Chapter 9
Solving the Discretizable Molecular Distance Geometry Problem by Multiple Realization Trees

Pedro Nucci, Loana Tito Nogueira, and Carlile Lavor

Abstract The discretizable molecular distance geometry problem (DMDGP) is a subclass of the MDGP, where instances can be solved using a discrete algorithm called branch-and-prune (BP). We present an initial study showing that the BP algorithm can be used differently from its original form, where the initial atoms are fixed and the branches of the BP tree are generated until the last atom is reached. Particularly, we show that the use of multiple BP trees may explore the search space faster than the original BP.

Keywords Distance geometry • Branch-and-prune • Realization tree

9.1 Introduction

The molecular distance geometry problem (MDGP) basically consists of obtaining all feasible three-dimensional structures for a molecule when some of its interatomic distances are given [2,4,6,8]. For the case when all interatomic distances are known, the problem can be solved in linear time [3]. Otherwise, the problem is classified as NP-hard [11].

P. Nucci (✉)
Navy Arsenal of Rio de Janeiro, Brazilian Navy, Brazil
e-mail: pedro.nucci@amrj.mar.mil.br

L.T. Nogueira
Instituto de Computação, Universidade Federal Fluminense, Rio de Janeiro, Brazil
e-mail: loana@ic.uff.br

C. Lavor
IMECC-UNICAMP, Campinas-SP, Brazil
e-mail: clavor@ime.unicamp.br

A. Mucherino et al. (eds.), *Distance Geometry: Theory, Methods, and Applications*,
DOI 10.1007/978-1-4614-5128-0_9,

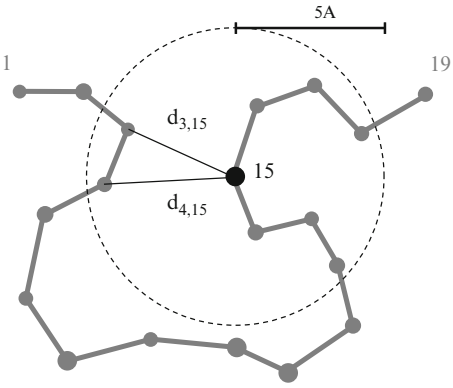

Fig. 9.1 F-type distances

Formally, the MDGP can be described as follows. Given an atomic sequence $1,2,\ldots,n$, and a set S of all pairs of atoms (i,j) such that the distance d_{ij} is known, find the feasible Cartesian coordinates $x_1,\ldots,x_n \in \mathbb{R}^3$ of the atomic sequence (which can be seen as a linear sequence of bonded atoms) so that

$$\|x_i - x_j\| = d_{ij}, \quad \forall (i,j) \in S. \tag{9.1}$$

By supposing the validity of some properties for the known interatomic distance set (usually compatible with proteins, a very important class of macromolecules), the problem has a discrete search space and is called discretizable molecular distance geometry problem (DMDGP) [5].

The following assumptions turn the MDGP into a combinatorial problem (DMDGP), for a given atomic ordering:

1. d_{ij} is known for all pairs of atoms (i,j), with $1 \leq j-i \leq 3$
2. Angles between vectors $(x_{i+2}-x_{i+1})$ and $(x_{i+1}-x_i)$, where $1 \leq i \leq n-2$, are never a multiple of π.

The set S of pair of atoms with known distances may be partitioned in two subsets: the set E, corresponding to all pairs of atoms (i,j), where $1 \leq j-i \leq 3$, and the set F of all pairs of atoms (i,j), where $j-i>3$ (see Fig. 9.1).

In [7], a branch-and-prune (BP) algorithm has been proposed for solving the DMDGP. In this chapter, we provide an alternative way for solving this problem, making use of the BP algorithm.

The main idea in the BP algorithm is to explore the search space by using torsion matrices of each atom and to eliminate the infeasible positions as soon as possible. The torsion matrix B_i related to the atom i can be calculated as follows:

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

for $i \geq 4$, where $\omega_{i-3,i}$ is a torsion angle, $\theta_{i-2,i}$ is a bond angle, and $d_{i-1,i}$ is a bond length. Matrices B_1, B_2, and B_3 are given by

$$B_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad B_2 = \begin{bmatrix} -1 & 0 & 0 & -d_{1,2} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$B_3 = \begin{bmatrix} -\cos\theta_3 & -\sin\theta_3 & 0 & -d_{2,3}\cos\theta_3 \\ \sin\theta_3 & -\cos\theta_3 & 0 & d_{2,3}\cos\theta_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{9.2}$$

With the product $B_1 B_2 \ldots B_i$, one can easily obtain the positions for the atom i which is consistent with all E-type distances. Each atom $i \geq 4$ has two possible torsion matrices B_i (calculated by using $\sin\omega_{i-3,i} = +\sqrt{(1-\cos^2\omega_{i-3,i})}$) or B'_i (calculated by using $\sin\omega_{i-3,i} = -\sqrt{(1-\cos^2\omega_{i-3,i})}$). The first three atoms have only one torsion matrix, which implies that there are 2^{i-3} feasible coordinates, a priori, for each atom $i > 3$. The BP algorithm behaves like a tree search algorithm (such as depth or breadth-first search): at each level i, each torsion matrix B_i and B'_i is multiplied by the previous matrix product $B_1 \ldots B_{i-1}$, thus providing us with two positions x_i and x'_i (*branching*), and positions that are not consistent (according to a constant error tolerance ε) with the related F-type distances are discarded (*pruning*).

We introduce now some definitions. Let M be a molecule defined by a sequence of n atoms $1, 2, \ldots, n$. An *interval* $[a,b]$ of M is any subsequence $\{a, \ldots, b\}$ of atoms of M, with $1 \leq a \leq b \leq n$. The size of $[a,b]$ is defined by $b-a$. A *realization* $R_{a,b}$ is a function $R_{a,b} : [a,b] \mapsto \mathbb{R}^3$ that associates each atom of an interval to a point in $\mathbb{R}^3$. We say that $R_{a,b}$ is *infeasible* for a given instance of the DMDGP when $\exists (i,j) \in S$ such that $i, j \in [a,b]$ and $d_{ij} \neq ||R_{a,b}(j) - R_{a,b}(i)||$; otherwise, it is *feasible*. $R_{a,b}$ is *complete* if $[a,b] = [1,n]$ otherwise, it is *partial*. The idea of working with partial realizations was previously exploited in [10], for investigating a conjecture on the DMDGP, and in [9], for the development of a parallel version of the BP algorithm, even though no formal definitions were given in the latter.

A *realization tree* $T_{a,b}$ is a rooted tree with two properties:

1. Each level k of $T_{a,b}$ corresponds to one atom of the interval $[a,b]$, given by $atom(k)$.
2. Each node at level k contains a coordinate vector for $atom(k)$ corresponding to the atom of the root node.

We use $\left|\left|T_{a,b}\right|\right|$ to denote the number of nodes in $T_{a,b}$. We also use $T^+_{a,b}$ and $T^-_{a,b}$ to denote, respectively, the tree growth direction from left to right and from right to left. As it can be seen, the BP algorithm (given in Algorithm 2) yields a realization tree containing all nodes visited by the algorithm.

Algorithm 2 The Branch-and-Prune algorithm

```
1: Branch-And-Prune(T, node)
2: j ← Level(n_0)
3: if (j < n) then
4:     Compute torsion matrices B_j and B'_j, by using 9.1
5:     Obtain the accumulative torsion matrix C_{j−1} of Father(node)
6:     C_j ← C_{j−1}B_j;  C'_j ← C_{j−1}B'_j
7:     x_j ← C_j [0,0,0,1]^T; x'_j ← C'_j[0,0,0,1]^T
8:     // test F-type distances related to j
9:     valid ← true; valid' ← true
10:    for ((i, j) ∈ F) do
11:        if ((‖x_j − x_i‖^2 − d^2_{i,j})^2 > ε) then
12:            valid ← false
13:        end if
14:        if ((‖x_j' − x_i‖^2 − d^2_{i,j})^2 > ε) then
15:            valid' ← false
16:        end if
17:    end for
18:    // create node with valid positions
19:    if (valid) then
20:        create node z containing C_j and x_j
21:        mark z as son of node
22:        mark node as father of z
23:        T = T ∪ {z}
24:        BranchAndPrune(T, z)
25:    end if
26:    if (valid') then
27:        create node z' obtaining C_j' and x_j
28:        mark z' as son of node
29:        mark node as father of z'
30:        T = T ∪ {z'}
31:        BranchAndPrune(T, z')
32:    end if
33: else
34:    for (each leaf node of T) do
35:        solution stored in parent nodes from level n to 1, output by back-traversal
36:    end for
37: end if
```

In the next sections, we will use more than one realization tree to solve the DMDGP, showing that this strategy can improve the BP algorithm performance. The rest of this work is structured as follows. In Sect. 9.2, we motivate alternative uses of the BP algorithm through one simple theoretical example. In sects. 9.3 and 9.4, we present a technique for merging realization trees, which is the main contribution of this work. Section 9.5 provides a heuristic method that controls the growth of the realization trees. In Sect. 9.6, we describe a methodology and show some computational experiments where the presented techniques are considered. Section 9.7 provides our conclusions.

9.2 BP May Be Used in Different Ways

One must notice that the BP algorithm may be used to explore the search space by other means than the original procedure where the initial atoms of the sequence are fixed and branching on the tree is performed until the last atom is positioned. We are going to show that the BP algorithm presented in [7] actually provides a framework for solving DMDGP instances in various ways. Despite DMDGP is already proven to be NP-hard [5], it is still important to know how to solve its instances as fast as possible, even if its asymptotic behavior does not change.

Let us first use a simple example as motivation, showing that we can use BP in two ways with different performances for the same DMDGP instance. Let us consider a DMDGP instance where $n = 6$ and $F = \{(2,6)\}$. When we execute the BP as described in [7], the algorithm starts placing atoms 1, 2, and 3. Then, it branches two possible positions for atom 4, and so on, until it reaches the last atom. At this level, it must branch all eight possibilities in order to check the feasibility of each one through the information provided by $d_{2,6}$. Only at the last level the algorithm is able to reject some atomic positions.

However, we could solve the considered instance in the opposite direction along the sequence of atoms (for implementation purposes, without loss of generality, this could also be seen as executing the same BP algorithm for an "inverted" instance, where each atom label i is swapped by $7 - i$). In this alternative approach, BP starts fixing atoms 6, 5, and 4, then branching two positions for atom 3. When atom 2 is reached, four positions are computed, and the known distance $d_{2,6}$ can be tested for each of them. In case a node is pruned, this node will not have any child nodes on the subsequent level. In this way, BP explores fewer nodes if compared to the classical approach. In other words, the knowledge about $d_{2,6}$ allows the second approach to restrict its search space one level before the first approach, thus making the search faster.

The presented example makes it clear that solving an instance with BP by the usual way is not always the best approach—it mainly depends on the F distance set. The same analysis may be applied to the concept of interval, introduced earlier, since this can be seen as DMDGP instance as well. Each interval may be solved separately by the BP algorithm, yielding multiple partial realization trees to be combined later, forming complete realizations. Therefore, it is important to study the different ways of solving the DMDGP instances with BP by dividing instances in intervals and, then, by solving intervals in different directions.

9.3 Merging Two Partial Realizations

In order to solve DMDGP instances by using many intervals, we need to be able to combine solutions associated to each interval. If $R_{a,x}$ and $R_{b,y}$ $(a < b < x < y)$ are two feasible realizations sharing three non-colinear atoms (the existence of these atoms implies that $x - b \geq 3$), then we can combine them in order to obtain a single realization $R_{a,y}$.

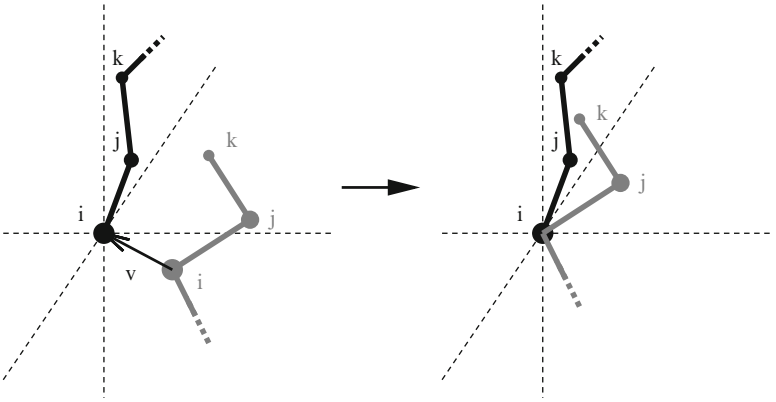

Fig. 9.2 Translation of $R'_{b,y}$ for aligning atom i

Arbitrarily we choose $R_{a,x}$ as a basis for constructing $R_{a,y}$. Thus, both realizations will have the same reference system, and $R_{a,y}$ will inherit all coordinates of $R_{a,x}$, that is, $\forall k \in [a,x]$, $R_{a,y}(k) = R_{a,x}(k)$. In order to complete the coordinate sequence of $R_{a,y}$, we still need to fill the remaining interval $[x+1,y]$, which will be done by applying Euclidean transformations over the coordinates of $R_{b,y}$.

Let i, j, and k be three atoms that belong to both realizations $R_{a,x}$ and $R_{b,y}$. In order to make the coordinates of interval $[x+1,y]$ satisfy all E-type distances, we must align $R_{b,y}$ to $R_{a,x}$, that is, to find $R'_{b,y}$ such that

$$\begin{cases} R_{b,y}(i) = R_{a,x}(i), \\ R_{b,y}(j) = R_{a,x}(j), \\ R_{b,y}(k) = R_{a,x}(k). \end{cases} \tag{9.3}$$

Initially, we consider $R'_{b,y}$ as a copy of $R_{b,y}$. Then, the first equation in Eq. (9.3) is achieved by applying a simple translation over $R'_{b,y}$ (Fig. 9.2), whose translation vector v is given by

$$v = R_{a,x}(i) - R_{b,y}(i).$$

In order to satisfy the second equation in Eq. (9.3), after the translation, we need to apply a rotation around the axis perpendicular to the two vectors connecting the atoms i and j in each realization ($R_{a,x}$ and $R'_{b,y}$) (Fig. 9.3). These vectors are:

$$L_j = R_{a,x}(j) - R_{a,x}(i)$$

and

$$L'_j = R'_{b,y}(j) - R'_{b,y}(i).$$

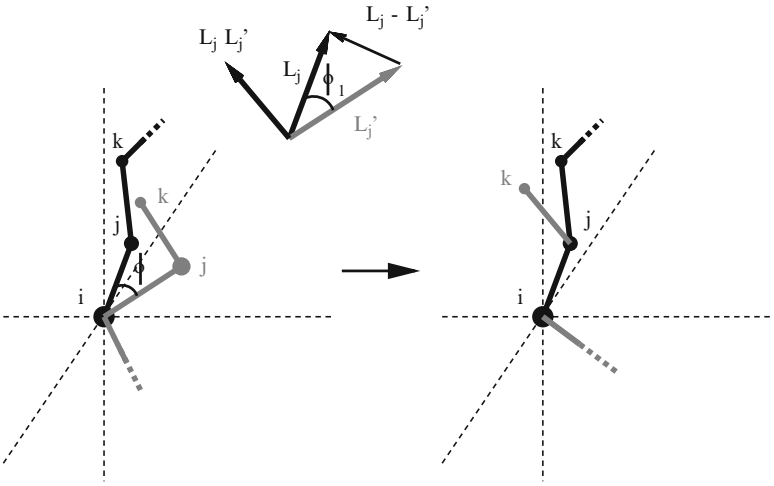

Fig. 9.3 Rotation of R'_{by} for aligning atom j

The rotation axis can be obtained through the cross product $L_j \times L_j^{'}$. The rotation angle is the one between the two vectors, and can be obtained by using the cosine law:

$$\phi_1 = \cos^{-1}\left(\frac{L_j^2 + L_j^{'2} - \left|L_j - L_j^{'}\right|^2}{2L_j L_j^{'}}\right).$$

For aligning the atom k (satisfying the last equation in Eq. (9.3)) we need another rotation. Atoms i and j—already aligned—determine the only possible rotation axis for $R_{b,y}^{'}$ in order to continue satisfying the first two equations. The rotation angle around this axis is calculated by using the two vectors connecting the atoms j and k in each realization, as follows:

$$L_k = R_{a,x}(k) - R_{a,x}(j)$$

and

$$L_k^{'} = R_{b,y}^{'}(k) - R_{b,y}^{'}(j).$$

However, we are not interested anymore in the angle formed by these two vectors, as in the previous case. Now, what matters is the angle between their projections over the perpendicular plane to the rotation axis (Fig. 9.4). For calculating these projections, we use the projection matrix M, oriented by vector L_j:

$$M = L_j L_j^T.$$

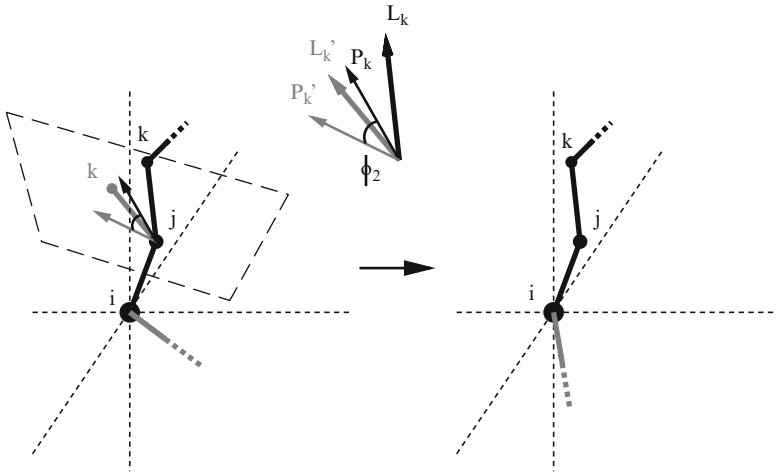

Fig. 9.4 Rotation of R'_{by} for aligning atom k

The projected vectors are given by:

$$P_k = ML_k$$

and

$$P_k^{'} = ML_k^{'},$$

and the angle between them may also be calculated by the cosine law:

$$\phi_2 = \cos^{-1}\left(\frac{P_k^2 + P_k'^2 - \left|P_k - P_k^{'}\right|^2}{2P_kP_k^{'}}\right).$$

9.4 Merging Two Realization Trees

Once we know how to combine two feasible realizations sharing three non-colinear atoms, we can combine two realization trees sharing three atoms, which cannot be colinear by definition of DMDGP. If $T_{a,x}$ and $T_{b,y}$ are two realization trees sharing at least three atoms, the realizations generated by their combination will fill the interval $[a,y]$.

According to the growth direction of the trees, different kinds of merging may occur, described below. Algorithm 3, presented next, provides a way for merging trees $T_{a,x}$ and $T_{b,y}$, so yielding realizations for the interval $[a,y]$, and is applicable to all three kinds of merging.

Algorithm 3 MergeTrees

```
1: MergeTrees(T_{a,x},T_{b,y})
2: create an empty realization list L_{a,y} = {}
3: for (each realization R_{a,x} in T_{a,x}) do
4:     for (each realization R_{b,y} in T_{b,y}) do
5:         align R_{b,y} to R_{a,x} as described in Sect. 9.3
6:         fail = false
7:         for (each (i, j) ∈ F such that a ≤ i ≤ x and b ≤ j ≤ y) do
8:             if (| ||R_{b,y}(j) − R_{a,x}(i)|| − ε| > d_{i,j}) then
9:                 fail = true
10:                exit loop
11:            end if
12:        end for
13:        if (not fail) then
14:            create a realization R_{a,y} such that
15:                R_{a,y}(i) = R_{a,x}(i) if i ∈ [a,x],
16:                R_{a,y}(i) = R_{b,y}(i) otherwise
17:            insert R_{a,y} into L_{a,y}
18:        end if
19:    end for
20: end for
21: return L_{a,y}
```

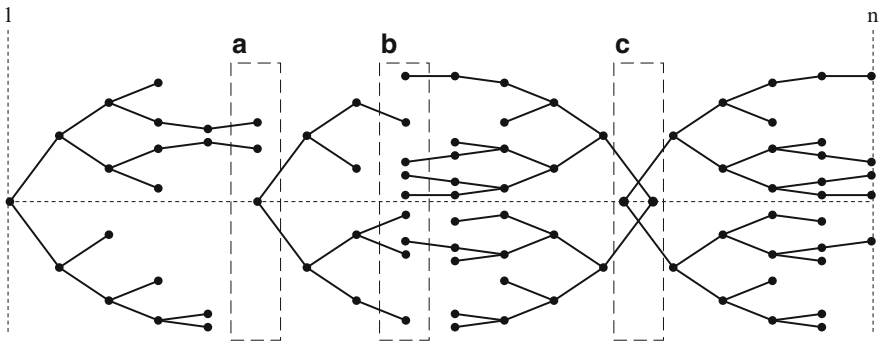

Fig. 9.5 Realization trees merging

Root–Root Merging. When two trees $T^{+}_{a,x}$ and $T^{-}_{b,y}$ grow in opposite direction and overlay their roots, satisfying $x - b \geq 3$, they can be combined from their initialization (Fig. 9.5c).

Leaf–Root Merging. This kind of merging occurs between trees T_1 and T_2 growing in the same direction, when T_1 reaches the atom related to the root of T_2 (Fig. 9.5a). In order to be able to merge them, it is necessary to expand T_1 into two levels, so that we have $T_1 = T_{a,x}$ and $T_2 = T_{b,y}$, $x - b \geq 3$.

Leaf–Leaf Merging. This case happens when two trees grow one in direction of the other (Fig. 9.5b). Let us suppose that at some point, two trees (one negative,

the other positive) reach the same atom i. In order to share three atoms, they need to grow at most two levels more. This can be done in three ways: the positive tree growing two levels, the negative tree growing two levels, or both growing one level. Thus, we will have $T^{-}_{a,x}$ and $T^{+}_{b,y}$, such that $x - b \geq 3$. From the performance point of view, the tree that is about to undergo more prunings should have higher growth priority. Even though two levels may seem to not be significant, it is worth to emphasize that, for big amounts of leafs, growing one level may be very expensive (in the order of the total amount of nodes in the tree until the previous level).

In Algorithm 3, we initially combine each realization in $T_{a,x}$ with each realization in $T_{b,y}$. Clearly, the total amount of combinations is the product of the amount of leaves in each tree. Considering that both trees share exactly three atoms (this condition is enough for merging them), by naming n as $y - a$, the amount of leaves for $T_{a,x}$ and $T_{b,y}$ is $O(2^{x-a})$ and $O(2^{n-(x-a)+3})$, respectively. Thus, the total amount of combined realizations is $O(2^n)$. Then, each combined realization is verified according to the F distance set, whose size is $O(n^2)$. Finally, the complexity of the algorithm for combining $T_{a,x}$ and $T_{b,y}$ is $O(n^2 2^n)$, which is greater than the exponential complexity of the original BP algorithm. In Sect. 9.6, we will see that, according to our computational results, this worst-case analysis does not seem to entail practical significance.

9.5 Growth Control

When we solve one instance by using multiple realization trees $T_1, T_2, \ldots, T_x$, each pair of subsequent trees will undergo one of the three kinds of merging described in Sect. 9.4. In case of pairs (T_i, T_{i+1}) that undergo root–root merging, there is only one way for performing the merging. The same happens in the case of pairs undergoing root–leaf merging, since only the tree that undergoes the merging on its leaves can grow inside the interval delimited by the roots (the tree which undergoes merging on its leaves has to grow until it reaches the root of the other tree). However, in leaf–leaf merging cases, the atom in which the merging will occur is not previously known, and it depends on the growth of both trees.

Aiming at minimizing the algorithm's execution time and the total amount of nodes (of both trees), we may consider the following heuristic: we give growth priority to the tree with fewer leaves. In other words, at each step, we verify which tree has fewer leaves, and we let it grow by one level (changing its amount of leaves for the next step). This procedure is repeated until their leaves reach the same atom. However, this is a greedy method, which does not consider the possibility of allowing the other tree to grow first. For example, it might be more convenient to let a tree grow if it is about to apply a large pruning in a few steps. Algorithm 4 summarizes this approach.

Algorithm 4 GrowthControl

```
GrowthControl(I)
initialize trees T^+_{a,x} and T^-_{b,y}
while (atom(x) ≠ atom(b)) do
    if (T^+_{a,x} has more leaves than T^-_{b,y}) then
        make T^+_{a,x} to grow one level
    else
        make T^-_{b,y} to grow one level
    end if
    grow more 2 levels in T^+_{a,x} or T^-_{b,y} to enable merging
end while
return MergeTrees(T^+_{a,x},T^-_{b,y})
```

9.6 Computational Experiments

In this section, we will consider some artificial instances, automatically generated by computer programs, and real instances, produced from protein structures obtained from the protein data bank (PDB) [1]. PDB is a public database where three-dimensional conformations of proteins and nucleic acids are stored. We have selected only structures generated by NMR. Our goal is to study in which cases the use of multiple realization trees is more efficient than the original BP algorithm. For accomplishing this, we have implemented two methods which use two realization trees in a primitive way (without any previous analysis of the F set for determining in which atom the trees start and in which directions they grow). Then, we have compared both methods to the original BP algorithm, in positive and negative directions. Our analysis did not consider the quality of the solutions which are found, since all methods fully explore the search space of instances, thus reaching the same set of solutions.

All algorithms were implemented in C++, using the standard template library (STL). Experiments were executed on an Intel Core2Duo 2.2GHz, with 2GB RAM.

We introduce a graphical representation which allows us to view how the F set pairs are distributed along the molecule. We do this through the plot of $x \times P(x)$, where x is an atom of the molecule and $P(x)$ is the function expressing the sum of the interval lengths related to F-set pairs whose last atom to be reached by BP is x. Considering $F_x^+ = \{(i,x) \in F\}$ and $F_x^- = \{(x,j) \in F\}$, $P(x)$ is defined, for each direction, by the following formulae:

$$P^+(x) = \sum_{(i,x) \in F_x^+} (x - i)$$

and

$$P^-(x) = \sum_{(x,j) \in F_x^-} (j - x).$$

Figure 9.6 shows the F-set of tested artificial instances, described through an arc representation (each pair $(i, j) \in F$ is represented by an arc that connects atoms i and j), their respective plots of $P^+(x)$ and $P^-(x)$, and the execution time for the following methods (also listed in Table 9.1):

Method 1 One positive tree $T^+_{1,n}$ (original BP), implemented with breadth-first tree search;

Method 2 One positive tree $T^+_{1,n}$ (original BP), implemented with depth-first tree search;

Method 3 One negative tree $T^-_{1,n}$ (original BP), implemented with breadth-first tree search;

Method 4 One negative tree $T^-_{1,n}$ (original BP), implemented with depth-first tree search;

Method 5 Two trees in opposite directions, growing from their extremities towards the center, with growth control (leaf–leaf merging);

Method 6 Two trees in opposite directions, growing from the center towards their extremities (root–root merging).

From the test results with artificial instances, we can observe some facts. The variability of instances has showed that different cases require different approaches, where the direction and the amount of trees play an important role (see Fig. 9.6). The bad performance of methods with two trees for instances (b) and (c) is not due to the growth of trees, but to the merging process, since both trees have many leaves in their merging point.

Method 5, which uses the growth control heuristic for two trees, behaves in a versatile manner, allowing the tree with fewer leaves to traverse a greater part of the molecule, and is not so sensitive to less uniform distributions of F-type distances as the BP algorithm is, thus obtaining good performances for instances such as (f) and (a).

The same methods have been tested in instances produced from real protein data. For this, we created a DMDGP instance from a PDB file that contains a known protein structure, by taking all atomic coordinates from the main chain (protein backbone) for determining those inter-atomic distances that are inside a cut-off radius of 5 Å. Figure 9.7 and Table 9.2 provide further details about the used instances and the execution time of our methods (for instances generated from real data, the arc representation is not clear, due to the big amount of atoms and F-type distances, so it was not used).

In these tests, as in tests with artificial instances, the use of two trees has been efficient in certain cases and has showed some advantages over the original BP. However, in order to justify the use of more than one tree, due to its computational cost, it is necessary that the trees are placed in strategic positions along the molecule (as it happened for method 5 for instance 1SFV), so that F-type distances can be used as soon as possible, causing prunings and having yielding few leaves in the moment of merging. As it has been showed for artificial instances, the cases where molecules have some interval with low amount of F-type distances to be

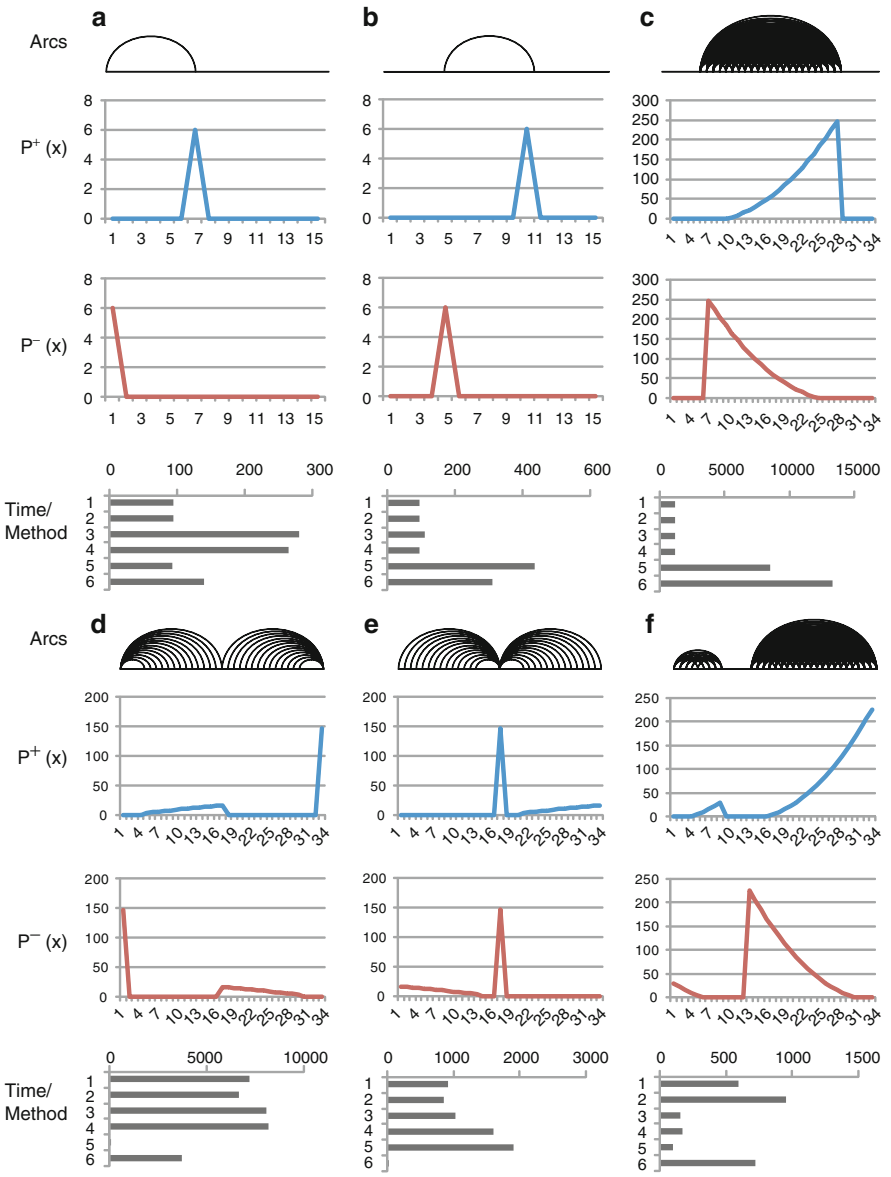

Fig. 9.6 Tests with artificial instances

Table 9.1 Tests with artificial instances

		Execution time of each method (ms)					
Instance	n	1	2	3	4	5	6
a	16	94	94	281	266	93	140
b	16	94	94	109	94	438	312
c	35	1,141	1,171	1,125	1,172	8,485	13,391
d	35	7,203	6,672	8,079	8,203	16	3,719
e	35	922	844	1,031	1,609	1,906	15
f	35	594	953	156	172	93	718

traversed (as instances (1DFN-a) and (1DFN-b)) give greater growth priority for one of the trees, what cannot be foreseen by the original BP. Method 5, whose trees have controlled growth, has showed that it can deal better with this kind of F-set topology.

9.7 Conclusions

We have presented some initial studies that enable solving the DMDGP with multiple realization trees. Through Euclidean transformations as translations and rotations, we discussed about possible ways for combining realizations of distinct intervals that share at least three atoms. We studied the three possible cases for merging realization trees, and, by using this technique, we presented an algorithm that deals with these three cases. This algorithm is what actually allows the use of multiple trees for solving the DMDGP. Moreover, we presented a heuristic for the multiple tree strategy, consisting of regulating the growth of trees that will undergo leaf–leaf merging.

We have made tests with artificial instances and instances generated from real protein data, by comparing the BP algorithm, which produces one realization tree, with two primitive methods using two realizations trees. For both artificial instances and instances generated from proteins, the use of two trees (despite the simplicity of the implemented methods) has showed good performance in comparison to the original BP algorithm. For each instance, depending on the topology of the F set, the methods that use only one tree had a high performance variability according to the direction of growth along the molecule. However, the heuristic method that consider two trees was not so sensitive to the F-set topology, having, most of the time, a performance which is similar to the method of one tree in its most efficient direction (with no need to detect which is the direction).

The results presented here reinforce the interest about studying alternative solving approaches for the DMDGP. Our intention here was twofold: (1) to show that the BP algorithm itself does not assure the best performance, depending on

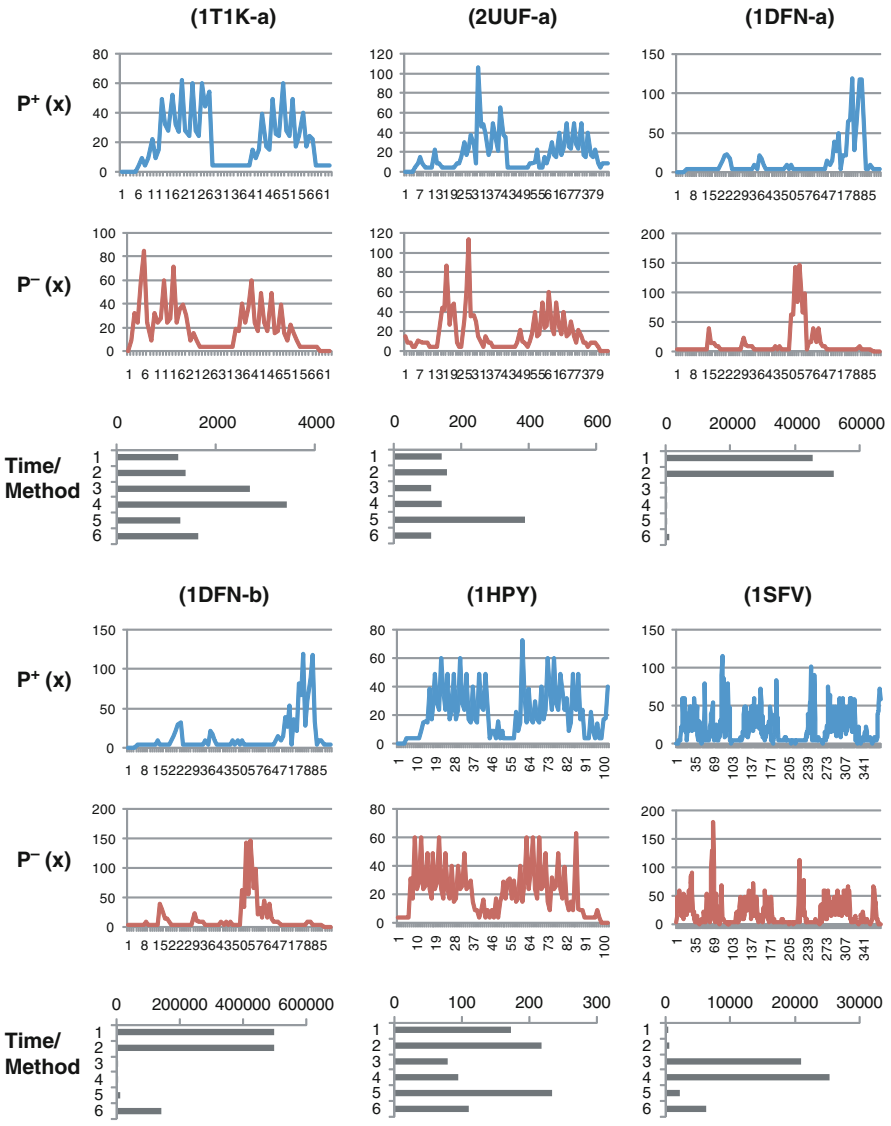

Fig. 9.7 Tests with instances generated from real protein data

the direction of its growth along the molecule and (2) it may be used by more complex strategies, as solving the molecule with multiple realization trees, which provide advantages over the traditional approach. The multiple trees strategy lets us think about performance (not only completeness and correctness) when solving the DMDGP, stimulating investigation of heuristics for the DMDGP.

Table 9.2 Tests with instances generated from real protein data

		Execution time for each method (ms)					
Instance	n	1	2	3	4	5	6
1T1K-a (Insulin)	63	1,250	1,391	2,688	3,437	1,281	1,641
2UUF-a (Thrombin)	84	141	156	109	141	391	109
1DFN-a (Human defensin)	90	45,500	52,031	78	94	328	922
1DFN-b (Human defensin)	90	*	*	969	1,172	8,453	141,672
1HPY (PTH hormone)	102	172	219	78	94	234	110
1SFV (Phospholipase A2)	372	343	453	20,985	25,328	2,109	6,234

* not concluded, due to memory allocation demands

Acknowledgments The authors would like to thank the Brazilian research agencies FAPESP, FAPERJ, and CNPq, for the financial support.

References

1. Berman, H.M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T.N., Weissig, H., Shindyalov, I.N., Bourne, P.E.: The protein data bank. Nucleic Acids Res. **28**, 235–242 (2000)
2. Crippen, G., Havel, T.: Distance Geometry and Molecular Conformation. Wiley, New York (1988)
3. Dong, Q., Wu, Z.: A linear-time algorithm for solving the molecular distance geometry problem with exact interatomic distances. J. Global Optim. **22**, 365–375 (2002)
4. Lavor, C., Liberti, L., Maculan, N.: Molecular distance geometry problem. Encyclopedia of Optimization, pp. 2305–2311, 2nd edn. Springer, New York (2009)
5. Lavor, C., Liberti, L., Maculan, N., Mucherino, A.: The discretizable molecular distance geometry problem. Comput. Optim. Appl. **52**, 115–146 (2012)
6. Lavor, C., Liberti, L., Maculan, N., Mucherino, A.: Recent advances on the discretizable molecular distance geometry problem. Eur. J. Oper. Res. **219**, 698–706 (2012)
7. Liberti, L., Lavor, C., Maculan, N.: A branch-and-prune algorithm for the molecular distance geometry problem. Int. Trans. Oper. Res. **15**, 1–17 (2008)
8. Liberti, L., Lavor, C., Mucherino, A., Maculan, N.: Molecular distance geometry methods: from continuous to discrete. Int. Trans. Oper. Res. **18**, 33–51 (2010)
9. Mucherino, A., Lavor, C., Liberti, L., Talbi, E.-G.: A parallel version of the branch and prune algorithm for the molecular distance geometry problem. In: IEEE Conference Proceedings, ACS/IEEE International Conference on Computer Systems and Applications (AICCSA10), pp. 1–6. Hammamet, Tunisia (2010)
10. Nucci, P.: Heuristicas para o problema molecular de geometria de distancias aplicado a proteinas. Undergraduate Dissertation, Fluminense Federal University
11. Saxe, J.B.: Embeddability of weighted graphs in *k*-space is strongly NP-hard. In: Proceedings of 17th Allerton Conference in Communications, Control and Computing, pp. 480–489. Monticello, IL (1979)