# A Parallel Version of the Branch & Prune Algorithm for the Molecular Distance Geometry Problem

A. Mucherino* C. Lavor† L. Liberti‡ E.-G. Talbi*

*INRIA, Lille Nord Europe, France. Emails: antonio.mucherino@inria.fr, el-ghazali.talbi@lifl.fr
†Dept. of Applied Mathematics, State University of Campinas, Brazil. Email: clavor@ime.unicamp.br
†LIX, Ecole Polytechnique, Palaiseau, France. Email: liberti@lix.polytechnique.fr

***Abstract*—We consider the Molecular Distance Geometry Problem (MDGP), which is the problem of finding the conformation of a molecule from some known distances between its atoms. Such distances can be estimated by performing experiments of Nuclear Magnetic Resonance (NMR). Unfortunately, data obtained during these experiments are usually noisy and affected by errors. In particular, some of the estimated distances can be wrong, typically because assigned to the wrong pair of atoms. When particular assumptions are satisfied, the problem can be discretized, and solved by employing an ad-hoc algorithm called Branch & Prune (BP). However, this algorithm has been proved to be less efficient than a meta-heuristic algorithm when the percentage of wrong distances is large. We propose a parallel version of the BP algorithm which is able to handle this kind of instances. The scalability of the proposed algorithm allows for solving very large instances containing wrong distances. Implementation details of the algorithm in C/MPI are discussed, and computational experiments, performed on the nation-wide grid infrastructure Grid5000, are presented.**

## I. INTRODUCTION

The DISTANCE GEOMETRY PROBLEM (DGP) [4], [9], [18] is the problem of identifying the coordinates of a set of points which satisfy a given set of constraints based on some relative distances between the points. Let $X = \{x_1, x_2, \dots, x_n\}$ be a set of points in the three-dimensional space, and let us suppose that, for some pairs of points $x_i$ and $x_j$, with $i \neq j$, the distance $d_{ij}$ between the points is known. The DGP is the problem of finding the coordinates of the points in $X$ such that

$$||x_i - x_j|| = d_{ij}, \qquad (1)$$

where $d_{ij}$ are the known distances.

In its basic form, the DGP is a constraint satisfaction problem. However, it is usually reformulated as a global continuous optimization problem (refer, for example, to [2], [8], [15]). The set of constraints (1) is transformed into a penalty function, which is basically the sum of as many terms as the available constraints. Each term is able to measure the satisfaction of each constraint: each constraint is satisfied if and only if the corresponding term of the penalty function is zero. Therefore, finding the global minimum of such a penalty function is equivalent to solving DGP instances. Over the years, different penalty functions associated to DGPs have been proposed. One of the most used is the Largest Distance Error (LDE):

$$LDE(X) = \frac{1}{m} \sum_{\{i,j\}} \frac{|\,||x_i - x_j|| - d_{ij}\,|}{d_{ij}},$$

where $m$ is the total number of known distances.

We are interested in DGPs arising from biology. Distances between pairs of atoms of a molecule can be estimated through experiments of Nuclear Magnetic Resonance (NMR), and the conformation of the molecule can be identified by solving a DGP. This application is of relevant interest, because discovering the conformation of a molecule can give insights on its dynamics, and therefore on its function. Molecules of interest are proteins, which perform several vital functions in cells of living beings. The DGP related to molecules is usually referred to as MOLECULAR DGP (MDGP).

Different techniques have been proposed for solving MDGPs. The reader is referred to [12], [14] for a survey. In this paper, we will consider the subclass of instances of the MDGP which can be discretized. Under certain assumptions, indeed, the domain of the penalty function can be reduced from a continuous to a discrete set. This does not reduce the complexity of the problem, because the MDGP is NP-hard even after the discretization [11], [17]. However, the optimization problem to be solved gets combinatorial, and an ad-hoc algorithm, called BRANCH & PRUNE (BP) [13], can be employed for its efficient solution. Under these hypotheses, we refer to the problem as DISCRETIZABLE MDGP (DMDGP) [11].

After the discretization, the domain of the penalty function can be seen as a binary tree containing positions for the atoms of the considered molecule. The BP algorithm is based on the exploration of this binary tree. At each iteration of the algorithm, two new positions for the current atom are computed, and their feasibility is checked. Then, the search proceeds along branches where feasible positions are found, whereas branches containing infeasible positions are pruned. This pruning phase allows to reduce the branches of the binary tree very quickly, allowing the algorithm to focus its researches on the feasible branches only.

A major issue arising when considering distances obtained by NMR is the following. Experiments of NMR are usually affected by noise and, as a consequence, wrong distances might be provided. Some of the obtained distances, indeed, could be assigned to the wrong pair of atoms, because of noisy information or errors while performing the experiments [5]. This brings to the definition of a set of constraints in which some of them are *wrong*. Even a few distances assigned to an incorrect pair of atoms can spoil the solutions to the corresponding MDGPs. Indeed, if we require that one of the wrong distances must be satisfied, this may result in the violation of many other (good) constraints. In general, the identification of wrong distances can be difficult [1].

In the BP algorithm, branches are pruned as soon as one infeasible position is found. However, if the considered instance contains at least one wrong distance, no branches can pass the pruning test, and no solutions are found. A strategy for overcoming this problem has been previously proposed in [16]. The pruning phase of BP is modified so that branches are pruned only after that a certain number of distances are violated. A counter of violated distances is 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 $\Delta$, the corresponding branch is finally pruned. However, the BP algorithm can be quite expensive when the threshold $\Delta$ is large. In [16], a comparison between the BP algorithm and a meta-heuristic search showed that the use of the meta-heuristic algorithm is preferable in these cases, even though there is no guarantee that the found solutions are optimal.

In this paper, we propose a parallel version of the BP algorithm, which is able to efficiently manage instances of the DMDGP where wrong distances are contained. The basic idea is to divide each instance in many sub-instances, whose solutions are successively combined in order to obtain the solutions of the original instance. The proposed algorithm is implemented in C programming language, and it makes use of the MPI library [6], [7]. Experiments on the nation-wide grid infrastructure Grid5000 [3] show that this parallel version of BP is able to speed up the solution of DMDGPs containing wrong distances.

The paper is organized as follows. In Section II we will describe the DMDGP in more details, and we will discuss the BP algorithm, where particular emphasis will be given to the considered strategy for managing wrong distances. In Section III we will present a parallel version of the BP algorithm, and, in Section IV, some implementation details of the proposed algorithm in a parallel environment are given. Finally, computational experiments are presented in Section V, and conclusions are drawn in Section VI.

## II. The sequential BP algorithm

Let us consider an instance of the MDGP. This instance can be represented through a weighted undirected graph $G = (V, E, d)$, where each vertex corresponds to an atom of the considered molecule, and there is an edge between two vertices if and only if the relative distance between the two corresponding atoms is known. The weight $d$ associated to the edge corresponds to the numerical value of the distance.

The set of instances of the DMDGP contains all the instances of the MDGP satisfying the following two assumptions, for a given ordering on $V$:

**Assumption 1**: $E$ contains all cliques on quadruplets of consecutive vertices, i.e.

$$\forall i \in \{4, \dots, n\} \quad \forall j, k \in \{i-3, \dots, i\} \qquad (j, k) \in E;$$

**Assumption 2**: the following strict triangular inequality

$$d_{i-2,i} < d_{i-2,i-1} + d_{i-1,i}$$

holds for all $i \in \{2, \dots, n-1\}$. In practice, Assumption 1 ensures that the distances between each possible pair of atoms in any quadruplet of consecutive atoms are known. Moreover, if Assumption 2 holds, there cannot be triplets of consecutive atoms that are perfectly aligned. When these two assumptions are satisfied for an entire conformation $X$, then, there exist only two possible positions in which each of its atoms can be placed, if the preceding atoms already have a fixed position. This leads to the definition of a binary tree of possible atomic positions, where solutions to the problem can be searched [11].

Let us consider a generic quadruplet of consecutive atoms $\{x_{i-3}, x_{i-2}, x_{i-1}, x_i\}$. Because of Assumption 1, all the possible distances in the quadruplet are known. Moreover, there are no aligned atoms, because of Assumption 2. In these hypotheses, the cosine of the torsion angle among these four atoms can be computed, and two values for the torsion angle can be obtained from the value of its cosine. Therefore, if we suppose that the three atoms $x_{i-3}$, $x_{i-2}$, $x_{i-1}$ are already placed somewhere, we can compute two possible positions for the atoms $x_i$. For more details, see [13].

A binary tree of atomic positions can be obtained with this methodology. All the positions on the tree are computed by exploiting the known distances from the three preceding atoms. However, other distances $d_{ij}$ may also be known (even though it is not required by Assumptions 1 and 2), and this allows to check the feasibility of the computed positions. If $x_i'$ is one possible position for $x_i$, then we can compare all the known distances between $x_i$ and $x_j$, with $j < i$, to the corresponding distances that can be computed between $x_i'$ and each $x_j$. If known and computed distances match:

$$|\, ||x_i' - x_j|| - d_{ij} \,| < \varepsilon,$$

for a given tolerance $\varepsilon > 0$, then $x_i'$ is feasible, otherwise it is not. We refer to this strategy for checking the feasibility of the atomic positions as *pruning test*.

When wrong distances are included in the instance, the pruning test could give an incorrect answer. Supposing that we are checking the feasibility of the computed position $x_i'$ for the atom $x_i$, if one of the distances $d_{ji}$, for some $j < i$, is wrong, then $x_i'$ can be declared infeasible even though it might not be. This can force the pruning of a branch where there are actually solutions. The following strategy has been proposed in [16] for overcoming this problem. Once a certain threshold $\Delta$ on the maximum allowed number of violated distances has been set up, each branch is pruned only after that the actual number of violated distances gets larger than $\Delta$. When the value of $\Delta$ increases, the number of *feasible* branches of the tree increases as well. In order to reduce the effects of this phenomenon, the value of the threshold $\Delta$ can be updated during the execution of BP, when solutions are found in which less than $\Delta$ distances are violated.

Note that, if some of the distances related to the quadruplets of consecutive atoms $\{x_{i-3}, x_{i-2}, x_{i-1}, x_i\}$ are wrong, then it might be impossible to compute the two atomic positions for the atom $x_i$, because the distances in this quadruplet may be incompatible to each other. For example, let us suppose that the distance $d_{i-2,i}$ between $x_{i-2}$ and $x_i$ is wrong. As a consequence, we may not be able to compute the cosine of the torsion angle among these four atoms. This can be considered as a signal which can reveal the presence of wrong distances: there is at least one wrong distance in the considered quadruplet. In order to find out which distance could be wrong, we can check the validity of the triangular inequality in correspondence with the triangles that can be defined in the quadruplet. By hypothesis, we require the knowledge of all the distances in the quadruplet, and therefore we have enough information for checking the triangular inequalities. If some of them is not satisfied, then at least one distance in the triangle is wrong. For this reason, we will consider only wrong distances which are not contained in the quadruplets of consecutive atoms, because they can be more easily identified and corrected. In particular, isolated distances are the most difficult to be recognized as wrong.

For solving instances of the DMDGP, we consider the BP algorithm [13], which is strongly based on the binary tree structure of the combinatorial problem. Algorithm 1 provides a sketch of this algorithm. $i$ is the current atom whose position is searched, whereas $n$ is the the total number of atoms of the considered molecule. $d$ represents the set of known distances. $\Delta$ is the threshold for the number of violated distances. Algorithm 1 is invoked iteratively, and one of the solutions to the problem is found when BP($n$,$n$,$d$,$\Delta$,$sol$) finds a feasible position for the last atom of the molecular conformation. The given output is the binary matrix $sol$ representing a set of complete paths on the binary tree. This is sufficient for reconstructing the conformations for the molecule which are solutions to the problem.

Note that, because of a symmetry property of the DMDGP, it is sufficient to explore only half of the binary tree. Indeed, the solutions contained in the other part of the tree can be simply reconstructed by symmetry. The reader can refer to [11] for more details.

## III. The parallel BP algorithm

The basic idea of this parallel version of BP is to split a generic instance of the DMDGP in sub-instances and to solve them independently on different processes. In order to obtain the sub-instances for $p$ processes, the first block of $n/p$ atoms of the original instance are assigned to the first process, the second block of $n/p$ atoms are assigned to the second one, and so on. If $n$ is not perfectly divisible by $p$, the remaining atoms can be either assigned to a certain process, or distributed among all the processes, depending on $p$. Only the distances regarding the atoms assigned to a process are included in the corresponding sub-instance. As a consequence, the distances between atoms that are assigned to different processes are not used, but they can be exploited later, when the local solutions are combined.

Algorithm 2 is a sketch of the proposed parallel version for the BP algorithm. The input and output parameters are the same that are considered in the sequential version of the algorithm (see Algorithm 1), plus the number $p$ of processes involved in the computation. Once the instance in input is split in $p$ sub-instances, $p$ parallel executions of the sequential ver-

**Algorithm 1** The BP algorithm (sequential version)

```
0: BP(in: $i$, $n$, $d$, $\Delta$; out: $sol$)
0: $counter = 0$   // counter of violated distances
0: $nsol = 0$   // number of solutions
  for ($k = 0, 1$) do
    compute the $k^{th}$ atomic position for the $i^{th}$ atom: $x_i^{(k)}$;
    check the feasibility of the atomic position $x_i^{(k)}$:
    if (the atomic position $x_i^{(k)}$ is NOT feasible) then
      update $counter$;
    end if
    if ($counter \leq \Delta$) then
      $sol(nsol, i) = k$;
      if ($i = n$) then
        $sol(nsol, *)$ contains a complete solution;
        $nsol = nsol + 1$;
      else
        BP($i + 1$,$n$,$d$,$\Delta$,$sol$);
      end if
    end if
  end for
  return $sol$;
```

**Algorithm 2** The BP algorithm (parallel version)

```
0: parallel BP(in: $i$, $n$, $d$, $\Delta$, $p$; out: $sol$)
0: split the instance in $p$ sub-instances:
0:         compute $i^{(k)}$, $n^{(k)}$, $d^{(k)}$ ($k = 0, \ldots, p-1$);
  for each process $k = 0, \ldots, p-1$ (in parallel) do
    call sequential BP($i^{(k)}$,$n^{(k)}$,$d^{(k)}$,$\Delta$,$sol^{(k)}$);
  end for
  broadcast $sol^{(k)}$ to the other processes (cascade schema);
  $sol = [sol^{(0)}, sol^{(1)}, \ldots, sol^{(p-1)}]$;
  build the binary tree associated to $sol$;
  remove infeasible solutions from $sol$;
  return $sol$;
```

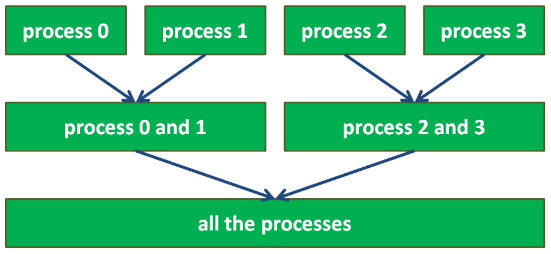

Fig. 1. The classic communication schema “cascade”, where only $\log_2 p$ communication phases are needed to the processes for sharing an information.

sion of BP are performed. When all the executions are ended, the found (local) solutions are given in output through $sol^{(k)}$. They are stored in terms of the binary choices $(0/1$, left/right branch) that are made at each iteration of the algorithm.

After the execution of the local BP algorithms, the local solutions need to be collected and distributed to all the processes. To this aim, we consider the classic *cascade* schema for the necessary communications among the processes (see Figure 1). This schema is very efficient, because it allows the spreading of all local solutions among the processes in only $\log_2 p$ phases of communication. In order to use this schema, the number $p$ of considered processes must be a power of 2.

During each communication phase, each process exchanges information regarding the local solutions found by the sequential BPs with other processes. After each such phase, therefore, we can start combining local solutions. For example, after the first communication phase (see Figure 1), the solutions found by processes 0 and 1 could be combined, as well as the solutions found by processes 2 and 3. This would produce two new sets of local solutions, that could be exchanged in the next communication phase. However, the final coordinates for the solutions can be produced only when all local solutions are collected. The coordinates produced when combining a partial number of solutions can be reused when the final solutions are built only if translations or rotations are applied to the set of points. Therefore, it is preferable that each single process does not perform any computation during the different communication phases. Only after, each process builds the final solutions, and checks their feasibility. This part of the algorithm is not executed in parallel.

In order to combine the local solutions, we consider a binary tree $T_f$ formed only by branches which passed the pruning test during the sequential executions of BP. Let us indicate with the symbol $T^k$ the binary tree containing the local solutions found by process $k$, with $0 \leq k < p$. Let $G^k = (V^k, E^k)$ be the corresponding undirected graph, where vertices $v \in V^k$ represent atomic positions, and edges connect vertices related to consecutive atomic positions. In order to combine the local solutions found by two processes $k_1$ and $k_2 = k_1 + 1$, we can construct a new binary tree as follows. Let $\hat{V}$ be the set of vertices of the corresponding graph, which contains all the vertices $v$ in $V^{k_1}$ and $V^{k_2}$, where the vertices $v$ of $V^{k_2}$ are also duplicated as many times as the number of

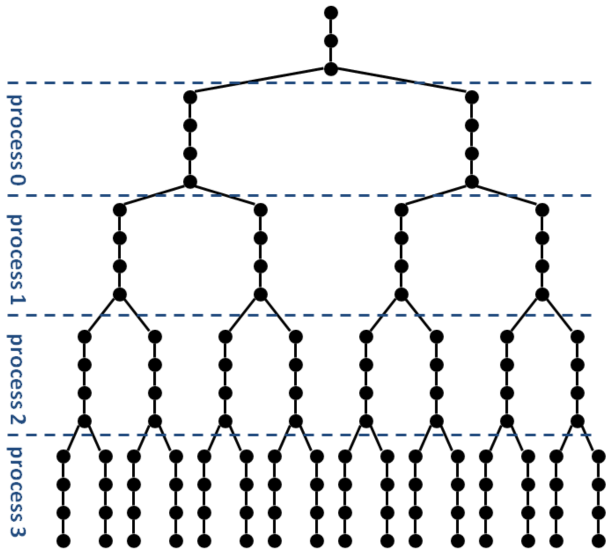

Fig. 2. The final tree $T_f$ of solutions obtained by combining the subtrees $T^k$ related to the local solutions found on $k$ processes. Some of the branches of $T_f$ could be infeasible.

leaf vertices in $V^{k_1}$, and new labels are assigned to them. The set of edges $\hat{E}$ is obtained similarly, and, for each leaf vertex $v_l$ of $V^{k_1}$, a new edge is added between $v_l$ and the various copies of the first vertex of $V^{k_2}$. If this procedure is performed recursively considering all the graphs $G^k$, then the final tree $T_f$ is constructed. Note that distances related to atoms previously assigned to different processes can be used for pruning branches of $T_f$ and removing infeasible solutions.

Figure 2 gives a representation of this tree in the case in which 4 processes are considered. For an easy representation of $T_f$, we suppose that all the local calls to BP provide 2 solutions, even though, in general, the number of solutions found by each process can be different.

By the symmetry property of the DMDGP [11], each local BP can be employed for computing only the non-symmetric solutions for its sub-instance. Then, the other symmetric solutions can be found by exploiting this property. Indeed, if $sol$ contains non-symmetric solutions only, then all the symmetric solutions can be identified by computing the complement of each binary variable used for their representation. Note that the graphs $G^k$, for each process $k$, must contain both symmetric and non-symmetric solutions in order to build a complete final tree $T_f$.

## IV. Implementation details

For the parallel implementation of the BP algorithm, we use the MPI library [6], [7]. The communications based on the cascade schema are implemented by employing pairs of blocking MPI standard routines for sending and receiving messages.

In the parallel BP, the message to be shared contains the number of solutions found by each process, and variables containing the local solutions. In order to compact the information

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |

Fig. 3. A byte containing information regarding 8 different local solutions. If this byte is related to the atom $x_i$, and $x'_i$ and $x''_i$ are the two possible positions for $x_i$, then $x'_i$ was chosen in the $2^{nd}$, the $3^{rd}$ and the last two solutions (from right to left); in all the other cases, $x''_i$ was chosen.

to be shared and to speed up the communication phases, a unique vector of integer numbers is used for representing all found solutions. This is possible because each solution can be represented by a binary vector: the first bit of each integer can be used for representing the first solution, the second bit for the second solution, and so on. Since 4 bytes are usually used for representing integer numbers on computer machines, each vector of integers can only store 32 solutions. In order to consider more solutions, more than one integer should be considered in correspondence with the same atom. However, in our experience, we never found instances related to real molecules where the total number of solutions is so large. Therefore, in our implementation, we consider a single vector of $n$ integer numbers, where all the solutions are stored bit per bit. An example of interpretation of a single byte of an integer number belonging to this vector is showed in Figure 3.

The number of solutions found by each process is also an important information to be shared. However, in our implementation, this information is not contained in the messages exchanged by the processes, because it can be easily computed by each process once the vectors of integer numbers are all collected. Indeed, from the values of the integer numbers, we can check which bits have been activated (1) and which ones kept instead their initial value (0). In the example in Figure 3, we can see that the last two bits (from right to left) are not activated. If they are not activated for all the integers of the same local vector, then only 6 solutions have been found by the sequential call to BP.

Note that this strategy for computing the number of solutions may fail if the last solution which is stored is represented by a sequence of 0s. This solution, in theory, may be found by the local calls to BP. However, since our implementation of BP performs an exploration of the binary trees which considers left branches (associated to 0) before right branches (associated to 1), if this solution is included in the solution set, then it is the first one to be stored. As a consequence, the only case in which our strategy can fail is never verified.

## V. Computational experiments

The computational experiments are carried out on the nation-wide grid infrastructure Grid5000 [3]. It is a grid formed by various clusters geographically distributed in 9 different sites in France. Just recently, a new site located in Brazil, Porto Alegre, has been added to the grid. Grid5000 connects more than 5000 cores. For more information about Grid5000, the interested reader can visit the official website (`https://www.grid5000.fr`).

In our experiments, we consider a unique cluster of Grid5000 which is located in Lille, France, in order to consider

| $n = 5000$ | | | | | | | |
|---|---|---|---|---|---|---|---|
| $\Delta/p$ | 1 | 2 | 4 | 8 | 16 | 32 | 64 |
| 0 | 3.21 | 1.30 | 0.78 | 0.54 | 0.40 | 0.37 | 0.36 |
| 1 | 3.25 | 1.32 | 0.77 | 0.55 | 0.40 | 0.36 | 0.40 |
| 2 | 3.57 | 1.36 | 1.06 | 0.80 | 0.56 | 0.51 | 0.53 |
| 4 | 5.68 | 2.16 | 2.10 | 1.60 | 0.86 | 0.77 | 0.75 |

| $n = 7500$ | | | | | | | |
|---|---|---|---|---|---|---|---|
| $\Delta/p$ | 1 | 2 | 4 | 8 | 16 | 32 | 64 |
| 0 | 4.73 | 3.15 | 1.83 | 1.25 | 0.99 | 0.88 | 0.93 |
| 1 | 4.76 | 3.14 | 1.84 | 1.30 | 0.96 | 0.89 | 0.93 |
| 2 | 6.15 | 5.30 | 4.49 | 2.48 | 1.20 | 1.04 | 0.96 |
| 4 | 9.53 | 8.85 | 5.58 | 2.74 | 2.21 | 1.67 | 1.55 |

| $n = 10000$ | | | | | | | |
|---|---|---|---|---|---|---|---|
| $\Delta/p$ | 1 | 2 | 4 | 8 | 16 | 32 | 64 |
| 0 | 13.38 | 5.49 | 3.57 | 2.49 | 1.99 | 1.72 | 1.57 |
| 1 | 13.47 | 5.51 | 3.29 | 2.23 | 1.74 | 1.55 | 1.53 |
| 2 | 19.16 | 8.65 | 4.29 | 3.09 | 3.22 | 2.91 | 2.24 |
| 4 | 127.51 | 22.74 | 13.99 | 10.27 | 11.79 | 11.55 | 14.72 |

TABLE I
The CPU time, in seconds, needed for carrying out our computational experiments on a set of artificial instances.

nodes of the grid with the same features. We use a SGI Altix Xe 310 cluster with 46 nodes, each of them composed by 2 CPUs (Intel Xeon E5440 QC 2.83 GHz, 4 MB, 1333 MHz). Summing up, this cluster has 92 CPUs, and we use 64 of them in our experiments (because of the employed communication schema, a power of 2 of processes must be considered). The reservation system for the resources of the grid makes sure that the used CPUs are completely devoted to our experiments.

We implemented the parallel BP in C programming language. We used the MPI library [6], [7], version MPICH2 1.0.5p4, and the GNU C compiler version 4.2.4. The efficient routines provided by MPI for making the involved processes communicate and the flexibility of C programming language in the management of bitwise operations make C and MPI the best possible choices for the implementation of our parallel BP.

The instances we use in the computational experiences are artificially generated, so that they resemble conformations of protein molecules [10]. Moreover, in order to simulate instances containing a small part of wrong distances, a predetermined number of distances are modified and arbitrarily changed to a wrong value. As previously remarked, errors in the distances $d_{i-3,i}$, $d_{i-2,i}$ and $d_{i-1,i}$ can be easily identified when using this discrete approach, and therefore we never modify these distances.

Table I shows some computational experiments with three different instances, having size $n = 5000$, 7500 and 10000. The total number of known distances is 84195, 135475, 183444, respectively. For each instance, we introduce 1, 2 or 4 wrong distances. We also consider the instance as is, without introducing any errors. The value of the threshold $\Delta$ is set up accordingly to the number of introduced wrong distances (in general, however, the precise number of wrong distances may not be known a priori). For each instance and for each $\Delta$, one sequential execution of BP is provided, as well as six parallel executions in which a power of 2 processes $p$

(with $2 \leq p \leq 64$) is used. For each experiment, the total computational time, expressed in seconds, is given.

The experiments show that the parallel version of BP is able to find the solutions to DMDGPs more efficiently. For all the experiments with the same instance and the same $\Delta$, we always obtained the same set of final solutions, and, when the number $p$ of processes involved in the computation increases, this set of solutions is obtained in a shorter time. This reduction of time is more evident when 2 processes are used instead of 1, or 4 processes are used instead of 2. When $p$ increases, the gain in computational time is less evident and, in a few cases, executions with more processes took a time which is slightly larger than the one needed for some executions with less processes. This is due to the fact that, as $p$ increases, the sub-instances considered on each process get smaller, whereas the number of branches of the final binary tree $T_f$ increases. As a consequence, the parallel part of the algorithm (the calls to the sequential BPs) tends to be less expensive than the sequential part, in which all the processes work on the same task: building $T_f$ and giving in output the final set of solutions.

## VI. Conclusions

We presented a parallel version of the BP algorithm for the DMDGP. Instances of the problem are divided in sub-instances that are solved independently by the various processes involved in the computation. Then, after the communication of the local solutions, they are combined and the final set of solutions of the original instance is found. We performed some computational experiments on the nation-wide grid infrastructure Grid5000, and they showed the scalability of the proposed algorithm. Large instances containing wrong distances can be solved in parallel in a few seconds.

## Acknowledgments

The authors wish to thank Jean-Claude Charr for his help in organizing the computational experiments on Grid5000. Also, we would like to thank the Brazilian research agencies FAPESP and CNPq, the French research agency CNRS and École Polytechnique, for financial support.

## References

[1] B. Berger, J. Kleinberg and T. Leighton, *Reconstructing a Three-Dimensional Model with Arbitrary Errors*, Journal of the ACM **46**, 212–235, 1999.

[2] P. Biswas, K.-C. Toh, and 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**, 1251–1277, 2008.

[3] R. Bolze, F. Cappello, E. aron, M. Daydé, F. Desprez, E. Jeannot, Y. Jégou, S. Lantéri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, E-G. Talbi and I. Touche, *Grid'5000: a Large Scale and Highly Reconfigurable Experimental Grid Testbed*, International Journal of High Performance Computing Applications **20** (4), 481–494, 2006.

[4] G.M. Crippen and T.F. Havel, *Distance Geometry and Molecular Conformation*, John Wiley & Sons, New York, 1988.

[5] W. Gronwald, H.R. Kalbitzer, *Automated Structure Determination of Proteins by NMR Spectroscopy*, Progress in Nuclear Magnetic Resonance Spectroscopy **44** (1–2), 33–96, 2004.

[6] W.D. Gropp, Ewing Lusk, *User's Guide for* `mpich`*, a Portable Implementation of MPI*, Mathematics and Computer Science Division, Argonne National Laboratory, 1996.

[7] W. Gropp, E. Lusk, N. Doss, A. Skjellum, *A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard*, Parallel Computing **22** (6), 789–828, 1996.

[8] A. Grosso, M. Locatelli, F. Schoen, *Solving Molecular Distance Geometry Problems by Global Optimization Algorithms*, Computational Optimization and Applications **43**, 23–27, 2009.

[9] T.F. Havel, *Distance Geometry*, In: D.M. Grant and R.K. Harris (Eds.), Encyclopedia of Nuclear Magnetic Resonance, Wiley, New York, 1701-1710, 1995.

[10] C. Lavor, *On Generating Instances for the Molecular Distance Geometry Problem*, In: L. Liberti and N. Maculan (Eds.), Global Optimization: from Theory to Implementation, Springer, New York, 405–414, 2006.

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

[12] 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.

[13] 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.

[14] L. Liberti, C. Lavor, A. Mucherino, N. Maculan, *Molecular Distance Geometry Methods: from Continuous to Discrete*, to appear in International Transactions in Operational Research, 2010.

[15] J.J. Moré, Z. Wu, *Global Continuation for Distance Geometry Problems*, SIAM Journal on Optimization **7**, 814–836, 1997.

[16] A. Mucherino, L. Liberti, C. Lavor, and N. Maculan, *Comparisons between an Exact and a MetaHeuristic Algorithm for the Molecular Distance Geometry Problem*, ACM Conference Proceedings, Genetic and Evolutionary Computation Conference (GECCO09), Montréal, Canada, 333–340, 2009.

[17] J.B. Saxe, *Embeddability of Weighted Graphs in $k$-space is Strongly NP-hard*, Proceedings of $17^{th}$ Allerton Conference in Communications, Control, and Computing, Monticello, IL, 480–489, 1979.

[18] T. Schlick, *Molecular Modelling and Simulation: an Interdisciplinary Guide*, Springer, New York, 2002.