On Suitable Parallel Implementations of the Branch & Prune Algorithm for Distance Geometry

A. Mucherino[*], C. Lavor[†], L. Liberti[‡], E.-G. Talbi[§]

Grid5000 Spring School 2010, Lille

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). When particular assumptions are satisfied, the problem can be discretized, and solved by employing an ad-hoc algorithm called Branch & Prune (BP). We analyse and discuss possible parallel versions for the BP algorithm, and we provide some computational experiments for the parallel version which appeared to be the most promising. Computational experiments, performed on the nation-wide grid infrastructure Grid5000, show that the considered parallel version of the BP algorithm is able to solve very large instances of the discretized MDGP. The experiments also show the scalability of the parallel algorithm.

1 Introduction

The Distance Geometry Problem (DGP) [3, 7, 12, 14] 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 $X = \{x_1, x_2, \ldots, x_n\}$ 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

$$||x_i - x_j|| = d_{ij}, \tag{1}$$

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. 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):

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

[*]INRIA, Lille Nord Europe, France. Email: antonio.mucherino@inria.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
[§]INRIA, Lille Nord Europe, France. Email: el-ghazali.talbi@lifl.fr

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).

We 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 [9, 13]. However, the optimization problem to be solved gets combinatorial, and an ad-hoc algorithm, called Branch & Prune (BP) [11], can be employed for its efficient solution. Under these hypotheses, we refer to the problem as Discretizable MDGP (DMDGP) [9].

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.

In this short communication, we discuss different possible parallel versions of the BP algorithm. Moreover, we choose the version which seems the most promising, and we provide computational experiments which show its capability in solving DMDGP instances into a reduced amount of time. The experiments also show the scalability of the considered parallel version.

The remaining is organized as follows. In Section 2 we will give a short description of the DMDGP and we will present the BP algorithm. In Section 3 we will discuss possible parallel versions of the BP algorithm, and then we will focus our attention on only of them. Computational experiments will be discussed in Section 4, and conclusions will be given in Section 5.

2 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.

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

Assumption 2: the following strict triangular inequality

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

holds for all $i \in \{2, \ldots, n-1\}$. 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

Algorithm 1 The BP algorithm (the sequential version on the right, and the parallel version on the left)

```
0: BP(in: i, n, d; 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 feasible) then
       sol(nsol, i) = k;
       if (i = n) then
         sol(nsol, *) contains a complete solution;
         nsol = nsol + 1;
       else
         BP(i + 1,n,d,sol);
       end if
     end if
   end for
   return sol;
```

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

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 [9]. For more details, the reader is referred to [9, 10, 11].

All the positions on the binary 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*.

For solving instances of the DMDGP, we consider the BP algorithm [11], which is strongly based on the binary tree structure of the combinatorial problem. In the sequential version of the algorithm, i represents the current atom whose position is searched, n is the the total number of atoms of the considered molecule, and d represents the set of known distances. The BP algorithm is invoked iteratively, and one of the solutions to the problem is found when BP(n,n,d,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.

3 Possible parallel implementations for the BP algorithm

The BP algorithm is an intrinsically parallel algorithm. Since the basic idea is to explore a binary tree of atomic positions, a possible parallel version of the algorithm could simply distribute the branches of the binary tree among the processes involved in the computation, and then collect the different solutions from

the various processes. However, even though all the sub-trees on the different processes can potentially have the same size, the pruning phase might remove a different number of branches from each sub-tree, in this way causing a non-balanced computational load on the processes. Therefore, even though this parallel version of the algorithm is very natural and easily implementable, it does not ensure a good scalability.

Another possible implementation for the BP algorithm is the following. A generic instance of the DMDGP can be divided in many sub-instances which can be solved independently on different processes by local calls to BP. This is also a classic approach to parallelize an algorithm, but, in this particular case, it is not trivial. Indeed, while creating the sub-instances, distances regarding atoms assigned to two different sub-instances are lost. However, such distances may be important for removing infeasible solutions from the final solution set. Therefore, they need to be considered later when local solutions are combined. This parallel version of the algorithm potentially have better scalability properties of the previous one.

Given an instance of the DMDGP, p sub-instances can be generated and solved independently on p different processes by BP. In order to combine the local solutions, a binary tree T_f can be considered, which is 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. For combining 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 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. Distances related to atoms previously assigned to different processes can be used for pruning branches of T_f and removing infeasible solutions.

In the previous page, a sketch of this parallel version of the BP algorithm is given. The input and output parameters are the same that are considered in the sequential version of the algorithm, 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 version 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 are collected and distributed to all the processes. To this aim, we consider the *cascade* schema for the necessary communications among the processes. 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.

4 Computational experiments

The computational experiments are carried out on the nation-wide grid infrastructure Grid5000 [2]. 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 nodes of the grid with the same features. We use a SGI Altix Xe 310 cluster with 46 nodes,

n	m	1	2	4	8	16	32	64
1000	14338	0.09	0.04	0.03	0.03	0.02	0.02	0.03
2500	36795	0.56	0.30	0.17	0.12	0.10	0.09	0.09
5000	84195	3.21	1.30	0.78	0.54	0.40	0.37	0.36
7500	135475	4.73	3.15	1.83	1.25	0.99	0.88	0.93
10000	183444	13.38	5.49	3.57	2.49	1.99	1.72	1.57

Table 1: The CPU time, in seconds, needed for carrying out our computational experiments on a set of artificial instances.

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.

The local solutions found by each process are stored bit by bit on arrays of integer numbers. This strategy allows for reducing the amount of data which needs to be exchanged by the processes. In order to efficiently implement the needed bitwise operations, we choose to implement our parallel BP in C programming language (GNU C compiler version 4.2.4). Moreover, we used the MPI library [5, 6], version MPICH2 1.0.5p4, for performing an efficient communication among the processes involved in the computation. The instances used in the computational experiences are artificially generated, so that they resemble conformations of protein molecules [8].

Table 1 shows some computational experiments with three different instances, having size n from 1000 to 10000. For each instance, 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 in the table.

The experiments show that the considered parallel version of BP is able to find the solutions to DMDGPs more efficiently. For all the experiments related to the same instance, we always obtained the same final set of solutions, and, when the number p of processes involved in the computation increases, this set of solutions is generally 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.

5 Conclusions

We discussed possible parallel versions of the BP algorithm, which we use for solving instances of the DMDGP. Moreover, we presented one of such parallel versions in more details, and computational experiments have been discussed. In the considered parallel version, 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. The computational experiments showed the scalability of the proposed parallel version of the BP algorithm.

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] 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.

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

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

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

[10] C. Lavor, A. Mucherino, L. Liberti, N. Maculan, *Discrete Approaches for Solving Molecular Distance Geometry Problems using NMR Data*, to appear in International Journal of Computational Biosciences, 2010.

[11] 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.

[12] 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.

[13] 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.

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