Deformation Profile Scripts

v.1.0.1, 2010-12-17

Jose Almeida Cruz
Eric Westhof

Universite de Strasbourg
Architecture et Reactivite de l'ARN
Institut de Biologie Moleculaire et Cellulaire du CNRS,
67084 Strasbourg Cedex
France

(Email: j.cruz@ibmc.u-strasbg.fr)

The Deformation Profile Metric is described in detail in:

Parisien M, Cruz JA, Westhof E, Major F. 2009. "New metrics for comparing and assessing discrepancies between RNA 3D structures and models". RNA 15(10): 1875-1885.

1 Introduction

The Deformation Profile (DP) is a metric of discrepancy between two RNA structures of the same molecule. The DP is described in detail in [Parisien 2009].

The Deformation Profile Scripts (DPS) are a set of Python scripts that compute the DP between two PDB files [Berman 2000] and produce a graphical representation of DP in a matrix.

The reference structure against which all comparing models are compared will be referred in this manual as 'reference'. A comparing structure will be referred as 'model'.

All scripts and documentation can be obtained from the authors.

All suggestions, ideas, bug reports, help requests or comments are warmly welcomed by the authors.

2 Requirements and installation

Before running the DPS make sure that you have the following software installed:

- Python 2.5 or later (www.python.org)
- Biopython (biopython.org)
- NumPy (numpy.scipy.org)

Although we only tested the DPS in Linux (Ubuntu), Windows XP Professional and Mac OS X, we believe that it can run seamlessly in any environment supporting the above requirements.

To install and run DPS just go through the following steps:

1. Decompress the file DPS_vX.X.X.tar.gz

```
$ tar -zxvf DPS_vX.X.X.tar.gz
```

2. A directory dps_X.X.X will be created. Enter this directory:

```
$ cd  dps_X.X.X
```

3. Call DPS with Python:

```
$ python dp.py
```

If the DPS is installed and working you should see something like this:

```
-------------------
Deformation Profile
-------------------

Usage:
        dp.py <reference pdb> <comparing pdb>
        dp.py -c <config_file>
        dp.py -o <pdb_file>

For more information see 'dps_manual.pdf'
```

3 Quick start

To quickly start using DPS in "fast mode" just type:

```
$ python dp.py <reference pdb> <model pdb>
```

This command takes two PDB files with the reference and model structures, computes the DP, creates a DP matrix corresponding to the divergence between both models and saves all data in a text file. Check the Output section for more details on those files.

In fast mode the DPS assume that both PDB files correspond to the same structure, thus, both will have chains (or strands) with the same names and roughly the same bases in each chain. Additionally it's assumed that the user wants to compare the first model of each file. Based on these assumptions DPS will perform a simple sequence alignment in each chain to obtain a match between bases in both files. If the PDB files don't have the same chains, or the bases are not in the same order, or we want to compare other models of the PDB files than we must tell DPS how to compare the models using a configuration file. To do that type:

```
$ python dp.py –c <config file>
```

The following sections describe in more detail how to build a configuration file.

If all we want is to quickly inspect the residue contents of a PDB file we can just type:

```
$ python dp.py –o <pdb file>
```

The DPS will display a hierarchical list of models, chains and bases contained in the PDB file.

4 Comparing two similar models

All examples in the following sections refer to files available in the directory "examples" that can be fond under the installation directory.

Let us start with the most simple example, that is, we want to compare two similar (i.e, having the same chains and nucleotides) files 'a.pdb' and 'b.pdb' that sit in our "examples" directory. To do that just go to the directory:

```
$ cd /<installation directory>/examples/ex1
```

And type:

```
$ python ../../dp.py a.pdb b.pdb
```

This command will produce two files:

- b.svg – DP matrix in SVG format
- b.dat – All DP values in text format.

Note that the output files are always named after the second file (the model) in the call.

If we want to customize this job, lets say, change the output directory, change the structure alignments, define a list of models to compare, and so on, we'll have to use a configuration file. Configuration files are text files that tell the program how do we want it to work (in this case, DPS configuration files are small python scripts that set a number of internal variables).

The most simple configuration file can be found in "ex1.cfg". It reproduces the same behavior of the preceding command and only have two lines:

```
ref_model = ("a.pdb", 0)
cmp_model = [("b.pdb", 0)]
```

The first line instructs the DPS to use "a.pdb" to use the 1st model, (model zero), in the file has the reference structure. The second line instructs the DPS to use "b.pdb" has the comparison model. Note the mandatory square brackets in the second line. They are required to allow the definition of a list of models as we'll see above.

To run the DPS with this configuration file, just type:

```
$ python ../../dp.py –c ex1.cfg
```

Not surprisingly the result will be the same as the first command.

5 Comparing many similar models in single run

Imagine that we have a directory with many models that we want to compare with the reference and save the results in a specific output directory. This is the scenario of example 2. To test it just type:

```
$ cd /<installation directory>/examples/ex2
$ python ../../dp.py –c ex2a.cfg
```

With this command the DPS will take three model files located in "ex2/cmp" and compare each of them with the reference file located in "ex2/ref". All output files are saved in the "ex2/out".

The configuration file "ex2a.cfg" is similar to the previous ones with a few minor additions:

1. A new parameter was introduced, "out_dir", that specifies the directory where to save results.
2. All file names contain the directory where they are located.
3. The "cmp_model" variable is now a list with three files.

While this example is fairly straightforward, it is often painful to type all file names like in the example. To overcame this inconvenient one can use a separate file containing a list of model files. Such a list can be easily created in any spreadsheet or text editor (the "file_list.txt" is an example list). The only rules to follow are:

- Each model file should be in a separate line.
- The file name and the model number must be separated by ';'.

To instruct the DPS to use "file_list.txt" instead of individual we must use the option "cmp_list":

```
cmp_list = "./file_list.txt"
```

If you have some python skills, and all model file names follow some regular pattern (as in our example b1, b2, b3, ...), you can replace the hard coded list of names with a few lines of code:

```
cmp_model = []
for i in xrange(1, 4):
    cmp_model += [("./cmp/b%d.pdb" %i, 0)]
```

Remember that the configuration file is plain Python, so any valid Python code is accepted.

6 Comparing non similar models

Very often the DPS cannot automatically figure out how to compare two structures. This may occur for one or more of the following reasons:

- Structures have different chain names.
- The order of the bases is not the same in both files.
- Structures differ in too many bases.
- The sequences to compare are just different.

In these cases issue we have to explicitly tell the DPS how to align the bases of both structures. This can be done with the parameter “aligns” which consists on a list of matching strands in both structures. Each of those strands is defined as:

```
(<ref. chain>, <ref. start base>, <model chain>, <model start base>, <number of bases to consider>)
```

The following example defines two strands. The first one spans from base 1 until 10 of chain “A” in both reference and model structures. The second spans from base 1 until 5 of chain “B” in reference structure and from base 10 until 14 of chain “C” in the model structure:

```
aligns = [("A", 1, "A", 1, 10), ("B", 1, "C", 10, 5)]
```

Graphically, this alignment, could represented like this:

```
Reference:     A:1 A:2 A:3 A:4 A:5 A:6 A:7 A:8 A:9 A:10 B:1  B:2  B:3  B:4  B:5
Model:         A:1 A:2 A:3 A:4 A:5 A:6 A:7 A:8 A:9 A:10 C:10 B:11 B:12 B:13 B:14
```

When defining alignments it's important to consider the following aspects:

- Any number of matching strands is allowed.
- Base numbers must be exactly as defined in the PDB files.
- Gaps in the base numbering must be considered when defining the strand length.
- It is not mandatory to include in the alignment all the bases of both structures. However, bases not included will not be considered in the DP.

- Comparison between bases is done using all atoms with the same name in both bases. This way, when comparing different bases only the backbone and the sugar atoms are guarantee to participate in the comparison.

See example “ex3” to have a simple yet diverse example of how "aligns" parameter works.

7 Drawing a Secondary Structure

To help interpreting the DPS' results is often useful to draw in the DP matrix the different structural domains of the molecule in study. The parameters “helices”, “loops” and “draw” instruct the DPS to draw the desired structural domains in the DP matrix.

To exemplify the use of those parameters consider the following alignment:

```
Reference:     A:1 A:2 A:3 A:4 A:5 A:6 A:7 A:8 A:9 A:10 B:1  B:2  B:3  B:4  B:5  B:6
Model:         A:1 A:2 A:3 A:4 A:5 A:6 A:7 A:8 A:9 A:10 C:10 B:11 B:12 B:13 B:14 B:15

#:             0   1   2   3   4   5   6   7   8   9   10   11   12   13   14   15
SEQUENCE:      C   G   A   U   A   A   U   C   C   C    A    U    U    U    C    A
Sec. Struct:   .   (   (   .   (   (   (   .   .   .    )    )    )    )    )    .
```

This alignment represents a hairpin with a bulging 'U' in the 5' strand of the helix. To draw the square representing this helix in the DP matrix we would have to define the following parameters:

```
aligns = [("A", 1, "A", 1, 10), ("B", 1, "C", 10, 6)]
helices = [("H1", 1, 5, 10, 5)]
loops = [(“Loop X”, 7, 3)]
draw = [“H1”, “Loop X”]
```

The parameters “helices” consists on a list of helices each one defined as:

```
(<helix name>, <5' strand start coord.>, <5' strand length>, <3' strand start coord.>, <3' strand length>)
```

Note that:

- Helix names can be any arbitrary string;
- Strand coordinates correspond to the absolute position of the base in the alignment (starting with 0) and have no relation with base numbers.
- Strand length can have different lengths to allow the inclusion of bulges and internal loops.

In a similar way, “loops” consists on a list of loops each one defined as:

```
(<loop name>, <loop start coord.>, <loop length>)
```

Note that:

- Loop names can be any arbitrary string;
- Loop coordinates correspond to the absolute position of the base in the alignment (starting with 0) and have no relation with base numbers.

The “draw” parameter tells the DPS which helices, loops and inter-domain regions to draw. Excluding a domain name from the "draw" parameter can easily prevent the drawing of a given domain without having to delete it from the "helices" or "loops" lists.

See “ex4a.cfg” for an example on how to use these parameters. In this example the following **fake** secondary structure, with two helices and 5 loops (all single stranded regions in this case), is defined:

```
0         1         1
01234567890123456789
..(.(..))...((..))..
```

Finally, a word about inter-domain comparisons. The DP matrix makes evident any inter domain discrepancy between the reference and the model. For example, two helices that form a coaxial stack in the reference structure but not in the model will display higher values in the DP matrix in the intersection region between them. To draw those regions of interest it suffices to add a new instruction to the “draw” parameter:

```
draw = [“H1”, “H2”, “H1xH2” ]
```

The entry “H1xH2” tells the DPS to draw a square in the all intersections between helical strands of helices H1 and H2 (in this case there are 4, 2x2, of those regions)

See “ex4b.cfg” for an example on how to draw several inter-domain regions.

8 Outputs

By default, the DPS produces two output files:

- <model>.svg – Graphic representation of the DP Matrix in Scalable Vector Graphics format. Most of modern image processing software can manipulate SVG files, my favorite, though, is Inkscape (www.inkscape.org).
- <model>.dat – Contains the values of the DP matrix in a text, easy to parse format .
- To a complete description of the data and how it is computed see [Parisien 2009].

9 Configuration File Reference

This is a complete description of all valid DPS parameters.

```
#
# Deformation Profile (DP) configuration file.
#

# - - - - - - - - - -
# Parameter: 'out_dir' (OPTIONAL)
# Description: Target directory for all resulting files.
#
# Value: STRING
#              STRING - Any valid directory path
#
# If missing or "" (empty) the script will assume as default the directory of the comparing model.

out_dir = "."


# - - - - - - - - - -
# Parameter: 'quiet_out' (OPTIONAL)
# Description: Turns off the output messages.
#
# Value: BOOLEAN
#
# Default value = False.

quiet_out = True


# - - - - - - - - - -
# Parameter: 'quiet_err' (OPTIONAL)
# Description: Turns off the output of error messages.
#
# Value: BOOLEAN
#
# Default value = False.

quiet_err = True


# - - - - - - - - - -
# Parameter: 'ref_model' (MANDATORY)
# Description: PDB file containing the reference structure against which all comparisons will be
#             computed.
#
# Value: (STRING, INT)
#              STRING - Any valid PDB file full path name.
#              INT - Indicates which model to use (see comment).
#
# This parameter must be a tuple (file name, model). As some PDB files contain several models we
# need to specify which one to use. If you don't know or don't care just use a '0'.

ref_model = ("/my_path/x.pdb", 0)
```

```
# - - - - - - - - - -
# Parameter: 'cmp_model' (OPTIONAL, see 'cmp_list')
# Description: List of PDB files containing the structures to be compared with the reference one.
#
# Value: [(STRING, INT), ..., (STRING, INT)]
#               STRING - Any valid PDB file full path name.
#               INT - Indicates which model to use (see comment).
#
# Although optional, either 'cmp_model' or 'cmp_list' MUST be defined.
#
# This parameter is a list of tuples. Each tuple follows the same rules as defined in 'ref_model'
# parameter.

cmp_model = [("/my_path/y1.pdb", 0),
             ("/my_path/y2.pdb", 0),
             ("/my_path/y3.pdb", 0)]


# - - - - - - - - - -
# Parameter: 'cmp_list' (OPTIONAL, see 'cmp_model')
# Description: File containing a list of PDB files to be compared with the reference structure.
#
# Value: STRING
#               STRING - Any valid text file full path name.
#
# Although optional, either 'cmp_list' or 'cmp_model' MUST be defined.
#
# This parameters allows a separated file with the list of comparing structures.
# The file should contain one pdb full file name per line followed by the model (separated by ";").
# If no model is specified 0 will be assumed (see 'model_list.txt' for an example).

cmp_list = "model_list.txt"


# - - - - - - - - - -
# Parameter: 'aligns' (OPTIONAL)
# Description: List of alignment directives.
#
# Value: [(STRING1, INT1, STRING2, INT2, INT3)...(STRING1, INT1, STRING2, INT2, INT3)]
#               STRING1 - Any valid chain name in the reference model.
#               INT1 - First residue to consider in the reference chain.
#               STRING2 - Any valid chain name in the comparing model.
#               INT2 - First residue to consider in the comparing chain.
#               INT3 - Number of bases to consider.
#
# Sometimes the reference and comparing models have different chain or residue names. This parameter
# allows one to easily specify a correspondence between reference and comparing bases.
#
# Bases will be compared using COMMON ATOMS ONLY. This means that, if residue type don't match,
# only part of the atoms (mainly backbone atoms) will be used in the comparison.
#
# If missing or [] (empty) the program will perform a pairwise local sequence alignment between
# chains with the same name. Bases will be matched by their names. We use a slightly modified
# Smith-Waterman local alignment algorithm and only consider the longest non-gaped alignments.

aligns = [("A", 1, "A", 1, 10),
          ("B", 1, "C", 10, 5)]


#
# secondary structure definition
#

#
# NOTE: all bases' coordinates are 0-bases 'alignment' coordinates.
#

# - - - - - - - - - -
# Parameter: 'helices' (OPTIONAL)
# Description: List of double stranded domains (DSD) to be plotted.
#
# Value: [(STRING, INT1, INT2, INT3, INT4)...(STRING, INT1, INT2, INT3, INT4)]
#               STRING - Any unique arbitrary name to identify the DSD.
#               INT1 - First residue of the 5' strand of DSD.
#               INT2 - Number of base of the 5' strand of DSD to consider.
#               INT3 - First residue of the 3' strand of DSD.
#               INT4 - Number of bases of the 3' strand of DSD to consider.
```

```
#
# Lengths of the 5' and 3' strands can be different due to bulged bases.
# We don't care about individual base pairs. We just want to draw squares around DSD regions.

helices = [("H1", 0, 4, 10, 4), ("H2", 5, 2, 10, 2)]


# - - - - - - - - - -
# Parameter: 'loops' (OPTIONAL)
# Description: List of single stranded domains (SSD) to be plotted.
#
# Value: [(STRING, INT1, INT2)...(STRING, INT1, INT2)]
#              STRING - Any unique arbitrary name to identify the SSD.
#              INT1 - First base of the SSD.
#              INT2 - Number of bases to consider.

loops = [("L1", 7, 3)]

# - - - - - - - - - -
# Parameter: 'draw' (OPTIONAL)
# Description: Indicates which DSD, SSD or relation between them should be plotted.
#
# Value: [STRING, ..., STRING]
#              STRING - Each string defines a rectangular region on the plot. The syntax must be:
#                 '<domain_name>[x<domain_name>][:<alt_name>]'
#               - 'domain_name' MUST correspond to a domain defined in 'helices' or 'loops' parameters.
#               - 'alt_name' is an alternative name to be displayed. Can be anything.
#
# Examples:
#       "H1" - draws a square around the intersection of the bases belonging to H1 DSD.
#       "H1:helix I" - same as previous but displays 'helix I' instead of 'H1' (default).
#       "H1xH2" - draws a square around the intersection of the bases belonging to H1 and H2 DSDs.
#       "H1xH2:I x II" - same as previous but displays 'Ix II' instead of 'H1xH2' (default).
#       "L1xH2" - draws a square around the intersection of the bases belonging to L1 SSD and H2 DSDs.

draw = ["H1:x", "H2:y", "L1", "H1xH2"]

# - - - - - - - - - -
# Parameter: 'matrix' (OPTIONAL)
# Description: Tells to program whether to write or not matrix data output.
#
# Value: BOOL - 'True' - Writes data / 'False' - Doesn't write data.
#
# If missing it will write data by default. This can be useful if one only wants to produce
# matrix plots. Either 'matrix' or 'svg' parameters must be True.

matrix=False

# - - - - - - - - - -
# Parameter: 'svg' (OPTIONAL)
# Description: Tells to program whether to plot or not matrix graphics.
#
# Value: BOOL - 'True' - Plots matrix / 'False' - Doesn't plot matrix.
#
# If missing it will plot by default. This can be useful if one only wants to write text data.
# Either 'matrix' or 'svg' parameters must be True.

svg=True
```

10 History

Version: 1.0.1 – 2011-04-01
- BUG: Missing import in module 'dp_2d.py' prevented correct display of error messages.
- CHANGE: Diagram colors changed from white-to-green/yellow-to-red to white-to-yellow/yellow-to-red.
- CHANGE: List of atoms updated
- ADDED: Added 'quiet_err' and 'quiet_out' flags to silence the output.

Version: 1.0.0 – 2009-08-25
- First version

11 Legal

11.1 Disclaimer and Copyright

11.2 Acknowledgments

During the execution of this work Jose Almeida Cruz was supported by the Ph.D. Program in Computational Biology of the Instituto Gulbenkian de Ciencia, Portugal (sponsored by Fundacao Calouste Gulbenkian, Siemens SA, and Fundacao para a Ciencia e Tecnologia; SFRH/BD/33528/2008).

12 References

Berman H.M., Westbrook J., Feng Z., Gilliland G., Bhat T.N., Weissig H., Shindyalov I.N., Bourne P.E. 2000. "The Protein Data Bank". NAR 28: 235-242

Parisien M, Cruz JA, Westhof E, Major F. 2009. “New metrics for comparing and assessing discrepancies between RNA 3D structures and models”. RNA 15(10):1875-1885.

All PDB files used in examples are free adaptation of the PDB file 1A1T.pdb available at (www.rcsb.org/pdb/explore/explore.do?structureId=1a1t).