Comparisons between an Exact and a Meta-Heuristic Algorithm for the Molecular Distance Geometry Problem

Antonio Mucherino
Leo Liberti
LIX, École Polytechnique,
Palaiseau, France
{mucherino,liberti}@
lix.polytechnique.fr

Carlile Lavor
Dept. of Applied Mathematics,
State University of Campinas,
Campinas-SP, Brazil
clavor@ime.unicamp.br

Nelson Maculan
COPPE
Systems Engineering,
Federal University of
Rio de Janeiro,
Rio de Janeiro, Brazil
maculan@cos.ufrj.br

ABSTRACT

We consider the Discretizable Molecular Distance Geometry Problem (DMDGP), which consists in a subclass of instances of the distance geometry problem related to molecular conformations for which a combinatorial reformulation can be supplied. We investigate the performances of two different algorithms for solving the DMDGP. The first one is the Branch and Prune (BP) algorithm, an exact algorithm that is strongly based on the structure of the combinatorial problem. The second one is the Monkey Search (MS) algorithm, a meta-heuristic algorithm that is inspired by the behavior of a monkey climbing trees in search for food supplies, and that exploits ideas and strategies from other meta-heuristic searches, such Genetic Algorithms, Differential Evolution, and so on. The comparison between the two algorithms is performed on a set of instances related to protein conformations. The used instances simulate data obtained from the Nuclear Magnetic Resonance (NMR), because the typical distances provided by NMR are considered and a predetermined number of wrong distances are included.

Categories and Subject Descriptors

G.1.6 [**Optimization**]: Global optimization; G.2.1 [**Combinatorics**]: Combinatorial algorithms; J.3 [**Life and medical sciences**]: Biology and genetics; I.2.8 [**Problem Solving, Control Methods, and Search**]: Graph and tree search strategies, Heuristic methods

General Terms

algorithms, performance, experimentation

Keywords

distance geometry, protein molecules, combinatorial optimization, branch and prune, monkey search.

1. INTRODUCTION

The distance geometry problem is the problem of finding the coordinates of the points forming a three-dimensional conformation when some of the relative distances between such points are known. An interesting application of the problem is to the protein conformations, in which the points of the conformation represent the atoms of the protein molecule. In such a case, the considered problem is usually referred to as MOLECULAR DISTANCE GEOMETRY PROBLEM (MDGP).

Over the years, many methods have been proposed for solving the MDGP. Most of them are based on a continuous formulation of the problem. In its basic form, the MDGP can be seen as a constraint satisfaction problem, where the following constraints need to be verified:

$$||x_i - x_j|| = d_{ij} \quad \forall i, j.$$

In the formula, x_i and x_j are the coordinates in the space of the i^{th} and j^{th} atoms of the protein conformation, and d_{ij} is a known distance. The MDGP can be also seen as a global optimization problem, in which the function to be optimized is a penalty function that evaluates the differences between known distances d_{ij} and computed distances $||x_i - x_j||$. Different objective functions have been proposed, one of the most used is Largest Distance Error (LDE):

$$LDE(\{x_1, x_2, \dots, x_n\}) = \frac{1}{|m|} \sum_{\{i,j\}} \frac{||x_i - x_j|| - d_{ij}}{d_{ij}}, \quad (1)$$

where m is the total number of known distances. Note that, in order to use the LDE function, none of the distances d_{ij} can be exactly 0. Moreover, once a position is given to all the atoms of the protein conformation, if the value of the LDE function is 0, then the set of given distances is feasible and the final conformation satisfies all of them. For a survey on methods and algorithms for the MDGP refer to [7].

Recently, a new approach to the MDGP has been proposed, that is tailored to particular geometric properties of the protein conformations. Because of these properties, the MDGP can be reformulated as a combinatorial optimization problem in correspondence with a subclass of its instances. This subclass contains all the instances in which the distances $d_{i-3,i}$, $d_{i-2,i}$ and $d_{i-1,i}$ are known and in which there are no triplets of aligned atoms. If the backbones of the protein molecules are considered, then most of the corresponding instances of the MDGP can be reformulated as a combinatorial problem. In this case, a solution to the MDGP can be represented as a vector of binary variables. The combinatorial reformulation of the MDGP is referred to as Discretizable Molecular Distance Geometry Problem (DMDGP).

The DMDGP can be solved by applying methods for combinatorial optimization, where the objective function is, for example, the LDE function (1). However, if n is the number of considered atoms, there are 2^{n-3} possible solutions: the number of solutions is huge when n is large. This suggests the use of meta-heuristic searches for solving the optimization problem. A possible choice is the recently proposed Monkey Search (MS) [11], a meta-heuristic method inspired by the behavior of a monkey climbing trees in its search for food. MS has already been successfully employed for solving difficult global optimization problems, including combinatorial problems.

Another approach for solving the DMDGP is the one proposed in [9]. A Branch and Prune (BP) algorithm has been developed for the solution of the combinatorial reformulation of the MDGP. The algorithm mimicks the structure of the combinatorial problem closely. A tree of possible solutions is explored starting from its top where the first atom of the conformation is placed, and the search proceeds by placing the following atoms one per time. As soon as a branch of this tree is found to be infeasible, then it is pruned and the search is backtracked. Because of the pruning phase, the size of the tree is reduced quickly and therefore an exhaustive search on the remaining branches is not too computational demanding. Hence, BP falls into the category of the exact methods for solving the DMDGP.

The aim of this paper is to compare these two algorithms for the solution of the DMDGP. The BP algorithm implements an exact method, whereas the MS algorithm implements a meta-heuristic search. Both methods are valid alternatives for solving the problem. We will investigate the performances of the two algorithms and we will discuss the advantages and disadvantages in using the one or the other algorithm.

The instances used in this paper will be chosen in order to analyze how the two algorithms are able to manage experimental errors. In fact, experimental techniques, such as the Nuclear Magnetic Resonance (NMR), can provide a small subset of distances that are completely wrong. Therefore, the complete set of distances forming an instance is not feasible in general if real data are used. We will investigate how the two algorithms manage these data and if they are able to find solutions in which only the *good* distances are considered.

Our computational experiences will show that the two algorithms can be considered as complementary. While the performances of the BP algorithm are much better than the performances of MS when the instances contain few wrong distances, this situation is completely inverted when the number of wrong distances is large. The evaluation of the performances is made in terms of CPU time and quality of the of the found solutions.

The paper is organized as follows. In Section 2, we introduce the combinatorial reformulation of the distance geometry problem. In Section 3, we will briefly describe the BP algorithm and a possible extension of this algorithm for taking into account instances containing wrong distances. In Section 4, we explain the basic structure of the meta-heuristic MS algorithm. Computational experiences related to the two discussed algorithms are presented in Section 5. In particular, in Section 5.1 we describe the method we used for generating instances of the problem in which a certain percentage of distances is wrong. Finally, in Section 5.2 we present the details of our computational experiments, and we discuss the advantages and disadvantages in using the one or the other algorithm for solving the DMDGP. Conclusions are given in Section 6.

2. THE DISCRETIZABLE DISTANCE GEOMETRY PROBLEM

The distance geometry problem, and, in particular, the Molecular Distance Geometry Problem (MDGP), can be formulated as a combinatorial problem if some assumptions are satisfied. Given an instance,

- all the distances $d_{i-3,i}$, $d_{i-2.i}$ and $d_{i-1,i}$ must be known;
- each triplet of consecutive atoms $\{x_{i-2}, x_{i-1}, x_i\}$ cannot be perfectly aligned.

If both these two assumptions are satisfied, then it is possible to prove that the cosine of the torsion angle among four consecutive atoms $\{x_{i-3}, x_{i-2}, x_{i-1}, x_i\}$ of the protein can be computed. If the atoms x_{i-3}, x_{i-2}, x_{i-1} are already placed into a fixed location, then, by exploiting all the known distances and the value of the torsion angle, the exact position of the atom x_i can be obtained. Unfortunately, the value of the torsion angle is not available, but only its cosine, which brings to two possible values for the angle. Because of this uncertainty, each atom x_i can be placed in two different positions.

Once the first three atoms of a protein have been placed, the cosine of the torsion angle among the first four atoms can be computed and two possible positions for the fourth atom, x_4 and x'_4, can be identified. Then, in correspondence with the two choices for the fourth atom, two different pairs of possible atomic positions for x_5 can be computed. Hence, 4 possible positions in total are possible for the atom x_5. By iterating this procedure, the generic atom x_i can be assigned to 2^{i-3} positions. Therefore, for a protein shaped as a chain of n atoms, there are 2^{n-3} different three-dimensional conformations that can be solutions to the problem. We refer to the combinational reformulation of the problem as Discretizable Molecular Distance Geometry Problem (DMDGP). More details about the combinatorial reformulation can be found in [9, 10].

A solution to the combinatorial problem can be seen as a vector of n binary variables, where n is the number of atoms in the molecule. The i^{th} binary variable represents the choice between the two possible positions for the atom x_i. The quality of the solutions can be evaluated through, for example, the LDE function (1). The search domain can be seen as a set of binary vectors, and also as a binary tree. In the latter representation, all the possible atomic positions correspond to a node of the binary tree, and a path from the top of the tree (where it is placed the first atom) to its root

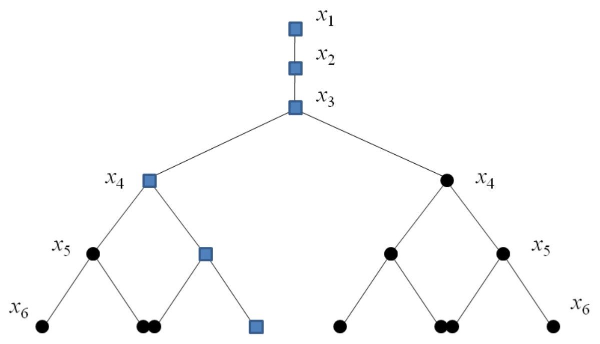

Figure 1: The graph G_{BP} in correspondence with an instance with 6 atoms.

represents a possible solution to the problem. Each vector of binary variables corresponds to one and only one complete path on the binary tree.

Instances obtained by experimental techniques and related to the backbones of the protein conformations belong to the subclass of instances of the DMDGP in most of the cases. Indeed, in this case, the two aforementioned assumptions are almost always satisfied. The short range distances $d_{i-3,i}$, $d_{i-2,i}$ and $d_{i-1,i}$ between the atoms of the protein backbones are usually detected by experimental techniques. Moreover, there is a very low probability that three consecutive atoms are perfectly aligned. Therefore, the problem of finding the coordinates of the atoms of a protein backbone is, in most of the cases, a combinatorial problem.

In the following two sections, we will discuss two possible approaches for solving the combinational problem. The first one is the Branch and Prune algorithm (Section 3), which is an exact method that exploits the binary tree structure of the combinatorial reformulation. The second one is the Monkey Search, a meta-heuristic algorithm inspired by a monkey in search for food resources, in which ideas from other meta-heuristic algorithms are also employed.

3. THE BRANCH AND PRUNE ALGORITHM

The Branch and Prune (BP) algorithm has been recently proposed in [9, 10]. It is a very efficient algorithm for solving the combinatorial reformulation of the distance geometry problem, and it is strongly based on the tree structure of the reformulated problem. As previously observed, a binary tree containing all the possible atomic positions can be defined and solutions to the problem can be found by exploring such a tree. Formally, the binary tree can be represented as an undirected graph $G_{BP} = (V_{BP}, E)$, where the nodes in V_{BP} represent possible atomic positions. Groups of nodes represent positions for the same atom x_i, and therefore the nodes in V_{BP} can be organized in layers (see Figure 1). There is always an edge between one node in a layer and two nodes of the following layer. Nodes are connected by an edges in E if the two atomic positions depend on each other. Figure 1 shows a path defined by the nodes marked by the boxes. For example, the chosen position for x_6 is connected to a position for x_5, because there are only two positions for x_6 for a given x_5. Only paths in which the atomic positions are connected to the previous and the following positions are allowed, and each complete path represents a solution.

Algorithm 1 BP algorithm.

```
0: BP($i$, $n$, $d$)
0: compute the first atomic position for the $i^{th}$ atom: $x_i$;
0: check the feasibility of the atomic position $x_i$:
  **if** ($| \, ||x_i - x_j|| - d_{ij} \, | < \varepsilon, \forall j < i$) **then**
    the atomic position $x_i$ is feasible;
    **if** ($i = n$) **then**
      a solution is found;
    **else**
      BP($i + 1$,$n$,$d$);
    **end if**
  **else**
    the atomic position $x_i$ is pruned;
  **end if**
  compute the second atomic position for the $i^{th}$ atom: $x'_i$;
  check the feasibility of the atomic position $x'_i$:
  **if** ($| \, ||x'_i - x_j|| - d_{ij} \, | < \varepsilon, \forall j < i$) **then**
    the atomic position $x'_i$ is feasible;
    **if** ($i = n$) **then**
      a solution is found;
    **else**
      BP($i + 1$,$n$,$d$);
    **end if**
  **else**
    the atomic position $x'_i$ is pruned;
  **end if**
```

This is how the BP algorithm works [9, 10]. The first three atoms are placed in the following three positions:

$$\begin{aligned} x_1 &= (0, 0, 0), \\ x_2 &= (-d_{12}, 0, 0), \\ x_3 &= (d_{23} \cos\theta_3 - d_{12}, d_{23} \sin\theta_3, 0), \end{aligned}$$

where the generic d_{ij} represents the known distance between the i^{th} and the j^{th} atom, and where θ_i denotes the generic bond angle (that can be computed from the known distances).

Starting from the atom x_4, two possible positions can be obtained for each atom. During the generic step of the algorithm, the possible positions x_i and x'_i for the i^{th} atoms are computed, by exploiting the information obtained from the distances $d_{i-3,i}$, $d_{i-2,i}$ and $d_{i-1,i}$. In this way, a part of the tree G_{BP} is generated. However, this part of the tree could contain only infeasible solutions. There can be indeed known distances from the atom x_i to any other atom of the conformation that may result to be violated. Therefore, the comparison between these known distances and the distances computed by using one of the possible choices for x_i can reveal that some of the parts of the binary tree are infeasible. In such a case, it is useless to continue generating some parts of the tree, and they can be pruned as soon as they are discovered to be infeasible.

Algorithm 1 provides a sketch of the BP algorithm. The algorithm is invoked iteratively, starting from the atomic position 4. The input parameters are i, the current atom whose position is searched, n, the total number of atoms, and d, the set of known distances. One of the solutions to the problem is found when BP(n,n,d) finds a feasible position at least for the last atom of the conformation. The condition $| \, ||x_i - x_j|| - d_{ij} \, | < \varepsilon$, for all $j < i$ and where $\varepsilon > 0$ is a given tolerance, represents a pruning test, which we employ for discovering infeasible atomic positions. Actually, more than one pruning test can be used, but we will consider only one of them in the experiments described in this paper. Other pruning tests and their effectiveness are discussed in [8].

If a given instance contains a set of distances that are feasible, then the BP algorithm is able to find all the solutions to the problem. Each solution is obtained when the last atom of the conformation is placed in its position: this means that all the previous atoms were placed in a possible position and that they were all feasible. Note that an optimization problem is solved by BP and no objective function is used. The symmetry results proved in [10] suggest that, if the DMDGP has solutions, then it has an even number of solutions. Then, an objective function (such as the LDE function (1)) could be employed for discriminating among the found solutions. However, due to the symmetry, pairs of solutions can have the same objective function value.

In the hypothesis that the set of given instances is infeasible, then the BP algorithm, as it is described above, can provide no solutions to the problem. This hypothesis is realistic, because data obtained from experimental techniques often contain a small percentage of wrong measurements. Since the BP algorithm prunes all the branches containing at least an infeasible position, the basic algorithm needs to be extended for dealing with wrong distances.

A possible extension is the following. At each step of the algorithm, the two possible positions x_i and x_i' are computed and the pruning test is applied. Instead of pruning the current branch as soon as it is discovered to be infeasible, a counter of violated distances can be set up and updated every time an atomic position is found to be infeasible. Then, when the number of violated distances gets greater than a certain predetermined threshold, the corresponding branch can be pruned.

This variant of the BP algorithm is able to deal with set of distances that are infeasible. If an estimate of the wrong distances is available, an accurate threshold V_D on the violated distances can be set, so that the BP algorithm is able to find solutions in which at most V_D distances are violated, corresponding to the wrong ones. The choice of the threshold play an important role. Indeed, too small thresholds could force the BP algorithm to prune too early, and no solutions are found. For example, if there are 10 wrong distances, and $V_D = 9$, the algorithm cannot find solutions. On the other hand, if the threshold is too large, then the infeasible branches are pruned too late. This brings to two consequences. First, the computational cost grows with T_D because the branches are pruned later and later as the threshold increases. Second, the number of found solutions grows, because more violated distances are allowed. If no estimates on the number of wrong distances are available, then more executions of the algorithm can help in locating the optimal T_D value.

4. THE MONKEY SEARCH ALGORITHM

The Monkey Search (MS) is a meta-heuristic method for global optimization [11]. It mimicks the behavior of a monkey climbing trees in its search for food resources. The trees the monkey climbs contain solutions to the optimization problem to be solved, and the quality of the solutions reflects the quality of the food discovered by the monkey. The branches connecting two solutions on the trees represent the perturbation that can be applied to one solution

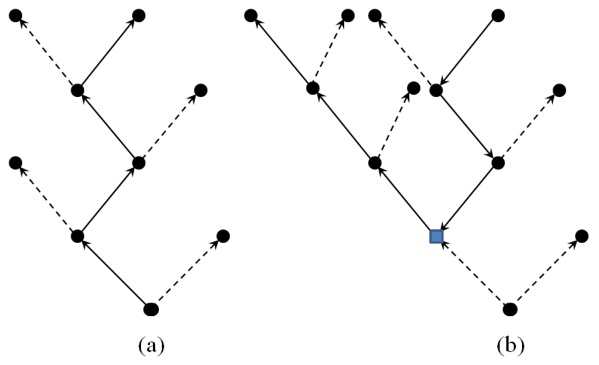

Figure 2: Two representations of the monkey behavior. The monkey climbs a tree for the first (a) and the second (b) time.

for obtaining another solution. The monkey climbs the trees and it is able to remember where it previously found good food. This allows it to come back on the same tree and explore close branches, with the hope of improving the quality of the discovered food.

Formally, each tree the monkey climbs can be represented as a weighted directed graph $G_{MS} = (V_{MS}, A, w)$. V_{MS} is the set of nodes of the tree, representing feasible solutions. There is an arc (u, v) between two nodes $u, v \in V_{MS}$ of the tree if v can be obtained by applying a specific perturbation to the solution u. The associated weight $w_{uv} \in \Re$ is the difference in objective function value between v and u. This information is used during the search, because, while climbing G_{MS}, the monkey will prefer paths that optimize the objective function value. Paths bringing from the root to the top of the tree can be located if particular sets of arcs are chosen to be climbed.

Algorithm 2 shows how the monkey climbs each tree G_{MS}. The inputs given to Algorithm 2 are the following ones: the graph G_{MS}, which is empty at this stage because it is built as the search proceeds; a feasible solution to the problem v_0, which is used as seed for the tree; the parameter maxpaths, which specifies how many paths on the tree the monkey is allowed to explore; and, finally, the parameter maxlevels, which specifies the predetermined maximum possible length of each tree path. At the end of the algorithm, y_{best} contains the best solution found during the exploration of the tree G_{MS}. It is worth noting that this procedure produces trees G_{MS} with probability 1 because of the likelihood to recover exactly the same solution.

Figure 2 gives graphic representations of the monkey behavior described in Algorithm 2. In Figure 2(a), the monkey climbs a new tree for the first time. At each step, two new solutions are generated and placed on two nodes of the tree. The dashed arcs are the ones the monkey rejects, and all the others form a path on the tree. When the top of the tree is reached (level = maxlevels), the monkey climbs the chosen arcs in the opposite direction and marks them (the weights w are modified with the best improvement available in the direction of the corresponding arc). At a certain point, the monkey restarts climbing up (see the node marked by a box in Figure 2(b)). Then, new solutions are generated and one of them is chosen, until the top of the tree is reached again.

Algorithm 2 MS algorithm.

```
0: MS($G_{MS} = (V_{MS}, A, w)$,$v_0$,maxpaths,maxlevels)
0: let $V = \{v_0\}$;
0: let $y = v_0$, $y_{best} = y$;
0: let level = 0, npaths = 0;
0: let termination = false;
  while (termination = false) do
    # climbing up new tree branches
    let npaths = npaths + 1;
    while (level <= maxlevels) do
      $x' =$ random_perturb($y$);
      $x'' =$ random_perturb($y$);
      if ($f(x') < f(y_{best})$) then
        let $y_{best} = x'$;
      end if
      if ($f(x'') < f(y_{best})$) then
        let $y_{best} = x''$;
      end if
      let $V = V \cup \{x', x''\}$;
      let $A = A \cup \{(y, x'), (y, x'')\}$;
      let $w_{yx'} = f(y) - f(x')$;
      let $w_{yx''} = f(y) - f(x'')$;
      choose_arc($w_{yx'}$,$w_{yx''}$);
      let $y =$ chosen($x'$,$x''$);
      let level = level + 1;
    end while
    if (npaths < maxpaths) then
      # climbing down
      let $w_{best} = w_{y_{\text{level}}, y_{\text{level}-1}}$;
      let $p = 0$;
      while (level $\geq 0$ and $p < 0.90$) do
        let level = level − 1;
        let $w_{current} = w_{y_{\text{level}}, y_{\text{level}-1}}$;
        if ($w_{current} > w_{best}$) then
          let $w_{best} = w_{current}$;
        else
          let $w_{y_{\text{level}}, y_{\text{level}-1}} = w_{best}$;
        end if
        let $p =$ uniform random number in $[0, 1]$;
      end while
      # climbing up pre-visited tree branches
      let $y = y_{current}$
      while ($y$ is not a leaf node) do
        choose_arc($w_{yx'}$,$w_{yx''}$);
        let level = level + 1;
      end while
    else
      let termination = true
    end if
  end while
```

When the tree has been climbed the maximum number of times (maxpaths), G_{MS} is completely generated.

In this paper, the MS algorithm is used for solving instances of the DMDGP. A solution to the problem is given by a vector of binary variables whose length equals the number of atoms forming the considered protein conformation. Solutions to the DMDGP will be considered as food resources and their quality will be evaluated through the LDE function (1).

The perturbations used in MS for generating new solutions are taken from other meta-heuristic methods for global optimization. In the implementation of MS used in this paper, 8 different perturbations are employed. For instance, new solutions are generated by applying the crossover operator to the current solution y and to the current best solution y_{best}. This perturbation is taken from the Genetic Algorithms [3]. Other perturbations are taken in the same way from methods such as Simulated Annealing [5], Harmony Search [2], Differential Evolution [15], and so on. Local perturbations are also employed, where the value of a single variable is modified in order to explore close solutions. Other perturbations are also inspired by the problem to be solved. For instance, our implementation considers the following perturbation. A subset of consecutive binary variables representing a part of a solution is chosen, and all the values of such variables are inverted. This perturbation is inspired by the studies on the symmetries of the solutions of the DMDGP [10]. The most successful perturbations are adaptively chosen during the search.

MS uses the following strategy for avoiding that the search gets stuck at local minima. The monkey is allowed to climb the same tree only a predetermined number of times (maxpaths). When a tree cannot be climbed anymore, a new tree G_{MS} is started. As a consequence, Algorithm 2 is invoked several times during the execution of the MS algorithm, and every time with a different seed solution v_0. Changing the tree G_{MS} allows to explore completely new solutions. Moreover, a list containing the best solutions found while exploring an entire tree is kept during the search. At the start of the algorithm, each tree is started from a totally random seed, in order to spread the search in different parts of the domain. Then, after a predetermined number of trees, seeds start to be taken from the list of best solutions. Thus, the size of the list of best solutions and the number of trees that are started from a random seed represent two parameters of the MS algorithm. The MS algorithm is stopped when the monkey is not able to find any better solutions after having climbed a predetermined number of trees.

The generation of a random feasible solution to be used as seed solution is, in general, a difficult task. When there are indeed constraints to be satisfied, it can be quite hard to find solutions satisfying all the constraints. As for example, in the application of MS presented in [13], the optimization problem to be solved have been reformulated before applying the search: the constraints have been removed and penalty terms have been added to the objective function. In the case of DMDGP, there are not constraints, and a random solution is generated by creating a random sequence of values 0 and 1.

The MS algorithm has been successfully applied for solving different global optimization problems. In [12], optimal clusters of molecules governed by the Lennard Jones and Morse potential energies were found by MS. In [4], the MS algorithm was used for solving instances of the Multidimensional Assignment Problem (MAP). Finally, in [13], MS was employed for solving a global optimization problem arising from a geometric model for the simulation of protein molecules. The reader is referred to the quoted paper for other details about the MS algorithm.

In this work, the MS algorithm is applied to the DMDGP, in order to compare this meta-heuristic search to the exact algorithm BP. If the instance given as input to MS contains a set of distances which is feasible, then the best solutions to the problem corresponds to values of the LDE function

(1) equal to 0. If the distances are instead infeasible, then the optimal solutions are the ones in which the minimum number of distances are violated (the wrong ones).

5. COMPUTATIONAL EXPERIENCES

The performances of the MS and the BP algorithms are compared on a set of instances of the DMDGP. Both the algorithms have been implemented in C programming language. All tests have been carried out on an Intel Core 2 CPU 6400 @ 2.13 GHz with 4GB RAM, running Linux. The codes implementing the BP and the MS algorithm have been compiled by the GNU C compiler v.4.1.2 with the `-O3` flag.

5.1 Generation of the instances

The instances we use in the computational experiences are generated from the known conformations of some protein molecule. The protein conformations are downloaded from the Protein Data Bank (PDB) [1, 14]. We consider a small set of monomeric proteins having a different number of amino acids. Once downloaded, we extracted all the atoms N, C_α and C belonging to the backbone of a protein and computed all the possible distances between them. Then, all the distances d_{ij} that are smaller than 6Å are considered to form an instance of the DMDGP. All the short range distances $d_{i-3,i}$, $d_{i-2,i}$ and $d_{i-1,i}$ are included in the generated instances, because they all fall below the threshold of 6Å.

In order to simulate data containing a small part of completely wrong distances, a predetermined percentage of distances are arbitrarily modified and changed to a randomly chosen value. In our experiments, we never modify the distances $d_{i-3,i}$, $d_{i-2,i}$ and $d_{i-1,i}$, for two reasons. First, shorter distances can be obtained with a greater precision, and hence the probability that these distances are wrong is lower. Second, the extension of the BP algorithm for considering wrong distances is not able to manage errors on the distances $d_{i-3,i}$, $d_{i-2,i}$ and $d_{i-1,i}$. In fact, these distances are used for building the tree G_{BP}: if some of these distances are wrong, then errors propagate on all the branches of the tree, and no solution to the problem can be found.

5.2 Experiments

We consider a small set of instances of the DMDGP generated by using the procedure detailed above. The proteins `1brv`, `1ppt`, `2erl` and `1dv0` have been selected from the PDB. In correspondence with each of them, an instance of the DMDGP has been generated which contains no errors. Then, different percentages of wrong distances have been introduced, ranging from 1% to 10% of the total number of available distances. The instances are labeled with the name of the original protein and the percentage of wrong distances. For example, the instance `2erl_01` corresponds to the protein `2erl`, and contains the 1% of wrong distances. In Table 1, some properties of the considered proteins are shown, such as the number n of backbone atoms they contain, and the number of distances m contained in the instances generated by the method which is described in Section 5.1.

Preliminary experiments have been carried out in order to tune the parameters of the meta-heuristic MS to the DMDGP. In the experiences presented in this section, the parameters maxpaths and maxlevels are set to 90 and 40, respectively. Moreover, the list containing the best solutions found during the search contains 10 conformations, and the first 40 trees have as seed a randomly generated solution. The search is stopped after that the monkey climbed 40 trees without improving any of the solutions in the list.

protein name	n	m
`1brv`	57	476
`1ppt`	108	912
`2erl`	120	1136
`1dv0`	135	1290

Table 1: Number n of atoms and number m of distances in correspondence with the considered proteins.

	BP			MS	
Instance	LDE	#Sol	CPU	LDE	CPU
`1brv_00`	1.39e-14	2	0.00	1.39e-14	˜21
`1ppt_00`	1.97e-14	2	0.05	1.97e-14	˜16
`2erl_00`	1.33e-14	2	0.08	1.33e-14	˜16
`1bv0_00`	8.63e-15	2	0.16	8.63e-15	˜41

Table 2: Comparisons between BP and MS on instances without wrong distances.

In Table 2, computational experiences are shown in which only the instances without wrong distances are considered. The basic BP algorithm is therefore used here, where a branch is pruned as soon as one atomic position is found to be infeasible. For each experiment, the LDE value of the best found solution, the total number of solutions #Sol and CPU time (in seconds) are reported in correspondence with the BP algorithm, whereas the LDE of the best found solution over 10 runs and the average CPU time are given in correspondence with the MS algorithm. As the table shows, the BP algorithm performs much better in terms of CPU time. The efficiency of BP is probably due to the fact that entire parts of the binary tree G_{BP} are soon pruned away when an infeasible atomic position is discovered. In this way, the domain of the search is reduced very quickly. The MS algorithm, instead, has a constant domain, whose cardinality is 2^{n-3}. This makes the search much more computationally expensive.

In Table 3, instances with a given percentage of wrong distances are considered. All the information provided in the previous table are shown in Table 3 as well. In the experiments related to the MS algorithm, the threshold T_D represents an estimate of the wrong distances contained in the instance. Even though the exact number of wrong distances is known because they are artificially generated, an estimated T_D value is used in order to simulate a realistic experiment. For example, the instances related to the protein `1brv` have 476 distances, and then about 4.8 distances should be wrong when it is supposed that the 1% of the distances are wrong. The estimated T_D values are approximated by the closer larger integer number. Note that an understimation of T_D could bring the algorithm to provide no solutions. The CPU time for the BP algorithm is limited to 1 hour: the experiments in which BP takes exactly one hour have been stopped before that BP ended. Therefore, the number of solutions #Sol, in this case, does not include all the possible solutions. Finally, the LDE value is computed by considering all the available distances, including the wrong ones. This could be avoided when BP is used,

	BP			MS	
Instance	LDE	#Sol	CPU	LDE	CPU
`1brv_01`	3.43e-04	2	0.02	3.43e-04	~35
`1brv_02`	9.78e-04	2	0.03	9.78e-04	~42
`1brv_03`	2.37e-03	4	0.28	2.37e-03	~33
`1brv_04`	2.17e-03	12	0.30	2.17e-03	~33
`1brv_05`	2.75e-03	10	0.43	2.75e-03	~36
`1brv_10`	7.44e-03	4	8.08	7.44e-03	~38
`1ppt_01`	4.55e-04	2	6.07	4.55e-04	~25
`1ppt_02`	1.05e-03	2	274.32	1.05e-03	~25
`1ppt_03`	2.36e-03	3	*1h*	2.36e-03	~35
`1ppt_04`	2.36e-03	1	*1h*	**2.30e-03**	~30
`1ppt_05`	-	0	*1h*	**2.71e-03**	~29
`1ppt_10`	-	0	*1h*	**7.93e-03**	~38
`2erl_01`	3.34e-04	2	0.29	3.34e-04	~87
`2erl_02`	1.01e-03	2	1.42	1.01e-03	~139
`2erl_03`	2.68e-03	2	24.14	2.68e-03	~143
`2erl_04`	2.28e-03	2	27.86	2.28e-03	~136
`2erl_05`	2.65e-03	2	29.51	2.65e-03	~114
`2erl_10`	-	0	*1h*	**8.21e-03**	~129
`1dv0_01`	3.53e-04	2	0.57	3.53e-04	~89
`1dv0_02`	1.08e-03	2	5.60	1.08e-03	~108
`1dv0_03`	2.78e-03	2	66.20	2.78e-03	~135
`1dv0_04`	2.57e-03	54	79.91	2.57e-03	~151
`1dv0_05`	3.09e-03	8	120.35	3.09e-03	~189
`1dv0_10`	-	0	*1h*	**8.26e-03**	~206

Table 3: Comparisons between BP and MS on instances containing wrong distances.

because it is known which ones are the violated distances. However, all the distances are considered even in this case for an easier comparison with the results obtained by MS. As for example, if the violated distances are not considered in the computation of the LDE function, then the LDE value for the solution found by BP in correspondence with the instance `1brv_01` and percentage 1% is 1.37e-14.

The results showed in Table 3 suggest that BP performs better than MS when the percentage of wrong distances is small. As this percentage increases, the CPU time related to BP increases much more than the CPU time related to MS. Therefore, starting from a certain percentage, which depends on the considered protein molecule, the MS algorithm starts to perform better than BP. As for example, BP is able to obtain the same solution in a much smaller amount of time in correspondence with the instance `1dv0_01`. Inversely, when the instance `1ppt_05` is considered, then MS provides a better solution in less time. In this case, BP is stopped after one hour of execution, and this is why it cannot provide the same solution as MS. The best found solutions for the instances `1dv0_01` and `1ppt_05` are shown in Figure 3.

The obtained results are due to the fact that the branches of the tree G_{BP} are pruned later during the BP algorithm as the threshold T_D increases. In this way, larger parts of the tree are in fact exhaustively explored. Therefore, the CPU time for carrying BP out strongly depends on T_D, and hence on the number of wrong distances that are included in the considered instance. Inversely, MS is less sensible to the number of wrong distances. Then, when there are many wrong distances, BP can be too slow, and MS could be preferable. On the other side, BP can always provide all the solutions to the problem, whereas MS is a meta-heuristic algorithm, and therefore it is able to provide a solution only after that a given number of runs are performed. The two methods can therefore be considered as complementary.

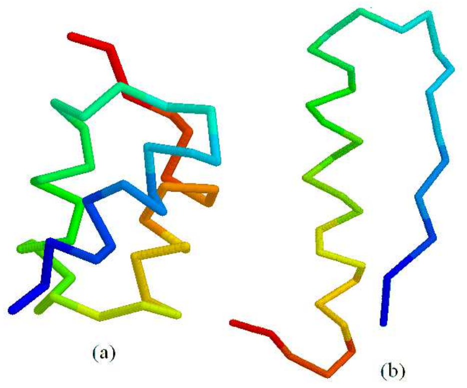

Figure 3: Two solutions: (a) the solution found by BP for the instance `1dv0_01`; (b) the solution found by MS for the instance `1ppt_05`.

6. CONCLUSIONS

We presented a comparison between two algorithms for solving the DMDGP. The first one is the BP algorithm, which implements an exact method that is strongly based on the structure of the combinatorial problem to be solved. The second one is the MS algorithm, a meta-heuristic inspired by the behavior of a monkey in search for food supplies. The instances used during the comparison simulate data obtained by experimental techniques, because they contain the typical distances these techniques are usually able to detect, and a certain percentage of wrong distances is included.

Our comparison showed that the BP algorithm performs much better than MS on instances where the percentage of wrong distances is low. Indeed, the pruning phase of BP allows to quickly reduce the domain of the search, so that an exhaustive search on the remaining feasible solution is not computationally expensive. However, as the number of wrong distances increases, the possibility for BP to prune is reduced, and the computational cost grows. The MS algorithm is instead not dependent on the percentage of wrong distances contained into the instances. Even though it is slower than BP on instances with few wrong distances, it is much faster than BP when the number of wrong distances is larger. Obviously, since MS is a meta-heuristic, there are no guarantees that the found solutions are the optimal ones, and usually more than one execution of the algorithm is needed.

The two considered algorithms are complementary. When the performances of one of them are excellent, the performances of the other are poor, and vice versa. This suggests that better results, in general, could be obtained by exploiting the features of both exact and meta-heuristic algorithms. The study of hybrid strategies can lead to the development of new algorithms which are in part exact and in part meta-heuristic, that could perform better than the two algorithms compared in this paper. As the performances of the MS algorithm are due to several strategies and ideas borrowed from other meta-heuristic searches, new algorithms could be developed that are based on the combination of different strategies, such as the ones exploited in BP and MS. Future research can be devoted in this direction, for solving the DMDGP and other difficult global optimization problems.

7. REFERENCES

[1] H.M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T.N. Bhat, H. Weissig, I.N. Shindyalov, P.E. Bourne, *The Protein Data Bank*, Nucleic Acids Research **28**, 235–242, 2000.

[2] Z.W. Geem, J.H. Kim, G.V. Loganathan, *A New Heuristic Optimization Algorithm: Harmony Search*, Simulations **76**(2), 60–68, 2001.

[3] D.E. Goldberg, *Genetic Algorithms in Search*, Optimization & Machine Learning, Addison-Wesley, 1989.

[4] A.R. Kammerdiner, A. Mucherino, and P.M. Pardalos, *Application of Monkey Search Meta-heuristic to Solving Instances of the Multidimensional Assignment Problem*, Lecture Notes in Control and Information Sciences **381**, 385–397, 2009.

[5] S. Kirkpatrick, C.D. Jr. Gelatt and M.P. Vecchi, *Optimization by Simulated Annealing*, Science **220**(4598), 671–680, 1983.

[6] C. Lavor, L. Liberti, and N. Maculan, *Computational Experience with the Molecular Distance Geometry Problem*, In: Global Optimization: Scientific and Engineering Case Studies, J. Pintér (Ed.), 213–225. Springer, Berlin, 2006.

[7] C. Lavor, L. Liberti, and N. Maculan, *Molecular distance geometry problem*, In: Encyclopedia of Optimization, C. Floudas and P. Pardalos (Eds.), 2^{nd} edition, Springer, New York, 2305–2311, 2009.

[8] C. Lavor, L. Liberti, A. Mucherino and N. Maculan, *On a Discretizable Subclass of Instances of the Molecular Distance Geometry Problem*, Proceedings of the Conference SAC09, Honolulu, Hawaii, March 8/12 (2009).

[9] L. Liberti, C. Lavor, and N. Maculan, *A Branch-and-Prune Algorithm for the Molecular Distance Geometry Problem*, International Transactions in Operational Research **15** (1): 1–17, 2008.

[10] L. Liberti, C. Lavor, and N. Maculan, *Discretizable Molecular Distance Geometry Problem*, Tech. Rep. q-bio.BM/0608012, arXiv, 2006.

[11] A. Mucherino and O. Seref, *Monkey Search: A Novel Meta-Heuristic Search for Global Optimization*, "Data Mining, System Analysis and Optimization in Biomedicine", AIP Conference Proceedings **953**, O. Seref, O.E. Kundakcioglu, P.M. Pardalos (Eds.), 162-173, 2007.

[12] A. Mucherino and O. Seref, *Modeling and Solving Real Life Global Optimization Problems with Meta-Heuristic Methods*, Advances in Modeling Agricultural Systems, Springer Optimization and Its Applications Series, P.J. Papajorgji, P.M. Pardalos (Eds.), November 2008.

[13] A. Mucherino, O. Seref, P.M. Pardalos, *Simulating Protein Conformations through Global Optimization*, arXiv e-print, arXiv:0811.3094v1, November 2008.

[14] Protein Data Bank: `http://www.rcsb.org/pdb/`

[15] R. Storn, K. Price, *Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces*, Journal of Global Optimization **11**: 341-359, 1997.