics, we can write

$$\sum_{i=1}^{PopSize(t)} (PE_i(t) + KE_i(t)) + buffer(t) = Z \tag{4}$$

where $PE_i(t)$ and $KE_i(t)$ denote the potential and kinetic energy of the molecule i at time t, respectively, $buffer(t)$ is the energy of the surrounding as well as the energy of the central buffer at time t and Z is a constant [29].

A chemical reaction can be either endothermic or exothermic. The initial buffer size (S_{BO}) can characterize these two types of chemical reactions. If $S_{\mathrm{BO}} > 0$, then the reaction is endothermic, and if $S_{\mathrm{BO}} = 0$, then it is exothermic. The second law says that the entropy of a system always tends to increase over time. The energy stored in a molecule is referred to as potential energy. When it is converted into kinetic energy, then the system becomes more disordered and entropy increases. When a chemical system becomes unstable with excessive energy, it undergoes some chemical reactions to obtain stable state. When a chemical substance goes to a stable condition, its potential energy is minimum. The algorithm is a step-wise searching process for minimum energy (optimal solution) [29].

4 Design of CRO for PFO problem

Chemical reaction optimization (CRO) is a population-based metaheuristic where the number of molecules may not be same in each iteration. When an ineffective collision occurs, the number of molecules remains same. On the other hand, when an effective collision occurs, the number of molecules may be either increased or decreased. In CRO, a molecule (M) represents one specific solution of any optimization problem. Essentially, CRO consists of three stages: initialization, iteration and the final stage. The initialization stage begins with the initialization of the different parameters of CRO algorithm. The iteration stage consists of some operations between molecules. In the iteration stage, if any stopping criterion is not met, then the algorithm proceeds to the final stage. The final stage determines a globally optimal solution from the local optimal solutions using its objective function value and terminates the algorithm [31].

4.1 Population initialization and algorithmic description

The initial population is selected by creating random directions. For PFO problem, the population can be termed as a set of direction arrays. If the length of an amino acid sequence is l, then the length of the direction array will be $l-1$ because the first position is fixed for the center of the solution structure. In cubic lattice, there are six directions: {L, R, U, D, F, B} where L: left, R: right, U: up, D: down, F: forward, B: backward. Every array-based solution is represented by randomly generating a number between 1 and 6. The corresponding direction of the cubic lattice according to the random value is then taken as an element of the direction array. The generation process of a solution structure of sequence {H, H, P, H} is shown in Fig. 2. In step 1, we can see that the first position is fixed and no direction has not been assigned to it. In step 2, upward direction has been selected randomly and has been assigned to the resultant array. The same process continues to step 4. After assigning all the direction values, a complete structure is formed. After generating a complete solution, it is checked with the existing solutions. If the same solution exists in the set, then the solution is discarded; otherwise, it is accepted and added to the set as a new solution.

Algorithm 1 CRO_{PFO}

```
1: procedure $CRO_{PFO}(sequence, popSize)$
2: Set $popSize, KELossRate, molCol, buffer, initialKE,$
   $numHit, minHit, \alpha$ and $\beta$
3: Create $popSize$ number of molecules
4: $population$ = search_space_creation($sequence, popSize$)
5: while $initialKE$ != 0 do
6:     Generate $b \in [0, 1]$
7:     for $i = 1$ to $popSize$ do
8:         if $b > molCol$ then
9:             if $numHit - minHit > \alpha$ then
10:                $\{dec1, dec2\}$ = decomposition(
                   $sequence, population[i]$)
11:                $dec1$ = repair($sequence, dec1$)
12:                $dec2$ = repair($sequence, dec2$)
13:                $numHit{+}{+}$
14:            else
15:                $onWall$ = onWallIneffective($sequence, population[i]$)
16:                $onWall$ = repair($sequence, onWall$)
17:                $numHit{+}{+}$
18:            end if
19:        else
20:            Randomly select $population[j]$, $j \in [0, popSize]$
21:            if $initialKE \le \beta$ then
22:                $syn$ = synthesis($sequence, population[i], population[j]$)
23:                $syn$ = repair($sequence, syn$)
24:                $numHit{+}{+}$
25:            else
26:                $\{in1, in2\}$ =interMolecularInEffective(
                   $sequence, population[i], population[j]$)
27:                $in1$ = repair($sequence, in1$)
28:                $in2$ = repair($sequence, in2$)
29:                $numHit{+}{+}$
30:            end if
31:        end if
32:        $hpcom$ = HandP_Compliance($sequence, population[i]$)
33:        $hpcom$ = repair($sequence, hpcom$)
34:        $eval$ = evaluate($sequence, population[i]$)
35:        $eval$ = repair($sequence, eval$)
36:        Check for any new minimum solution
37:        $newP = popSize{+}{+}$
38:        $population[newP]$=
           closest($dec1, dec2, onWall, syn, in1, in2$)
39:        $\{sequence, population\}$ = OppositePosition
           $\{sequence, population\}$
40:    end for
41:    $initialKE = initialKE - KELossRate$
42: end while
43: Output: the best solution and its energy value
44: end procedure
```

After the initialization process, a conditional loop starts executing until the stopping criterion is not satisfied. The algorithm meets the stopping criterion when the molecule (solution) has lost its kinetic energy (initialKE) completely. The molecule has lost its kinetic energy according to KELossRate that is initialized at very first. A threshold value (b) is generated between 0 and 1 randomly to determine whether the reaction will be uni-molecular or

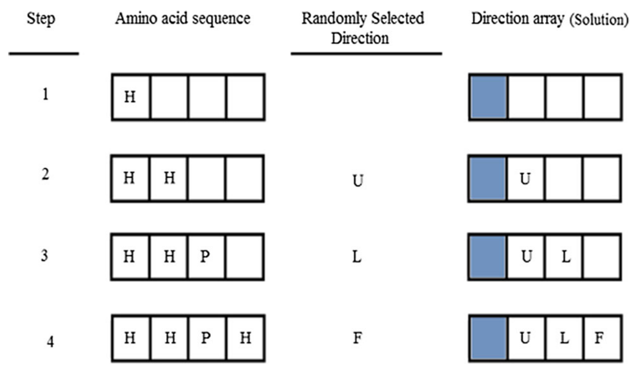


Fig. 2 Population initialization

inter-molecular. Every solution of the population is iterated and modified by one of the four operators of CRO algorithm. The value of b decides which operator will be used to modify the solutions. If $b >$ molCol, a uni-molecular collision is selected; otherwise, inter-molecular collision is selected. In the case of uni-molecular collision, if the criterion of decomposition is met, the decomposition is executed; otherwise, on-wall ineffective collision is performed. On the other hand, in the case of an inter-molecular collision if the criterion of synthesis is met, then synthesis is done; otherwise, inter-molecular ineffective collision (IMIC) is performed to modify the selected solutions. When new minimum solution is found, the solution is placed into the last position of the *population* array using a variable, *newP*.

The strategy of CRO is totally dependent on the behavior during a chemical reaction. The very first stage of CRO generates initial population along with the parameters: popSize, KELossRate, molCol, initialKE and two thresholds (α and β). The parameters and their definitions are given in Table 1. In the second stage, one particular elementary reaction out of four occurs in every iteration. The threshold value α is used to select which uni-molecular reaction will be triggered. If $numHit - minHit \geq \alpha$, then decomposition reaction is triggered; otherwise, on-wall ineffective collision reaction is triggered. On the other hand, the second threshold value β is also used to select which inter-molecular reaction will be triggered. If $initialKE \leq \beta$, then synthesis reaction is triggered; otherwise, inter-molecular ineffective collision reaction is triggered. At the end of each iteration, we have to check the termination criterion of the algorithm. The algorithm terminates when the molecule (solution) has lost its kinetic energy completely, which means the initialKE of the molecule has turned to be 0. In each iteration, the molecule has lost its kinetic energy according to KELossRate. The final and the last stage executes when termination criterion of iteration stage satisfies. In the final stage, the algorithm terminates and best solution is found. The pseudo-code of this procedure is given in Algorithm 1.

4.2 Reaction operators

In the proposed algorithm, we have redesigned the basic four operators of CRO for the PFO problem. Along with these operators, two extra operators have been designed. All of these operators are described below.

4.2.1 On-wall ineffective collision

This is a molecular reaction that is invoked frequently in the initial iteration phase to update the solutions. The operator selects a solution and takes the length of the direction array of the solution. Then, the process makes a copy of the solution and selects a random pointer in the range of the length that specifies how many positions will be modified. Another random variable is generated within range 1–6. The selected position is altered with the corresponding direction value pointed by the new random variable. When all the positions are updated, the new

Table 1 Description of parameters of CRO algorithm

Parameters	Description
popSize	The total population size of the problem that indicates the number of molecules
initialKE	Initial kinetic energy
KELossRate	Loss rate of kinetic energy (KE)
molCol	A parameter to choose whether the chemical reaction is uni-molecular or inter-molecular
buffer	Energy of surroundings
α, β	Threshold values the intensification and diversification
Number of hits (numHit)	The total number of hits a molecule has taken
Minimum structure (MinStruct)	The molecule structure that has minimum potential
MinimumPE (MinPE)	When a molecule attains its MinStruct, MinPE is its corresponding potential energy value
Minimum hit number (minHit)	It is the number of hits when a molecule has MinStruct

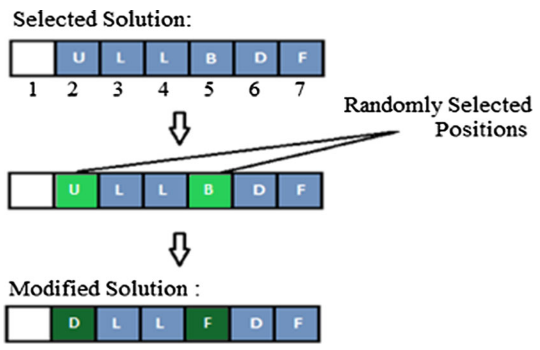


Fig. 3 On-wall ineffective collision

(updated) solution is returned as output. Figure 3 depicts the process. First, a solution is selected and then positions 2 and 5 are selected randomly and the direction values in the corresponding positions have been changed to their opposite directions. Here, direction U (up) is changed to direction D (down) and direction B (backward) is changed to direction F (forward). The modified solution is the output of this process. Algorithm 2 gives the pseudo-code of the process.

Algorithm 2 OnWallIneffectiveCollision

```
1:  procedure $onWallIneffective(sequence, solution)$
2:  $n$ = length of $sequence$
3:  Duplicate $solution$ to form $newSolution$
4:  Set $nPoint$ randomly select from $1, 2, 3, ..., n$
5:  $r = Rand(0, (n - nPoint))$
6:  for $i = r$ to $(r + nPoint)$ do
7:      $select = Rand(1, 6)$
8:      $newSolution[i]$ = direction_array$[select]$
9:  end for
10: Output: $newSolution$
11: end procedure
```

4.2.2 Decomposition

Decomposition process breaks a selected solution into two new solutions. Since the process produces two new solutions from an existing one, in PFO problem the new solutions may contain overlapping. Each time a solution is generated, the solution is repaired immediately. In this process, a solution is randomly selected. Two duplicate solutions are produced from the selected solution. Then, a position is randomly selected that divides the solutions into two parts. The first half of one new solution is changed to opposite directions, and the second half of another solution is modified by opposite directions. In Fig. 4, a pointer has been set randomly after position 3. The segment before the pointer is considered as the first portion, and the remaining part constitutes the second portion. According to rule, the first portion of the first solution and the second portion of the second solution are selected. The directions of the selected portions have been changed to their opposite directions, such as U to D, L to R, F to B and vice versa. The remaining portions of the two new solutions remain same as in their parent. The process generates two new solutions as output. Algorithm 3 shows the pseudo-code of the process.

Algorithm 3 Decomposition

```
1: procedure $decomposition(sequence, solution)$
2: $n$ = length of $sequence$
3: Duplicate $[(n/2 + 1)...n]$ values from solution to form $newSolution1$
4: Duplicate $[0...n/2]$ values from solution to form $newSolution2$
5: Generate $[0...n/2]$ opposite direction values of the $solution$ and append $newSolution1$
6: Generate $[(n/2 + 1)...n]$ opposite direction values of the $solution$ and append $newSolution2$
7: Output: $newSolution1$ and $newSolution2$
8: end procedure
```

4.2.3 Synthesis

In this process, a new solution is produced from two existing solutions. First, two solutions are selected randomly. Then, the solutions are divided into parts. How many parts will be created is selected randomly. Then, the energy value is calculated for each part of the solutions. The parts of the two solutions are compared. The portion that shows the highest energy value is selected as a part of the new solution. The new solution often shows higher energy value than the selected solutions. Figure 5 shows the synthesis process. In the figure, a pointer is set after position 4 randomly which breaks each solution into two parts. Then, for each part we have calculated the energy value. Now if we compare the parts, then the first part of the second solution and the second part of the first solution

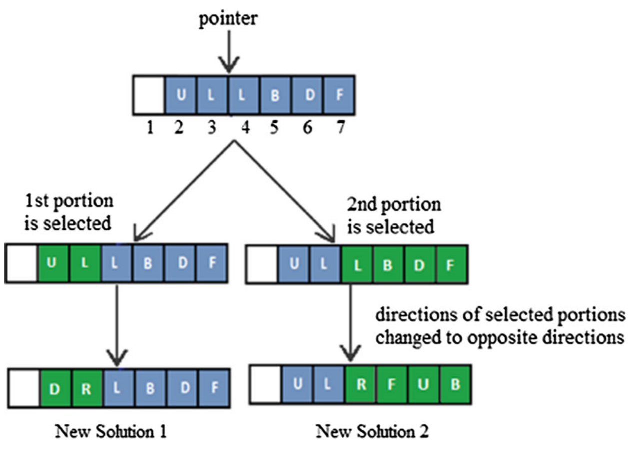


Fig. 4 Decomposition

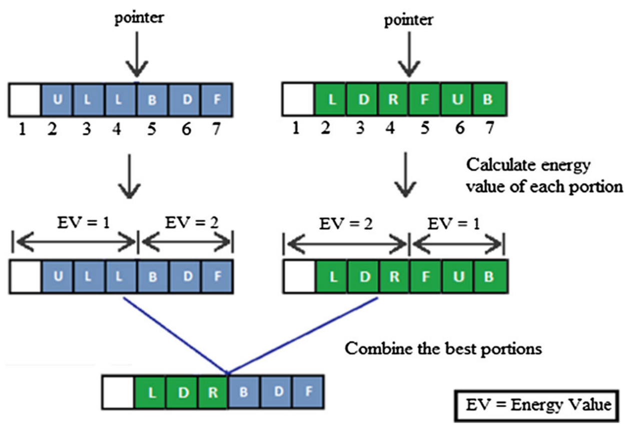


Fig. 5 Synthesis

are selected due to higher energy values. The combined solution is then returned as output. Algorithm 4 depicts the pseudo-code of the synthesis operation.

Algorithm 4 Synthesis

```
 1: procedure synthesis($sequence, solution1, solution2$)
 2: $n$ = length of $sequence$
 3: Duplicate $solution1$ to form $newSolution$
 4: Set $partitionNum$ randomly select from $1, 2, 3, ..., n$
 5: $partitionLen = (n/partitionNum)$
 6: for $i = 1$ to $partitionNum$ do
 7:   for $j = i*partitionLen$ to $i*partitionLen+partitionLen$ do
 8:     if $solution1$ has larger number of HH contacts than $solution2$ then
 9:       $newSolution[j] = solution1[j]$
10:     else
11:       $newSolution[j] = solution2[j]$
12:     end if
13:   end for
14: end for
15: Output: $newSolution$
16: end procedure
```

4.2.4 Inter-molecular ineffective collision

The nature of this process is to select two solutions that collide with each other. The collision produces two new solutions by changing some positions. The structural view of the two new solutions may be hugely different from their parents. The difference depends on the decision that how many positions will be changed during this process. The process selects an increment value randomly. Each of the two solutions is selected sequentially. Starting from the first position, some positions are selected by the increment value. The selected positions are then changed to their opposite directions. A pictorial view of this process is shown in Fig. 6. First, 2 has been chosen randomly as an increment value. Starting from the first position, every position according to the increment value is selected, which are colored by deep blue and deep green for the first solution and second solution of the figure, respectively. The selected positions are then changed to their opposite directions such as U to D, L to R, F to B and vice versa. Algorithm 5 shows the pseudo-code of this process.

Algorithm 5 Inter-molecular Ineffective Collision

```
 1: procedure $interMolecularInEffective(sequence, solution1, solution2)$
 2: $n$ = length of $sequence$
 3: Duplicate $solution1$ to form $newSolution1$
 4: Duplicate $solution2$ to form $newSolution2$
 5: Set $nPoint$ randomly select from $1, 2, 3, ..., n$
 6: $r = Rand(0, (n - nPoint))$
 7: for $i = r$ to $(r + nPoint)$ do
 8:   $newSolution1[i] = opposite\_Direction[solution1[i]]$
 9:   $newSolution2[i] = opposite\_Direction[solution2[i]]$
10:   $i = i + incrementValue$
11: end for
12: Output: $newSolution1$ and $newSolution2$
13: end procedure
```

4.3 H&P compliance

The hydrophobic and polar (H&P) compliance mechanism determines the center of each individual structure and the distance of each H and P monomer from the center. It checks whether the hydrophobic (H) amino acids are in the core positions and the polar (P) amino acids are in the remote portion of the core or not. If H amino acids are not in the core positions, it restructures the conformation such that the hydrophobic amino acid remains in the core position. Similarly, it reforms the conformation such that the polar amino acid remains in the remote part of the core of the structure. If (P_c, Q_c, R_c) represents the core position and (P_d, Q_d, R_d) represents dth amino acid, then H&P compliance is calculated as follows:

$$HP_s = \frac{\sum_{d=1}^{length} (P_c - P_d)^2 + (Q_c - Q_d)^2 + (R_c - R_d)^2}{N_s} \tag{5}$$

where N_s = number of amino acids in a structure and

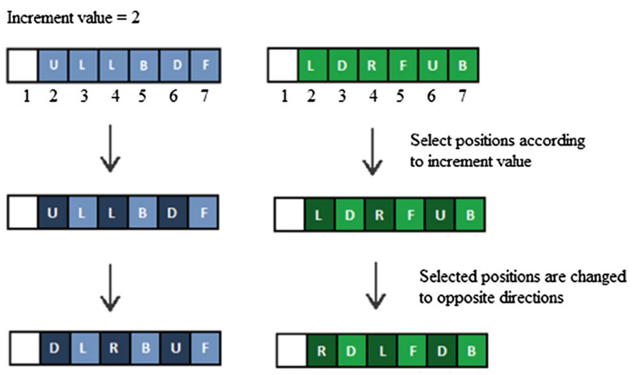

Increment value = 2
U L L B D F
1 2 3 4 5 6 7
L D R F U B
Select positions according to increment value
Selected positions are changed to opposite directions
D L R B U F
R D L F D B

Fig. 6 Inter-molecular ineffective collision

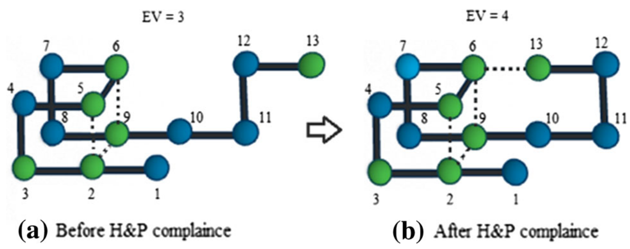


Fig. 7 H&P compliance

$$P_c = (P_{\max} - P_{\min})/2$$
$$Q_c = (Q_{\max} - Q_{\min})/2$$
$$R_c = (R_{\max} - R_{\min})/2$$

HP_s calculates the closeness of any amino acid from the core position. After determining distance, this mechanism changes the direction of each monomer in such a way that the H monomers tend to small distance from the core position and P monomers are folded to far distance from the center of the specific structure. Essentially, this mechanism tries to mimic the behavior of real protein folding. The H&P compliance process is depicted in Fig. 7.

Here, the last monomer is hydrophobic amino acid and it locates on the outer side. According to H&P compliance, H monomer tends to be inner part and P monomer tends to be in the outer part. Now if we change the direction of 13th monomer from right to left, then we can see that all the H monomers seem to be inner part and P monomers in the outer part. The energy value has also increased 3–4 due to this movement. Algorithm 6 gives the pseudo-code of the process.

Algorithm 6 H&P Compliance

```
 1: procedure $HandP\_Compliance(sequence, solution)$
 2: $n$ = length of $sequence$
 3: Duplicate $solution$ to form $newSolution$
 4: for $i = 1$ to $n$ do
 5:   for $j = i - 1$ to 1 do
 6:     if $sequence[j]$ is $H$ monomer then
 7:       try to move every possible empty position in cubic
          lattice and calculate $HP_s$
 8:       if $HP_s$ is minimum then
 9:         move $newSolution[j]$ to the free lattice point
10:       end if
11:     end if
12:     if $sequence[j]$ is $P$ monomer then
13:       try to move every possible empty position in cubic
          lattice and calculate $HP_s$
14:       if $HP_s$ is maximum then
15:         move $newSolution[j]$ to the free lattice point
16:       end if
17:     end if
18:   end for
19: end for
20: Output: $newSolution$
21: end procedure
```

4.4 Evolution mechanism

The evolution mechanism tries to move each individual monomer in a protein using every possible direction. For each amino acid, it inspects empty positions of the cubic lattice points. After that, it changes the position of amino acid to one of the empty positions and determines whether the energy value of the entire structure increases or not. If the energy value increases, then new structure is accepted; otherwise, it is rejected. Figure 8a gives a sample structure with 13 monomers. The energy value is 3 and the structure does not contain any overlapping in positions. The main principle of evolution mechanism is to increase the performance of each structure by moving the monomers in every possible direction. In this mechanism, we try to move each monomers from its monomer, gradually moving toward the first monomer. For example, in the case of 13th monomer, if we move the 13th monomer in every possible direction, it can be seen that the energy value does not increase. As the energy value cannot be increased due to 13th monomer, the mechanism immediately switches to the previous monomer. If we change the direction of the 12th monomer from right to left, then we can observe that energy value increases from 3 to 5 as shown in Fig. 8b. This direction changing process continues to the first monomer, the process tries to move the monomer and increases the performance. After that, we consider the 12th monomer and continue this process 12th to first monomer. We continue this process for every individual monomer with respect to each previous individual monomer. Algorithm 7 depicts the pseudo-code of this process.

Algorithm 7 Evolution

```
 1: procedure $evaluate(sequence, solution)$
 2: $n$ = length of $sequence$
 3: Duplicate $solution$ to form $newSolution$
 4: for $i = n$ to 1 do
 5:   for $j = i$ to 1 do
 6:     if $newSolution[j]$ has free lattice point and energy
        value increases then
 7:       move $newSolution[i]$ to the free lattice point
 8:     end if
 9:   end for
10: end for
11: Output: $newSolution$
12: end procedure
```

4.5 Repair mechanism

During execution of the operators of CRO_{PFO}, many invalid structures may be created and those may contain overlapping in the same position. The number of overlapping positions can be one or more. Such structures cannot be accepted to solve PFO problem. For this reason, we have designed a repair function that works on the invalid

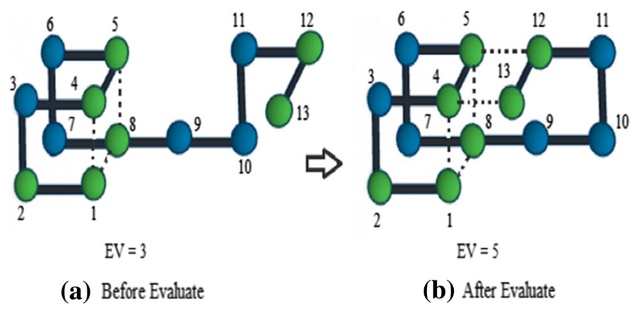


Fig. 8 Evolution mechanism

structures and produces valid ones. The two major tasks of the repair mechanism are overlap checking and applying backtracking algorithm to create a valid structure. The working process of repair mechanism is as follows. A checking process checks the structures to observe that if the structures contain overlapping or not. The procedure starts checking from the last position to the first position of any particular monomer. If any lattice point contains more than one monomer, then the procedure returns true. A backtracking algorithm is applied to the protein structure for removing the overlapping. The backtracking algorithm sets a pointer to the previous monomer of the overlapping monomer. Then, the algorithm finds free adjacent lattice points of the previous monomer and tries to move the overlapping monomer to the free positions. If there exists any free lattice point, then the algorithm moves the overlapping monomer to the free position, and if there is no free lattice point, then the algorithm moves to the next previous monomer. This process continues until the structure becomes overlapped free. For example, if we find any overlapping for any individual monomer, we start checking from the previous position to first position of that monomer. If we find any free lattice point position, then we move the overlapping monomer to that free position. We continue this process for each individual monomer with respect to its previous all monomers until we get a valid structure. Algorithm 8 gives the pseudo-code of the process, and Fig. 9 depicts the pictorial view, where an invalid structure has been repaired using repair mechanism. In the case of Fig. 9, if we start checking from the last monomer,

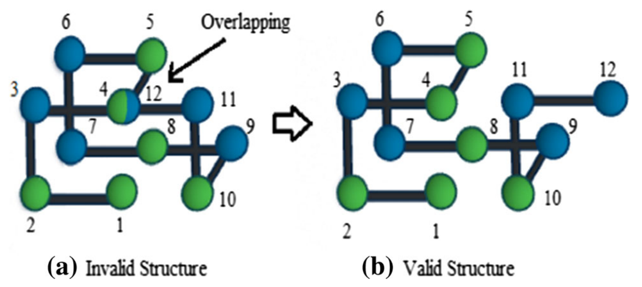


Fig. 9 Repair mechanism

it can be seen that the last monomer produces overlapping with another monomer. After removing overlapping according to the above steps, we have found a valid structure that has also shown on the right side of Fig. 9. The repair mechanism is invoked after the execution of any of the four operators and it repairs any invalid structure.

```
Algorithm 8 Repair
1: procedure $repair(sequence, solution)$
2:   $n$ = length of $sequence$
3:   Duplicate $solution$ to form $newSolution$
4:   for $i = 1$ to $n$ do
5:     if overlap exists then
6:       for $j = i - 1$ to 1 do
7:         if $newSolution[j]$ have free lattice point then
8:           move $newSolution[i]$ to the free lattice point
9:         end if
10:      end for
11:    end if
12: end for
13: Output: $newSolution$
14: end procedure
```

5 Experimental result

We compared the E_{best}, E_{mean} values and the standard deviation of our proposed algorithm with the results of GAAM [5]. Here, the E_{mean} value has been multiplied by -1 which means higher E_{mean} value shows better efficiency. The proposed algorithm was tested on four different datasets. The results are shown in Tables 4, 5, 6 and 7. In the tables, the sequence number is generated by a set number followed by a serial number of the set, such as S1.1 represents set 1 and the first sequence of the set.

5.1 Experimental setup

We described the proposed CRO algorithm in Sect. 4.1. The experimental issues have been started as follows. The algorithm was implemented in Java programming language and executed using an Intel Core i5 computer with 2.60 GHz CPU and 4 GB RAM under windows operating system. Our developed algorithms were tested on 4 datasets [5] which are shown in Table 2. The values of the parameters used in the algorithm are shown in Table 3. The molCol value varies according to the length of protein sequences. In this case, the algorithm performs 200 iterations in one run.

$$molCol = \begin{cases} 1, & \text{for length} \leq 36 \\ 0.5, & \text{for length} > 36 \end{cases} \tag{6}$$

Table 2 Test sequences

Set	Protein sequence	Len
1	HPH2P2H4PH3P2H2P2HPH3PHPH2P2H2P3HP8H2	48
	H4PH2PH5P2HP2H2P2HP6HP2HP3HP2H2P2H3PH	48
	PHPH2PH6P2HPHP2HPH2PHPHP3HP2H2P2H2P2HPHP2HP	48
	PHPH2P2HPH3P2H2PH2P3H5P2HPH2PHPHP4HP2HPHP	48
	P2HP3HPH4P2H4PH2PH3P2HPHPHP2HP6H2PH2PH	48
	H3P3H2PHPH2PH2PH2PHP7HPHP2HP3HP2H6PH	48
	PHP4HPH3PHPH4PH2PH2P3HPHP3H3P2H2P2H2P3H	48
	PH2PH3PH4P2H3P6HPH2P2H2PHP3H2PHPHPH2P3	48
	PHPHP4HPHPHP2HPH6P2H3PHP2HPH2P2HPH3P4H	48
	PH2P6H2P3H3PHP2HPH2P2HP2HP2H2P2H7P2H2	48
2	P2H5P3H2P5H2P3HP6HPHP3HP2HP2HP5HP4H2PH2P2HP2HP	64
	P2HPHP2HP2H3PH4P2H3P4HPHP3HPHP3HPHP5HPHP2HPHP3HP2HP2	64
	HPH2P2H2PHP5H3PH4P2HP2HPH2P3HPHP2H3PH2PHP5H8P3	64
	HP2H2P2HP2HPHP2HP4HP6HPHPH3P2HPHP3HPHP2H2P2HP2HP2HPH3PH	64
	HP3H2P2HPHP3HP3HPH2P3H2PHPH2PHP2HP3HP2HPH3P2HP2HP2H3PH4	64
	HP2H2PH4P6H2P2HP4H2P3HP2HPH2PHP4H2P4HP5HP4HPH2	64
	P4HP3HP3H4PH2P5HP2HPH2PHPHP5HP10H4P4H2P2H	64
	P3H3P2HPHP2HP2H2P3HP2HP2H2PHP3HP7HPH3PH5P2H2P3HP2H	64
	HP2HP2H3P4HPHP3HPH2PH5P4HPHPHP4HPHP3H2PHP4HP2H2PHP	64
	P2HP2HP2H3P3HPHP2HP2HP6HP2H3P2HP2HP2HPHP6H3P5HPHP	64
3	HPHP2H2PHP2HPH2P2HPH	20
	H2P2HP2HP2HP2HP2HP2HP2H2	24
	P2HP2H2P4H2P4H2P4H2	25
	P3H2P2H2P5H7P2H2P4H2P2HP2	36
	P2HP2H2P2H2P5H10P6H2P2H2P2HP2H5	48
	H2PHPHPHPH4PHP3HP3HP4HP3HP3HPH4PHPHPHPH2	50
	P2H3PH8P3H10PHP3H12P4H6PH2PHP	60
	H12PHPHP2H2P2H2P2HP2H2P2H2P2HP2H2P2H2P2HPHPH12	64
4	P2H3PH3P3HPH2PH2P2HPH4PHP2H5PHPH2P2H2P	46
	PHPH3PH3P2H2PHPH2PH3PHPHPH2P2H3P2HPHP4HP2H P2H2P2HP2H	58
	P2H2P5H2P2H2PHP2HP7HP3H2PH2P6HP2HPHP2HP5H3P4H2PH2P5H2P4H4PHP8H5P2HP2	103
	HP5HP4HPH2PH2P4HPH3P4HPHPH4P11HP2HP3HPH2P3H2P2HP2HPHPHP8HP3H6P3H2P2H3P3H2PH5P9HP4HPHP4	136

Table 3 Parameters of CRO algorithm

Symbol	Value
popSize	500
initialKE	100
KELossRate	0.5
molCol	Eq. 6
α	Rand [10, 100]
β	Rand [10, 100]
numHit	0
minHit	0

5.2 Effect of control parameters

The effect of the control parameters was tested on the first dataset of sequences. Because the set contains a large number of sequences and the length is medium. We set three values for each control parameter: $popSize \in \{200, 500, 1000\}$ and $molCol \in \{0.4, 0.5, 0.6\}$. Besides this, the *initialKE* and the *KELossRate* were set to 100 and 0.5 during the performance test. The default values were set to $popSize = 500$, $molCol = 0.5$, $initialKE = 100$ and $KELossRate = 0.5$ and we calculated the E_{mean} and E_{std} for all the values of control parameters. Table 4 shows the results and 30 independent runs were performed to compute the values. From the table, it can be seen that the

Table 4 The effect of control parameters for dataset 1 with $initialKE = 100$, $KELossRate = 0.5$ and the number of runs was 30

Seq.	$popSize = 200$		$popSize = 1000$		$popSize = 500$		$popSize = 500$	
	$molCol = 0.5$		$molCol = 0.5$		$molCol = 0.4$		$molCol = 0.6$	
	E_{mean}	E_{std}	E_{mean}	E_{std}	E_{mean}	E_{std}	E_{mean}	E_{std}
S1.1	32.00	0.00	32.00	0.00	32.00	0.00	31.95	0.21
S1.2	33.95	0.21	34.00	0.00	33.70	0.78	34.00	0.00
S1.3	34.00	0.00	34.00	0.00	33.95	0.21	33.80	0.51
S1.4	32.90	0.43	33.00	0.00	33.00	0.00	33.00	0.00
S1.5	32.00	0.00	32.00	0.00	32.00	0.00	32.00	0.00
S1.6	32.00	0.00	32.00	0.00	31.90	0.30	32.00	0.00
S1.7	31.90	0.30	32.00	0.00	32.00	0.00	31.80	0.51
S1.8	31.00	0.00	31.00	0.00	31.00	0.00	31.00	0.00
S1.9	33.90	0.30	34.00	0.00	34.00	0.00	34.00	0.00
S1.10	32.80	0.51	33.00	0.00	32.90	0.43	32.75	0.73

Table 5 Results of datasets 1 and 2 with $initialKE = 100$, $KELossRate = 0.5$, the number of runs was 50 and $NE = 4 \times 10^5$ (whereas $NE = 4 \times 10^6$ for GAAM [5])

Seq.	CRO_{PFO}			GAAM [5]			Seq.	CRO_{PFO}			GAAM [5]		
	E_{best}	E_{mean}	E_{std}	E_{best}	E_{mean}	E_{std}		E_{best}	E_{mean}	E_{std}	E_{best}	E_{mean}	E_{std}
S1.1	32	32.00	0.00	32	31.82	0.38	S2.1	32	32.00	0.00	32	30.86	0.60
S1.2	34	34.00	0.00	34	33.08	0.77	S2.2	38	38.00	0.00	37	35.12	0.71
S1.3	34	34.00	0.00	34	33.26	0.44	S2.3	45	45.00	0.00	45	43.54	0.37
S1.4	33	33.00	0.00	33	32.22	0.54	S2.4	42	42.00	0.00	41	39.74	0.59
S1.5	32	32.00	0.00	32	31.58	0.49	S2.5	42	42.00	0.00	42	40.62	0.72
S1.6	32	32.00	0.00	32	31.18	0.38	S2.6	35	35.00	0.00	34	33.52	0.50
S1.7	32	32.00	0.00	32	30.62	0.56	S2.7	28	28.00	0.00	28	28.00	0.00
S1.8	31	31.00	0.00	31	30.38	0.48	S2.8	38	38.00	0.00	38	36.54	0.54
S1.9	34	34.00	0.00	34	33.02	0.37	S2.9	41	41.00	0.00	40	38.00	0.60
S1.10	33	33.00	0.00	33	32.28	0.45	S2.10	31	31.00	0.00	31	31.00	0.00

values have been changed a little due to the change in parameters. For $popSize = 200$, the E_{mean} and E_{std} values have changed a little from the highest values. On the other hand, for $popSize = 1000$, all the E_{mean} and E_{std} values have attained the highest values. But for $molCol = 0.4$ and $molCol = 0.6$, the E_{mean} and E_{std} values have also changed a little from the highest values. The E_{mean} and E_{std} have been calculated as a statistical measure. The E_{std} value is calculated as follows:

$$E_{std} = \sqrt{\frac{1}{N_R}\sum_{i=1}^{N_R}(E_i - E_{mean})^2} \tag{7}$$

where N_R is the number of runs which is set to 30 and E_i is the selected energy value in each run. E_{mean} is the mean energy value in all the iterations and the value is calculated by the following equation:

$$E_{mean} = \frac{\sum_{i=1}^{N_R} E_{best}}{N_R} \tag{8}$$

It means that the E_{best} is the best energy value of each run.

5.3 Comparison with existing algorithm

The state of the art for the PFO in HP cubic lattice model is GAAM [5]. This method was proposed in 2016 by Bošković and Brest. They compared their proposed algorithm with adaptive genetic algorithm with phenotypical crowding [26], memetic algorithm with self-adaptive local search [32], ant colony optimization algorithm [33], multi-objective genetic algorithm with feasibility rules [34], genetic algorithm with systematic crossover [24], estimation of distribution algorithm [4] and clustered memetic algorithm with local heuristics [35]. The performance of GAAM was best of all the compared algorithms. So, we compared our proposed algorithm with the GAAM. The results are shown in Tables 5, 6 and 7. In the tables, the sequence number was generated by a set number followed by a serial number of the set, such as S1.1 represents set 1 and the first sequence of the set. We computed the E_{best}, E_{mean} and E_{std} for dataset 1. The results are shown in left part of Table 5. All the sequences of this dataset are of 48 lengths long. The E_{mean} indicates the mean energy value of all independent

Table 6 Results of dataset 3 with *initialKE* $= 100$, *KELossRate* $= 0.5$, the number of runs was 50 and NE $= 4 \times 10^5$ (whereas NE $= 4 \times 10^6$ for GAAM [5])

Seq.	$\mathrm{CRO}_{\mathrm{PFO}}$			GAAM [5]		
	E_{best}	E_{mean}	E_{std}	E_{best}	E_{mean}	E_{std}
S3.1	11	11.00	0.00	11	11.00	0.00
S3.2	13	13.00	0.00	13	13.00	0.00
S3.3	9	9.00	0.00	9	9.00	0.00
S3.4	18	18.00	0.00	18	18.00	0.00
S3.5	31	31.00	0.00	31	31.00	0.00
S3.6	34	34.00	0.00	34	32.62	0.48
S3.7	55	55.00	0.00	55	52.40	0.66
S3.8	59	59.00	0.00	59	57.62	0.60

Table 7 Results of dataset 4 with *initialKE* $= 100$, *KELossRate* $= 0.5$ (for S4.1 and S4.2), *KELossRate* $= 0.05$ (for S4.3 and S4.4), the number of runs was 50, NE $= 4 \times 10^5$ (for S4.1 and S4.2) and NE $= 4 \times 10^6$ (for S4.3 and S4.4) (whereas NE $= 4 \times 10^6$ (for S4.1–S4.3) and NE $= 4 \times 10^7$ (for S4.4) for GAAM [5])

Seq.	$\mathrm{CRO}_{\mathrm{PFO}}$			GAAM [5]		
	E_{best}	E_{mean}	E_{std}	E_{best}	E_{mean}	E_{std}
S4.1	35	35.00	0.00	35	34.42	0.49
S4.2	44	44.00	0.00	44	41.92	0.49
S4.3	58	58.00	0.00	53	50.80	0.99
S4.4	75	73.98	0.46	71	68.54	1.20

runs, and E_{std} represents the standard deviation of all runs. From the table, it can be observed that $\mathrm{CRO}_{\mathrm{PFO}}$ shows better E_{mean} values for all sequences. Our proposed algorithm has obtained not only better E_{mean} values but also better E_{std} values for all the sequences with respect to GAAM. We computed the E_{best}, E_{mean} and E_{std} for dataset 2. The results are shown in the right side of Table 5. All the sequences of this dataset are 64 lengths long. For the sequences of both datasets 1 and 2, we set *initialKE* $= 100$ and *KELossRate* $= 0.5$. The *molCol* value was set to 0.5. The number of independent runs was 50 for each individual sequence. It means that $\mathrm{CRO}_{\mathrm{PFO}}$ performed 200 iterations in one run. In one full iteration the algorithm performs 500 evaluations (*popSize* $= 500$). Now we describe the calculation process of number of evaluations (NE) as follows:

- If we consider the worst case, in Algorithm 1 within *popSize* and (if $b >$ *molCol* or not) loops the *repair* operator executes maximum 2 times (whether $b >$ *molCol* or not). Either line numbers 11–13 or line numbers 27–29 of Algorithm 1.
- In each iteration within *popSize* loop, but outside the (if $b >$ *molCol*) loop the *repair* operator executes 2 times (line numbers 33–35 of Algorithm 1).

Table 8 Results of dataset 3 under runtime 12 h

Seq.	Runtime = 12 h					
	$\mathrm{CRO}_{\mathrm{PFO}}$			GAAM [5]		
	E_{best}	E_{mean}	E_{std}	E_{best}	E_{mean}	E_{std}
S3.1	11	11.00	0.00	11	11.00	0.00
S3.2	13	13.00	0.00	13	13.00	0.00
S3.3	9	9.00	0.00	9	9.00	0.00
S3.4	18	18.00	0.00	18	18.00	0.00
S3.5	31	31.00	0.00	31	31.00	0.00
S3.6	34	34.00	0.00	34	33.96	0.14
S3.7	55	55.00	0.00	55	54.46	0.50
S3.8	59	59.00	0.00	59	59.00	0.00

From these two points, we can say that in each iteration within *popSize* loop *repair* executes 4 times which is maximum. Actually, it will be less than 4 in average case. By considering the worst case the number of evaluations in one run is $200 \times 500 \times 4 = 4 \times 10^5$. From Table 5, it can be noticed that our proposed algorithm provides better E_{best} values for the sequences S2.2, S2.4, S2.6, S2.9 and for other sequences it gives same E_{best} values as GAAM. For all sequences of this dataset, $\mathrm{CRO}_{\mathrm{PFO}}$ has obtained better E_{mean} and E_{std} values than GAAM. Our proposed algorithm has also showed the less number of solution evaluations (NE) than GAAM [5].

Table 6 shows the E_{best}, E_{mean} and E_{std} for dataset 3. Dataset 3 contains sequences of length 20–64. Here, we used three different parameters. For dataset 3 we set *initialKE* $= 100$ and *KELossRate* $= 0.5$. The *molCol* value was set 1 for *length* ≤ 36 and 0.5 for *length* > 36. Fifty independent runs were performed for each individual sequence. From this table, we can notice that our proposed algorithm provides better E_{mean} and E_{std} for sequences S3.6, S3.7 and for other sequences, the proposed algorithm has shown same E_{best} values as GAAM [5]. Here, the values in boldface indicate the better performance of our algorithm.

Table 7 shows the E_{best}, E_{mean} and E_{std} for dataset 4. Dataset 4 contains sequences of length 46–136. Here, two different parameters were used. For sequences of length 46 and 58, we set *KELossRate* $= 0.5$, for length 103 and 136, the *KELossRate* $= 0.05$. The *molCol* value was set 0.5 for all sequences. Fifty independent runs were performed for each individual sequence. For all sequences of Table 7, $\mathrm{CRO}_{\mathrm{PFO}}$ has obtained better E_{mean} and E_{std} values. In particular, when the sequence length is long, $\mathrm{CRO}_{\mathrm{PFO}}$ obtained better E_{best}, E_{mean} and E_{std} values than GAAM.

Table 8 shows the E_{best}, E_{mean} and E_{std} for dataset 3. Dataset 3 contains sequences of length 20–64. The stopping condition was 12 h for dataset 3 and our algorithm

Table 9 A performance measure of datasets 1 and 2 with $initialKE = 100$, $KELossRate = 0.5$ and the number of runs was 30

Seq.	CRO$_{PFO}$					Seq.	GAAM [5]				
	E_{best}	E_{mean}	E_{std}	Speed	Div		E_{best}	E_{mean}	E_{std}	Speed	Div
S1.1	32	32.00	0.00	13,843.51	4.31	S2.1	32	32.00	0.00	14,214.46	3.34
S1.2	34	34.00	0.00	12,248.94	3.44	S2.2	38	38.00	0.00	13,214.54	3.59
S1.3	34	34.00	0.00	13,724.23	3.67	S2.3	45	45.00	0.00	11,854.14	3.11
S1.4	33	33.00	0.00	11,876.71	2.50	S2.4	42	42.00	0.00	12,385.43	2.72
S1.5	32	32.00	0.00	14,978.51	4.01	S2.5	42	42.00	0.00	12,745.10	4.48
S1.6	32	32.00	0.00	12,487.27	7.31	S2.6	35	35.00	0.00	13,547.41	5.73
S1.7	32	32.00	0.00	15,104.18	3.32	S2.7	28	28.00	0.00	14,496.98	4.61
S1.8	31	31.00	0.00	11,871.74	4.55	S2.8	38	38.00	0.00	13,482.48	4.25
S1.9	34	34.00	0.00	14,814.24	6.34	S2.9	41	41.00	0.00	12,578.59	2.49
S1.10	33	33.00	0.00	12,541.71	5.34	S2.10	31	31.00	0.00	14,271.14	3.69

Table 10 A performance measure of dataset 3 with $initialKE = 100$, $KELossRate = 0.5$ and the number of runs was 30

Seq.	CRO$_{PFO}$				
	E_{best}	E_{mean}	E_{std}	Speed	Div
S3.1	11	11.00	0.00	19,286.78	5.15
S3.2	13	13.00	0.00	18,215.45	5.52
S3.3	9	9.00	0.00	18,548.91	3.65
S3.4	18	18.00	0.00	16,436.36	3.75
S3.5	31	31.00	0.00	13,749.31	4.47
S3.6	34	34.00	0.00	12,473.32	2.72
S3.7	55	55.00	0.00	10,734.25	3.11
S3.8	59	59.00	0.00	9218.42	3.59

Table 11 Mean time comparison of dataset 1 with $initialKE = 100$, $KELossRate = 0.5$ and the number of runs was 20

Seq.	Runtime (s)		
	CRO$_{PFO}$	GAAM [5]	*Speed up*
S1.1	223.97	251	1.12
S1.2	463.86	549	1.18
S1.3	372.68	532	1.42
S1.4	294.24	525	1.78
S1.5	298.56	358	1.19
S1.6	402.47	619	1.53
S1.7	756.37	2447	3.23
S1.8	268.74	317	1.17
S1.9	439.46	945	2.15
S1.10	374.32	412	1.10

obtained the best result. From all of the tables, we can observe that the results of our proposed algorithm are better than the GAAM especially for the mean energy (E_{mean}) and the standard deviation (E_{std}) values for all the datasets.

5.4 Performance measure

In order to do a fairer comparison, we performed some more statistical tests over the datasets such as the diversity of the final population and the speed (number of sequence evaluations per second) of our proposed algorithm. Diversity measures the structural differences in each population. The mean diversity of final population has been calculated by following equation:

$$\text{div} = \frac{2}{\text{TR} * (p-1) * p} \sum_{i=1}^{\text{TR}} \sum_{m=1}^{p} \sum_{n=m+1}^{p} D(a_m, a_n) \tag{9}$$

Here, $p = popSize$ and $D(a_m, a_n)$ represent the Hamming distance between sequence a_m and a_n. The E_{mean} value has been multiplied by -1 which means higher E_{mean} value shows better efficiency. The statistical tests were done over the datasets 1, 2 and 3 to check the speed and diversity in population. It seems that speed is a number of function evaluations per second. Tables 9 and 10 show the results.

Table 12 Mean time comparison of dataset 2 with $initialKE = 100$, $KELossRate = 0.5$ and the number of runs was 20

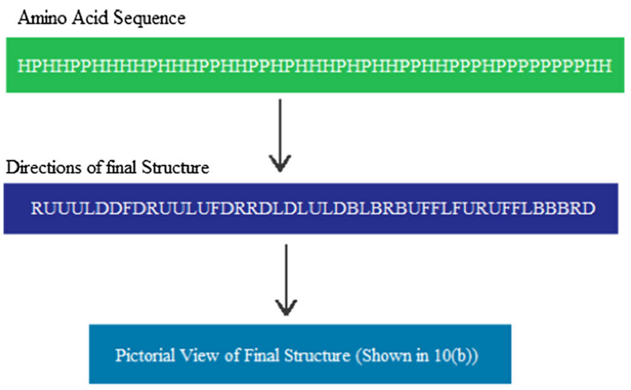
Seq.	Runtime (s)		
	CRO$_{PFO}$	GAAM [5]	Speed up
S2.1	588.774	1061	1.8
S2.2	533.193	2171	4.07
S2.3	967.16	1481	1.53
S2.4	1373.645	1763	1.28
S2.5	2241.341	2373	1.05
S2.6	567.112	1288	2.27
S2.7	1180.383	1337	1.13
S2.8	1122.502	1890	1.68
S2.9	1361.756	2239	1.64
S2.10	610.531	1207	1.97

Table 13 Mean time comparison of dataset 3 with *initialKE* = 100, *KELossRate* = 0.5 and the number of runs was 20

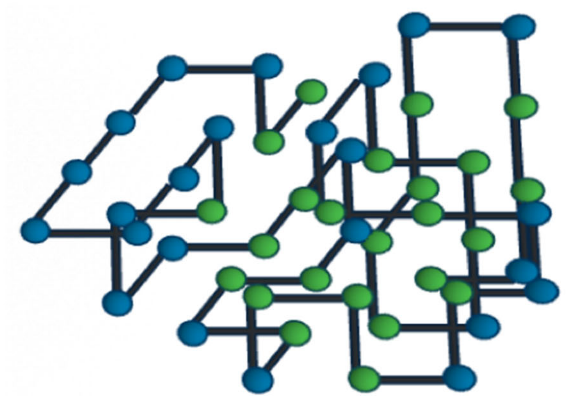
Seq.	Runtime (s)		
	CRO_{PFO}	GAAM [5]	Speed up
S3.1	36.917	87	2.36
S3.2	49.117	178	3.62
S3.3	28.751	211	7.39
S3.4	983.917	1295	1.32
S3.5	430.713	459	1.06
S3.6	263.303	448	1.70
S3.7	2452.67	2974	1.21
S3.8	2889.21	3692	1.28

5.5 Execution time comparison

In this subsection, the execution time comparison of datasets 1, 2 and 3 was performed. The sequences are of different lengths. The default parameter settings were used for the sequences. Speed up value indicates the rate of increase performance of CRO_{PFO} algorithm. The speeds of the algorithm for different datasets with respect to GAAM [5] are shown in Tables 11, 12 and 13. From the tables, it can be noticed that the highest speed up value = 7.39 and lowest speed up value = 1.05 of our proposed CRO_{PFO} algorithm. Here, the stopping condition was the maximum target energy value or until the *initialKE* is not reached 0. The speed up values show that the proposed algorithm is better than GAAM due to less execution time and high speed up values. From these tables, it can be observed that our proposed algorithm takes less time for evaluating the amino acid sequences than GAAM [5] and provides highest energy values most of the times. In our developed algorithm, we worked with the directions of structures, not with the real structure. Because the final direction array provides sufficient information of the structures. We know that for the PFO problem input is the amino acid sequence which is a combination of H and P monomers as defined earlier. For an example, HPHHPPHHHHPHHHPPHHPPHPHHHPHP HHPPHHPPPHPPPPPPPPHH is an input sequence of 48 lengths. According to CRO_{PFO} algorithm, a random population of fixed size is created first. Then, our algorithm applies the four operators of CRO and the additional mechanisms on the population which is a set of direction arrays. After all the iterations and modifications, the best

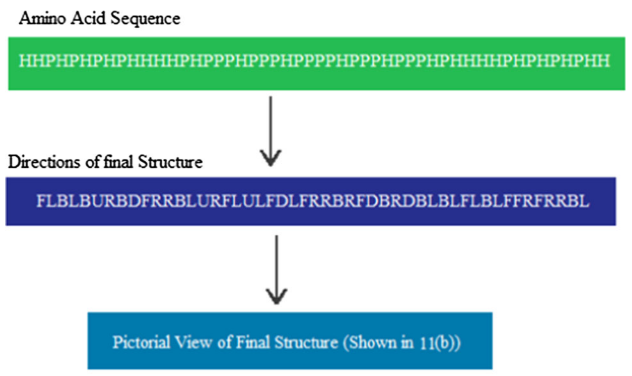

Amino Acid Sequence
HPHHPPHHHHPHHHPPHHPPHPHHHPHPHHPPHHPPPHPPPPPPPPHH
Directions of final Structure
RUUULDDFDRUULUFDRRDLDLULDBLBRBUFFLFURUFFLBBBRD
Pictorial View of Final Structure (Shown in 10(b))

(a) PFO problem at a glance

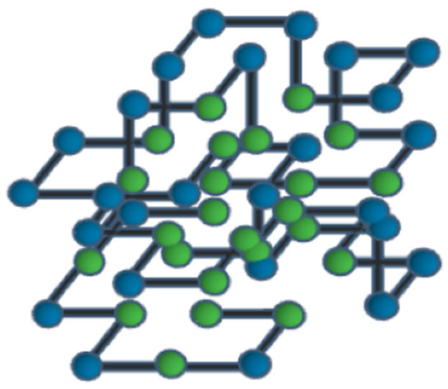


(b) Structure of final directions with $E_c = (-1) * F_c = 32$

Fig. 10 Progress of energy value of dataset 1 with respect to time (s)


Amino Acid Sequence
HHPHPHPHPHHHHPHPPPHPPPHPPPPHPPPHPPPHPHHHHPHPHPHPHH
Directions of final Structure
FLBLBURBDFRRBLURFLULFDLFRRBRFDBRDBLBLFLBLFFRFRRBL
Pictorial View of Final Structure (Shown in 11(b))

(a) PFO problem at a glance



(b) Structure for final directions with $E_c = (-1) * F_c = 34$

Fig. 11 Progress of energy value of dataset 1 with respect to time (s)

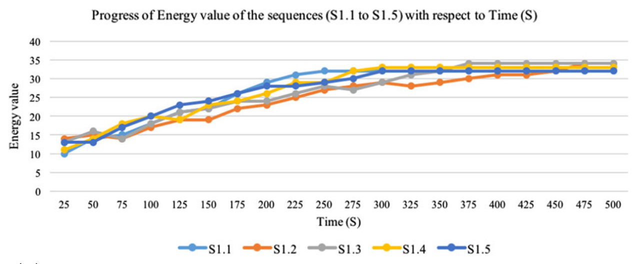
Table 14 Performance analysis of datasets 1–4 with *initialKE* = 100, *KELossRate* = 0.5 and the number of runs was 30

Seq.	With *H&P* and without *Evolution* mechanism			With *Evolution* and without *H&P* mechanism			Without *H&P* and *Evolution* mechanism			With *H&P* and *Evolution* mechanism		
	E_{best}	E_{mean}	E_{std}	E_{best}	E_{mean}	E_{std}	E_{best}	E_{mean}	E_{std}	E_{best}	E_{mean}	E_{std}
S1.1	28	24.78	1.41	27	25.64	2.28	17	15.88	13.12	32	32.00	0.00
S1.2	30	27.66	1.51	28	26.22	1.41	19	14.12	12.71	34	34.00	0.00
S1.3	29	27.12	1.18	28	26.12	1.64	18	15.97	12.97	34	34.00	0.00
S1.4	29	26.84	1.58	27	25.74	2.13	17	15.74	13.27	33	33.00	0.00
S1.5	28	25.46	1.52	26	24.54	2.07	16	14.57	12.93	32	32.00	0.00
S1.6	29	26.88	1.82	28	25.68	2.23	16	15.03	13.32	32	32.00	0.00
S1.7	27	24.32	1.61	27	25.52	1.89	15	13.72	12.49	32	32.00	0.00
S1.8	29	25.12	1.46	29	26.48	1.97	15	13.81	12.07	31	31.00	0.00
S1.9	28	24.88	2.22	26	23.84	2.08	18	16.94	14.98	34	34.00	0.00
S1.10	27	25.82	1.71	25	23.26	2.15	18	16.39	14.13	33	33.00	0.00
S2.1	29	26.98	2.22	27	25.18	1.51	17	15.38	12.37	32	32.00	0.00
S2.2	32	29.76	2.89	30	28.66	1.93	19	18.01	17.92	38	38.00	0.00
S2.3	40	37.12	2.91	39	36.52	2.28	23	21.72	20.19	45	45.00	0.00
S2.4	38	34.88	3.11	37	34.14	2.39	20	18.23	16.71	42	42.00	0.00
S2.5	37	36.42	1.64	36	34.38	1.79	21	19.82	17.09	42	42.00	0.00
S2.6	31	29.84	1.86	31	29.84	1.86	17	15.73	14.43	35	35.00	0.00
S2.7	26	25.36	1.21	25	24.18	1.43	13	11.14	9.86	28	28.00	0.00
S2.8	33	31.62	1.97	31	29.52	1.67	18	17.24	15.45	38	38.00	0.00
S2.9	36	33.44	2.52	35	32.46	2.32	19	17.03	16.73	41	41.00	0.00
S2.10	28	26.86	1.71	27	25.91	1.89	14	13.92	12.14	31	31.00	0.00
S3.1	10	9.24	0.58	10	9.52	0.38	5	4.29	4.08	11	11.00	0.00
S3.2	12	11.12	0.81	11	10.44	0.51	6	5.17	4.91	13	13.00	0.00
S3.3	9	8.52	0.39	9	8.32	0.53	4	3.73	3.08	9	9.00	0.00
S3.4	16	15.14	0.78	15	14.74	0.89	8	7.15	6.97	18	18.00	0.00
S3.5	28	25.48	1.55	27	24.68	1.21	14	12.83	11.48	31	31.00	0.00
S3.6	30	28.86	1.62	28	25.54	2.27	16	15.73	14.57	34	34.00	0.00
S3.7	49	45.38	2.81	47	44.76	1.73	26	24.15	23.81	55	55.00	0.00
S3.8	51	47.82	3.07	49	45.28	2.93	27	24.73	23.49	59	59.00	0.00
S4.1	31	29.87	1.89	30	29.08	1.09	16	14.84	13.86	35	35.00	0.00
S4.2	40	38.64	2.97	38	36.84	2.23	21	19.64	18.47	44	44.00	0.00
S4.3	54	52.46	2.25	53	51.91	2.07	26	24.71	23.54	58	58.00	0.00
S4.4	71	68.32	2.88	69	66.67	2.93	34	32.78	31.09	75	73.83	0.89

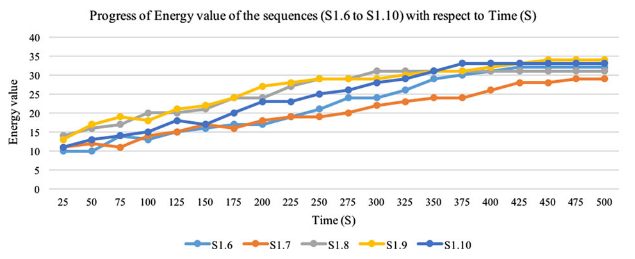
structure is selected among all the structures. The sequence of directions of the final structure is RUUULDDFDRU ULUFDRRDLDLULDBLBRBUFFLFURUFFLBBBRDB. The pictorial view for the PFO problem is given in Fig. 10a. Here, the green monomers represent hydrophobic amino acid and the blue monomer represents the polar amino acids. Another example can be given for the sequence HHPHPHPHPHHHHPHPPPHPPPHPPPPHPP PHPPPHPHHHHPHPHPHPHH that is a sequence of 50 length. After completing all the steps according to our algorithm, the best structure is selected among all the structures. The sequence of directions of the best structure is FLBLBURBDFRRBLURFLULFDLFRRBRFDBRDBL BLFLBLFFRFRRBL. The pictorial view of PFO problem for this sequence is given in Fig. 11a.

5.6 Performance analysis

In Table 14, it can be observed that four types of results have been shown to analyze the performance of our proposed algorithm. First, we calculated the energy value of the sequences with H&P compliance and without evolution mechanism. Second, the energy values were calculated of the sequences with evolution mechanism and without H&P compliance.



(a) Progress of energy value of the sequences S1.1 to S1.5 with respect to Time (S)



(b) Progress of energy value of the sequences S1.6 to S1.10 with respect to Time (S)

Fig. 12 Progress of energy value of dataset 1 with respect to time (s)

After that, we found out the energy value of the sequences without two mechanisms. It can be noticed that without these two types of mechanisms the algorithm cannot find out the target best energy values and most of the cases the obtained energy values are more than 49

In Fig. 12a, b we have shown the convergence results of the algorithm with time. From the graphs, it can be noticed that the energy value of each sequence (S1.1–S1.10) is increased with respect to the time. Within the interval of time, the energy value of each sequence is increased sequentially in most of the cases and finally gets the best energy value. The convergence curves show that our proposed algorithm has progressed with time and found out the best results.

6 Conclusions

The protein folding optimization (PFO) is a familiar NP-hard optimization problem, and here, we have used chemical reaction optimization (CRO) algorithm to solve this problem. CRO is a recent population-based metaheuristic algorithm that has already been successfully applied to many well-known optimization problems. It is a nature-inspired algorithm that mimics the behavior of chemical reactions. The main challenge of this algorithm is the search space formulation. In PFO problem, the actual search space is massive with respect to the length of the protein and the size of the actual search space increases exponentially with the protein length. So, it is a very challenging task to select a limited number of structures from the huge number of possible choices. We have redesigned four basic operators of CRO algorithm to solve PFO problem. Additionally, we have also designed two extra mechanisms, evolution and H&P compliance. These two extra mechanisms help to raise the performance of the algorithm dramatically. Our algorithm includes a repair mechanism that produces valid structures from invalid structures. The performance of our proposed algorithm is appreciable. We have compared the results of our algorithm with the genetic algorithm with an advanced mechanism (GAAM) which is a state-of-the-art population-based algorithm, and from the results, it is clear that the performance of our algorithm is better than GAAM.

A particular study on parameters of CRO may yield better results and less execution time for this problem. Since there is no fixed rule for setting of the parameters of CRO, finding the right values for the parameters is a tough task. So more experiment and study on parameters may give better results in the case of PFO problem.

Compliance with ethical standards

Conflict of interest All authors declare that they have no conflict of interest.

References

1. Santana R, Larrañaga P, Lozano JA (2008) Protein folding in simplified models with estimation of distribution algorithms. IEEE Trans Evolut Comput 12(4):418–438
2. Brändén CI, Tooze J (1999) Introduction to protein structure. Taylor & Francis, Milton Park
3. Huang HJ, Lee KJ, Yu HW, Chen CY, Hsu CH, Chen HY, Tsai FJ, Chen CYC (2010) Structure-based and ligand-based drug design for HER 2 receptor. J Biomol Struct Dyn 28(1):23–37
4. Dal Palu A, Dovier A, Pontelli E (2005) Heuristics, optimizations, and parallelism for protein structure prediction in clp (fd). In: Proceedings of the 7th ACM SIGPLAN international conference on principles and practice of declarative programming. ACM, pp 230–241
5. Bošković B, Brest J (2016) Genetic algorithm with advanced mechanisms applied to the protein structure prediction in a hydrophobic-polar model and cubic lattice. Appl Soft Comput 45:61–70
6. Garman EF (2014) Developments in X-ray crystallographic structure determination of biological macromolecules. Science 343(6175):1102–1108
7. Dill KA (1985) Theory for the folding and stability of globular proteins. Biochemistry 24(6):1501–1509
8. Berger B, Leighton T (1998) Protein folding in the hydrophobic-hydrophilic (HP) model is NP-complete. J Comput Biol 5(1):27–40
9. Guo YZ, Feng EM, Wang Y (2007) Optimal HP configurations of proteins by combining local search with elastic net algorithm. J Biochem Biophys Methods 70(3):335–340
10. Mansour N, Kanj F, Khachfe H (2010) Evolutionary algorithm for protein structure prediction. In: 2010 Sixth international conference on natural computation, vol 8. IEEE, pp 3974–3977

11. Nemhauser G, Wolsey L (1988) The scope of integer and combinatorial optimization. In: Wolsey LA, Nemhauser GL (eds) Integer and combinatorial optimization. Wiley, New York, pp 1–26
12. Lin CJ, Su SC (2011) Protein 3d HP model folding simulation using a hybrid of genetic algorithm and particle swarm optimization. Int J Fuzzy Syst 13(2):140–147
13. Lam AY, Xu J, Li VO (2010) Chemical reaction optimization for population transition in peer-to-peer live streaming. In: IEEE congress on evolutionary computation. IEEE, pp 1–8
14. Saifullah CK, Islam MR (2016) Chemical reaction optimization for solving shortest common supersequence problem. Comput Biol Chem 64:82–93
15. Lam AY, Li VO, James J (2012) Real-coded chemical reaction optimization. IEEE Trans Evolut Comput 16(3):339–353
16. Lam AY, Li VO (2010) Chemical reaction optimization for cognitive radio spectrum allocation. In: 2010 IEEE global telecommunications conference GLOBECOM 2010. IEEE, pp 1–5
17. Pan B, Lam AY, Li VO (2011) Network coding optimization based on chemical reaction optimization. In: 2011 IEEE global telecommunications conference-GLOBECOM 2011. IEEE, pp 1–5
18. Xu J, Lam AY, Li VO (2010) Chemical reaction optimization for the grid scheduling problem. In: 2010 IEEE international conference on communications. IEEE, pp 1–5
19. Xu J, Lam AY, Li VO (2011) Stock portfolio selection using chemical reaction optimization. In: Proceedings of international conference on operations research and financial engineering (ICORFE 2011), pp 458–463
20. James J, Lam AY, Li VO (2011) Evolutionary artificial neural network based on chemical reaction optimization. In: 2011 IEEE congress of evolutionary computation (CEC). IEEE, pp 2083–2090
21. Xiao J, Li LP, Hu XM (2014) Solving lattice protein folding problems by discrete particle swarm optimization. J Comput 9(8):1904–1913
22. Mansour N, Kanj F, Khachfe H (2012) Particle swarm optimization approach for protein structure prediction in the 3D HP model. Interdiscip Sci Comput Life Sci 4(3):190–200
23. Khimasia MM, Coveney PV (1997) Protein structure prediction as a hard optimization problem: the genetic algorithm approach. Mol Simul 19(4):205–226
24. König R, Dandekar T (1999) Improving genetic algorithms for protein folding simulations by systematic crossover. BioSystems 50(1):17–25
25. Chatterjee S, Smrity RA, Islam MR (2016) Protein structure prediction using chemical reaction optimization. In: 2016 19th international conference on computer and information technology (ICCIT). IEEE, pp 321–326
26. Custódio FL, Barbosa HJ, Dardenne LE (2014) A multiple minima genetic algorithm for protein structure prediction. Appl Soft Comput 15:88–99
27. Shmygelska A, Hoos HH (2003) An improved ant colony optimisation algorithm for the 2D HP protein folding problem. In: Conference of the Canadian society for computational studies of intelligence. Springer, pp 400–417
28. Thilagavathi N, Amudha T (2015) Aco-metaheuristic for 3D-HP protein folding optimization. ARPN J Eng Appl Sci 10(11):4948–4953
29. Lam AY, Li VO (2012) Chemical reaction optimization: a tutorial. Memet Comput 4(1):3–17
30. Lam AY, Li VO (2010) Chemical-reaction-inspired metaheuristic for optimization. IEEE Trans Evolut Comput 14(3):381–399
31. Bechikh S, Chaabani A, Said LB (2015) An efficient chemical reaction optimization algorithm for multiobjective optimization. IEEE Trans Cybern 45(10):2051–2064
32. Bazzoli A, Tettamanzi AG (2004) A memetic algorithm for protein structure prediction in a 3D-lattice HP model. In: Workshops on applications of evolutionary computation. Springer, pp 1–10
33. Shmygelska A, Hoos HH (2005) An ant colony optimisation algorithm for the 2D and 3D hydrophobic polar protein folding problem. BMC Bioinform 6(1):30
34. Garza-Fabre M, Rodriguez-Tello E, Toscano-Pulido G (2015) Constraint-handling through multi-objective optimization: the hydrophobic-polar model for protein structure prediction. Comput Oper Res 53:128–153
35. Islam MK, Chetty M (2013) Clustered memetic algorithm with local heuristics for ab initio protein structure prediction. IEEE Trans Evolut Comput 17(4):558–576