MD-jeep: an Implementation of a Branch & Prune Algorithm for Distance Geometry Problems

A. Mucherino[1], L. Liberti[2], and C. Lavor[3]

[1] INRIA Lille Nord Europe, Villeneuve d'Ascq, France,
antonio.mucherino@inria.fr
[2] LIX, École Polytechnique, Palaiseau, France,
liberti@lix.polytechnique.fr
[3] Dept. of Applied Mathematics, State University of Campinas, Campinas-SP, Brazil,
clavor@ime.unicamp.br

Abstract. We present `MD-jeep`, an implementation of a Branch & Prune (BP) algorithm, which we employ for the solution of distance geometry problems related to molecular conformations. We consider the problem of finding the conformation of a molecule from the distances between some pairs of its atoms, which can be estimated by experimental techniques. We reformulate this problem as a combinatorial optimization problem, and describe a branch and prune solution strategy. We discuss its software implementation, and its complexity in terms of floating-point operations and memory requirements. `MD-jeep` has been developed in the C programming language. The sources of the presented software are available on the Internet under the GNU General Public License (v.2).

1 Introduction

The Distance Geometry Problem (DGP) [4, 5, 8, 13] is the problem of finding the coordinates of a set of points from some known distances between pairs of such points. There are different real-life applications where a DGP needs to be solved, and the most interesting and challenging application arises in biology. Distances between pairs of atoms of a molecule can be estimated through experiments of Nuclear Magnetic Resonance (NMR), and such distances can be used for formulating a DGP. DGPs arising in biology are usually referred to as Molecular DGPs (MDGPs — whence the name of our software).

Proteins are important molecules, because they perform many important functions in living beings. There is a web database, the Protein Data Bank (PDB) [1] (web address: `http://www.rcsb.org/`), which is completely devoted to the three-dimensional conformations of proteins. In fact, the conformation of a protein can give insights on its dynamics in the cells, and therefore on its function. Currently, the conformation and the function of many proteins are still not known: each gene of recently sequenced genomes is potentially able to code for a protein (or for more than one), but the corresponding conformation and function are still unknown. One way of solving this problem is to isolate each of such proteins and to perform experiments of NMR in order to obtain a subset of distances between pairs of their atoms. The successive step is to solve an MDGP.

The MDGP is, in its basic form, a constraint satisfaction problem, where molecular conformations, that satisfy all the constraints based on the distances, must be identified. This problem is often reformulated as a global continuous optimization problem, where a penalty function, measuring the satisfaction of the set of constraints, needs to be minimized. Different penalty functions have been proposed over the years for the MDGP, and all of them contain several local minima, where many traditional nonlinear descent methods can easily get stuck at. The most common penalty function is the Largest Distance Error (LDE):

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

where $X = \{x_1, x_2, \ldots, x_n\}$ is a three-dimensional conformation of n atoms, and m is the number of known distances d_{ij}.

Many techniques have been proposed for the MDGP, and the reader is referred to [8, 13] for a survey. However, there are only a few software for the MDGP that are freely available for the scientific community. As an example, `DGSOL` (`http://www.mcs.anl.gov/~more/dgsol/`) is based on the idea of approximating the penalty function (in the continuous reformulation of the problem) with a sequence of smoother functions converging to the original objective function [14]. Other available software products are based on general meta-heuristic searches for global optimization. `Xplor-NIH` (`http://nmr.cit.nih.gov/xplor-nih/`) has been particularly designed for solving MDGPs arising from NMR experiments [19], and it includes different functionalities. In particular, for the solution of MDGPs, it makes use of heuristic methods (such as Simulated Annealing) and local search methods (such as Conjugate Gradient Minimization). Finally, `TINKER` (`http://dasher.wustl.edu/tinker/`) is a package for molecular modeling and design. It includes many force fields for attempting the prediction of protein conformations from their chemical structure only. One of its functionalities, however, is to solve MDGPs. `TINKER` implements the method for distance geometry proposed in [6]. Solutions are found by applying the meta-heuristic Simulated Annealing and they are successively equilibrated with Molecular Dynamics techniques.

Ever since 2006 [7–13, 15–17], we have been working on a combinatorial reformulation of the MDGP. When some particular assumptions are satisfied [12], the domain of the penalty function can be discretized, and, in particular, it can be seen as a binary tree containing positions for the atoms of the considered molecule. Therefore, the optimization problem to be solved becomes combinatorial, and we refer to this combinatorial reformulation as the Discretizable MDGP (DMDGP). Both the MDGP and the DMDGP are NP-hard [7, 18].

In order to solve instances of the reformulated problem, we employ a Branch & Prune (BP) algorithm [12], which is strongly based on the binary tree structure of the penalty function domain. The basic idea is to construct the binary tree during the execution of the algorithm. At each iteration, two new nodes of the tree are added, which represent two new positions for a current atom x_i. Then, the feasibility of the two positions is checked, and branches of the tree containing infeasible positions are pruned. This pruning phase allows for reducing the binary tree very quickly, and for solving the DMDGP in a reasonable amount of time. The BP algorithm has been shown to provide very accurate solutions on sets of instances related to protein conformations.

In this paper, we present the software `MD-jeep`, which is an implementation of the BP algorithm in the C programming language. We present in detail the implementation strategies that are employed for an efficient execution of the BP algorithm. In particular, we describe the strategy we consider for reducing to the minimum possible the memory requirement and the floating-point operations. Computational experiments are shown, and implementation issues regarding future versions of the software are also discussed. `MD-jeep` is distributed under the GNU General Public License (v.2) and it can be downloaded from the following web address: `http://www.antoniomucherino.it/en/mdjeep.php`.

The rest of the paper is organized as follows. In Section 2 we describe the BP algorithm and we discuss several implementation details regarding the development of `MD-jeep`. In Section 3 we present some computational experiments obtained by using the developed software, and show how the outputs it provides can be visualized by using visualization software. In Section 4 we discuss some implementation issues related to future versions of `MD-jeep`. Section 5 concludes the paper.

2 An implementation of the BP algorithm

2.1 The DMDGP and the BP algorithm

Let $G = (V, E, d)$ be a weighted undirected graph, where vertices in $V = \{1, 2, \ldots, n\}$ correspond to the atoms of the considered molecule, and there is an edge between two vertices if and only if the corresponding distance is known. The weights d associated to the edges provide the numerical value of the known distances. Instances of the DMDGP must satisfy the following two assumptions, for a given ordering on V:

- $\{1, 2, 3\} \subset V$ must be a clique, and, for each atom $x_i \in V$ with rank $i > 3$, the set E must contain the three edges $(i-1, i)$, $(i-2, i)$ and $(i-3, i)$;
- for each triplet of consecutive atoms x_i, x_{i-1} and x_{i-2}, the triangular inequality on the corresponding distances must hold strictly:

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

When these two assumptions are satisfied, a binary tree of atomic positions can be built and explored for solving the DMDGP (see Figure. 1). In fact, if the positions for the first $i-1$ atoms are already known, then there are only two possible positions for the atom x_i, because of the two assumptions. The binary tree can be simply built by repeating recursively the same procedure on all the

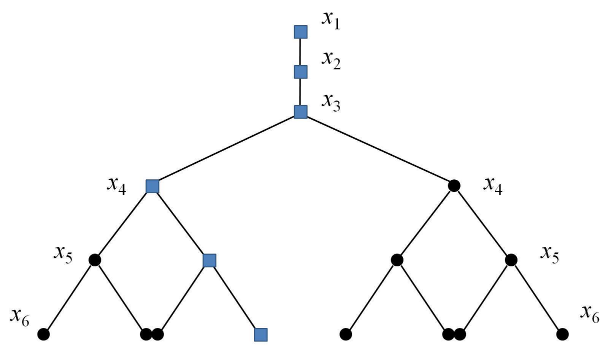

Fig. 1. An example of binary tree for a small molecule with $n = 6$ atoms. The boxes show a complete path on tree, which corresponds to a solution to the DMDGP.

atoms forming the molecule (we provide more details regarding this procedure in Section 2.4). Note that the binary tree has n layers, and all the possible positions for the same atom x_i can be found on the layer i of the tree.

The BP algorithm [12], that we use for solving instances of the DMDGP, is strongly based on the structure of this binary tree. At each iteration of the algorithm, two new positions for the current atom are computed by exploiting the distances that must be known because of the assumptions. However, other distances (which are not required by the assumptions) may also be known, and they can be used for checking the feasibility of the computed atomic positions. Therefore, during the search, branches of the binary tree are pruned as soon as one of its positions are discovered to be infeasible. This pruning phase helps in reducing the binary tree quickly, so that an exhaustive search of the remaining branches is not computationally expensive.

More details on the assumptions of the DMDGP, on the construction of the binary tree and on the BP algorithm can be found in [7,12]. Alg. 1 is a sketch of the BP algorithm. The input parameters for the algorithm are i, the current atom whose positions are searched, n, the total number of atoms forming the molecule, and d, the subset of available distances. The condition $|\,||x_i - x_j|| - d_{ij}\,| < \varepsilon, \forall j < i$, represents the pruning test that we employ for discovering infeasible atomic positions. Since a perfect match on the floating-point arithmetic of a computer machine is impossible, a tolerance ε is used (usually set to 0.001). Note that the algorithm invokes itself recursively for working on the successive atoms of the molecule. The output provided by the BP algorithm is the set of solutions to the DMDGP.

2.2 Input arguments

`MD-jeep` is written in the C programming language. It accepts as input a list of distances on pairs of atoms of a molecule through a text file with a predefined

Algorithm 1 The BP algorithm.

```
0: BP($i$, $n$, $d$)
  for ($k = 1, 2$) do
    compute the $k^{th}$ 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 branch containing $x_i$ is pruned;
    end if
  end for
```

format. In particular, since we mainly work with protein conformations, some additional information related to these molecules can also be specified in the input file. Such additional information are currently not used during the execution of the BP algorithm, but they are included in the output files so that other software can use them, together with the solutions provided by the BP algorithm.

The general format of each single row of the input text file must be:

```
i   j   l   u   i_atom   j_atom   i_amino   j_amino ,
```

where

– `i`	the label of the first atom to which the distance refers;
– `j`	the label of the second atom to which the distance refers;
– `l`	the lower bound on the distance;
– `u`	the upper bound on the distance;
– `i_atom`	the name of the atom `i`;
– `j_atom`	the name of the atom `j`;
– `i_amino`	the name of the amino acid the atom `i` belongs to;
– `j_amino`	the name of the amino acid the atom `j` belongs to.

Note that the names of the amino acids can be expressed in the standard 3-digit code (e.g.: *glycine* is GLY). Naturally, `i_amino` and `j_amino` regard protein conformations only. If there are no amino acids in the molecule, or if the names of the amino acids are unknown, the symbol UNK (*unknown*) can be used.

It is important to note that the BP algorithm is currently able to solve DMDGPs where exact distances d are provided, rather than lower and upper bounds. However, the decision to include in the input text file two values `l` and `u` has been taken in order to guarantee the compatibility of this format with the future versions of the software, when lower and upper bounds will be considered. Currently, only instances in which the lower bound `l` coincides with the upper bound `u` can be solved.

2.3 Instance preprocessing

Once the input text file is read, some checks are performed before invoking the BP algorithm. First of all, we need to verify if the instance in memory is actually a DMDGP. In order to check this, the first assumption of the DMDGP is verified: for each atom x_i, the distances between x_i and the three preceding atoms must be known. Instead, we do not spend computational time for checking the second assumption, for which all the triangular inequalities on the triplets on consecutive atoms must hold strictly. We avoid this check because the probability for this assumption not being satisfied is zero. If the distances contain errors or noise, the (non-strict) triangular inequality can be checked for all the possible triplets of atoms of the molecule. This is a necessary condition for the compatibility of the distances given in input, and can be performed by `MD-jeep` by setting the appropriate option.

2.4 An efficient implementation

The data from the input text file are stored into a predefined data structure, `PROBL`, where each distance is represented by all the information provided on the generic row of the input file. As a consequence, an array of n elements of this data structure represents an entire instance of the DMDGP.

Let us suppose that the distance between the two atoms `i` and `j` is needed sometimes during the execution of the BP algorithm. In order to find information on the distance, it is necessary to scan the array `PROBL` until the corresponding distance is found (if it is actually included in the considered instance). To avoid scanning this array every time a distance is needed, we use a matrix of pointers which is able to provide the location in `PROBL` of the needed distance by using the two labels `i` and `j`. Of course, in this way, a bi-dimensional array of n^2 integers needs to be defined, but it is worth using this memory for speeding the algorithm up.

Before invoking the BP algorithm, the angles θ among consecutive triplets of atoms are computed, as well as the cosine of each torsion angle ω that is defined by each quadruplet of consecutive atoms (details about these computations are given in [7]). Each $\cos(\omega)$ implies the definition of two possible values for ω, which in turn implies two possible atomic positions for the corresponding atom. At each iteration of the algorithm, the two atomic positions are computed as follows. The matrix:

$$B'_i = \begin{bmatrix} -\cos\theta_{i-2,i} & -\sin\theta_{i-2,i} & 0 & -d_{i-1,i}\cos\theta_{i-2,i} \\ \sin\theta_{i-2,i}\cos\omega_{i-3,i} & -\cos\theta_{i-2,i}\cos\omega_{i-3,i} & -\sin\omega_{i-3,i} & d_{i-1,i}\sin\theta_{i-2,i}\cos\omega_{i-3,i} \\ \sin\theta_{i-2,i}\sin\omega_{i-3,i} & -\cos\theta_{i-2,i}\sin\omega_{i-3,i} & \cos\omega_{i-3,i} & d_{i-1,i}\sin\theta_{i-2,i}\sin\omega_{i-3,i} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

is considered for computing the first position for x_i, and the matrix:

$$B''_i = \begin{bmatrix} -\cos\theta_{i-2,i} & -\sin\theta_{i-2,i} & 0 & -d_{i-1,i}\cos\theta_{i-2,i} \\ \sin\theta_{i-2,i}\cos\omega_{i-3,i} & -\cos\theta_{i-2,i}\cos\omega_{i-3,i} & \sin\omega_{i-3,i} & d_{i-1,i}\sin\theta_{i-2,i}\cos\omega_{i-3,i} \\ -\sin\theta_{i-2,i}\sin\omega_{i-3,i} & \cos\theta_{i-2,i}\sin\omega_{i-3,i} & -\cos\omega_{i-3,i} & d_{i-1,i}\sin\theta_{i-2,i}\sin\omega_{i-3,i} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

is considered for the second position. Note that the only difference between B'_i and B''_i is the sign of the sine of the torsion angle $\omega_{i-3,i}$. In the following discussion, we will consider the symbol B_i for referring to any of the two matrices, and the symbols B'_i and B''_i when it will be important to discriminate between the two matrices. In order to obtain the two sets of coordinates for x_i, the two matrices are multiplied by all the preceding matrices B_j, $\forall j < i$:

$$Q'_i = B_4 \cdots B_{i-1} B'_i \qquad Q''_i = B_4 \cdots B_{i-1} B''_i \tag{2}$$

and the coordinates for the two positions for x_i are given by:

$$[Q'_i(1,4), Q'_i(2,4), Q'_i(3,4)] \qquad [Q''_i(1,4), Q''_i(2,4), Q''_i(3,4)]\,,$$

where $Q'_i(k,h)$ and $Q'_i(k,h)$ refer to the element (k,h) of the two matrices.

Let us analyze the complexity of this procedure. All the matrices that are needed for computing an atomic position can be associated to the corresponding vertex of the binary tree. For each atomic position, we need to compute the matrix B'_i or the matrix B''_i, and then we need to compute the matrix Q'_i or Q''_i, respectively. In the worst case (in which BP never prunes), we would need memory for representing the full unpruned tree: the memory requirement would be $O(2 \times 2^{n-3})$, where n is the number of considered atoms. This memory requirement is huge for large molecules. Moreover, every time a new atomic position is computed for x_i, the product among $i-3$ matrices needs to be performed. For all the atomic positions belonging to the same layer i of the binary tree, the complexity is $O(i-3)$.

In order to reduce both memory requirement and floating-point operations, we consider the following strategy. The matrices B_i are needed for computing the matrices Q_i, from which the coordinates of the atomic positions can be extracted. However, the matrix Q_i related to the atom x_i can also be computed as:

$$Q_i = Q_{i-1} B_i, \tag{3}$$

where Q_{i-1} is the matrix related to x_{i-1}. Therefore, instead of considering all the matrices B_j, with $j < i$, only the matrix Q_{i-1} can be exploited for calculating Q_i. This brings to two consequences. First, we can avoid to keep in memory all the matrices B_i, because they are never used again after the computation of Q_i. Secondly, by using the expression (3) instead of (2), the calculation of each atomic position, on any layer i of the binary tree, only needs the computation of the product between two 4×4 matrices.

If all the matrices Q_i are kept in memory, the new memory requirement in the worst case is $O(2^{n-3})$, which is still too large. Let us analyze a single iteration of the BP algorithm. Two new matrices Q'_i and Q''_i are computed for identifying the two possible positions for x_i. Both Q'_i and Q''_i depend by the preceding matrix Q_{i-1}, which, after this computation, will never be used again. Until the search is not backtracked on higher layers of the binary tree, the matrix Q_{i-1} keeps the coordinates of the atom x_{i-1}. However, when the search is backtracked, the memory for Q_{i-1} can be released and used for storing the new matrix Q_{i-1}

corresponding to the branch currently being explored. In this way, the memory requirement is decreased to $O(n-3)$.

The only evident flaw of this strategy is that found solutions are lost when the search is backtracked: once a solution is found, the algorithm can continue the exploration of the remaining branches of the binary tree, and the arrays where the matrices Q_i are stored are overwritten. For this reason, we print on text files the solutions as soon as they are found. If only the best solution is required by the user, we allocate memory for storing only another solution, where we keep the best solution ever found during the search, and we print it at the end of the execution.

2.5 Solutions in PDB format

The solutions found by the BP algorithm are printed in text files in PDB format. Details about this format can be found on the web site of the Protein Data Bank (`http://www.rcsb.org/`). The advantage in using this format is that it is compatible with many other software for the management or for the visualization of molecules. In particular, we use RasMol (`http://www.rasmol.org/`) for visualizing the conformations obtained by the BP algorithm.

3 Experiments

`MD-jeep` was compiled by the GNU C compiler v.4.1.2 with the `-O3` flag. We performed the following experiments on an Intel Core 2 CPU 6400 @ 2.13 GHz with 4GB RAM, running Linux.

The instances we consider were generated artificially by employing a commonly used technique [3, 7, 20]. We chose a subset of proteins from the PDB and we extracted the backbone atoms from these molecules, i.e. the sequence of atoms $\mathrm{N}-\mathrm{C}_\alpha-\mathrm{C}$. We computed all the possible distances between pairs of such atoms, and we kept only the distances smaller than 6Å. This is done for simulating distances obtained through experiments of Nuclear Magnetic Resonance (NMR). Actually, in order to simulate real NMR data [9, 10], such distances should be mainly related to hydrogen atoms, and they should be noisy. However, the aim of the presented experiments is only to show how the developed software works. For considering more realistic (and more complex) instances of the DMDGP, the BP algorithm can be adapted as described, as an example, in [9–11, 15, 17].

Table 1 shows some computational experiments on a set of generated instances, which can be downloaded at the same address as the sources of `MD-jeep`. The label given to the each instance is the label of the corresponding downloaded PDB file. In the table, n is the total number of atoms contained into the protein conformation, m is the total number of available distances, #Sol is the number of solutions found by the BP algorithm, *best* LDE is the penalty function value (1) in correspondence with the best found solution, and, finally, the CPU time is given in seconds for each experiment. All the experiments show that this implementation of the BP algorithm is able to find very accurate solutions to the

instance name	n	m	#Sol	*best* LDE	CPU time
`1crn`	138	846	2	5.79e-14	0.00
`1hoe`	222	1259	2	7.26e-14	0.00
`1jk2`	270	1816	8	5.63e-14	0.01
`1a70`	291	1628	2	3.25e-13	0.00
`1fs3`	372	2209	2	1.84e-13	0.01
`1mbn`	459	3200	8	2.08e-10	0.00
`1rgs`	792	4936	8	1.55e-13	0.06
`1m40`	1224	13823	2	2.46e-13	0.03
`1bpm`	1443	9303	2	8.87e-14	0.03
`1n4w`	1610	10920	2	2.58e-13	0.04
`1mqq`	2032	13016	8	5.40e-13	0.09
`1rwh`	2265	14057	2	4.49e-14	0.12
`3b34`	2790	19563	4	4.91e-12	0.15
`2e7z`	2907	27706	2	1.22e-12	0.18

Table 1. Computational experiments on a set of artificially generated instances.

problem (the best LDE is very close to 0 in all the cases), while the computational time is only a small fraction of seconds, even when the largest instances are solved.

By setting the appropriate options, our software can provide the found solutions in PDB format. The results can then be analyzed by using visualization software for molecular conformations. In Figure 2 we show two different representations of the best found solution corresponding to the instance `1mbn`. These two

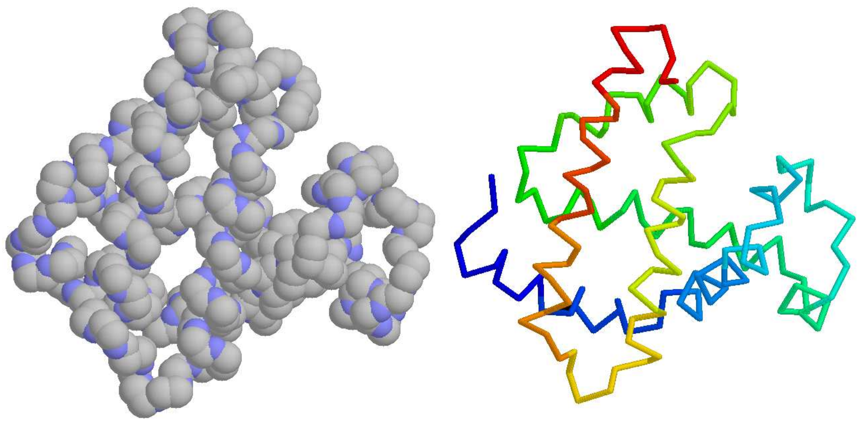

Fig. 2. Two different ways to represent with RasMol one of the solutions obtained by the BP algorithm.

pictures have been created by using the software RasMol (`http://www.rasmol.org/`), which accepts as input a protein conformation in PDB format. In the picture on the left, all the atoms of the molecule are represented by small spheres having different colors. The choice of the colors is made by RasMol by analyzing the additional information that we inserted in our output files (in this specific case, the labels for the atoms). In the picture on the right, only the trace of the protein backbone is represented.

4 New developments

We recently proposed an extention of the DMDGP, to which we refer as the Discretizable Distance Geometry Problem (DDGP) [16]. In the DDGP in $\Re^3$, the assumptions for the discretization are relaxed:

– $\{1, 2, 3\} \subset V$ must be a clique, and, for each atom $x_i \in V$ with rank $i > 3$, there must exist three vertices j, k, h such that

$$j < i,\, k < i,\, h < i, \qquad (j,i), (k,i), (h,i) \in E, \qquad d_{jk} < d_{kh} + d_{hi}.$$

This new assumption allows for discretizing a larger subclass of distance geometry problems, which are not necessarily related to molecular conformations. In [16], a wide discussion on the main differences between the DMDGP and the DDGP is presented. We point out that the DDGP can be also defined in spaces with several dimensions.

Future versions of `MD-jeep` will also solve DDGPs. Even though the DDGP can be seen an extention of the problem that `MD-jeep` is currently able to solve, the extention of `MD-jeep` is not trivial. In particular, the strategy which is implemented for the computation of the atomic positions cannot be used anymore: such a strategy can be employed only when all torsion angles ω are defined by consecutive quadruplets of atoms. As a consequence, we need to use an alternative strategy.

Let us suppose that all the atoms with rank smaller than x_i have been already placed somewhere and that the two possible positions for x_i need to be found. By the new assumption, there are three atoms x_j, x_k and x_h that precede x_i and for which the three distances d_{ji}, d_{ki}, d_{hi} are known. Therefore, three spheres having center in x_j, x_k and x_h and radius d_{ji}, d_{ki} and d_{hi}, respectively, can be defined. Since the triangular inequality $d_{jk} < d_{kh} + d_{hi}$ holds, the intersection of these three spheres can result in two different points, which are the two possible positions for the atom x_i.

The intersection of the three spheres can be computed by solving two linear systems, as explained in [2]. As a consequence, two linear systems need to be solved for finding the coordinates of each atomic position on the binary tree. It is important to note that, differently from the strategy based on the torsion angles, round-off errors can more easily propagate when solving these linear systems. The main reason is that the new assumption for the DDGP does not require the consecutivity among the three preceding atoms used for placing the current atom x_i. Therefore, spheres having different sizes are generally intersected, and this helps the propagation of numerical errors.

In order to keep low the propagation of errors, we plan to implement two main strategies. First, spheres having very different diameters bring to the definition of linear systems where the coefficient matrix is badly-scaled: elements on the rows or on the columns of the matrix can be much larger than the others. Therefore, we need to use a strategy for scaling the coefficient matrix before the solution of the linear system. Secondly, the lost of the consecutivity assumption allows us to choose which distances to use for building the binary tree, and which distances to use for pruning. Since more than three distances between the current atom x_i and the predecessors may be available, there are different possible combinations of distances that can be used for computing the two atomic positions. We plan to develop a strategy for finding out which is the best triplet of distances, i.e. which is the triplet of distances for which the propagation of errors is as low as possible.

5 Conclusions

We presented `MD-jeep`, an implementation of the BP algorithm for solving instances of the DMDGP. We discussed many aspects related to the development of `MD-jeep`, from the input and output formats to the strategies that are considered for reducing the memory requirements and the floating-point operations. The presented software is freely downloadable and usable, and it is distributed under the GNU General Public License (v.2). Future releases of the software will consider more general DMDGPs (for example, considering lower and upper bounds on the distances) and the recently proposed DDGP.

Acknowledgments

The authors wish to thank Sonia Cafieri for the discussions on badly-scaled matrices, and Virginia Costa and Luiz M. Carvalho for their help in the development of the presented software. We would also like to thank the Brazilian research agencies FAPESP and CNPq, the French research agency CNRS and École Polytechnique, for financial support.

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. I.D. Coope, *Reliable Computation of the Points of Intersection of n Spheres in n-space*, ANZIAM Journal **42**, 461-477, 2000.
3. 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.
4. G.M. Crippen and T.F. Havel, *Distance Geometry and Molecular Conformation*, John Wiley & Sons, New York, 1988.

5. T.F. Havel, *Distance Geometry*, D.M. Grant and R.K. Harris (Eds.), Encyclopedia of Nuclear Magnetic Resonance, Wiley, New York, 1701-1710, 1995.
6. M.E. Hodsdon, J.W. Ponder and D.P. Cistola, *The NMR Solution Structure of Intestinal Fatty Acid-binding Protein Complexed with Palmitate: Application of a Novel Distance Geometry Algorithm*, Journal of Molecular Biology **264**, 585-602, 1996.
7. C. Lavor, L. Liberti, and N. Maculan, *Discretizable Molecular Distance Geometry Problem*, Tech. Rep. q-bio.BM/0608012, arXiv, 2006.
8. 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.
9. 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.
10. C. Lavor, A. Mucherino, L. Liberti, and N. Maculan, *Computing Artificial Backbones of Hydrogen Atoms in order to Discover Protein Backbones*, IEEE Conference Proceedings, International Multiconference on Computer Science and Information Technology (IMCSIT09), Workshop on Computational Optimization (WCO09), Mragowo, Poland, 751-756, 2009.
11. C. Lavor, A. Mucherino, L. Liberti, and N. Maculan, *An Artificial Backbone of Hydrogens for Finding the Conformation of Protein Molecules*, Proceedings of the Computational Structural Bioinformatics Workshop (CSBW09), Washington D.C., USA, 152–155, 2009.
12. 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.
13. 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.
14. J.J. Moré and Z. Wu, *Distance Geometry Optimization for Protein Structures*, Journal of Global Optimization **15**, 219–223, 1999.
15. A. Mucherino, C. Lavor, *The Branch and Prune Algorithm for the Molecular Distance Geometry Problem with Inexact Distances*, Proceedings of World Academy of Science, Engineering and Technology (WASET), International Conference on Bioinformatics and Biomedicine (ICBB09), Venice, Italy, 349–353, 2009.
16. A. Mucherino, C. Lavor, L. Liberti, *The Discretizable Distance Geometry Problem*, Optimization Letters, in revision.
17. 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.
18. 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.
19. C.D. Schwieters, J.J. Kuszewski, G.M. Clore, *Using Xplor-NIH for NMR Molecular Structure Determination*, Progress in Nuclear Magnetic Resonance Spectroscopy **48**, 47–62, 2006.
20. D. Wu and Z. Wu, *An Updated Geometric Build-Up Algorithm for Solving the Molecular Distance Geometry Problem with Sparse Distance Data*, Journal of Global Optimization **37**, 661–673, 2007.