

MDjeep

MDjeep is a software tool for Discretizable Distance Geometry (version 0.3.2).

Copyright (C) 2024, A. Mucherino, D.S. Goncalves, C. Lavor, L. Liberti, J-H. Lin, N. Maculan

GNU General Public License v.3 (see below).

Discretizable Distance Geometry consists of a subclass of problems for which the search space can be discretized and reduced to a tree. Given a graph $G = (V, E, d)$, with vertex set V , edge set E indicating whether the distance between two vertices is known or not, and a weight function d providing the numerical values for such distances, an instance of this problem (in dimension 3) falls in the discretizable subclass where there exists a *vertex order* on V such that:

1. the first 3 vertices in the order form a clique with exact distances;
2. for all other vertices with rank $i > 3$, there must exist three reference vertices j_1, j_2 and j_3 , such that:
 - $j_1 < i, j_2 < i, j_3 < i, (j_1, i) \in E, (j_2, i) \in E, (j_3, i) \in E$.

In this version, we suppose that only one of the three distances $d(j_1, i)$, $d(j_2, i)$ and $d(j_3, i)$ can be represented by an interval, while the others are supposed to be exact (ie, its lower and upper bounds are closer than the predefined error tolerance).

Two methods are currently implemented in **MDjeep** for the solution of the instances. The Branch-and-Prune (BP) algorithm is specifically designed to solve instances satisfying the discretization assumptions given above. The Spectral Projected Gradient (SPG) is an algorithm for local optimization, which may be either run alone, or as a refinement step in BP. For more information about these two algorithms, please refer to our list of publications below.

Since version 0.3.0, **MDjeep** is able to solve instances containing both exact and interval distance values. Although initially written for problems arising in the context of structural biology, **MDjeep** is a general solver capable to solve instances from various applications.

Since version 0.3.2, **MDjeep** accepts in input MDfiles (with **mdf** extension). These are text files containing some main specifications for loading the problem instances, and for running the solution methods:

syntax: `./mdjeep [options] mdfile.mdf`

The MDfile is supposed to contain the specifications for a certain number of predefined “fields”. Every field key-word is followed by its name; key-word and name need to be separated by a colon (:). Every value given in MDfiles appears on a single line and needs to respect the following syntax (blank characters and tabs cannot be included in names or values, as they both act as separators):

name [colon] value

After the definition of a field's name, every new line starting with the key-word "with" allows the user to set up one of the attributes of the field. Some attributes may have default values, so that it is not strictly necessary to specify them in the MDfile; other attributes are mandatory and their absence in the MDfile will cause the termination of MDjeep with code 1.

In the MDfile, the first mandatory field is "instance". Any string of characters (not including blanks and tabs) is a valid name for the instance. The attributes "file", "format" and "separator" need to be specified in the MDfile in the subsequent lines, starting with the key-word "with":

- *file*: it's the path and name of the distance file, where distances are arranged line by line
- *format*: it's the format that MDjeep expects to find for the distance file
- *separator*: this is the character that serves as a separator in the distance file

The format can include the following elements:

- **Id1** (nonnegative integer, mandatory), identifier of vertex 1 (in the line)
- **Id2** (nonnegative integer, mandatory), identifier of vertex 2 (in the line)
- **groupId1** (integer), group identifier of vertex 1
- **groupId2** (integer), group identifier of vertex 2
- **Name1** (char string), the name of vertex 1
- **Name2** (char string), the name of vertex 2
- **groupName1** (char string), the group name of vertex 1
- **groupName2** (char string), the group name of vertex 2
- **lb** (double, mandatory), the lower bound for the distance between vertex 1 and 2
- **ub** (double, mandatory), the upper bound for the distance between vertex 1 and 2

Notice that:

- if the distance file contains additional information that MDjeep does not need to load, the format element **ignore** can be used to skip this information;
- the integer vertex labels need to be consecutive, but the smallest label is not supposed to be equal to 0 (neither to 1); the only constraint for the smallest label is that it needs to be nonnegative;
- the format compatible with MDjeep versions 0.1 and 0.2 is **Id1 Id2 lb ub Name1 Name2 groupName1 groupName2**;
- the format introduced in MDjeep 0.3.0 is **Id1 Id2 groupId1 groupId2 lb ub Name1 Name2 groupName1 groupName2**;
- the separator is one single character, and it needs to be specified between single quotes; blank characters and tabs are always separators, so if not specified, the default separators are all blank characters and tabs.

Another mandatory field of the MDfile is the “method”. Two method names can be specified in the current version of MDjeep: either “bp”, or “spg” (see above). In both cases, a predefined set of attributes can then be specified on the subsequent lines of the MDfile through the key-word “with”. The reader can refer to the examples of MDfile provided with our instances to discover the several attributes that can be set up. Many of such attributes have default values: if not specified in the MDfile, the default value are automatically used. Other attributes are mandatory: when using SPG as a main method, for example, the path and name of the text file containing the starting point (attribute “startpoint”), as well as the maximum number of iterations (attribute “maxit”), both need to be specified. In this version of MDjeep, it is mandatory for the bp method to have a refinement method: this can be specified via the field “refinement”. Since only bp and spg are currently implemented in MDjeep, the only option for bp for a refinement method is currently spg. The key-word “with” can be invoked multiple times for the same attribute in the same MDfile: in such a case, the last specified value is the one that will actually be considered.

Notice that it is possible to include comments in the MDfiles: very line starting with the character # is ignored by MDjeep. Even if not specified as a separator, blank characters and tabs cannot be part of attribute values. They basically work as sort of “general separators”.

MDjeep options (you can access to this list by running MDjeep without arguments):

```

-1 | the specified method stops at the first solution
    (always true for SPG)
-l | specifies after how many solutions the method should stop
    (applies only to BP)
-sym | only one symmetric half of the tree is explored (for BP,
      argument may be 1 or 2)
-p | prints the best found solution in a text file
-P | prints all found solutions (in the same text file)
    | (when using -1, options -p and -P have the same effect)
-f | specifies the output format (default is "xyz", may be changed
    to "pdb")
-consec | verifies whether the consecutivity assumption is satisfied
-nomonitor | does not show the current layer number during the execution
            to improve performance
-r | obsolete, resolution parameter can now be specified in MDfile
    (method field)
-e | obsolete, tolerance epsilon can now be specified in MDfile
    (method field)
-v | obsolete, file formats can now be specified in MDfile
    (instance field)

```

Notice that the use of option -nomonitor can actually improve MDjeep performances; moreover, it is recommended to use it when redirecting stdout to a

file.

Since the current version of **MDjeep**, part of the parameters can be specified through the MDfile, another part through the input arguments. This separation is supposed to keep on one side the parameters that are related to a specific method (the method attributes in the MDfile) and on another side the parameters that are generic (the **MDjeep** input arguments), such as the printing parameters. As new methods will be included in **MDjeep**, this separation may be subject to change: we'll try our best to guarantee the compatibility for future versions of **MDjeep**.

Example of use for solving protein instances with low precision distances (proteinSet2) :

```
mdjeep -1 instances/0.3/proteinSet2/proteins.mdf
```

If **MDjeep** takes too long to solve your instance, you can terminate it with the ^C signal and verify the current partial solution in the output file (it will be created before termination if one of the two options **-p** or **-P** were used).

Recent changes

Version 0.3.2 vs 0.3.1

Box expanding technique

The main novelty in version 0.3.2 is given by the strategy for generating and updating the boxes used in the coarse-grained representation implemented in the BP algorithm to deal with instances containing interval distances. First of all, in the versions 0.3.x (with $x < 2$), the arcs that are used to define the boxes for the current vertex v are computed by using only one possible position for every reference vertex u . As a consequence, when the box is computed (in a way to entirely contain the arc), it cannot be guaranteed that it actually covers the entire portion of space for the vertex v where all reference vertices are satisfied. If another position for some of the reference vertices u is considered, then a “similar” arc can be computed, which however doesn’t “stand” in the same position in space: it actually “moves” wrt the first computed arc.

For this reason, the bound expanding technique (already implemented in a primitive version since **MDjeep** 0.3.0) is applied to every initially computed box as soon as they are created. The initial box is related to the arc where the current positions of the reference vertices u are considered, and where the middle distance of the only reference interval distance is selected. The expansion of a box is stopped only when the newly added positions in the box are not feasible w.r.t. all reference distances. With this bound expanding technique, the boxes are able to cover a larger portion of the space, where the selected positions are freely to move during the refinement step of the BP algorithm. Of course, the box is only a rough approximation of the true portion of space where the positions for a vertex are feasible. For this reason, every time SPG is invoked to

perform the refinement step, and the vertex positions are “moved” inside the boxes with the aim of reducing the overall error on the distances, all involved boxes are subsequently recentered, so that this rough approximation provided by the boxes is more accurate around the currently selected vertex position.

The box centering technique basically consists in creating a new box centered in the new selected vertex position and having the same size of the previous box over the 3 dimensions, and by intersecting it with this previous box. Then, the bound expanding technique is applied again to the result of the intersection to enlarge it until all reference distances can be satisfied.

Revision of DDF and BoxDDF

The DDF and BoxDDF functions have been revised so that they can also output the current partial error. As a consequence, the verification of the constraints in the BP algorithm is now performed after invoking such functions (the verification is not performed anymore directly by these two functions). This modification allowed to implement a new version of `bp_exact` where all possible triplets of discretization vertices may be tested and the one leading to the least error propagation is chosen. When the consecutivity assumption is satisfied, `bp_exact` initially chooses the 3 immediate preceding vertices: in this case, the verification of other triplets is performed only when it is detected that the triplet of immediate preceding vertices forms a flat angle.

Using information about symmetric vertices

In `bp_exact`, the verification on the value of the omega cosine (which can reveal that the angle is actually flat and therefore it is not necessary to branch at this layer of the search tree) is replaced by a new implementation which is mostly based on the symmetries of the search tree. Up to now, the symmetry theory has been fully developed only for DDGP instances satisfying the so-called “consecutivity assumption” (we say that these instances belong to the DMDGP class). Given a vertex v , and for a given selection of its reference vertices u , two new branches rooted at v can be defined: we know that both branches can contain a valid realization only if the vertex v is *symmetric*. Therefore, if during the exploration, in `bp_exact`, a valid realization was already found by exploring the first branch rooted at v , then it is not necessary to explore the second branch, unless it is symmetric (the verification of the symmetries is performed in the main by invoking the function `findSymmetries`). However, for the vertices that are symmetric, and for the instances which do not satisfy the consecutivity assumption, this theoretical result cannot be exploited. In the current version of `MDjeep`, when the information about the symmetries cannot be exploited, we only use the information about cosine of omega, so that to prune the second branch of the vertex v only if this cosine is smaller than a predefined value (0.05) and if the exploration of the first branch had not led to the construction of any valid realizations.

Introducing the MDfile

The **MDjeep** file (MDfile, with extension **mdf**) is introduced in **MDjeep** 0.3.2, which allows us to provide, in one unique text file, the specifications necessary to load a DDGP instance, as well as to select the method we wish to use to solve it, with all its attributes. Apart from the instance name (that is subsequently used by **MDjeep** to make reference to the instance), the text file containing the distance list defining the instance can directly be specified in the MDfile, together with the format for every line of this distance list. This format specification allows us to identify in the distance list the several necessary elements, such as the vertex identifiers, the lower and upper bounds for the distances, and others.

Together with the new parser, several verifications on the input files (both MDfiles and distance lists) have been implemented. For the MDfiles, the precise syntax described in the README file needs to be respected. For the text file containing the distance lists (whose name can be specified in the MDfile), **MDjeep** verifies first of all that every line of the text files contains a list of elements of the same type (distinguishing among integer, real, and alphanumeric elements). It is subsequently verified that the type list in every line of the file is compatible with the format specified in the MDfile (for example, a string of alphabet characters cannot be considered as a valid lower bound for the distances). It is also verified that there are no different lines in the text file making reference to the same pair of vertices (this would imply that we have two different distances for the same pair of vertices). Before invoking the BP method, it is verified whether the discretization assumptions are satisfied (prerequisite for BP). To this aim, **MDjeep** verifies (in the given order) whether: 1. there are enough distances to perform the discretization, 2. there are enough exact distances, 3. the first 3 vertices form a clique of exact distances, 4. the DDGP assumptions are satisfied, 5. there exists at least one triplet of vertices for every vertex that forms a non-flat angle, 6. and (finally, only if it is detected that all distances are exact and precise) it is verified whether the consecutivity assumption is satisfied.

Launching SPG alone

With the idea to implement in the future other distance geometry methods inside **MDjeep** (as a main method or as a refinement method), it is now possible to launch with **MDjeep** the execution of the spectral projected gradient method (SPG), already implemented since **MDjeep** 0.3.0 but used then only as a refinement method. The selected solution method can be specified in the MDfile, together with its list of attributes (many attributes have their own default values, which will be used in case they won't be specified). The attributes startpoint (name of the file containing the starting point) and maxit (maximum number of iterations) are mandatory when using SPG as a main method.

New and old options

A new option (which can be specified through **MDjeep** input arguments) has

been added, which allows to specify the maximum number of solutions that the selected method should find (it currently applies only to BP, as SPG can provide one solution only). This option comes as an alternative to option `-1`, where the number of solutions is limited to 1; with the new option `-l`, the number of solution can be limited to any specified value. The default maximum number of solutions is set to 10 in MDjeep 0.3.2.

The options `-e`, `-r` and `-v` are now obsolete. The values of the tolerance `eps` (option `-e`) and the resolution parameter (option `-r`) can now be set up directly in the MDfile (which is read by MDjeep before verifying the other command line options). As for the option `-v`, the previous file formats used in MDjeep versions 0.1 and 0.2 can now be explicitly specified in the MDfile (the format is: `Id1 Id2 lb ub Name1 Name2 groupName1 groupName2`).

Version 0.3.1 vs 0.3.0

Verification on problem instances

The verification of the discretization assumptions is now performed by using external functions (functions of C file “vertex”), the functions are `initialClique`, `isDDGP` and `isDMDGP`. For MDjeep 0.3.1 to solve the input instance, it is necessary that the function `isDDGP` gives a positive answer; if the result of `isDMDGP` is negative, MDjeep can still solve the instance (this verification is in fact now optional, and performed automatically only when the instance is composed only by exact distances).

The verification of the existence of the symmetric vertices is also now performed by an external function of the “vertex” C file (function `findSymmetries`). The new implemented method has a lower complexity wrt the method implemented directly in the main of version 0.3.0 (old complexity: $|V|^3$, new worst-case complexity: $|V| * |E|$, where V is the instance vertex set, and E is its edge set).

Preselection of reference vertices

The computation of the reference vertices to be used in the BP algorithm is performed only once, and the triplets of reference vertices are kept in memory for the several recursive calls to BP. When more than one reference triplet can be selected, the *optimal* one is searched (basically, when all distances are exact, we avoid to select triplets leading to the definition of angles close to a multiple of π ; whereas if one distance is an interval, we simply take the triplet with two exact distances and the interval with the smallest range).

Resolution parameter

The resolution parameter is now disabled when stepping from a symmetric side to the other of the search tree (when the option `-sym` is not used). In this way, solutions obtained from both symmetric parts of the search tree will be contained in the solution set.

A new pruning device

A new pruning device, named **BoxDDF**, is integrated in **MDjeep** 0.3.1 in order to compute the distances between pairs of boxes “enveloping” pairs of vertices with known distance: if the available distance cannot be satisfied by the two boxes, then it is not necessary to call SPG in the attempt to refine the current solution (because the infeasibility cannot be eliminated as long as the vertex positions are constrained to be in these boxes). This improvement was suggested by Douglas S. Gonçalves.

A separated implementation for precise distances

In order to avoid lowering the performances (w.r.t. the performances that **MDjeep** 0.2 is able to give) when dealing with instances consisting of only exact (and very precise) distances, **MDjeep** 0.3.1 follows two separated paths for the solution of instances containing or not interval distances. The function **bp_exact** was therefore included, which is strongly inspired by the version of **bp** given in **MDjeep** 0.2. We also point out that the resolution parameter is now disabled when the instance at hand only contains exact distances.

Bug report

From the most recent to the oldest.

WARNING: The very last bug fix made **MDjeep** slower when exploring the entire search tree for certain instances.

MDjeep 0.3.2 (current version)

Since the version 0.3.0, **MDjeep** attempts avoiding to generate solutions that are too close to other found solutions. This is regulated by the resolution parameter. However, the comparisons were actually performed too early, i.e. at tree levels where potentially the two compared solutions could still consistently diverge. This bug was discovered by Therese Malliavin and was fixed in the commit “version 0.3.2 patch 2”.

The arclength in `splitOmegaIntervals` was not properly computed. The strategy for expanding the boxes is probably responsible for alleviating the negative impacts of this bug. This bug was fixed in the commit “version 0.3.2 patch 2”.

The method implemented in the main function for the identification of triplets of reference vertices was raising a false warning. This warning is supposed to warn the user about the collinearity of the reference vertices. The bug was discovered by Wagner Rocha and has been fixed in the commit “version 0.3.2 patch”.

The `cosomega` function could have been stuck in an infinite loop in some particular conditions. This bug was discovered by Simon Hengeveld and has been fixed in the commit “version 0.3.2 patch”.

MDjeep 0.3.0

The method implemented for the computation of the boxes was giving wrong results in some particular cases. The bug was probably not detected during the development of MDjeep 0.3.0 because SPG had just been included in order to “correct” a potential error propagation.

The use of the bound expansion feature in SPG was allowing the generation of solutions that were not included in the original set of boxes.

Both bugs have been fixed for the release of MDjeep 0.3.1.

References

If you use and refer to this software in your publications, please cite the appropriate paper(s). Follows a list of main publications:

1. A. Mucherino, J-H. Lin, *An Efficient Exhaustive Search for the Discretizable Distance Geometry Problem with Interval Data*, IEEE Conference Proceedings, Federated Conference on Computer Science and Information Systems (FedCSIS19), Workshop on Computational Optimization (WCO19), Leipzig, Germany, 135-141, 2019. [PDF](#).
2. A. Mucherino, D.S. Gonçalves, L. Liberti, J-H. Lin, C. Lavor, N. Maculan, *MD-jeep: a New Release for Discretizable Distance Geometry Problems with Interval Data*, IEEE Conference Proceedings, Federated Conference on Computer Science and Information Systems (FedCSIS20), Workshop on Computational Optimization (WCO20), Sofia, Bulgaria, 289-294, 2020.
3. A. Mucherino, J-H. Lin, D.S. Gonçalves, *A Coarse-Grained Representation for Discretizable Distance Geometry with Interval Data*, Lecture Notes in Computer Science 11465, Lecture Notes in Bioinformatics series, I. Rojas et al (Eds.), Proceedings of the 7th International Work-Conference on Bioinformatics and Biomedical Engineering (IWBBIO19), Part I, Granada, Spain, 3-13, 2019.
4. D.S. Gonçalves, A. Mucherino, C. Lavor, L. Liberti, *Recent Advances on the Interval Distance Geometry Problem*, Journal of Global Optimization 69(3), 525-545, 2017.
5. D.S. Gonçalves, A. Mucherino, *Discretization Orders and Efficient Computation of Cartesian Coordinates for Distance Geometry*, Optimization Letters 8(7), 2111-2125, 2014.
6. V. Costa, A. Mucherino, C. Lavor, A. Cassioli, L.M. Carvalho, N. Maculan, *Discretization Orders for Protein Side Chains*, Journal of Global Optimization 60(2), 333-349, 2014.
7. L. Liberti, C. Lavor, N. Maculan, A. Mucherino, *Euclidean Distance Geometry and Applications*, SIAM Review 56(1), 3-69, 2014.

8. D.S. Gonçalves, A. Mucherino, C. Lavor, *An Adaptive Branching Scheme for the Branch & Prune Algorithm applied to Distance Geometry*, IEEE Conference Proceedings, Federated Conference on Computer Science and Information Systems (FedCSIS14), Workshop on Computational Optimization (WCO14), Warsaw, Poland, 463-469, 2014.
9. A. Mucherino, C. Lavor, L. Liberti, N. Maculan (Eds.), *Distance Geometry: Theory, Methods and Applications*, 410 pages, Springer, 2013.
10. C. Lavor, L. Liberti, A. Mucherino, *The interval Branch-and-Prune Algorithm for the Discretizable Molecular Distance Geometry Problem with Inexact Distances*, Journal of Global Optimization 56(3), 855-871, 2013.
11. A. Mucherino, *On the Identification of Discretization Orders for Distance Geometry with Intervals*, Lecture Notes in Computer Science 8085, F. Nielsen and F. Barbaresco (Eds.), Proceedings of Geometric Science of Information (GSI13), Paris, France, 231-238, 2013.
12. A. Mucherino, C. Lavor, L. Liberti, *The Discretizable Distance Geometry Problem*, Optimization Letters 6(8), 1671-1686, 2012.
13. A. Mucherino, C. Lavor, L. Liberti, *Exploiting Symmetry Properties of the Discretizable Molecular Distance Geometry Problem*, Journal of Bioinformatics and Computational Biology 10(3), 1242009(1-15), 2012.
14. C. Lavor, L. Liberti, N. Maculan, A. Mucherino, *The Discretizable Molecular Distance Geometry Problem*, Computational Optimization and Applications 52, 115-146, 2012.
15. C. Lavor, L. Liberti, A. Mucherino, *On the Solution of Molecular Distance Geometry Problems with Interval Data*, IEEE Conference Proceedings, International Workshop on Computational Proteomics (IWCP10), International Conference on Bioinformatics & Biomedicine (BIBM10), Hong Kong, 77-82, 2010.
16. A. Mucherino, L. Liberti, C. Lavor, *MD-jeep: an Implementation of a Branch & Prune Algorithm for Distance Geometry Problems*, Lectures Notes in Computer Science 6327, K. Fukuda et al. (Eds.), Proceedings of the 3rd International Congress on Mathematical Software (ICMS10), Kobe, Japan, 186-197, 2010.

The complete list of publications can be found on [this page](#).

Licence

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY

ITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <https://www.gnu.org/licenses/>.
