Protein structure prediction from inaccurate and sparse NMR data using an enhanced genetic algorithm

Md. Lisul Islam[a], Swakkhar Shatabda[b], Mahmood A. Rashid[c,d,*], M. G. M. Khan[d], M Sohel Rahman[e]

[a] *Department of Computer Science, Indiana University, Bloomington, USA*
[b] *Department of Computer Science and Engineering, United International University, Dhaka, Bangladesh*
[c] *Institute for Integrated & Intelligent Systems, Griffith University, Brisbane, Australia*
[d] *School of Computing, Information and Mathematical Sciences, The University of the South Pacific, Suva, Fiji*
[e] *Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh*

Abstract

Nuclear Magnetic Resonance Spectroscopy (most commonly known as NMR Spectroscopy) is used to generate approximate and partial distances between pairs of atoms of the native structure of a protein. To predict protein structure from these partial distances by solving the Euclidean distance geometry problem from the partial distances obtained from NMR Spectroscopy, we can predict three-dimensional (3D) structure of a protein. In this paper, a new genetic algorithm is proposed to efficiently address the Euclidean distance geometry problem towards building 3D structure of a given protein applying NMR's sparse data. Our genetic algorithm uses *(i)* a greedy mutation and crossover operator to intensify the search; *(ii)* a twin removal technique for diversification in the population; *(iii)* a random restart method to recover from stagnation; and *(iv)* a compaction factor to reduce the search space. Reducing the search space drastically, our approach improves the quality of the search. We tested our algorithms on a set of standard benchmarks. Experimentally, we show that our enhanced genetic algorithms significantly outperforms the traditional genetic algorithms and a previously proposed state-of-the-art method. Our method is capable of producing structures that are very close to the native structures and hence, the experimental biologists could adopt it to determine more accurate protein structures from NMR data.

*Corresponding author
Email address: mahmood.rashid@griffith.edu.au (Mahmood A. Rashid)

Keywords: Protein structure prediction; Sparse Data; Molecular Distance Geometry; Nuclear Magnetic Resonance Spectroscopy; Genetic algorithms

1. Introduction

Protein structure prediction remains one of the highly researched and challenging problems in molecular biology for several decades. Proteins are virtually involved almost in every process within the living cell. It is hypothesized that the 3D native structure–the most stable structure with minimum free energy in a particular environment–of a protein mostly determines its functionality [1] with some exceptions. Determining this native structure has significant importance in rational drug design. Nuclear magnetic resonance (NMR) spectroscopy is a widely applied technique of predicting native structure of proteins. NMR Spectroscopy generates incomplete inter-atomic distance dataset for a given target protein. To find a 3D structure from this inter-atomic dataset we need to solve the molecular distance geometry problem (MDGP)–from a given set of Euclidean distances between the atoms in a protein, the MDGP tries to find the Cartesian coordinates of the atoms.

In reality, NMR Spectroscopy can produce inter-atomic distances with a significant degree of inaccuracy only on a subset of the pairs of atoms that are spatially close to each other. Eventually, we are addressing a variant of the MDGP with missing and inaccurate data by effectively setting the upper and lower bounds of only a subset of the Euclidean distances. Some computational approaches [2, 3, 4, 5, 6, 7] applied sparse and inaccurate data on real instances to solve such problems. We noticed that the complete search methods like spatial branch and bound (sBB) and stochastic methods like variable neighbourhood search (VNS) can solve the problem only for small sized proteins (up to 50 amino acids) [8, 9] however, fail quickly in case of larger proteins.

In this paper, we present an enhanced genetic algorithm to solve the MDGP for incomplete and inaccurate NMR data. We first present GMT3R which combines *(i)* a greedy mutation operator (GM) to intensify the search, *(ii)* a twin removal technique (TR) to diversify the population, and *(iii)* a random restart method (RR) to overcome stagnation. We also enhanced GMT3R further to GMT3R$^+$ that exploits a greedy crossover operator along with a compaction factor to reduce the search space. This fusion of the compaction factor and the crossover operator in the sequel helps the search to converge quickly and improves the solution quality dramatically. Experimental outcomes on proteins

containing within the range of 50–2147 amino acids shows that our algorithms outperform the standard genetic algorithms and the state-of-the-art algorithms proposed thus far. Some preliminary results of GMT3R (denoted there as GreMuTRRR) were presented in [10].

2. Background

2.1. Distance Geometry Problem

In the MDGP, we are given the lower and upper bounds of the inter atomic distances. For each pair of atoms (i, j), let us assume that the lower (upper) bound of the distance between them is l_{ij} (u_{ij}). So, if the real distance between them is d_{ij}, then we have the following:

$$l_{ij} \leq d_{ij} \leq u_{ij}, \forall (i,j) \in E$$

Here, E denotes the set of the inter atomic distances. For this given set of bounds on the inter-atomic distances, the task is to find a set of Cartesian coordinates $C \equiv c_1, c_2, \cdots, c_n \in \mathbb{R}^3$ of atoms of a molecule. Here, these coordinates correspond to three dimensional points in the Cartesian space, i.e., $c_i \equiv (x_i, y_i, z_i) \in \mathbb{R}^3$. Now, we define a pairwise error function e_{ij} that finds the deviation of the inter-atomic distances in C with that given in the NMR data. Formally, e_{ij} is defined as follows:

$$e_{ij} = max\{l_{ij} - ||c_i - c_j||, ||c_i - c_j|| - u_{ij}, 0\}$$

Note that, we have used the similar notations and the problem model originally proposed in [2]. The values of the upper limits and the lower limits for the distance pairs are taken as suggested in the original paper. This suggestion is however supported in other work in the literature as well [11, 12].

The problem described in [2] is a global minimization problem with an objective function:

$$f(C) = \left(\frac{1}{|E|} \sum_{(i,j) \in E} e_{ij}^2 \right)^{1/2}$$

Here, the NMR Spectroscopy data E, is sparse.

2.2. Genetic Algorithm

A Genetic Algorithm (GA)–duly inspired by the biological evolution–is a population-based search algorithm comprised of a number of sub-algorithms. GAs are widely used for different search optimization problems in various domain. It basically starts with a set of randomly generated initial solutions, also known as initial population. Each individual in the population, also called a chromosome, carries the encoded properties which are eventually altered in the evolution process.

It maintains an iterative process to move through generations. In each generation, the individuals in the population are allowed to participate in generating of new individuals using different operators which also mimic the process of natural evolution like mutation, recombination or survival of the fittest. A generic recombination, widely known as crossover and a mutation operator are illustrated in Figures 1 and 2, respectively.

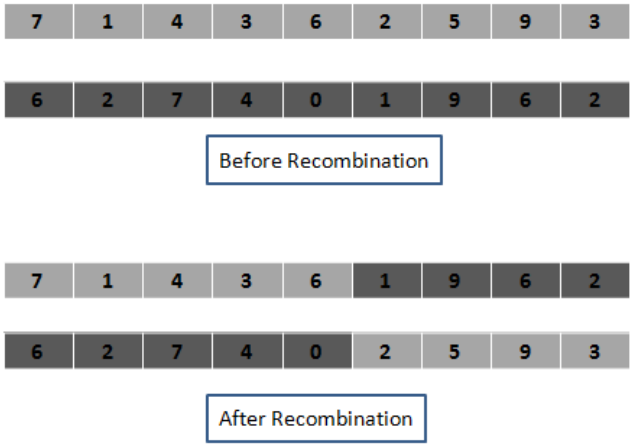


Figure 1: Genetic Crossover operator

The fitness of each of the individuals is evaluated in each generation. Generally, the fitness of an individual is obtained from the value of the optimization function for that individual solution which also indicates how well that particular solution addressing the problem. Usually, the more fit individuals are selected to breed among them to generate even fitter individuals for the next generation. This repetitive evolution process is controlled by some termination strategy such as a threshold on number of generations to run or attainment of

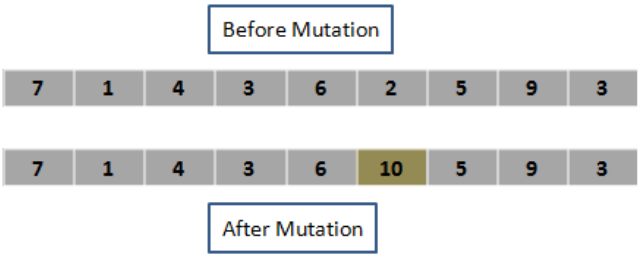


Figure 2: Genetic Mutation operator

a certain quality in solutions.

3. Related work

Some variants of Euclidean distance geometry problem are applied to different problems in various domains such as, wireless ad hoc network localization [13], inverse kinematic problem [14], multidimensional scaling [15], and protein structure determination [7]. In [16], the authors present a survey on MDGP and they claim that once the backbone (only the alpha carbons) of the protein is determined, the whole structure containing other atoms such as carbon, nitrogen can easily be found out by solving another instance of MDGP.

The variant of MDGP where the all the pairwise distances of atoms—$(i, j) \in E = \{1, 2, \cdots\}^2$ and $d_{ij} = l_{ij} = u_{ij}$—are taken into account, a polynomial time algorithm is required to find an exact solution [17]. The problem is solvable by a linear time algorithm [4] even, when some of the pairwise distances are missing. Nevertheless, the variant of MDGP is NP-hard [18] given that the data is sparse and inaccurate. A survey on applying computational methods solving this variant of MDGP, is presented in [3].

Spatial branch and bound [19, 20] and variable neighborhood search (VNS) [21] methods amongst the general purpose methods, are not scalable [9]. Smoothing based methods such as DGSOL [7, 18] also fail for larger instances of the problem. In [22], the hybridization of VNS and DGSOL provided better results for larger instances but resulted into a slow algorithm. In another work [23], a combinatorial build-up algorithm was proposed. However, one point is notable here that all of these methods were tested only on the dense instances. The graph decomposition methods [2] and the NLP formulations [24] are amongst the other notable methods applied to address this problem.

In [25], the authors dealt with a variant of the MDGP without considering the erroneous or missing data. They have solved the MDGP in two steps. First, they use a Branch and Prune algorithm to find the coordinates of the backbone hydrogen atoms and then follow it up with another algorithm that solves a system of linear equations by utilizing the knowledge-base on bond length and bond angles previously obtained to find other atoms such as carbon, nitrogen etc. in the protein structures. However, the assumption that the NMR provides exact distance measure, is unrealistic.

In [5], the authors presented a comparison between an exact method (Branch and Prune) and a meta-heuristics based method (Monkey Search) to solve MDGP. They perturbed and introduced errors in the distance data. Voller et al. [26] surveyed on geometric buildup approaches to problems with sparse but exact distances and other approaches that deal with inexact distances or distance bounds. In another work, Lavor (*et al.*) [27] considered interval distances and solved the MDGP problem using pre-decided manual atom sequence in the backbone structure.

4. Our Methods

In each generation of the evolution, individuals from the population are selected using *tournament selection* to act as parents and take part in recombination using one-point crossover to produce offspring to be embraced in the next generation. We have applied a *Greedy Crossover* strategy where the crossover point of the participating parent is chosen greedily. Mutation operators are also applied with some probability to the newly devised offspring and a probabilistic choice is made between *Greedy Mutation* and *Random Mutation*. Individual with the best fitness is always monitored in the next generation to ensure elitism. Recurrent twin removal procedure is activated to diversify the search and random restart is also triggered occasionally to recover from stagnation. Our algorithm reaches at convergence when no substantial amount of improvement in quality of global best individual in the population is encountered for a given number of iteration.

Note that we in fact, present two version of our algorithms, namely, *GMT3R* and *GMT3R*$^+$. As the name indicates the latter is an extended version of the former where we have infused a greedy crossover operator as well as an interesting compaction factor to reduce the search space for the problem. In the following subsections, different constituents of our algorithms are described

in details. In Algorithm 1, we present the outline of $GMT3R^+$ identifying the components that are inactive in $GMT3R$ in comments.

Algorithm 1: GMT3R$^+$()

```
1  intensificationCounter = 0
2  stagnationCounter = 0
3  intensificationProbability = 0.8
4  d = 5 // crossover points to be considered
5  r = 50 /* number of candidate values for a chromosome used in greedyMutate
   subroutine */
6  SSCF = 0.273V − 1.746, where V is the number of atoms in the predicted
   protein structure. //In GMT3R, we considered SSCF=1
7  Initialize the population, P randomly considering SSCF
8  while termination criteria is not fulfilled do
9      P_new = {globalBest}
10     for each individual X ∈ P do
11         ⟨X_1, X_2⟩ = tournamentSelection(P)
12         X_new = GreedyCrossOver(X_1, X_2)
13         //In GMT3R, traditional CrossOver was used with d = 1
14         add X_new to P_new
15     end
16     for each individual X in P_new do
17         if rand(0, 1) ≤ intensificationProbability then
18             greedyMutate(X) /* while mutating an Individual we considered
               proper value of SSCF,which was 1 in GMT3R */
19         else
20             randomMutate(X) // SSCF was also used
21         end
22     end
23     find the individual X_best ∈ P_new with best fitness
24     if fitness(globalBest) < fitness(X_best) then
25         globalBest = X_best
26         stagnationCounter = 0
27     else
28         stagnationCounter + +
29     end
30     if intensificationCounter ≥ nonDiverseThreshold then
31         activate twinRemoval(P_new) procedure
32         intensificationCounter = 0
33         stagnationCounter = 0
34     else
35         intensificationCounter + +
36     end
37     if stagnationCounter ≥ stagnationThreshold then
38         activate randomRestart(P_new) procedure
39         stagnationCounter = 0
40     end
41     P = P_new
42 end
43 return globalBest
```

Table 1: Value of the parameter *SSCF*

Protein Id	**V**	**SSCF**
1PTQ	50	15
1LFB	77	20
1F39	101	40
1AX8	130	45
1RGS	264	60
1TOA	277	60
1KDH	356	75
1BPM	481	130
1MQQ	679	200
A1PTQ	402	130
A1LFB	641	145
A1F39	767	180
A1AX8	1003	250

4.1. Search Space

A Protein structure is referred to as the bio-molecular structure conformed by linear sequence of α-amino acids held together by peptide bonds in a polypeptide chain. These chemical peptide bonds cannot juxtapose any pair of these α-amino acids in a closer vicinity to each other than 3.8Å. Hence, the upper bound of the length of linear sequence of these α-amino acids in a certain protein structure is approximated to $3.8 \times V$ in each directions of the 3D search space when these atoms are aligned along a straight line. Here, V is the number of constituents amino acid atoms in the target protein. Since, very rarely a protein structure takes a shape of straight chain of amino acids, we further reduce the search space dividing the upper bound of the 3D space in each direction by a parameter, *SearchSpaceCompactionFactor(SSCF)* to achieve faster convergence. For a certain protein structure, the *SSCF* parameter is set to a value that is commensurate with the number of amino acids present in the structure. The values used for the parameter *SSCF* in different protein structure are listed in Table 1. We have tried with 5 different values for SSCF in each protein instance and the value that gives the best fitness are reported in Table 1. A linear relationship between the value of SSCF and the number of atoms V in a protein structure has been observed (Figure 3). By using linear regression, in Figure 3, we inferred the approximate relationship as the following equation:

$$SSCF = 0.273V - 1.746$$

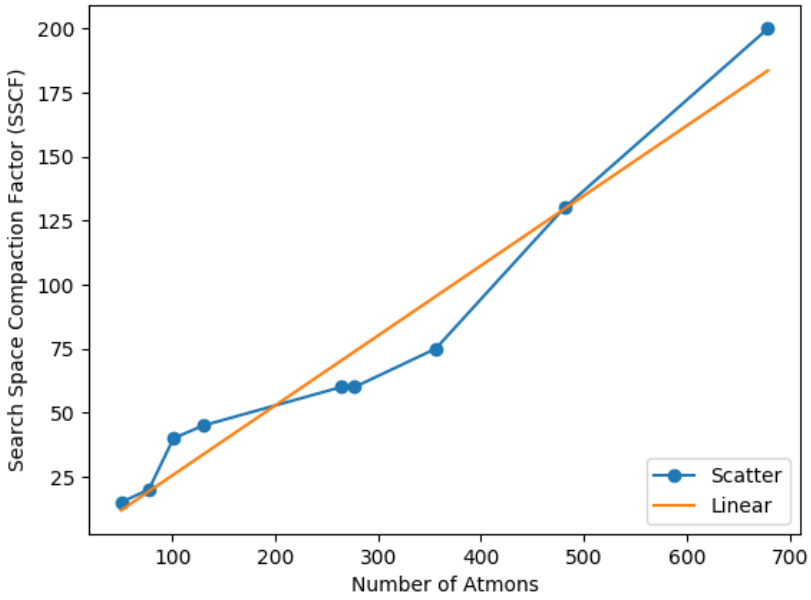


Figure 3: SSCF Values presented against the size of the Protein Instances. From the plot, a linear relationship between the value of SSCF and the number of atoms in a protein structure has been observed.

4.2. Encoding

For the MDGP in protein structure prediction, a prospective solution will contain coordinates of V number of the 3D points—presumably the 3D coordinates of atoms in the protein structure—in Cartesian 3D search space. In our algorithm, we have encoded each individual of the population with $3V$ number of real valued genes. Thus, an individual X comprises $3V$ number of genes in its genotype and hence, can be represented as an ordered list of length $3V$:

$$X = \left\{ \underbrace{x_1, y_1, z_1}_{1^{st}\,Atom}, \underbrace{x_2, y_2, z_2}_{2^{nd}\,Atom}, \ldots, \underbrace{x_i, y_i, z_i}_{i^{th}\,Atom}, \ldots, \underbrace{x_v, y_v, z_v}_{V^{th}\,Atom} \right\}$$

So, each triplet comprising three consecutive genes in an individual's genotype represents 3D coordinates of an amino acid atom in the protein structure. Each of the gene is set to a value drawn randomly from the range $[0,(3.8 \times V)/\text{SSCF}]$, as $(3.8 \times V)/\text{SSCF}$ is the upper bound of the search space in each direction of cartesian axis. Note that in *GMT3R* the compaction factor,

i.e., SSCF is not used and hence, the range becomes [0,$(3.8 \times V)$].

4.3. Fitness Evaluation

We have computed the euclidean distance between each pair of atoms present in the individual's phenotype where each triplet in the chromosome represents the coordinate of an amino acid in 3D space. We have approximately quantified the upper bound, u_{ij} and lower bound, l_{ij} of the distances of each pair of amino acids (i, j). The fitness of an individual (i.e., X) is defined by Equation 1 below:

$$Fitness(X) = \left(\frac{1}{|E|} \sum_{(i,j) \in E} e_{ij}^2 \right)^{1/2} \tag{1}$$

Here, $e_{ij} = max\{l_{ij} - \|c_i - c_j\|, \|c_i - c_j\| - u_{ij}, 0\}$ is the error associated with the constraints $l_{ij} \leq \|c_i - c_j\| \leq u_{ij}$ and $|E|$ denotes the number of distance pairs given. In literature [3], this is defined as the *Largest Distance Error (LDE)*.

4.4. Genetic Operators

Genetic operators aid the evolutionary process to optimize the fitness (defined by the objective function) of the individuals and evolve towards a pool of individuals with better fitness values. Genetic diversity within the population is ensured by the *Mutation* operator whereas intensification among the better candidates is assured by the *Crossover* operator. We have used three genetic operators in our proposed algorithms: *Random Mutation*, *Greedy Mutation*, and *Greedy Crossover*.

4.4.1. Random Mutation

Mutation operator helps to attain diversified individuals in the population and it is critical for the evolutionary process to guide through looking for different alternatives in the search space. It also assists overcome the local optima by precluding the individuals from becoming identical to each other. We adopted uniform mutation in our method. We have altered the value of a gene and set it to a new value from the pre-specified range of the search space ($[0, (3.8 \times V)/SSCF]$). We have applied random mutation operator on an individual according to the parameter, mutation rate, which set to a very small value of 0.015 to avoid primitive random search. A sketch of the pseudo-code for *Random Mutation* is given in Algorithm 2.

Algorithm 2: Random Mutation (Individual X)

```
mutationProbability = 0.015
for each gene X(i) in the genotype of X do
    if rand(0,1) ≤ mutationProbability then
        X(i) = U(0,1) * ((3.8 × V)/SSCF)
        //In GMT3R, SSCF was always set to 1
    end
end
return X
```

4.4.2. Greedy Mutation

We have infused ideas from stochastic local search paradigm in our algorithms by incorporating a mutation operator called *Greedy Mutation*. In *Greedy Mutation*, we greedily set the allele of a particular gene in an individual to a new value. We try and plug in r different values from the range $[0, (3.8 * V)/SSCF]$ ($[0, (3.8 * V)]$, for *GMT3R*) as the new value of that particular gene and choose the value v that optimizes the fitness of the individual most. If by setting v as the new value of that particular gene makes the individual more fitter than it was previously, we finally take and plug it into the individual's genotype. Otherwise, the previous value of the gene is retained with some probability p. We have set the value of p to 0.9. In *Greedy Mutation*, we need to plug-in r different values and calculate the fitness of the individual in each occasion and thus, it demands substantial amount of computational time. Therefore, we have fine tuned the parameter r and finally taken 50 as its value despite higher values of r tend to better optimize the fitness function and provide better results with some penalty in the execution time. We also made selection between the *Random Mutation* and *Greedy Mutation* with a probability, $intensificationProbability$ (= 0.8). The pseudo-code for *Greedy Mutation* is outlined in Algorithm 3.

4.4.3. Greedy Crossover

Crossover operator intensifies the search into a region of the search space containing individuals with better fitness values. It predominantly exploits genetic information of the individuals with better fitness values found thus far along the evolutionary process to produce new offspring. Thus, by combining individuals with better fitness values, crossover is more likely to produce offspring that are better than the parents. We have adopted a *tournament selection* scheme with tournament size being equal to 5 to elect two individuals from the population that will take part in the recombination/crossover process and produce

Algorithm 3: Greedy Mutation (Individual X)

```
1  //r = 50, number of candidate values tried out for a chromosome in an
   individual
2  i ←randomly selected gene index from the chromosome X
3  S ← {}
4  populate S with r random values for the gene i
5  v ← arg min_{v∈S} fitness(X, v)
6  if fitness(X, v) improves the fitness of individual X then
7  |  X(i) ← v
8  else
9  |  with probability p = 0.9, keep the original value
10 end
11 return X
```

offspring that eventually, are going to be included in the next generation of evolution. Chromosomes in the genotype of participating parents are recombined by one-point crossover in which a crossover point is randomly selected and then segments of the chromosomes across the crossover point are interchanged to produce new offspring. We try different indices chosen randomly (d) as the crossover points to generate offspring. These indices are carefully chosen so as to have a value that is a multiple of 3 to ensure that the triplet boundaries are not violated. The crossover point that generates the fittest offspring, is eventually chosen as the final crossover point and the fitter of the two newly evolved offspring is entered into the next generation. The pseudo-code is presented in Algorithm 4. Note that the greedy crossover is employed only in *GMT3R*$^+$.

Algorithm 4: GreedyCrossOver (Individual X_1, Individual X_2)

```
1  X_new = φ
2  C = {}
3  populate C with random d candidate points selected for crossover
4  i = selectBestCrossoverPoint(C)
5  ⟨X_new1, X_new2⟩ = recombineAt(X_1, X_2, i)
6  if fitness(X_new1) ≥ fitness(X_new2) then
7  |  X_new = X_new1
8  else
9  |  X_new = X_new2
10 end
11 return X_new
```

4.5. Twin Removal

In our proposed algorithms, we have also applied a twin removal procedure periodically to outspread and disperse the search within the search space to

ensure diversification among the individuals. Individuals with identical genetic information are identified as twins and surely they do not provide any useful avenue to look for in the search space. The similarity measure, based on which two individuals X_1 and X_2 are tagged as twins, is as follows:

$$Similarity(X_1, X_2) = e^{-\frac{\|X_1 - X_2\|^2}{2\sigma^2}} \tag{2}$$

Here, the parameter σ is set to the value of 75. The $Similarity(X_1, X_2)$ function will always give us a value in the interval [0,1]. The value of the similarity function closer to 1 indicates that the X_1 and X_2 are genetically more similar. We define the acceptable threshold value of similarity to 0.8–if the $Similarity(X_1, X_2)$ is greater than 0.8, we declare X_1 and X_2 as the twins and randomly reinitialize one of them. We have activated this *Twin Removal* after every hundred evolutionary generations.

Algorithm 5: Twin Removal

```
similarityThreshold = 0.8
for each pair of individuals (X_i, X_j) in the population do
    if Similarity(X_i, X_j) ≥ similarityThreshold then
        declare X_i and X_j as Twins
        reinitialize X_j
    end
end
```

4.6. Random Restart

If the search fails to improve the so far global best solution in terms of fitness over a predefined amount of iteration, we activate the *Random Restart* procedure. In this procedure, we rank each individual in the population according to its fitness and take one-third of the individuals with higher fitness values and then remove and re-initialize the rest of the individuals. After every 100 generations, we inspect the improvement of fitness of the global best individual over the immediate previous passage of 100 generations. If that progress is less than a threshold (0.001), we trigger this *Random Restart* procedure. *Random Restart* will aid the search process to recover and come out of stagnation if there is any.

Algorithm	Greedy Mutation	Random Restart	Twin Removal	Greedy Crossover	Compaction Factor
Basic GA	x	x	x	x	x
GMT3R	✓	✓	✓	x	x
GMT3R$^+$	✓	✓	✓	✓	✓

Table 2: Different Variants of the algorithms compared.

5. Results and discussion

5.1. Implementation and Experiment

We have used JDK 1.6 to implement GMT3R and GMT3R$^+$ in Java programming language. We run all of our experiments on an Intel 3.3GHz core $i3$ machine with 2GB of RAM running on Windows 7 Operating System. We first report the comparison of results among GMT3R$^+$, GMT3R and a basic implementation of the genetic algorithm referred to as BasicGA henceforth. BasicGA lacks the features incorporated in GMT3R$^+$ and GMT3R, e.g., greedy mutation, twin removal, random restart, compaction factor etc. BasicGA differs with GMT3R$^+$ (see Algorithm 1) in Lines 26-35 since it does not contain twin removal or random restart features and in Line 3, as value of $greedyMutationRate$ is set to 0.0 for Basic GA. Also, in Line 9, traditional one-point crossover has been used in BasicGA whereas in GMT3R$^+$, greedy crossover operator has been used. Notably, in comparison to GMT3R, GMT3R$^+$ differs in Line 9: in case of GMT3R, traditional one-point crossover operator has been used whereas greedy crossover has been incorporated for GMT3R$^+$. Also, recall that, in GMT3R$^+$, the Search Space Compaction Factor has been applied both for initializing the population in Line 4 and while applying mutation in Lines 14 and 15. All other parameters have been kept the same for a fair comparison. A comparative summary of the different components used for these algorithms is presented in Table 2.

5.2. Data Set

We have considered two sets of protein instances for our experiments: *(i)* protein structures with backbone only atoms and *(ii)* protein structures with all atoms. These large-sized protein benchmarks were introduced in [28]. We have calculated the pair-wise distances among the atoms after extracting the structures from the PDB [29]. The equations 3 and 4 are applied to calculate lower and upper bounds of the distances.

$$l_{ij} = (1 - \epsilon)\hat{d}_{ij} \tag{3}$$

$$u_{ij} = (1 + \epsilon)\hat{d}_{ij} \tag{4}$$

Here, $\hat{d}_{ij}$ is the real distance between point c_i and point c_j in the known structure of the protein sequence and value of ϵ is set to 0.08 as was used in literature [2]. Each point, c_i represents coordinates of an atom taken from the PDB file. In case of backbone only atoms, we considered $\alpha-$carbon atoms only. Since NMR is not capable of producing all the distance data, to make the data sparse and more realistic we have proceeded as follows. Firstly, we have considered only the distances that are $\leq 6\mathring{A}$. Subsequently, to make data sparse, we have incorporated 70% of the distances that are $\leq 6\mathring{A}$ in our experimental dataset by random selection. The effect of creating the dataset with other values, are analysed in Results section (see Section 5.4.4). Note that we are using upper and lower limits for the error in the distance pair as suggested in [12, 11] and proposed in the original model [2].

5.3. Results

The LDE values for BasicGA, GMT3R and GMT3R$^+$ for backbone-only and full-atomic instances are reported in Table 3 and Table 4, respectively. Each benchmark protein is associated with its corresponding PDB ID and the number of atoms considered in the structure. All the experiments were run 10 times for 2000 generations and the best and mean fitness values for each of the algorithms have been presented in both the tables. The best values for each protein instances are highlighted in bold-faced font. Best fitness values attained by each of the algorithms are reported and depicted in Figure 4 for backbone only instances. As the fitness values are in minute scale, we have plotted the logarithm of fitness value on the vertical axis. The best values achieved by different algorithms in the literature are reported in Table 5. For all the 4 proteins for which we performed the experiments, GMT3R$^+$ produced significantly better results compared to the other methods. We also tried to compare our results with that of buildup [4] and dgsol [30]. However, dgsol software fails to produce any satisfactory structures when run with the backbone only instances as input.

5.4. Discussions

From Tables 3- 4 and also from the Figure 4, we can clearly see that GMT3R$^+$ significantly outperforms GMT3R and BasicGA in all instances of protein structures. GMT3R$^+$ has been able to achieve smaller values for both

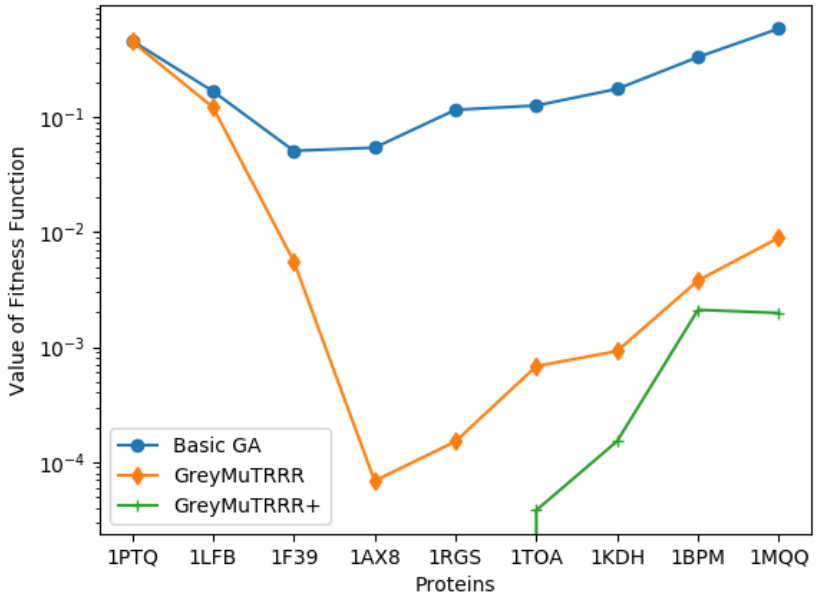


Figure 4: Best fitness values achieved by BasicGA, GMT3R and GMT3R$^+$. The fitness values are in minute scale, therefore, the logarithmic values of the fitness are plotted on the vertical axis.

Table 3: Best and mean Fitness of 10 runs of 2000 generations, each with a population size of 50 (with backbone atoms only)

Protein Id	V	BasicGA		GMT3R		GMT3R$^+$	
		Best	Mean	Best	Mean	Best	Mean
1PTQ	50	0.46111	0.47517	0.45435	0.46596	*0*	**0**
1LFB	77	0.16798	0.17681	0.12199	0.12834	*0*	**1.41E-04**
1F39	101	0.05095	0.05808	0.00558	0.01199	*0*	**4.38E-07**
1AX8	130	0.05426	0.05826	6.86E-05	0.00291	*0*	**7.22E-05**
1RGS	264	0.11582	0.12208	1.51E-04	9.42E-04	*0*	**0.00115**
1TOA	277	0.12616	0.12832	6.82E-04	0.00176	*3.81E-05*	**3.15E-04**
1KDH	356	0.17595	0.18304	9.23E-04	0.00259	*1.52E-04*	**9.29E-04**
1BPM	481	0.33291	0.34453	0.00379	0.33818	*0.00211*	**0.00364**
1MQQ	679	0.57934	0.58248	0.00830	0.00899	*0.00198*	**0.00321**

cases of best and mean fitness for all the protein instances. Note also that GMT3R also performs far better than the BasicGA. We also have run Student *t-test* with confidence level, $\alpha = 0.05$ to verify the statistical significance of the difference of results between the two competing algorithms GMT3R$^+$ and GMT3R. The rest of the section describes the effects of different components

Table 4: Best and mean Fitness of 10 runs of 2000 generations, each with a population size of 50 (including all atoms)

Protein Id	V	BasicGA		GMT3R		GMT3R$^+$	
		Best	Mean	Best	Mean	Best	Mean
1PTQ	402	0.70129	0.70363	0.17224	0.17941	*0.14879*	**0.15773**
1LFB	641	1.43464	1.45502	0.28914	0.28914	*0.13819*	**0.14384**
1F39	767	1.78401	1.79736	0.16805	0.14564	*0.14559*	**0.14825**
1AX8	1003	2.46104	2.48064	0.43211	0.43099	*0.15749*	**0.15750**

Table 5: Comparison of of results with state-of-the-art algorithms (including all atoms)

Protein Id	V	**buildup [4]**	**dgsol [30]**	GMT3R	GMT3R$^+$
1PTQ	402	1.80	0.541	0.17224	**0.14879**
1LFB	641	1.84	0.391	0.28914	**0.13819**
1AX8	1003	1.83	0.433	0.43099	**0.15749**
1F39	1534	1.89	0.474	0.16805	**0.14559**

featured in GMT3R and GMT3R$^+$.

5.4.1. Effects of Greedy Mutation

In our proposed algorithm, the role of the greedy mutation in optimizing the fitness function has been instrumental. A significant improvement over fitness value has been achieved by applying this mutation strategy which resembles popular strands from stochastic local search paradigm. Figures 5-6 depict the effect of Greedy Mutation for the protein structures 1AX8 and 1RGS. In each of these figures, we have plotted fitness values, with and without Greedy Mutation, against the number of generations to get the *fitness curves*. From Figures 5-6, we can clearly observe that significant improvement in fitness values over the course of evolution has been attained by applying the Greedy Mutation.

We have also run experiments to investigate and find a suitable value for the parameter r, which denotes the number of trials made in deciding the new value of a gene selected for mutation. Greater values of r tend to produce better fitness values but at the expense of a higher computational cost. Higher value of r ensures extensive local search in the neighborhood of an individual and hence better optimizes the value of the particular gene considered. As the value of r grows higher, we need to try out higher number of alternatives for the gene and plug in each of those values in the genotype and recompute the phenotype (euclidean distance values) which requires substantial amount of execution time. We have run experiments for the protein structure 1TOA using 20, 30, 50, 70 and 80 as the candidate values for r and the execution time and best fitness

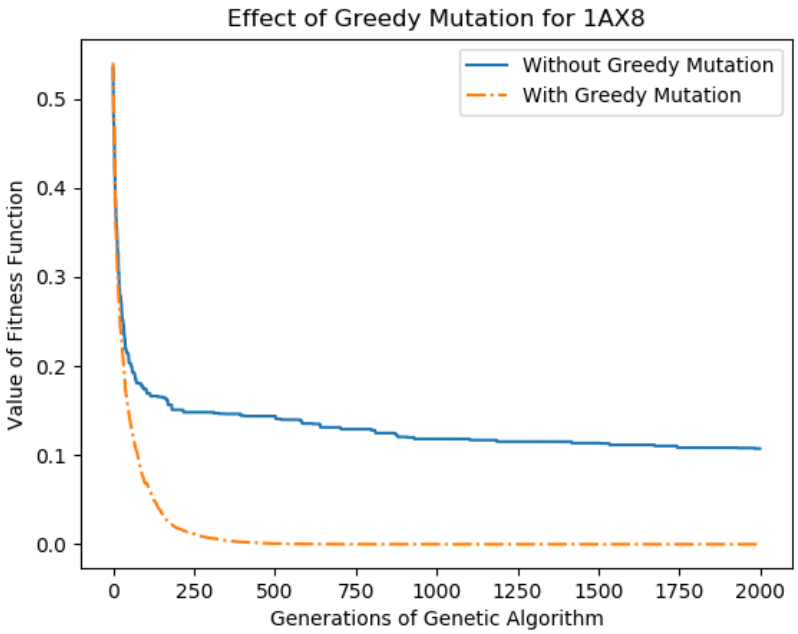


Figure 5: Fitness value against the number of generations for 1AX8. Fitness values are plotted with and without Greedy Mutation against the number of generations to get the *fitness curve*. It shows the effect of greedy mutation on achieving fitness function.

values are reported in the Table 6. From Table 6 we can see that the execution time per generation increases as the value for r grows higher and also better fitness values are achieved with higher values of r. To meet this trade off between better quality of fitness value and realistic execution time, we have chosen 50 as the ultimate value of r to be used for the entire evolution process.

Table 6: Execution time in seconds per generation and best fitness value attained after 2000 generations for the protein structure 1TOA

r	Execution time per generation (s)	Best fitness
20	2.18	5.71E-4
30	2.73	1.78E-4
50	3.06	5.34E-5
70	3.18	6.99E-5
80	3.23	1.15E-5

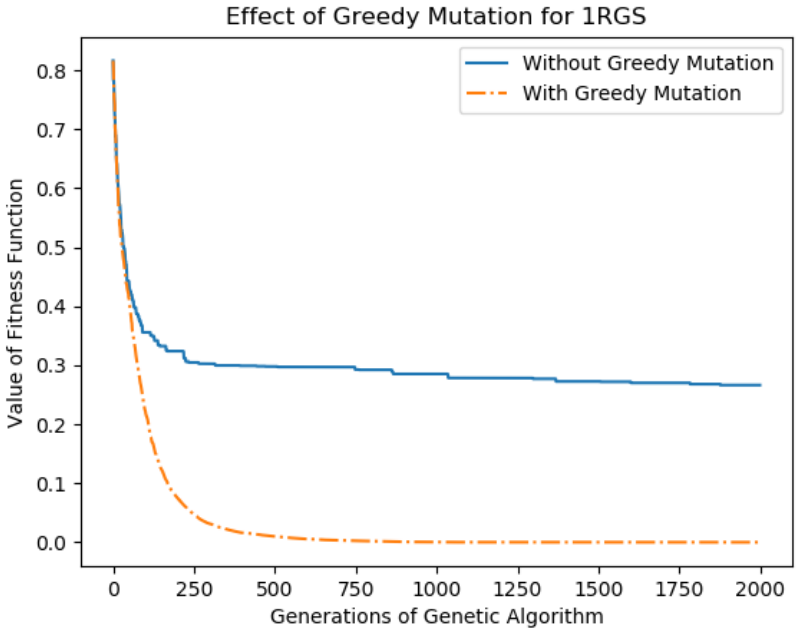


Figure 6: Fitness value against the number of generations for 1RGS. Fitness values are plotted with and without Greedy Mutation against the number of generations to get the *fitness curve*. It shows the effect of greedy mutation on achieving fitness function.

5.4.2. Effects of Search Space Compaction Factor

The *SSCF* (Search Space Compaction Factor) parameter contributes significantly in optimizing the fitness function for $GMT3R^{+}$. The effect of *SSCF* is clearly visible in Figure 7 where we have plotted two fitness curves, one with optimized value of SSCF for the protein instance 1AX8 and other without any search space compaction (i.e., setting *SSCF* value to 1). Figure 7 clearly shows that the fitness curve with compacted search space starts the evolution with much better fitness values with noteworthy differences compared to the fitness curve without any compaction applied to the search space. Since the protein structures very rarely form a linear shape of connected amino acids, the search space in every direction results in much faster convergence which is clearly evident in Figure 7.

Attainment of better fitness value by applying Greedy Mutation can be attributed to the proper choice of the value of parameter *SSCF*. We have experimentally tuned the parameter *SSCF* for each instance by plugging in different

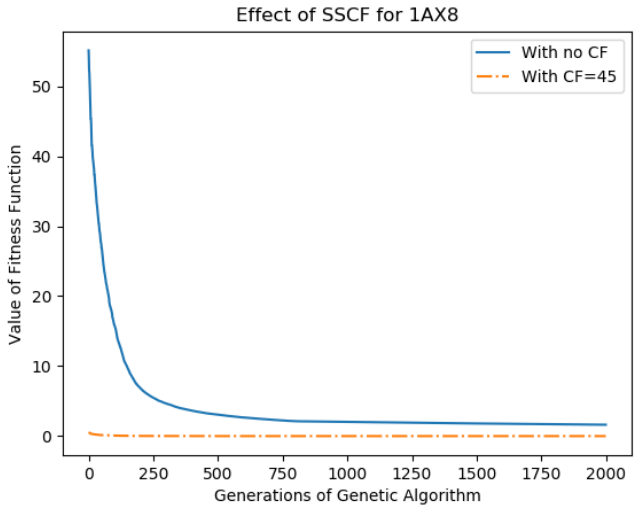


Figure 7: Effect of Search Space Compaction Factor. The plot clearly shows that the fitness curve with compacted search space starts with much better fitness values compared to the fitness curve without any compaction applied to the search space.

values for *SSCF*. We then plotted the corresponding fitness values against different values for *SSCF* and chose the best one. Figure 8 shows the different values tried for the parameter *SSCF* and the corresponding fitness for the protein structure 1KDH.

5.4.3. Effects of Twin Removal

We have also applied the twin removal procedure periodically after every 100 generations to ensure diversification among the individuals in the population. *Twin Removal* greatly contributes to the search process and the effect is clearly evident from Figure 9 where two fitness curves have been plotted for the protein structure of 1AX8, one with the twin removal procedure and the other without it.

5.4.4. Effects of Greedy Crossover

In GMT3R$^+$, we have adopted a greedy crossover strategy which greedily chooses the crossover point between two participating parents to produce offspring and include the offspring with better fitness in the next generation. In greedy crossover strategy, we tried d different indexes as crossover point, each of which is set to a value that is a multiple of 3. Figure 10 depicts the effect of

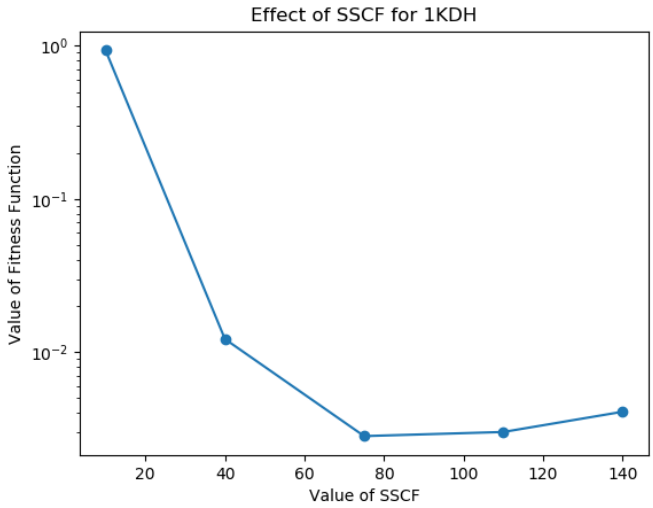


Figure 8: Fitness values against different SSCF values in logarithmic scale for 1KDH. Experiments are carried out with different SSCF values to find a proper value.

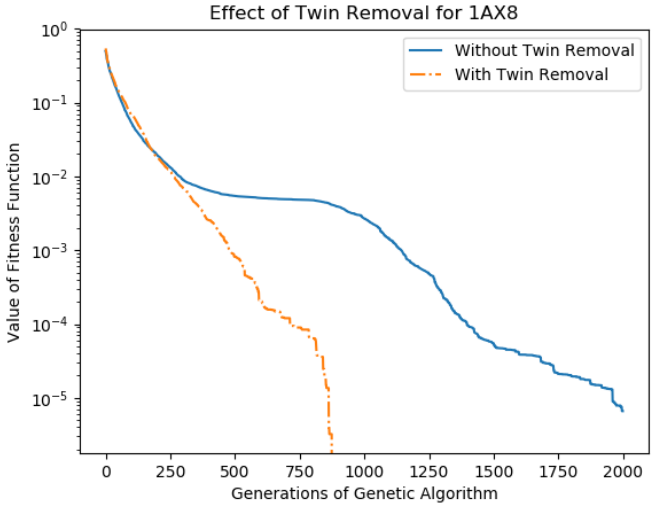


Figure 9: After applying Twin Removal the fitness values against the number of generations for 1AX8 in logarithmic scale. A Twin Removal procedure was applied after every 100 generations to diversify the population.

the greedy crossover for the protein structure 1AX8 where two fitness curves are plotted: one with greedy crossover and the other without it. Note that in the

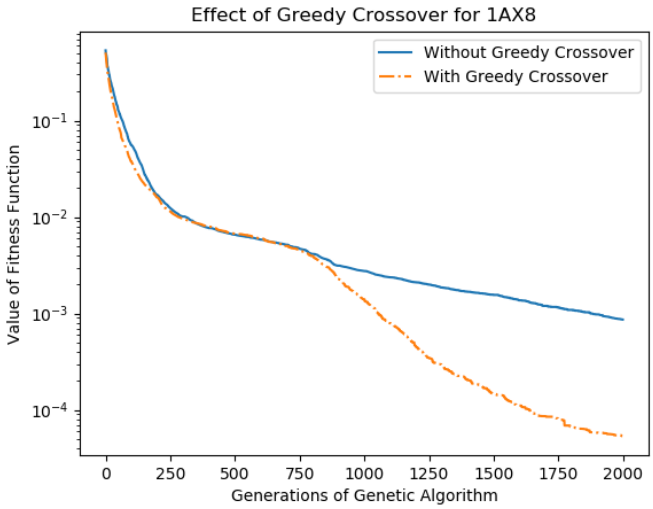
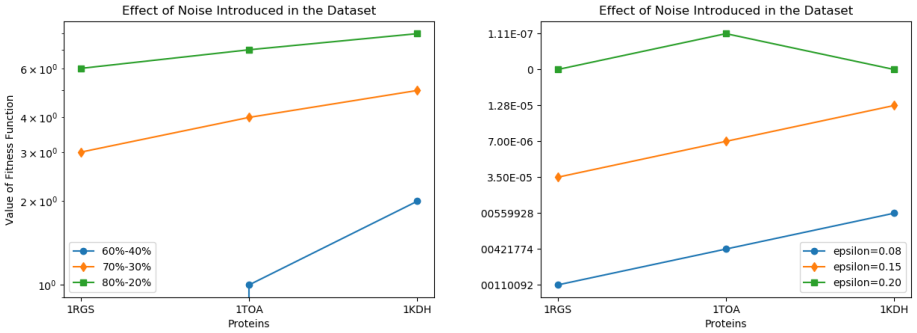


Figure 10: After applying Greedy Crossover the fitness values against number of generations for 1AX8 in logarithmic scale. Greedy Crossover improved the results only after a certain number of generations.

latter, instead of the greedy crossover, the one-point crossover (which can be attained eventually by setting d to 1) has been applied. The fitness curve with greedy crossover clearly achieves better results as can be seen in the figure. At this point a brief discussion is in order. As can be seen from the figure, greedy crossover could improve the results only after a certain number of generations have passed (in case of 1AX8 it is about 1000 generations). This phenomenon can be understood from the operation and effectiveness of the crossover operator itself. At initial stages, the individuals are not of enough good quality and the parts that are put together in by the crossover operations do not usually result in a better individual. As the search makes progress, sub-optimal solutions form and they together have an effect on quality of the solutions produced in crossover which random crossover fails to exploit.

5.4.5. Effect of Noisy Dataset

To see the effect of the noise introduced in the dataset, we have experimented using different values of the noise. In our original experiment, we have retained only 70% of the distance pairs, however, we tested GMT3R$^+$ with varying values percent of retention ranging from 60% to 80% . The results are shown in



(a) (b)

Figure 11: (a) Effect of noise or percentage of retaining distance pairs from the original data set on proteins. It is observed that lowering down the percent of retaining distance pairs in the dataset from the original dataset makes the problem relatively easier to solve. (b) Effect of noise in the value of ϵ on distance pairs from the original data set. The average fitness values are plotted and noted that the value of ϵ (0.08) we applied in our experiment is achieving the best results in terms of fitness.

Figure 11 (a). As expected, lowering down the number of distance pairs in the dataset from the original dataset makes the problem relatively easier to solve and hence the best performing dataset is created with 40% error, i.e., retaining 60% of original distance pairs. We also demonstrate the effect of varying the value of ϵ on the dataset for different proteins. The average fitness function values are reported in Figure 11 (b). As we may note that, the current value of ϵ is achieving the best results in terms of fitness function value.

6. Conclusion

In this work, we have proposed enhanced and scalable genetic algorithms to solve the molecular distance geometry problem for protein structure determination using sparse and inaccurate NMR data. We have applied *(i)* a greedy mutation operator to intensify the search, *(ii)* a twin removal technique for diversification in the population and, *(iii)* a random restart method to recover from the stagnation. We have also infused the search space compaction factor as well as a greedy crossover operator in our algorithms. We have shown that our algorithm significantly outperforms standard genetic algorithms and state-of-the-art algorithms on a standard set of benchmark proteins. Our method is capable of producing structures that are very close to the native ones (see Figure 12).

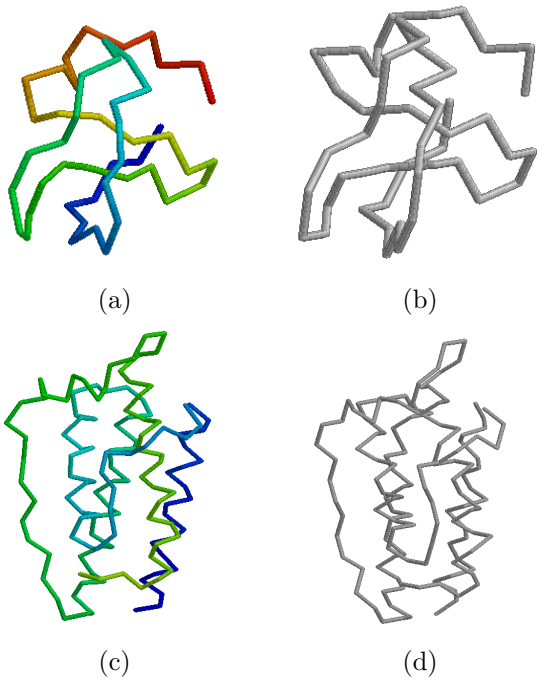


Figure 12: Backbone native structure and the model found by GMT3R$^+$ for the protein 1PTQ (a and b) and protein 1AX8 (protein c and d).

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this article.

References

[1] J. M. Berg, J. L. Tymoczko, L. Stryer, Biochemistry: international edition, WH Freeman & Company Limited, 2006.

[2] M. Souza, C. Lavor, A. Muritiba, N. Maculan, Solving the molecular distance geometry problem with inaccurate distance data, BMC bioinformatics 14 (Suppl 9) (2013) S7.

[3] L. Liberti, C. Lavor, N. Maculan, A. Mucherino, Euclidean distance geometry and applications, SIAM Review 56 (1) (2014) 3–69.

[4] D. Wu, Z. Wu, An updated geometric build-up algorithm for solving the molecular distance geometry problems with sparse distance data, Journal of Global Optimization 37 (4) (2007) 661–673.

[5] A. Mucherino, L. Liberti, C. Lavor, N. Maculan, Comparisons between an exact and a metaheuristic algorithm for the molecular distance geometry problem, in: Proceedings of the 11th Annual conference on Genetic and evolutionary computation, ACM, 2009, pp. 333–340.

[6] L. Liberti, C. Lavor, A. Mucherino, N. Maculan, Molecular distance geometry methods: from continuous to discrete, International Transactions in Operational Research 18 (1) (2011) 33–51.

[7] J. J. More, Z. Wu, Distance geometry optimization for protein structures, Journal of Global Optimization 15 (3) (1999) 219–234.

[8] L. T. Hoai An, Solving large scale molecular distance geometry problems by a smoothing technique via the gaussian transform and d.c. programming, Journal of Global Optimization 27 (4) (2003) 375–397. doi:10.1023/A:1026016804633. URL https://doi.org/10.1023/A:1026016804633

[9] C. Lavor, L. Liberti, N. Maculan, Computational experience with the molecular distance geometry problem, in: Global Optimization, Springer, 2006, pp. 213–225.

[10] M. L. Islam, S. Shatabda, M. S. Rahman, Gremutrrr: A novel genetic algorithm to solve distance geometry problem for protein structures, CoRR abs/1411.4246. URL http://arxiv.org/abs/1411.4246

[11] P. J. Nichols, A. Born, M. A. Henen, D. Strotz, J. Orts, S. Olsson, P. Güntert, C. N. Chi, B. Vögeli, The exact nuclear overhauser enhancement: recent advances, Molecules 22 (7) (2017) 1176.

[12] B. Vögeli, The nuclear overhauser effect from a quantitative perspective, Progress in nuclear magnetic resonance spectroscopy 78 (2014) 1–46.

[13] C. Savarese, J. M. Rabaey, J. Beutel, Location in distributed ad-hoc wireless sensor networks, in: Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP'01). 2001 IEEE International Conference on, Vol. 4, IEEE, 2001, pp. 2037–2040.

[14] D. Tolani, A. Goswami, N. I. Badler, Real-time inverse kinematics techniques for anthropomorphic limbs, Graphical models 62 (5) (2000) 353–388.

[15] J. B. Tenenbaum, V. De Silva, J. C. Langford, A global geometric framework for nonlinear dimensionality reduction, Science 290 (5500) (2000) 2319–2323.

[16] C. Lavor, L. Liberti, N. Maculan, A. Mucherino, Recent advances on the discretizable molecular distance geometry problem, European Journal of Operational Research 219 (3) (2012) 698–706.

[17] G. M. Crippen, T. F. Havel, Distance geometry and molecular conformation, Vol. 74, Research Studies Press Taunton, UK, 1988.

[18] J. J. Moré, Z. Wu, Global continuation for distance geometry problems, SIAM Journal on Optimization 7 (3) (1997) 814–836.

[19] L. Liberti, S. Kucherenko, Comparison of deterministic and stochastic approaches to global optimization, International Transactions in Operational Research 12 (3) (2005) 263–285.

[20] A. Mucherino, L. Liberti, C. Lavor, Md-jeep: an implementation of a branch and prune algorithm for distance geometry problems, in: Mathematical Software–ICMS 2010, Springer, 2010, pp. 186–197.

[21] L. Liberti, M. Drazic, Variable neighbourhood search for the global optimization of constrained nlps, in: Proceedings of GO Workshop, Almeria, Spain, Vol. 2005, 2005.

[22] L. Liberti, C. Lavor, N. Maculan, F. Marinelli, Double variable neighbourhood search with smoothing for the molecular distance geometry problem, Journal of Global Optimization 43 (2-3) (2009) 207–218.

[23] Q. Dong, Z. Wu, A geometric build-up algorithm for solving the molecular distance geometry problem with sparse distance data, Journal of Global Optimization 26 (3) (2003) 321–333.

[24] B. Hendrickson, The molecule problem: Exploiting structure in global optimization, SIAM Journal on Optimization 5 (4) (1995) 835–857.

[25] C. Lavor, A. Mucherino, L. Liberti, N. Maculan, Discrete approaches for solving molecular distance geometry problems using nmr data, International Journal of Computational Biosciences 1 (1) (2010) 88–94.

[26] Z. Voller, Z. Wu, Distance geometry methods for protein structure determination, in: Distance Geometry, Springer, 2013, pp. 139–159.

[27] C. Lavor, L. Liberti, A. Mucherino, The interval branch-and-prune algorithm for the discretizable molecular distance geometry problem with inexact distances, Journal of Global Optimization 56 (3) (2013) 855–871.

[28] P. Biswas, K.-C. Toh, Y. Ye, A distributed sdp approach for large-scale noisy anchor-free graph realization with applications to molecular conformation, SIAM Journal on Scientific Computing 30 (3) (2008) 1251–1277.

[29] H. M. Berman, J. D. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, P. E. Bourne, The protein data bank, Nucleic Acids Research 28 (1) (2000) 235–242.

[30] Dgsol: Distance geometry optimization software, `http://www.mcs.anl.gov/~more/dgsol/`, accessed: 2015-11-19.