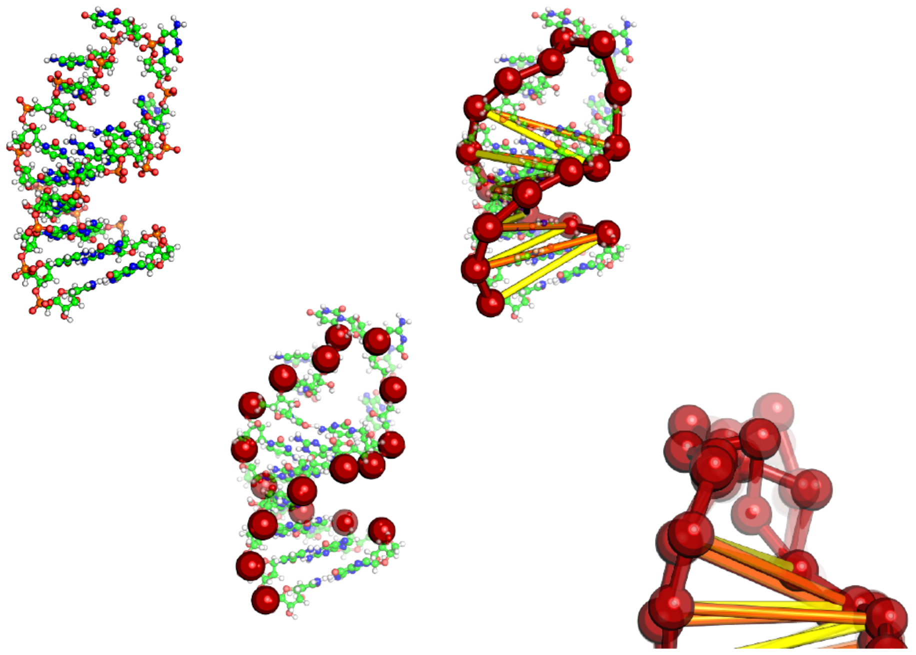
Reduced Molecular Dynamics Streamlined version 1.0 Users' Manual

Filip Leonarski
Joanna Trylska



Biomolecular
Machines
Laboratory

Contents

1 Introduction

RedMDStream is a free open source package for developing coarse–grained (CG) models of biomolecules (nucleic acids, proteins and their complexes) by **automatizing** and **streamlining** the most time consuming steps of the procedure. Since creating a CG model is often a very time consuming task, we have developed this software to help researchers efficiently explore a plethora of ways of designing such models (representation, bonding patterns, potential energy function). The focus of this tool is on **point–bead models**, **not bound to a lattice**, with **a small number of beads (1–4)** representing a residue.

RedMDStream provides routines to perform the following tasks (see also Fig. 1):

- Mapping a full–atomistic structure to a coarse-grained representation based on the user provided topology.
- Setting a network of bonded interactions (pseudo–bond, pseudo–angle, pseudo–dihedral) between beads based on their position in the sequence, spatial coordinates, structural information (2D/3D) and other properties.
- Defining a potential energy function for the bonded interactions defined above and the nonbonded interactions between all the other beads.
- Running molecular dynamics using a built–in RedMD simulation engine.
- Analyzing root mean square deviation, root mean square fluctuations and distance distributions for the resulting trajectories.
- Optimizing parameters of the coarse-grained model using metaheuristic or local search algorithms.

RedMDStream is an extension to the RedMD coarse–grained molecular dynamics software developed in the group of Joanna Trylska group in the years 2007–2014. The RedMD simulation engine is fully integrated into the RedMDStream software. Please note however that some additional tools bundled in the RedMD package (*e.g.*, structure preparation, analysis) are not included in the RedMDStream package to avoid redundancy. If needed please install them from the RedMD package available at the laboratory webpage.

RedMDStream is intended to be extensible. The source code is structured, therefore adding new methods and routines should be straightforward. The RedMDStream source code is written in C++ language, with some parts of the RedMD molecular dynamics engine written also in C language. RedMDStream is parallelized with OpenMP and MPI frameworks (see Sec. 1.4 for details). The code was tested on various Linux platforms. The RedMDStream package is distributed under the terms of the GNU Public License. A copy of the GPL is provided with the RedMDStream distribution and is also available at `http://www.gpl.org`.

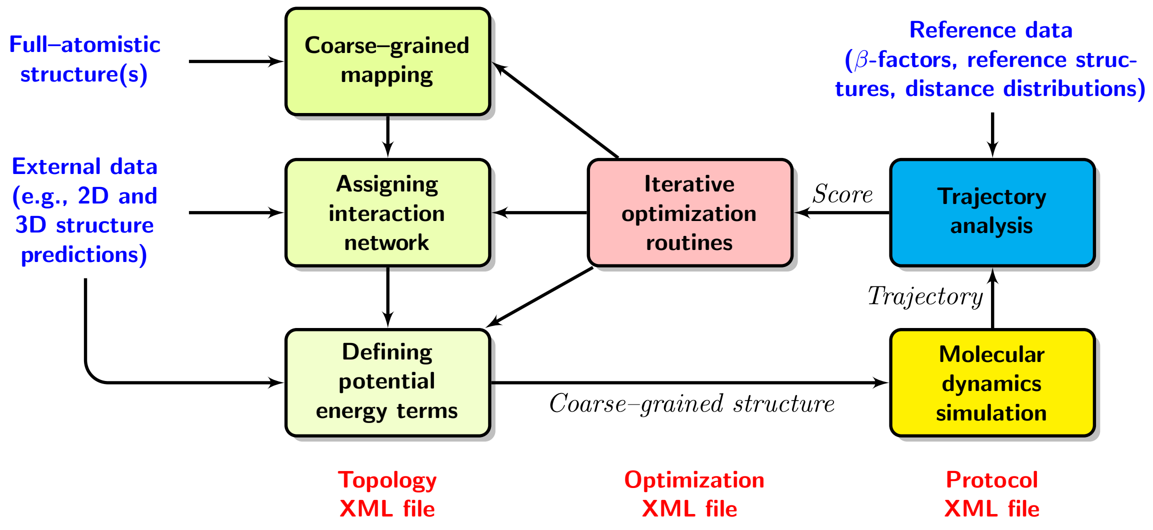

Full–atomistic structure(s)
Coarse–grained mapping
Reference data (β-factors, reference structures, distance distributions)
External data (e.g., 2D and 3D structure predictions)
Assigning interaction network
Iterative optimization routines
Score
Trajectory analysis
Trajectory
Defining potential energy terms
Coarse–grained structure
Molecular dynamics simulation
Topology XML file
Optimization XML file
Protocol XML file

Figure 1: Description of the RedMDStream operational workflow.

1.1 System Requirements

RedMDStream is a C++ Linux/UNIX application, installation on the Windows system should be possible using Cygwin. RedMDStream is available for download as a source code, to allow compilation for a particular system. It therefore requires:

- C/C++ compilers [1],
- certain basic library functions, which are usually supplied by the Linux/UNIX system, however the lack of `getline` function on older Mac OS X systems might unable compilation,
- `libxml2` library with headers for compilation (`libxml2-devel` or `libxml2-dev` package has to be installed on certain Linux distributions).

1.2 Installation

The source code of the RedMDStream package can be found at:

http://bionano.cent.uw.edu.pl/software

After downloading the package one should unpack the archive:

```
tar xzf RedMDStream-src.tar.gz
```

Enter into the RedMDStream sources directory:

```
cd RedMDStream
```

Execute a configuration script and compile the program:

```
./configure
make
make install
```

By default RedMDStream will be compiled with enabled OpenMP parallelization (if available). To compile without OpenMP, one should use the following commands:

```
./configure -disable-openmp
make
make install
```

One can also compile RedMDStream with the MPI parallelization support (turned off by default):

[1] RedMDStream was successfully tested with GNU gcc and g++ (v. 4.4.7 and 4.8.1), Intel C/C++ (v 14.0.1), Portland Group (v. 13.7) and IBM xlc (v. 11.1).

```
./configure --enable-mpi
make
make install
```

MPI and OpenMP parallelizations are mutually exclusive and both cannot be turned on at the same time.

The compiled executable will be stored in the `/usr/local/bin` folder (requires super-user privileges). To use a different location add `-prefix=` option with a directory name to the `configure` script (e.g., `./configure -prefix=/opt/RedMD`). To run RedMDStream simply write:

```
RedMDStream <input file name>
```

If necessary add a directory prefix, depending on the usage of the prefix option in the `configure` script, e.g.:

```
/opt/RedMD/bin/RedMDStream <input file name>
```

If `make` was run without `make install` then the RedMDStream executable can be found in `src` subdirectory of the RedMDStream folder.

To run the MPI version in parallel one should use `mpirun` (or a similar command depending on the MPI flavor), e.g. in MPICH2:

```
mpirun -np <total number of nodes/cores> RedMDStream <input file name>
```

1.3 Usage

If you want to design a new coarse-grained model for proteins or nucleic acids, RedMDStream will help you to find out how numerical parameters of the model, e.g. equilibrium distances or force constants, affect simulation of some test biomolecules, as well as find the best parameters. In this case you run RedMDStream with a optimization XML file (see Sec. 2.4.1):

```
RedMDStream <optimization XML file name>
```

To get practical insight into how RedMDStream optimization works, RedMDStream contains examples described in Sec 3.3.

When the model is done, you can use RedMDStream to simulate other molecules with the same force field. In that case you run protocol XML file (see Sec. 2.3):

```
RedMDStream <protocol XML file name>
```

There are as well examples in Sec. 3.2 that show sample protocol XML files.

1.4 Parallelization

Parallelization in the RedMDStream software is performed on two levels. In the first one multiple cores are assigned the same molecular dynamics simulation, to be simultaneously calculated by all cores. This level can be used in all the RedMDStream tasks that involve an MD simulation. However, since it requires frequent communication and synchronization between cores (every MD simulation step), its efficiency is limited to multi–core machines.

On the other hand the coarse-grained model optimization tasks (see 2.4) in RedMDStream often require performing multiple MD simulations in a single iteration of the optimization algorithm. Therefore, in these cases RedMDStream may assign each core a different MD protocol to process in a serial way. So the communication between cores is performed only after a CG MD protocol is executed, which results in a much lower frequency of communication in comparison to every single simulation step. CG MD protocol based optimization is the second level of parallelization, available in RedMDStream for all optimization modes, excluding simplex. It is more efficient, especially if larger number of cores is available or if multi–node clusters are involved.

The first work sharing level (one MD simulation on multiple cores) is available solely using **OpenMP**. The second level (multiple MD simulations on multiple cores) is also implemented using **OpenMP** and **MPI** for the evolutionary algorithm and particle swarm optimization, which are the best ones for parallelization.

1.5 Units

RedMDStream uses the following system of units:

Length	$\text{Å} = 10^{-10}\,\text{m}$
Mass	$\text{u} = 1.6605402(10) \cdot 10^{-27}\,\text{kg}$
Time	$\text{ps} = 10^{-12}\,\text{sec}$
Charge	$\text{e} = 1.602 \cdot 10^{-19}\,\text{C}$
Temperature	K
Angle	rad or deg °, see below
Energy	kcal/mol
Velocity	Å/ps
Momentum	Å/(ps u)
Force	kcal/(mol Å)

The reported angle and dihedral values are in degrees. However, for the calculations of energies and forces they are converted into radians. Therefore, if one provides parameters for a harmonic potential describing an angle, the equilibrium angle should be provided in degrees, but the force constant should be reported in $\frac{\text{kcal}}{\text{mol rad}^2}$.

1.6 Citing

If you find RedMDStream useful, please cite the following paper:

Filip Leonarski, Joanna Trylska; RedMDStream: streamlining development of residue scale coarse-grained models for proteins and nucleic acids (2014) *In preparation*

1.7 Contact

RedMDStream is a new application and its functionality is still being expanded. Therefore, we encourage every user to send us comments, questions and suggestions. Contact data for the authors are the following:

Filip Leonarski, MSc
fleon@cent.uw.edu.pl or

Joanna Trylska, PhD
joanna@cent.uw.edu.pl

Centre of New Technologies,
University of Warsaw
Banacha 2c
02-089 Warsaw, Poland
phone: +48 22 5543 683

2 Functionality

RedMDStream is a tool aimed to help designing CG potentials for proteins and nucleic acids. We have introduced certain file formats to describe the tasks. In this section we describe these file formats in detail. First, in Sec. 2.1 we introduce file formats to be used with RedMDStream and general rules regarding these files. In Sec. 2.1.1 we describe the framework in which these files are written, i.e., Extensible Markup Language (XML) version 1.0 [5]. XML is s well established format to organize data in a hierarchical way and exchange these data between various computer systems and applications. In Sec. 2.1.3 we introduce rules to build and use mathematical expressions in RedMDStream. Finally, we introduce three XML file formats used by RedMDStream:

- **Topology XML file** (Sec. 2.2)[2] defines a CG model, i.e., tells RedMDStream what type of reduced representation should be used, gives rules how to connect beads and potential energy terms. Based on this file RedMDStream will transform a full–atomistic structure into a CG one for further CG model optimization and simulations.
- **Protocol XML file** (Sec. 2.3) defines all the steps to prepare, execute and analyze CG MD simulation(s).

[2]Topology definition file should not be confused with the topology and structure (SXML) file in RedMD. The first one provides an abstract definition of a force field and the latter one contains a structure of a particular biomolecule, with information about bead positions and bonds.

- **Optimization XML file** (Sec 2.4) tells RedMDStream how to optimize a CG model: which parameters should be tuned, what are the measures to assess the quality of the CG model and how to search for the best solution.

Examples of these files are bundled with RedMDStream and are discussed in Sec. 3. These examples are designed to work as building blocks, and the uses may modify them to create own input files for RedMDStream.

2.1 Operation Modes and Input Files

RedMDStream can be executed in two different modes:

- **Simulation mode** In this mode a single simulation, with a defined coarse–grained model, is prepared, executed and analyzed. To use this mode provide a simulation protocol file as a RedMDStream input (see Section 2.3).

- **Optimization mode** In this mode multiple simulations are executed in a batch mode to find the optimal parameter set for a CG MD model of choice. To use this mode provide an optimization protocol file as a RedMDStream input (see Section 2.4).

2.1.1 XML Format Parsing

RedMDStream protocol and topology files are defined according to the XML 1.0 standard [5]. In an XML file data are organized into tags (each tag opens with the < sign and closes with the > sign). A tag can incorporate other tags or text in a manner shown below:

```
<TAG>
    <INNER_TAG>
       text
    </INNER_TAG>
</TAG>
```

always including an opening and closing of a tag. Missing a closing tag or trying to interlace tags (closing a tag before first closing all the enclosed tags) will result in a syntax error and stop the program execution. If a tag does not include any content it can be written in a shorter version:

```
<TAG />
```

which is equivalent to

```
<TAG></TAG>
```

This way closing of a tag is unnecessary. Tags may also include properties:

```
<TAG property1="value1" property2="value2" />
```

One tag may include only one property with a certain name, but can enclose multiple tags with the same name. Including some properties is obligatory, for example if one writes a tag to open a file, this would be a nonsense operation without providing a file name. If a tag without an obligatory property is found program execution will stop. Other properties are optional. If they are omitted, no error message will appear, but a default value will be assigned to the property. RedMDStream will give an error, if a tag is found that should not be enclosed in other tags, however if one includes a property that is not valid it is in general ignored. Care has to be taken to typographic errors in optional property names, since in such case a default value will be assigned and no error or warning message will be printed.

Everything inside the comment tags is being ignored:

```
<!-- ignored text -->
```

XML comments cannot be nested (one XML comment cannot be placed inside another one). It is allowed to place `--` inside XML comments only in a comment end.

2.1.2 Chemical File Formats

RedMDStream reads biomolecular data saved in the Protein Data Bank (PDB) [3] and Protein Data Bank Markup Language (PDBML)[23] formats. In the case of the PDB format, ver. 3.30[3] is used and the following entries are recognized: ATOM, HETATM, HELIX, SHEET, SSBOND and LINK. In the case of the PDBML format, we use the PDBx/mmCIF dictionary ver. 4.045[4][4] for tag description. The following PBDML tags are recognized: `atom_site`, `ndb_struct_na_base_pair`, `struct_conf`, `struct_conn` and `struct_sheet_range`. RedMDStream also reads the APBS PQR files [2][5] that include also information about the charges and radii of atoms. Please note that PDB is a fixed format file (it does matter in which column data are placed) and PQR is a free format file (only the order of data matters). RedMDStream saves and analyzes molecular dynamics trajectories in the popular CHARMM DCD format. [6]

2.1.3 Equation Parsing

RedMDStream has a built–in equation parser. One may use standard operations in the equation editor ($+ - * / \hat{}$) and mathematical functions (`log log10 exp sin cos tan sinh cosh tanh`).

One may also use the comparison operators[6] :

- `eq` or = Equals
- `neq` or != Not equals
- `lt` Less than
- `gt` Greater than
- `lte` Less than or equals
- `gte` Greater than or equals

[3]http://www.wwpdb.org/documentation/format33/v3.3.html

[4]http://mmcif.wwpdb.org/dictionaries/mmcif_pdbx_v40.dic/Index/

[5]http://www.poissonboltzmann.org/file-formats/biomolecular-structurw/pqr

[6]Since in XML standard lower/greater signs and & have special meaning such operations have to be described using literal acronyms.

Logical expressions and mathematical expressions can be mixed using ? and : operators:

`expr1` ? `expr2` : `expr3`

`expr1` can be a logical expression or mathematical expression. If it is true (logical) or not zero (math), `expr2` is evaluated, otherwise `expr3` is evaluated. Logical expressions can be also combined with each other using `and`, `or` (or |) and `not` (or !) operators.

The numbers are always parsed as real numbers (floating point), there is no integer mathematics available. Since internally RedMDStream uses the radian angle measures, to facilitate using the degree units, one may use the `deg` function, which changes the degree measure to radians. For example `deg(90.0)` will equal to $90°$ or $\frac{\pi}{2}$ in radians. π is available using `pi` literal, so one can also write `pi/2` instead of `deg(90.0)`. One can also access the Boltzmann constant in $\frac{\text{kcal}}{\text{mol K}}$ units by using `kB` (or `KB` and `kb`) literal in the equation mode.

2.1.4 Variables

Equations in RedMDStream can include not only numbers and operators, but also variables. The meaning of variables resembles functional, not imperative programming. If one defines a variable, its definition is stored internally. Only when the actual numerical value is required, it will be calculated according to the given definition. For example, one can define that a parameter for a certain potential connecting two atoms is equal their average mass divided by 5 times their distance in the reference structure. In RedMDStream the definition of this parameter `p` will be stored as:

```
<VAR name="p"> (atom1.mass + atom2.mass) / (5 * bond.dist) </VAR>
```

This tag tells RedMDStream how to calculate the value of `p`. Numerical `p` value is necessary if a structure is prepared for a molecular dynamics simulation. So when a particular bond is created, for designated beads, the value of `p` will be calculated for the bond and atoms and stored in the file. The `p` value will be independently calculated for each bond.

Variables in RedMDStream can be global or context–bound. Global variables can be used everywhere after they are declared. Context–bound variables are connected with a particular object: an atom (real or CG pseudo-atom), residue, bond or potential. Additional variables can be defined in atom type (see sec. 2.2.2) for CG beads or in potential definition (see sec. 2.2.4). Please refer to Tabs. 1 for details.

Variable names are case sensitive. Operator and function names can be typed in both all lower or all upper case. Operator literals, function names, as well as `kB` and `pi` are reserved names and should not be used to call variables. Variable names have to start with a letter (upper or lower case) and should be composed of letters, numbers and underscore `_`.

2.1.5 Interactions

A coarse–grained model should include a global interaction pattern in the molecule. Trying to deduce these patterns from the full–atomistic 3D structure might be a tedious task. Therefore, RedMDStream allows supplementing the structure with some higher–level information. For example, one may use the information about complementary pairing in nucleic acids to build an

Context	Variable	Description	Source
atom (atom1, atom2, ...)	mass	Atom mass[a]	PDB [3], PDBML [23], PQR [2]
	charge	Charge	PDB, PQR
	radius	Atom radius	PQR
	resSeq	Residue number in sequence	PDB, PDBML, PQR
	serialNum	Atom serial number	PDB, PDBML, PQR
	isHelix	0 = not in a helical conformation 1 = in a helical conformation	PDB, PDBML
	helixLength	length of helix	PDB, PDBML
	helixClass	helix class (see PDB spec.)	PDB, PDBML
	isStrand	0 = not in β-strand conformation 1 = is in β-strand conformation	PDB, PDBML
	totalMass	sum of masses of all atoms in the residue (not counting CG beads)	PDB, PDBML, PQR
	totalCharge	sum of charges of all atoms in the residue (not counting CG beads)	PDB, PQR
bond	dist	Bond distance in ref. structure	PDB, PDBML, PQR
	mu	Reduced mass (for bond only)	PDB, PDBML, PQR
	angle	Angle in ref. structure (in radians)	PDB, PDBML, PQR
	dist12	1-2 distance for angle in ref. structure	PDB, PDBML, PQR
	dist23	2-3 distance for angle in ref. structure	PDB, PDBML, PQR
	dih	Dihedral in ref. structure (in radians)	PDB, PDBML, PQR
	dist12	1-2 distance for dih. in ref. structure	PDB, PDBML, PQR
	dist23	2-3 distance for dih. in ref. structure	PDB, PDBML, PQR
	dist34	3-4 distance for dih. in ref. structure	PDB, PDBML, PQR
	interaction name	intensity of interaction (0 = none) (see Tab. 2)	see Tab. 2

[a]Calculated based on the PDB atom type or element, only for atomistic structures

Table 1: Variables used in the `atom` and `bond` contexts, depending on the type of the uploaded file. Please note that the 2D structure (a helix, strand) is based on the information encoded in the PDB/PDBML file. RedMDStream does not provide secondary structure reading software. dih. stands for dihedral and ref. for reference.

Interaction name	Description	Source
WC	Cis Watson-Crick hydrogen bonds in nucleic acids[a]	PDBML [23], RNAView [25], Dot-bracket
WC_AU	A–U Watson-Crick hydrogen bonds in nucleic acids	PDBML, RNAView
WC_CG	C–G Watson-Crick hydrogen bonds in nucleic acids	PDBML, RNAView
NonWC	Noncanonical hydrogen bonds in nucleic acids	PDBML, RNAView
Disulph	Disulphide bridges in proteins	PDB, PDBML
Salt	Salt bridges in proteins	PDBML
Hydrogen	Hydrogen bonds	PDBML
Link	Atom links	PDB

[a]Pair type I according to Leontis-Westhof[16] classification, it includes standard A–U and C–G, but also a wooble pair G–U or U–U pair, when interacting with the Watson–Crick edges in cis conformation. To get only standard A–U and C–G use WC_AU and WC_CG rules combined.

Table 2: Interaction names recogniezd by RedMDStream.

RNA model. This information might be inferred from a full–atomistic structure [25, 9]. It could be also found using a 2D prediction method [17] or experimental methods, like SHAPE. [24]

RedMDStream allows storing two body interactions in a given molecule. For each pair of interacting atoms a number in the range from 0.0 to 1.0 is stored, which describes the probability or intensity of a particular interaction. These interaction data can be fetched from certain file formats recognized by RedMDStream (see Tab. 2 for available names). At the moment only dot–bracket notation allows probabilistic data. For example, one can include multiple predictions of a 2D nucleic acid structure in one file and then create bonds between nucleotides according to a sum of probabilities of predictions that include this pair. Other standard formats use 1.0 if interaction is present and 0.0 otherwise. The interaction data can be also loaded from a text file.

Interactions can be later accessed using a special bond rule (see Sec. 2.2.3) or using a variable that exists in bond context called by interaction name (for example `bond.WC`, `bond.Salt`).

2.2 Topology

A coarse–graining process in RedMDStream is performed in the following way:

- Atoms from high–resolution structure are mapped into a coarse–grained representation based on "atom rules".
- Pseudo–bond[7] network is created based on "bond rules".
- Potential energy function terms are associated with particular bonds.

All the tasks mentioned above are referred in the manual as a "topology".

2.2.1 Topology XML File

The rules how to perform these tasks are encoded in the topology definition XML file. The file has to contain a single `TOPOLOGY` root tag and inside this tag one defines the variables (see Sec. 2.1.3), atom–CG bead mapping rules, CG atom types, pseudo–bond creation rules, and potential energy function definitions, see Fig. 2 for an example. The topology XML file can be also nested inside a protocol XML file. In such case its contest is placed inside a `TOPOLOGY` tag. All the rules below apply also to this situation.

2.2.2 Atom Mapping Rules

RedMDStream provides two ways to create a coarse–grain bead:

1. **Use an existing atom**
   A coarse–grain bead is associated with a particular atom present in the full–atomistic representation, for example, a phosphorus atom in nucleotides or α-carbon in amino acids. In this case the definition is the following:

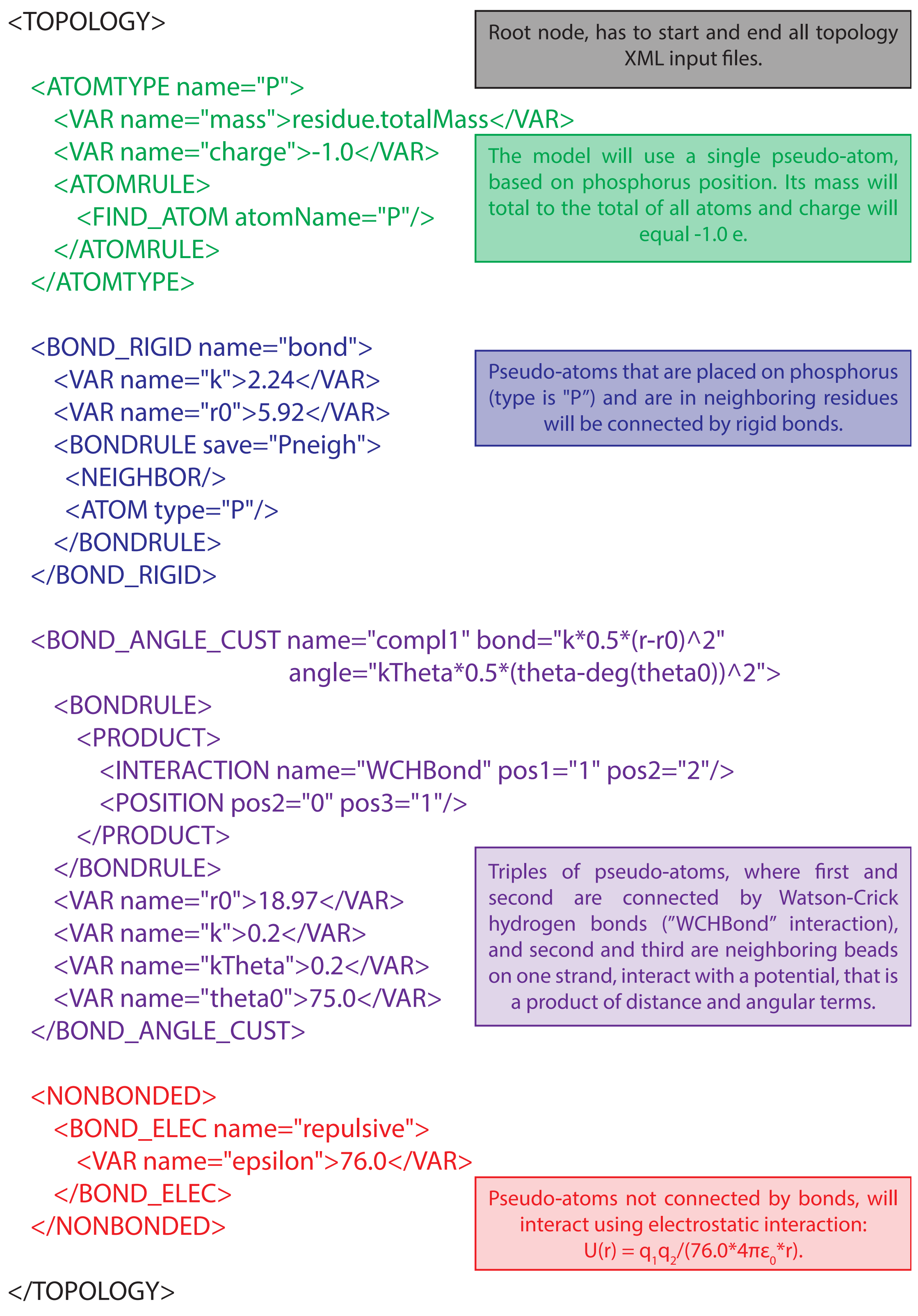
   ```
   <ATOMTYPE name="CGP">
      <VAR name="mass">300</VARIABLE>
      <VAR name="charge">-1.0</VARIABLE>
      <ATOMRULE>
         <FIND_ATOM atomName="P" />
      </ATOMRULE>
   </ATOMTYPE>
   ```

   This will create an atom of type `CGP` with mass of 300 u and charge -1.0 e, placed in the position of an atom named `P` in the PDB/PDBML file.[8]

   One can extend the ATOMRULE section to incorporate more conditions, for example the following code:

[7]Pseudo–bonds in this chapter refer as well to angles and dihedrals between CG beads.

[8]RedMDStream ignores leading and trailing spaces while searching the names in PDB or PDBML file entries. It will therefore not differentiate between "P ", " P " and " P" atom names.



```
<TOPOLOGY>

  <ATOMTYPE name="P">
    <VAR name="mass">residue.totalMass</VAR>
    <VAR name="charge">-1.0</VAR>
    <ATOMRULE>
      <FIND_ATOM atomName="P"/>
    </ATOMRULE>
  </ATOMTYPE>

  <BOND_RIGID name="bond">
    <VAR name="k">2.24</VAR>
    <VAR name="r0">5.92</VAR>
    <BONDRULE save="Pneigh">
     <NEIGHBOR/>
     <ATOM type="P"/>
    </BONDRULE>
  </BOND_RIGID>

  <BOND_ANGLE_CUST name="compl1" bond="k*0.5*(r-r0)^2"
                            angle="kTheta*0.5*(theta-deg(theta0))^2">
    <BONDRULE>
      <PRODUCT>
        <INTERACTION name="WCHBond" pos1="1" pos2="2"/>
        <POSITION pos2="0" pos3="1"/>
      </PRODUCT>
    </BONDRULE>
    <VAR name="r0">18.97</VAR>
    <VAR name="k">0.2</VAR>
    <VAR name="kTheta">0.2</VAR>
    <VAR name="theta0">75.0</VAR>
  </BOND_ANGLE_CUST>

  <NONBONDED>
    <BOND_ELEC name="repulsive">
      <VAR name="epsilon">76.0</VAR>
    </BOND_ELEC>
  </NONBONDED>

</TOPOLOGY>
```

Root node, has to start and end all topology XML input files.

The model will use a single pseudo-atom, based on phosphorus position. Its mass will total to the total of all atoms and charge will equal -1.0 e.

Pseudo-atoms that are placed on phosphorus (type is "P") and are in neighboring residues will be connected by rigid bonds.

Triples of pseudo-atoms, where first and second are connected by Watson-Crick hydrogen bonds ("WCHBond" interaction), and second and third are neighboring beads on one strand, interact with a potential, that is a product of distance and angular terms.

Pseudo-atoms not connected by bonds, will interact using electrostatic interaction: $U(r) = q_1 q_2/(76.0*4\pi\varepsilon_0*r)$.

Figure 2: Sample topology XML file

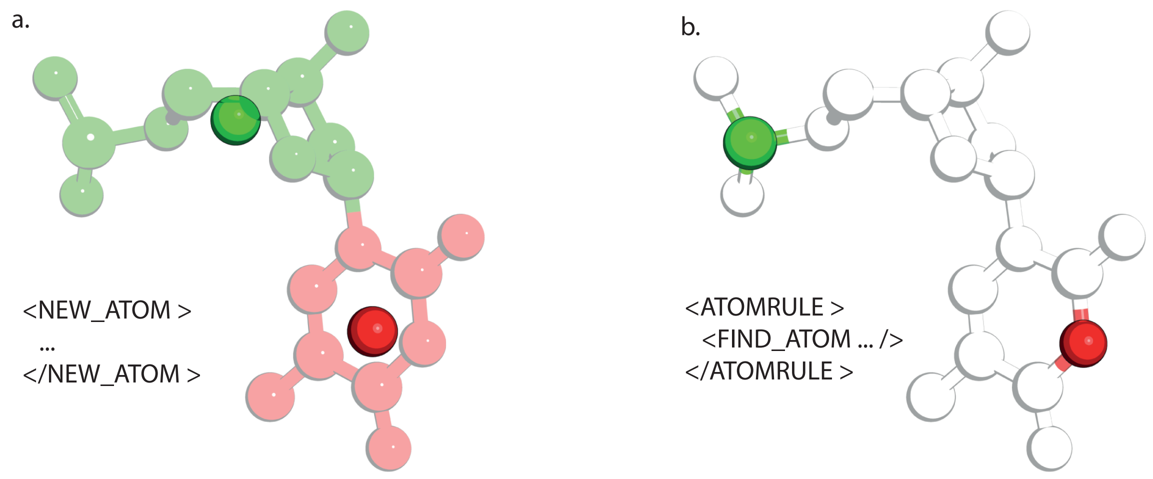


Figure 3: 3D structure of thymine nucleotide is presented in two different coarse–grained representations. Both representations model this nucleotide using 2 beads, one for backbone (green), one for base (red). (a) In the first representation all heavy atoms were categorized as either backbone (light green) or base (light red). Two beads were created, one in a position that is averaged of all atoms in the respective category. To achieve such bead placement one should use `NEW_ATOM` tag. (b) In the second representation phosphorus position was chosen as reference for backbone and N3 nitrogen position was chosen for base. All the other atoms are ignored. For such effect `ATOMRULE` and `FIND_ATOM` should be used.

```
<ATOMTYPE name="CAL">
   ...
   <ATOMRULE>
      <FIND_ATOM atomName="CA" resName="LYS" />
      <EXPR>atom.isHelix</EXPR>
   </ATOMRULE>
</ATOMTYPE>
```

finds all C_α atoms in lysine, which are part of a helix (based on the data in the PDB file). For variables available for use inside the expression rules see Tab. 1 and for rules on how to write equations in Sec. 2.1.3. The `EXPR` rule is satisfied if the enclosed expression is greater than 0.0[9].

Besides the atom and residue name, one can also specify the chain ID or range of residue IDs. This can be helpful in the design of multiscale or Go models. The following topology XML file entry:

```
<ATOMTYPE name="CAL">
   ...
   <ATOMRULE>
      <FIND_ATOM atomName="CA" resName="LYS"
                 chainID="A" minResID="7" maxResID="10" />
```

[9]Threshold of 0.000001 applies, values lower than threshold are considered zero.

```
    </ATOMRULE>
</ATOMTYPE>
```

will create the beads for C_α atoms of lysine but only for the residues in chain A from 7 to 10.

One may also combine the rules using the logical sum (a rule is satisfied, even if only one subrule is satisfied) or product (a rule is satisfied, only if all subrules are satisfied) and introduce negation:

```
<ATOMTYPE name="CAX">
   <VAR name="mass">300</VARIABLE>
   <VAR name="charge">0.0</VARIABLE>
   <ATOMRULE>
      <!-- AND is a default for rules inside ATOMRULE -->
      <AND>
         <OR>
            <FIND_ATOM atomName="CA" resName="LYS" />
            <FIND_ATOM atomName="CA" resName="HIS" />
         </OR>
         <NOT>
            <EXPR>atom.isHelix</EXPR>
         </NOT>
      </AND>
   </ATOMRULE>
</ATOMTYPE>
```

2. **Create a bead based on the average position of many atoms**
In this case a bead is created in a new coordinate position, for example, in the center of mass of a whole residue. This position is calculated for each residue, so at the moment it is not possible to build a bead corresponding to the center of mass of multiple residues (e.g., DNA models with a bead in the middle of a complementary pair cannot be constructed with RedMDStream at the moment). While creating a bead, properties of the atoms constituting it can be aggregated. For example:

```
<ATOMTYPE name="CGP">
   <NEW_ATOM atomName="LYS" centreOfMass="true">
      <RESIDUE name="LYS" />
      <VAR name="mass" aggr="sum" />
      <VAR name="tempFactor" aggr="avg" />
   </NEW_ATOM>
</ATOMTYPE>
```

creates a bead of type `LYS` (residue name = `LYS`), in the center of mass of this residue residue (`centreOfMass="false"` would create a bead in a geometrical center of a residue). This bead has two variables: `mass` – a sum of atom masses in the residue and `tempFactor` – an average of atom temperature factors as read from PDB or PDBXML.

A coarse-grain bead can be also composed only of selected atoms of the residue:

```
<ATOMTYPE name="CGP">
   <NEW_ATOM atomName="PEP" centreOfMass="true">
      <RESIDUE name="PHE" />
      <RESIDUE name="TYR" />
      <ATOM name="C" />
      <ATOM name="N" />
      <ATOM name="O" />
      <VAR name="mass" aggr="sum" />
      <VAR name="tempFactor" aggr="avg" />
   </NEW_ATOM>
</ATOMTYPE>
```

The above creates a bead of type `PEP` for the center of mass of a peptide bond of a phenylalanine or tyrosine (residue name = `PHE` or `TYR`). This bead has two variables: `mass` – a sum of atom masses in the group and `tempFactor` – an average of atom temperature factors. One can specify multiple `RESIDUE` tags, if the same type of bead should be created for different residue types. If multiple `RESIDUE` tags are listed, one can put `ATOM` tags for atoms that occur in one and do not occur in the other one. E.g. if one defined a side chain bead for the phenylalanine and tyrosine, a `ATOM` tag could be placed for oxygen from OH group in tyrosine. It will add the oxygen in tyrosine to the bead, but will not affect phenylalanine in any way (this `ATOM` tag will be ignored). Also for a bead formed based on a DNA nucleobase, one should include atom names for atoms present in guanine, cytosine, adenine and thymine (see examples of topologies for 2- and 3-bead per nucleotide DNA models in Sec. 3.1).

2.2.3 Bond Creation Rules

After specifying the beads and their positioning, the next step in designing the topology is to define the rules for placing the (pseudo-)bonds between the beads in the structure. These rules will be used when defining potential energy terms. Each rule has to have a name. An example rule:

```
<BONDRULE name="Pneigh">
   <NEIGHBOR />
   <ATOM type="P" />
</BONDRULE>
```

includes neighboring P (phosphorus) beads (1–2 for pseudo–bond, 1–3 for pseudo–angle, 1–4 for pseudo–dihedral). The following rule types are allowed inside the `BONDRULE` tag:

1. **Bead type**:

```
<BONDRULE name="R1">
   <ATOM type="P" />
</BONDRULE>
<BONDRULE name="R2">
   <ATOM type1="CA" type2="P" />
</BONDRULE>
```

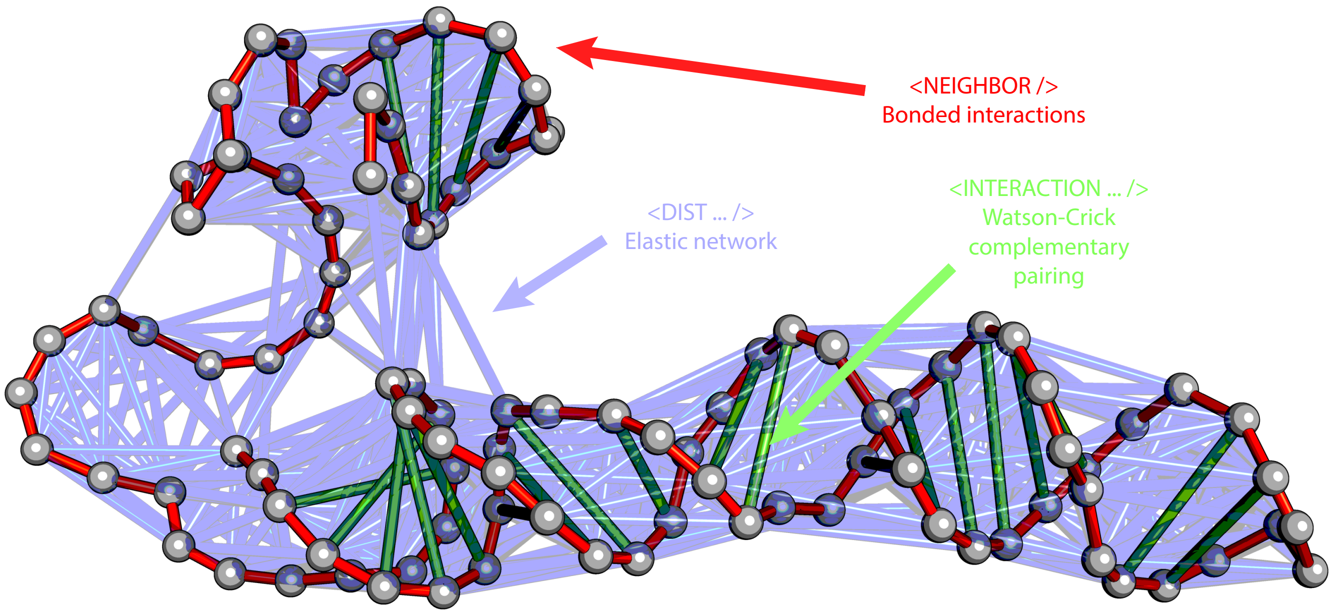


Figure 4: A fragment of the ribosomal RNA in one-bead per nucleotide elastic network model of Trylska et al. [22]. In this model: (a) beads are connected by pseudo-bonds according to their order in the sequence (red), (b) to their secondary structure connectivity (green) and (c) beads close to each other in reference structure are connected by breakable bonds. To achieve this representation in RedMDStream different bond rule XML tag should be used for each of the bonds: (a) `NEIGHBOR`, (b) `INTERACTION` and (c) `DIST`. The order of using the rules is also important because the rules (a) and (b) take precedence over the (c) distance rules.

The rule termed `R1` defines pseudo–bonds that connect beads of type `P`, defined with `ATOMTYPE` tag described in the previous section. The rule tt R2 defines pseudo–bonds that connect the `CA` type beads with the `P` type beads. If only `type1="CA"` is specified, then this rule would describe the interactions of the `CA` type beads with any beads. One can also set `type3` and `type4` for the angle and dihedral related rules.

If a single type is to be assigned to different atoms, more elaborate rules should be used:

```
<BONDRULE name="R3">
   <ATOM atom1="true" atom2="true">
       <FIND_ATOM resName="LYS" />
       <EXPR>atom.helixClass eq 3</EXPR>
   </ATOM>
</BONDRULE>
```

Inside the `ATOM` tag one can put all the tags used in Sec. 2.2.2 to find atoms. Setting switches `atom1`—`atom4` as `true` designates which atoms in the pseudo–bond/angle/dihedral should be checked according to this rule.

2. **Bead distance**:

```
<BONDRULE name="R4">
   <DIST min="0.0" max="10.0" />
</BONDRULE>
```

The rule termed `R4` allows pseudo–bonds in which the distance between the beads is in 0.0 — 10.0 Å range. The rule can be also applied in the context of an angle or dihedral potential energy term. In such case all the distances between atoms (1–2, 2–3, 3–4) have to match the minimum and maximum values. For only selected distances one should use a more general expression rules, described firther. The default `min` value is 0.0 and default `max` value is infinity.

3. **Interaction type**:

```
<BONDRULE name="R5">
   <INTERACTION name="WC" />
</BONDRULE>
```

connects beads interacting by `WC` bonds (Watson–Crick pairing), see Section 2.1.5 and interaction names in Tab. 2. Please remember that certain interaction classes are defined only with external files. Interactions in RedMDStream are only between two beads. For angles and dihedrals, one can use `pos1` and `pos2` attributes for the `INTERACTION` tag to set which beads in the angle or dihedral should be connected with a particular bead–bead interaction. `INTERACTION` can also have a `directional="true"` attribute that distinguishes the directionality. If the interaction file defines the interaction between bead 7 and bead 14, the rule will apply only to a pseudo–bond in which bead 7 is at the first position and bead 14 is at the second position. If `directional="false"` is set or no option is used, both 7–14 and 14–7 bead combinations will succeed.

4. **Position in the chain**:

```
<BONDRULE name="R6">
   <POSITION pos1="0" pos2="1" pos3="1" />
</BONDRULE>
<BONDRULE name="R7">
   <-- shortcut for
       POSITION pos1="0" pos2="1"
                pos3="2" pos4="3" -->
   <NEIGHBOR />
</BONDRULE>
<BONDRULE name="R8">
   <-- shortcut for
       POSITION pos1="0" pos2="0"
                pos3="0" pos4="0" -->
   <RESIDUE />
</BONDRULE>
```

This rule describes the connectivity of beads in the same chain (i.e., with the same chain ID) by taking into account their position in sequence controlled by the residue ID. The rule `R6` allows angles (or dihedrals) in which the second and third beads are from the residue following the first one. The `NEIGHBOR` tag describes the interactions between sequential beads and `RESIDUE` describes the interactions between the beads that belong to the same residue.

5. **Expression rules**:

```
<!-- bonds where first atom belongs
     to a right handed pi helix
     (see PDB specification for helix classes) -->
<BONDRULE name="R9">
   <EXPR> atom1.helixClass eq 3 </EXPR>
</BONDRULE>
<!-- angles where sum of 1-2 and 2-3 distances is
     greater than 10.0 A -->
<BONDRULE name="R10">
   <EXPR> (bond.dist12 + bond.dist23) gt 10.0</EXPR>
</BONDRULE>
<!-- angles bigger than 30.0 degree,
     since bond.angle is set in radians
     one should use deg(30.0) to convert
     degrees to radians -->
<BONDRULE name="R11">
   <EXPR> bond.angle gt deg(30.0) </EXPR>
</BONDRULE>
```

These rules allow more elaborate definitiona for creating bonds, see Sec. 2.1.3 for rules on how to write expressions and Tabs. 1 and 2 for the variable/interaction names. This rule is only satisfied if the enclosed expression is greater than 0.0[10].

6. **Shift other rules**:

```
<BONDRULE name="R12">
   <SHIFT shift1="1" shift2="1">
      <INTERACTION name="WC" />
   </SHIFT>
</BONDRULE>
```

The `SHIFT` rules apply to a particular case if the bonding pattern depends on some properties of neighboring beads. For example, the rule `R12` describes a pseudo–bond, only if the first–neighbor (in the anti–sense direction) of bead one and first–neighbor (in anti–sense direction) of bead two are connected by Watson–Crick canonical bond.

7. **Combine other rules**:

```
<BONDRULE name="R13">
   <!-- AND is default inside a BONDRULE
        and could be omitted in this example-->
   <AND>
      <OR>
         <ATOM type="CA" />
         <ATOM type="P" />
      </OR>
      <NOT>
         <EXPR>atom1.isHelix</EXPR>
      </NOT>
      <NOT>
         <EXPR>atom2.isHelix</EXPR>
      <NOT>
   </AND>
</BONDRULE>
```

One can use `OR`, `AND` and `NOT` tags to combine a rule by using the logical rules. If more than one rule is included inside a `BONDRULE`, `SHIFT` or `NOT` tag, it is implicitly assumed that the `AND` tag was used.

2.2.4 Potential Energy Functions

When bond rules are defined, the next step is to assign these rules to a potential energy function term as in the following example:

```
<BOND_HARM name="compl2">
   <BONDRULE name="compl2" />
   <VAR name="r0">bond.dist</VAR>
   <VAR name="k">5.0</VAR>
```

[10]Threshold of 0.000001 applies, the values lower than threshold are considered zero.

```
</BOND_HARM>
```

Here, a harmonic potential pseudo–bond is created between the beads described by the `compl2` bond rule (the potential and bond rule names can overlap). The potential parameters are defined with an equilibrium distance equal to the distance in a reference structure and a force constant equal to $5 \frac{kcal}{mol\text{Å}^2}$. In the parameter definition one can use the values connected to the bond (`bond.dist`, `bond.angle`, `bond.dihedral`), atoms forming the bond (`atom1.charge`, `atom2.mass`) or variables defined in the protocol/optimization file (which should be prefixed by a global, e.g. `global.PI`). Potential terms are assigned to bonds in order they appear in the file. If two potentials describe the interaction between the same beads, only the first in order will be used and the second one will be ignored.

RedMDStream supports the following potentials:

1. **Harmonic pseudo–bond potential**

$$V(r) = \frac{1}{2}k(r - r_0)^2 \tag{1}$$

tag: BOND_HARM, parameters to declare: `r0`$= r_0$ and `k`$= k$

2. **Harmonic pseudo–angle potential**

$$V(\theta) = \frac{1}{2}k_\theta(\theta - \theta_0)^2 \tag{2}$$

tag: ANG_HARM, parameters to declare: `theta0`$= \theta_0$ and `k`$= k_\theta$

3. **Harmonic pseudo–dihedral potential**

$$V(\alpha) = \frac{2}{2}k_\alpha(\alpha - \alpha_0)^2 \tag{3}$$

tag: DIH_HARM, parameters to declare: `alpha0`$= \alpha_0$ and `k`$= k_\alpha$

4. **Morse pseudo–bond potential**

$$V(r) = E_0(1 - \mathrm{e}^{-\alpha(r - r_0)})^2 \tag{4}$$

tag: BOND_MORSE, parameters to declare: `E0`$= E_0$, `r0`$= r_0$ and `alpha`$= \alpha$

5. **Lennard-Jones pseudo–bond potential**

$$V(r) = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] \tag{5}$$

tag: BOND_LJ, parameters to declare: `eps`$= \varepsilon$ and `sigma`$= \sigma$

6. **Rigid pseudo–bond**
This is a bond that has constant length during the simulation and is implemented with RATTLE [1] or SHAKE [20] — depending on the simulation method.
tag: BOND_RIGID, no need to declare parameters.

7. **Custom pseudo–bond potential** This is a user–defined potential. It is defined as:

```
<BOND_CUST name="custom1"
           formula="2*global.PI/log(r)*exp(-bond.dist*r*k)">
   <BONDRULE name="compl2" />
   <VAR name="k">5.0</VAR>
</BOND_CUST>
```

   with a similar definition for angle (ANG_CUST) and dihedral (DIH_CUST). RedMDStream precalculates a table with energies and force values based on the distance (angle, dihedral) and uses this table to interpolate the potential energies and forces during a simulation. One can also overwrite the default function argument name using the parameter attribute and cut-off using the cutoff attribute. To control granularity of the table one can use the following attributes to BOND_CUST and others:

   - tabMin is the smallest distance in the table (default is 0.0 for a pseudo–bond and angle, -π for a dihedral),
   - tabMax is the largest distance in the table (default is the cut–off value for a pseudo–bond, π for an angle and dihedral),
   - tabSteps is the number of steps in the table.

   For distances (angles/dihedrals) outside of the tabMin and tabMax range, the energies and forces for a boundary value are assigned. If the SXML output is used, RedMDStream will print in separate files the tabulated potentials in the column format r F E. This functionality can be helpful in visualizing custom potentials, as well as in diagnosing problems.

8. **Custom bond–angle potential**

$$V(r,\theta) = V_1(r)V_2(\theta) \tag{6}$$

   This is a product of two custom potentials where the first one is dependent on 1-2 pseudo–atom distance r and the second on 1-2-3 pseudo–atom angle θ. E.g.:

```
<BOND_ANGLE_CUST name="custom2"
           bond="0.5*k*(r-r0)^2"
           angle="0.5*kTheta*(theta-deg(theta0))^2">
   <BONDRULE name="compl2" />
   <VAR name="k">5.0</VAR>
   <VAR name="r0">15.0</VAR>
   <VAR name="kTheta">5.0</VAR>
   <VAR name="theta0">150.0</VAR>
</BOND_ANGLE_CUST>
```

This potential is tabulated using separate tables for pseudo–bond and angle terms. The settings for both tables are provided independently. BOND_ANGLE_CUST attributes bondTabMin, bondTabMax and bondTabSteps can be provided for the pseudo–bond part. BOND_ANGLE_CUST attributes angTabMin, angTabMax and angTabSteps play a similar role for the angle part.

2.2.5 Nonbonded Potential Energy Function

The nonbonded potentials are defined inside a NONBONDED section tags. They can be defined for a particular pair of beads (by specifying their type) or for all beads (the ”default” potential). An example of the NONBONDED section looks like this:

```
<NONBONDED>
   <BOND_LJ name="nonbLJ">
      <VAR name="eps">3.0</VAR>
      <VAR name="sigma">15.0</VAR>
   </BOND_LJ>
   <BOND_MORSE name="morseNB" type1="P" type2="CA">
      <VAR name="E0">3.0</VAR>
      <VAR name="r0">15.0</VAR>
      <VAR name="alpha">0.1</VAR>
   </BOND_MORSE>
</NONBONDED>
```

The nonbonded potential accepts the following potentials from the above defined (`BOND_LJ`, `BOND_MORSE`, `BOND_CUST`), as well as the containing sphere type potential (`B_SPHERE`) and Coulomb electrostatic potentials (`BOND_ELEC`) and (`BOND_ELEC2`):

1. **Containing sphere potential**

$$\vec{F} = A \cdot (R_o - r)^{-n} \hat{e} \quad \text{for} \quad r > R_{cut} \tag{7}$$

   This field assures that the molecules of the system do not move away from the system's center of mass farther than the containing sphere radius, R$_o$. $\vec{F}$ is a radial force acting on a given molecule in the region between the containing sphere radius (R$_o$) and the user-specified cut-off (R$_{cut}$). Parameters A and n ($n \geq 0$) allow the user to modulate the amplitude and radial dependence of $\vec{F}$.
   tag: `B_SPHERE`, parameters to declare: `A`$= A$ and `n`$= n$

2. **Standard Coulomb potential**

$$V(r) = \frac{1}{4\pi\varepsilon_0\varepsilon} \frac{q_1 q_2}{r} \tag{8}$$

   tag: `BOND_ELEC`, parameters to declare: `eps`$= \varepsilon$

3. **Shielded Coulomb potential**

$$V(r) = \frac{1}{4\pi\varepsilon_0\varepsilon' r} \frac{q_1 q_2}{r} \tag{9}$$

   This is the Coulomb potential with dielectric constant increasing with distance: $\varepsilon(r) = \varepsilon' r$
   tag: `BOND_ELEC2`, parameters to declare: `eps`$= \varepsilon'$

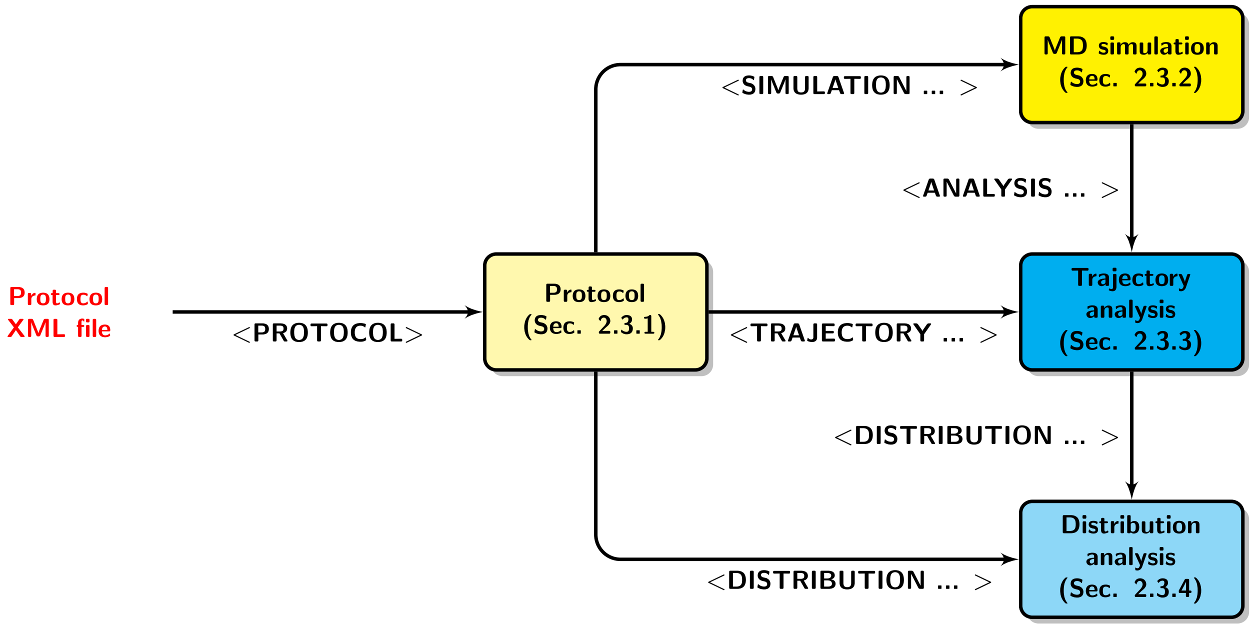


Figure 5: Relationship between different parts of the protocol XML file, along with the manual section number and XML tag to switch between the parts.

2.3 Simulation Protocol

The simulation protocol consists of a set of commands to prepare, execute and analyze a CG trajectory. The CG simulation is set with a particular topology defined in an XML file organized by a set of rules described in the previous section. The general layout of the protocol XML file is presented in Fig. 5.

2.3.1 Protocol XML File

The simulation protocol tag has to include a root tag `PROTOCOL`. Directives inside the `PROTOCOL` tag are executed in a sequential order. If any of the directives stop, the program execution is also stopped. The directives include:

1. **Load another protocol file**:

   ```
   <PROTOCOL file="filename" />
   ```

2. **Define variable for equation parsing**:

   ```
   <VAR name="PI">3.1415926</VAR>
   <VAR name="E">exp(1)</VAR>
   ```

   Expression for the variable is placed inside the tags.

3. **Print variable or expression value**:

   ```
   <PRINT var="var1" />
   <PRINT> log(var1 * pi)<PRINT />
   <PRINT description="Var2-4 sum"> var2+var3+var4<PRINT />
   ```

`description` is an optional attribute that can help understand the output.

4. **Calculate a distance (or angle/dihedral angle) distribution**:

```
<DISTRIBUTION name="distr1" min="0.0" max="100.0" bins="100">
   <EXPR> exp(-4*r) </EXPR>
</DISTRIBUTION>
```

Generates a distribution for the distances from 0.0 Å to 100 Å in 100 bins of an $f(r) = \exp(-4r)$ function and saves it in memory under the name `distr1`).

One can also load a distribution from a text file (in every line there is a bin position and number of counts separated by a space),

```
<DISTRIBUTION name="distr4">
   <TXTFILE file="filename" />
</DISTRIBUTION>
```

load a distribution from a binary file generated by RedMDStream:

```
<DISTRIBUTION name="distr5">
   <BINFILE file="filename" />
</DISTRIBUTION>
```

5. **Define a coarse–grained mapping and topology in the same file**:

```
<TOPOLOGY>
...
</TOPOLOGY>
```

or load from a different file:

```
<TOPOLOGY file="top.xml" />
```

See Section 2.2 for details on how to define topology.

6. **Load a molecule structure file**:

```
<MOLECULE name="mol1"/>
   <PDB file="file1.pdb" />
</MOLECULE>

<MOLECULE name="mol2"/>
   <PQR file="file2.pqr" />
</MOLECULE>

<MOLECULE name="mol3"/>
   <PDBML file="file3.xml.gz" />
</MOLECULE>
```

The above loads a PDB, PQR and PDBML (including gzip compressed PDBML). In addition, information about a secondary structure may be loaded:

(a) **Dot–bracket RNA file**:

```
<MOLECULE name="mol5"/>
   <PDB file="file5.pdb" />
   <RNA2D file="file5.pdb.out" />
</MOLECULE>
```

This is a standard format used to describe the RNA 2D structure by the structure prediction software [17] or structure analysis software [9]. A dot–bracket file may include one or multiple 2D structures, each in a separate line. Each structure might be accompanied by a floating point number that signifies the probability of finding such a structure. For such a case each particular interaction will have a probability calculated as a sum of probabilities of all structures it can be found in. For example in the following file:

```
123456
((..)) 3.0
(....) 2.0
((())) 1.0
```

the interaction between bases 2–5 will be created with $\frac{3.0+1.0}{3.0+2.0+1.0} = 0.66$ strength (it is always normalized to one). If no number is given in the line, we assume that all structures are equally probable. Lines which contain characters other than `().` or white spaces are ignored, before the first appearance of `().` .

(b) **RNAView tertiary structure information**:

```
<MOLECULE name="mol4"/>
   <PDB file="file4.pdb" />
   <RNAVIEW file="file4.pdb.out" />
</MOLECULE>
```

The above loads a tertiary structure (canonical and noncanonical pairings) from a `file4.pdb.out` file.

(c) **Custom interaction file**:

```
<MOLECULE name="mol6"/>
   <PDB file="file6.pdb" />
   <INTERACTION file="interaction_file" />
</MOLECULE>
```

Each line of the file should be formatted as follows: `interactionName chainID1 resID1 chainID2 resID2 probability`, with the last parameter being optional.

7. **Load an ensemble of structures** in the PDB format to generate a distance distribution:

```
<DIRECTORY name="dir" pdb=".pdb" scanSubdirs="true">
   <DISTRIBUTION name="distr2" min="0.0" max="100.0" bins="100"
                 bondrule="rule1">
       <TXTFILE save="file.txt" />
```

```
            <BINFILE save="file.distr" />
        </DISTRIBUTION>
    </DIRECTORY>
```

Analyzes all PDB files (with the extension .pdb) in a directory named `dir` and all its subdirectories. Creates a bead distance distribution named `distr2`, based on a bond named `rule1` (see Sec. 2.2). If `bondrule` is omitted, the distribution will be calculated for the distances between all beads. The distribution is calculated for the distances from 0.0 Å to 100.0 Å, in 100 bins and saved in memory as `distr2`, in a text file called `file.txt` and in a binary file called `file.distr`.

8. **Save a coarse–grained molecule structure**:

```
<OUTPUT mol="mol1" pdb="mol1.cg.pdb" />
<OUTPUT mol="mol1" sxml="mol1.sxml" />
<OUTPUT mol="mol1" pdb="mol1.r1.pdb" atomrule="rule1" />
<OUTPUT mol="mol1" pdb="mol1.r2.pdb" bondrule="rule2" />
```

The above saves the `mol1` coarse–grained structure to `mol1.cg.pdb` in the PDB format [3] and both the coarse–grained structure and topology to a file `mol1.sxml` (the RedMD structure XML file that contains information about beads and interactions) [8]. One can also filter the PDB file to select only these beads that conform with a particular atom rule (see Sec. 2.2.2). One can also store in the PDB file, with the CONECT statements, information about bonds satisfying a particular bond rule (see Sec. 2.2.3). This allows visualizing an effect of a bond rule on a sample molecule. For large molecular complexes, such as a ribosome, the residue number field width may become exceeded and this can lead to two different residues having the same number. In this case one can use the `useResSeqNum="false"` attribute for `OUTPUT` for RedMDStream to reassign residue numbers.

9. **Run an MD simulation**:

```
<SIMULATION mol="mol1" ...>
    ...
    <ANALYZE>
        ...
    </ANALYZE>
</SIMULATION>
```

The above means simulate a molecule named `mol1`, see Section 2.3.2 for detailed information on SIMULATION tag. The trajectory analysis commands are enclosed in the ANALYSIS tag and described in Section 2.3.3.

10. **Load and analyze a saved DCD trajectory**:

```
<TRAJECTORY mol="mol1" dcd="mol1.traj.dcd" cg="true">
    <ANALYZE>
        ...
    </ANALYZE>
</TRAJECTORY>
```

See Section 2.3.3 for details on the analysis of a simulation trajectory enclosed in the ANALYSIS tag.

2.3.2 Molecular Dynamics Simulation

For details on how a molecular dynamics simulation is performed (equations of motion, etc.) please refer to Ref. [8] or RedMD manual (available at the http://bionano.cent.uw.edu.pl/Software/RedMD webpage). The simulation execution tag depends on the simulation mode. For the integration (NVE simulation) one should start the `SIMULATION` section in the following way:

```
<SIMULATION mol="mol1" mode="integration" dt="10 fs">
    ...
</SIMULATION>
```

where all the properties are obligatory. A property named `dt` denotes the simulation time step. One can use the time units (fs or ps); if left unitless, picoseconds will be used. The user may also define how the simulation is saved by using the optional properties of the SIMULATION tag. By default the simulation is stored and analyzed in memory. To save a trajectory to a file one has to use the `dcd` property, for example:

```
<SIMULATION mol="mol1" mode="integration"
            dt="10 fs" dcd="mol1.dcd">
    ...
</SIMULATION>
```

If parallel calculations are performed, a suffix to a DCD file will be added, with a thread number or MPI rank. To avoid overwriting the files, if two RedMDStream processes are to be executed in one directory, one can use the `dcdProc="true"` option. With this option all the DCD files will be suffixed with a unique process number. If one would like to append the trajectory to an already existing DCD file, the option `dcdAppend="true"` will preserve the existing content of the file (otherwise it will be truncated). If the append option is in charge, the preexisting frames of the trajectory will also be used for the analysis.

Another simulation type is a simulation in the NVT ensemble with the Berendsen thermostat:

```
<SIMULATION mol="mol1" mode="berendsen" dt="10 fs" tau="1">
    ...
</SIMULATION>
```

where `tau` denotes the coupling constant in ps.

Langevin dynamics is also available:

```
<SIMULATION mol="mol1" mode="langevin" dt="10 fs"
            gamma="1" temp="300">
    ...
</SIMULATION>
```

with `gamma` the damping constant in ps^{-1} and `temp` the starting temperature for the initiation of Langevin dynamics engine.

and Brownian dynamics:

```
<SIMULATION mol="mol1" mode="brownian" dt="10 fs"
            hr="1.3" temp="300">
    ...
</SIMULATION>
```

where `hr` is the default hydrodynamic radius (in Å). The hydrodynamic radius can be also set as a `HR` variable for a particular atom type (see Sec. 2.2.2). It is important to set the starting temperature with `temp` for the proper initiation of the Brownian dynamics engine.

In the `SIMULATION` tag one can also define the temperature which will be used to initialize `velocities for` a molecule (default: 10 K) and tolerance for rigid bonds (default 0.001 Å):

```
<SIMULATION mol="mol1" mode="langevin" dt="10 fs" gamma="5.0"
            temp="310" rigid_tol="0.01">
   ...
</SIMULATION>
```

Inside the `SIMULATION` tag one can use the following commands:

1. **Run minimization**:

   ```
   <MINIMIZE />
   ```

   The structure is minimized using a limited memory Broyden-Fletcher-Goldfarb-Shanno (L-BFDS) method.

2. **Set temperature** for initialization (if no dynamics is performed), Berendsen bath coupling, Langevin dynamics or Brownian dynamics:

   ```
   <SET_TEMP val="310" />
   ```

   The temperature pattern can be also defined for certain steps. For heating from 10 K to 300 K during 10000 simulation steps and next a constant temperature simulation one can use the following command:

   ```
   <SET_TEMP start="10.0" end="310.0" steps="1000"/>
   ```

   or if one wants to use the time units instead:

   ```
   <SET_TEMP start="10.0" end="310.0" time="10 ps"/>
   ```

   If the `start` tag is not defined, RedMDStream will calculate the current temperature of the molecule and take it as a starting value.

   One can also use a custom expression to define temperature. In such a case it has to be enclosed between the opening and closing `SET_TEMP` tags (the `step` variable holds the current step value):

   ```
   <SET_TEMP>
      310.0 * exp (-step/10000)
   </SET_TEMP>
   ```

3. **Unfold a biomolecule**:

   ```
   <UNFOLD dist="4.8" />
   <UNFOLD topology="linear" dist="4.8" />
   ```

   The above unfolds a biomolecule to a circle (default) or linear conformation, with the bead distance equal to the `dist` parameter (compulsory). A small random number (-0.1 — 0.1 Å) is added to the z coordinate. Unfolding only affects a conformation present in a simulation and does not change the reference structure of a molecule, used e.g. as a reference in RMSD or RMSF calculations.

4. **Perform single point forces and energy calculation**:

   ```
   <SINGLE_POINT />
   ```

   For the loaded conformation the above calculates the energy of the molecule and forces acting on each bead. This can be helpful in understanding the force field mechanics and possible stability problems.

5. **Perform CG MD simulation**:

   ```
   <STEP steps="10000" />
   ```

   This performs 10000 steps of a simulation. One can also define the number of steps in time units:

   ```
   <STEP time="10 ns" />
   ```

   which can be more convenient for a longer simulation setup (fs/ps/us[11]/ms are available). There are also options to control the size of the saved trajectory. This can be defined either by setting the interval in the number of steps:

   ```
   <STEP time="10 ns" saveFreq="1000" />
   ```

   or by setting a target amount of frames to be saved in total:

   ```
   <STEP time="10 ns" saveFrames="1000" />
   ```

   The last option is not to save at all:

   ```
   <STEP time="10 ns" noSave="true" />
   ```

6. **Print system temperature**:

   ```
   <TEMPERATURE />
   ```

   Prints the current temperature of the system, calculated from the velocities, and includes information about the intended temperature used by the thermostat.

7. **Print system energy**:

   ```
   <ENERGY printAll="true"/>
   ```

[11]for microsecond

Prints the energy of the system. The output contains the total energy, as well as particular energy contributions (potential or kinetic) and terms (e.g., harmonic, Coulomb, etc.).

One can also save the energy value in the RedMDStream internal variable with the following tag:

```
<ENERGY saveAll="true"/>
```

to save various energy terms (the energy type name is used as a variable name) or

```
<ENERGY type="NONBONDED" var="nonb_enrgy"/>
```

where the energy type is set in the `type` attribute and the variable name in `var`. For possible energy types please first use the `printAll="true"` option.

8. **Print system conformation**:

```
<OUTPUT pdb="out1.pdb" />
<OUTPUT pdb="out2.pdb" force="true" />
<OUTPUT pdb="out3.pdb" momenta="true" />
<OUTPUT pdb="out4.pdb" velocity="true" />
<OUTPUT pdb="out5.pdb" atomrule="rule1" />
<OUTPUT pdb="out6.pdb" bondrule="rule2" />
```

The above outputs the current state of the molecule in a simulation. One can use the options to print force/momenta/velocity instead of coordinates in the PDB [3]. One can also filter the output using an atom rule `rule1` (see Sec. 2.2.2) or add the connectivity information in the PDB file according to a particular bond rule `rule2` (see Sec. 2.2.3).

9. **Analyze trajectory**:

```
<ANALYZE beg="0" end="1000" stride="20">
    ...
</ANALYZE>
```

For tags allowed inside `ANALYZE` see Section 2.3.3. The analyze keyword allows specifying the start and end frame for the analysis; this is the frame number in a trajectory, not the number of the simulation step or time; the numbering starts from 0, the stride specifies how many frames to omit. By default the analysis begins at first (zero) frame, ends at the last frame, and includes all frames (`stride` is 1).

2.3.3 Trajectory Analysis

RedMDStream provides three ways to analyze trajectories (see Ref. [14] on how each measure is defined):

1. **Structural fidelity by means of root mean square deviation (RMSD)**
RedMDStream calculates RMSD using the Kabsch algorithm [12] to account for the best translation and rotation (that minimizes RMSD) of a molecule prior to calculating the RMSD value. The calculated average RMSD from the whole trajectory can be saved to a variable:

```
<RMSD var="rmsd" />
```

or RMSD for each frame printed to a file:

```
<RMSD file="mol1.rmsd" />
```

2. **Structure mobility by means of root mean square fluctuations (RMSF)**
   To calculate average RMSF one can use the following tag:

   ```
   <RMSF var="rmsf" />
   ```

   To save RMSF to a file for each bead:

   ```
   <RMSF file="mol1.rmsf" />
   ```

   RedMDStream aligns the trajectory before calculating RMSF to remove translations and rotations of the molecule during simulation. By default the RMSF units ($Å^2$) are used, however one can also use the B-factor units by adding `align="true"`. This can be turned off by adding `align="false"` to the XML tag:

   ```
   <RMSF var="rmsf" align="false" bFactor="true" />
   ```

   RedMDStream can also compare RMSF with the one included in the original PDB/PDBML file (in this case always B-factor units are used):

   ```
   <RMSF var="rmsfDiff" align="true" />
   ```

   saves in a variable `rmsfDiff` the sum of absolute values of differences between the B-factors in each frame as compared to a reference. In a similar way RMSF can be compared with data stored in a file derived from analysis of a molecular dynamics trajectory, e.g. made using cpptraj:

   ```
   <RMSF var="rmsfDiff" file="file1" align="true" />
   ```

   `file1` should be a file with a number of lines equal to the number of CG beads in a system, ordered in the same way as in RedMDStream (e.g. as in a PDB file created from the structure). Each line should have two numbers: atom number (ignored) and RMSF value. One can impose the B-factor units with `bFactor="true"`.

3. **Distance distribution functions**
   Distance distribution functions for beads can be calculated from the trajectory using the DISTRIBUTION tag:

   ```
   <DISTRIBUTION min="0.0" max="100.0" bins="100"
                 bondrule="rule1" name="distr3" />
   ```

   Creates a bead distance distribution named `distr3`, based on a bond named `rule1` (see Sec. 2.2). If `bondrule` is omitted, the distribution will be calculated for distances between all beads. The distribution is calculated for the distances from 0.0 Å to 100.0 Å, in 100 bins and saved in memory as `distr3`. As a reference one can use also a previously saved distribution. In such case the distance range and bin number are taken from a reference distribution and should not be specified:

```
<DISTRIBUTION bondrule="rule1" name="distr4" ref="distr1" />
```

Distributions can be calculated also for angles and dihedrals:

```
<DISTRIBUTION bondrule="rule2" name="distr5"
              min="0.0" max="180.0" bins="100" type="angle">
   ...
</DISTRIBUTION>

<DISTRIBUTION bondrule="rule3" name="distr6"
              min="-50.0" max="50.0"
              bins="100" type="dihedral">
   ...
</DISTRIBUTION>
```

The angle and dihedral values are provided in degrees. To compare two distributions, one should provide the name of a reference distribution (previously defined by the `DISTRIBUTION` tag):

```
<DISTRIBUTION bondrule="rule1" name="distr3"
              ref="distr1">
   ...
</DISTRIBUTION>
```

In this case do not provide the beginning, end values and number of bins because these are taken to match the reference distribution. The analysis and comparison methods are described in Sec. 2.3.4.

2.3.4 Distribution Analysis

Distribution analysis is performed inside the `DISTRIBUTION` tag. The following options are available:

1. **Normalize distribution**:

   ```
   <DISTRIBUTION ...>
      ...
      <NORMALIZE />
      ...
   </DISTRIBUTION>
   ```

   Sums all bins equal to 1.0.

2. **Multiply distribution by a function**:

   ```
   <DISTRIBUTION ...>
      ...
      <MULTIPLY> ln(r) </MULTIPLY>
      ...
   </DISTRIBUTION>
   ```

Multiplies a distribution by a given function. r is always used as a parameter.

3. **Divide distribution by a function**:

```
<DISTRIBUTION ...>
   ...
   <DIVIDE> r^2 </DIVIDE>
   ...
</DISTRIBUTION>
```

Divides a distribution by a given function. r is always used as a parameter.

4. **Find maximum position**:

```
<DISTRIBUTION ...>
   ...
   <MAX_POSITION var="max" />
   ...
</DISTRIBUTION>
```

Stores the position of the most occupied bin in a variable provided in the attribute `var`.

5. **Calculate distribution difference using the Hodgkin index**:

```
<DISTRIBUTION ref="..." ...>
   ...
   <HODGKIN var="hi" />
   ...
</DISTRIBUTION>
```

Calculates the Hodgkin index [14, 11] :

$$\mathrm{HI} = 1 - \frac{2\sum_i d_1(i)d_2(i)}{\sum_i d_1^2(i) + \sum_i d_2^2(i)} \tag{10}$$

and stores it in a variable specified by the attribute `var`. The Hodgkin index is in the range from 0.0 to 1.0. For the variant of the Hodgkin index adopted in RedMDStream the low index means similarity and index close to 1 means dissimilarity. The above works only if a reference distribution is provided.

6. **Calculate distribution difference using the Kolmogorov–Smirnov measure**:

```
<DISTRIBUTION ref="..." ...>
   ...
   <K-S var="k-s" />
   ...
</DISTRIBUTION>
```

performs the Kolmogorov-Smirnov test of equality of the two distributions. This test finds the maximum difference between two cumulative distribution functions [19]. At the moment, we only calculate the value of the test itself and it is not changed into the probability space. When `K-S` is used, the measure is stored in a variable pointed by the `var` attribute. This value will be in the range from 0.0 to 1.0, with the lower values signifying better similarity of the two distributions. This option works only if a reference distribution is provided.

7. **Save distribution in binary format**:

```
<DISTRIBUTION ...>
   ...
   <BINFILE save="distr.bin" />
   ...
</DISTRIBUTION>
```

The binary format is only accessible by RedMDStream and allows avoiding small errors in the floating point number conversion from the internal representation to text and vice versa. The binary format is the preferred one for the distributions that are later to be read in by RedMDStream.

8. **Save distribution in text format**:

```
<DISTRIBUTION ...>
   ...
   <TXTFILE save="distr.txt" />
   ...
</DISTRIBUTION>
```

Saves the distribution in a text file; the first column is the position of the bin center and the second column – bin occupancy.

2.4 Model Optimization

The model optimization tools help to design models as well as give feedback how a particular modification of a model affects its ability to simulate biomolecules.

The optimization mode of RedMDStream helps to answer the following questions: (i) what is the optimal dihedral angle force constant to simulate a DNA helix? (ii) what is the effect of changing the positioning of a coarse-grained bead from the center of mass of an amino acid to its carbon-α? (iii) how changing one force field parameter affects the coarse-grained model? (iv) are there any correlations between the force field parameters? [14].

For the above purpose, a scoring function $f(x_1, ..., x_n)$ is defined:

$$f(x_1, ..., x_n) = \sum_{i}^{M} w_i s_i \tag{11}$$

The arguments of the function x_1 to x_n are the parameters that affect the behavior of the CG model. Most likely these are the parameters of the potential energy terms but also the variables that turn on and off particular interactions, switch between the bead types and positions or even protocol parameters (such as simulation temperature).

The value of the above function is a weighted sum of certain observables that assess/measure the quality of the model. These observables are derived from the analysis of the CG MD trajectory and include the distance distribution analyses, structural similarity and atom mobility as compared with a reference (see Sec. 2.3.3). RedMDStream provides a few methods that either find a best set of parameters x_1 to x_n or give an overview of how $f(x)$ is affected by changing x_1 to x_n.

Note the high complexity of the problem. First, the function $f(x)$ is in general not a single minimum function. Second, since $f(x)$ results from the simulations, its value is affected by uncertainty. Each simulation is started with a different draw of velocities and is of finite number of steps, so each trajectory may cover different parts of conformational space of the system. Our studies [14] show that the value of $f(x)$ of Eq. 11 may have a standard deviation of 10-20% if simulations are repeated.

To find optimal parameters two strategies are implemented in RedMDStream: metaheuristic and local optimization [10]. Metaheuristic algorithms, like evolutionary algorithm (see Sec. 2.4.2) or particle swarm optimization (see Sec. 2.4.3), look for a solution using certain rules that mimic the behavior of systems found in nature. On the other hand, local methods, like the simplex method (see Sec. 2.4.4), use a set of strict rules that explore the parameter space close to a given solution to systematically improve the model. In the optimization problem in CG MD models, this systematic improvement is a burden. First, the multi–minima nature of this problem sometimes requires the algorithm to "step back" and accept a worse solution to finally get to the best one. Second, the function uncertainty is a problem. Such systematic approach usually relies on the difference between two scores to find the direction. But when one subtracts two numbers of similar magnitude, the difference is usually small and can be smaller than the error associated with uncertainty, leading to a random behavior of the optimizer. Loosening the optimization rules is a good remedy in that case. The local optimization always accepts the better solution and metaheuristic approaches allow some wrong solutions in a next iteration.

RedMDStream also implements three methods that provide information about the relationship between the parameter and scoring function without performing the actual optimization. These methods are the parameter grid analysis, random analysis and gradient calculation.

2.4.1 Optimization XML File

The optimization control file defines how to perform optimization of a CG model. The `OPTIMIZATION` root tag has an attribute `method` that selects one of the available methods:

```
<OPTIMIZATION method="evo">
   ...
</OPTIMIZATION>
```

Inside this tag the following three sections must be declared (note that each can be declared only once):

1. **Arguments and scoring function** (`DATA`)
   This section defines the arguments for the function defined in Eq. 11:

   ```
   <DATA>
      <PARAMETER name="p1" min="0.0" max="1.0" />
      <PARAMETER name="p2" min="5.0" max="7.0" />
   ```

   as well as the terms, their limits/bounds[12] and relative weights for the function defined in this equation:

   ```
      <SCORE name="s1" value="rmsd" max="50" weight="0.7" />
      <SCORE name="s2" value="rmsf" min="20"
             max="100" weight="0.3" />
   </DATA>
   ```

   In the above example the simulation protocol knows the variables `p1` (in the range 0.0 — 1.0) and `p2` (in the range 5.0 — 7.0) and has to create the variables `rmsd` and `rmsf`. The former one will be taken with the weight of 70% and the values between 0 and 50. The latter one will be taken with the weight of 30% and the values between 20 and 100.

2. **Simulation protocol** (`PROTOCOL`)
   The simulation protocol is defined, either by referencing a file:

   ```
   <PROTOCOL name="run.xml" />
   ```

   or by embedding all the commands:

   ```
   <PROTOCOL>
      ...
   </PROTOCOL>
   ```

   The protocol can be set to be calculated multiple times to account for errors in calculating the function defined in Eq. 11:

   ```
   <PROTOCOL name="run.xml" runs="10"/>
   ```

[12]RedMDStream scores are normalized in the 0.0 to 1.0 range. Therefore, if s_i^l and s_i^u are allowed lower and upper bounds for the i-th score (by default 0.0 and 1.0), the score will be transformed using the equation: $s_i' = (s_i - s_i^l)/(s_i^u - s_i^l)$, if between the bounds and equal to 0.0 if lower than s_l and 1.0 if higher than s_u.

In the above case the protocol from the file `run.xml` will be executed 10 times and the scores will be averaged with their standard deviation recorded. Performing multiple runs may help determine the scale of the score uncertainty and give an idea how to overcome this uncertainty problem.

3. **Optimization options** (`OPTIONS`)
   This section controls the optimization process. Most of the provided options are specific for a particular optimization method and therefore they will be described later, in Secs. 2.4.2-2.4.7. Two tags are shared among all the methods:

   (a) `NORMALIZE_TOTAL`:

   ```
   <OPTIONS>
      <NORMALIZE_TOTAL />
      ...
   </OPTIONS>
   ```

   which normalizes the weights for the parameters in a way that their sum equals to 1.0 and the final score is in the range from 0.0 to 1.0.

   (b) `PROB_COMAPRISON`:

   ```
   <OPTIONS>
      <PROB_COMPARISON />
      ...
   </OPTIONS>
   ```

   toggles the probabilistic comparison of two CG MD models. Without this option RedMDStream uses only the average weighted score to compare the models. However, this type of comparison can be misleading if a score differs only by a small margin. If the MD simulation protocol is executed multiple times (for `runs` option higher than 1 in the `PROTOCOL` tag), RedMDStream records the score averaged from multiple MD runs, as well as the standard deviation over multiple runs, which estimate the error of finding the score. With both the average and its standard deviation, the Z-test can be applies to find the probability of the hypothesis that model 1 is better than model 2. According to this probability model 1 is returned as the answer to the compare query.

2.4.2 Evolutionary Algorithm

Evolutionary algorithm mimics Darwinian evolution to optimize multi–argument functions. It is a metaheuristic algorithm so it does not guarantee finding an optimal result, however it has been proven successful in solving many problems. [10]

The function arguments (x_1 to x_n in Eq. 11) in the evolutionary algorithm nomenclature are called *genes*[13] and their vector is considered a *chromosome*. The observables (s_i to s_M in Eq. 11) are considered *phenes* and their weighted sum a *phenotype*. The aim of this algorithm is to find a set of genes that result in the best phenotype. The general outline of the procedure is the following (see also Fig. 6 and Ref. [14] for details):

[13] In italic we type the first occurrence of a word that has a specific meaning in the jargon of a particular optimization type.

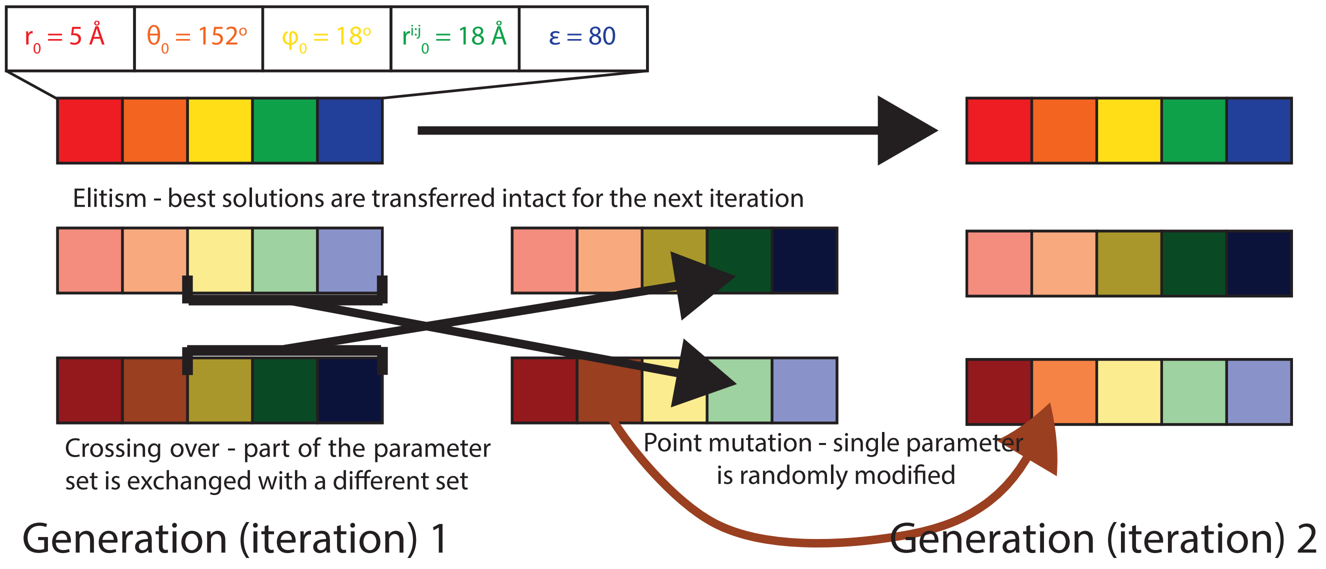

r_0 = 5 Å
θ_0 = 152°
φ_0 = 18°
$r^{i:j}_0$ = 18 Å
ε = 80
Elitism - best solutions are transferred intact for the next iteration
Crossing over - part of the parameter set is exchanged with a different set
Point mutation - single parameter is randomly modified
Generation (iteration) 1
Generation (iteration) 2

Figure 6: Schematic description of the evolutionary algorithm.

1. Initiation: create a pool of chromosomes and calculate their phenotypes.
2. *Elitism*: move the best solutions to the new pool without any modifications.
3. *Mating*: select pairs of chromosomes for mating, giving higher probability for the ones with a better phenotype.
4. *Crossing–over*: mix chromosomes to create a new gene pool.
5. *Mutations*: introduce random modifications to extend the gene pool.

General outline of the evolutionary algorithm file is:

```
<OPTIMIZATION method="evo">
   <PROTOCOL ... />
   <DATA>
      ...
   </DATA>
   <OPTIONS>
      <FILE save="optimize.result"
            saveTxt="optimize.result.csv" />
      <ITERATION size="16" number="128"/>
      <ELITE size="4" />
      <MATING type="tournament" />
      <MUTATION type="gauss" tolerance="0.1" />
      <CROSSOVER blending="1.0" />
```

```
    </OPTIONS>
</OPTIMIZATION>
```

The PROTOCOL and DATA tags were already described in Sec. 2.4.1. In the OPTIONS tag one can set the parameters of the evolutionary algorithm:

1. **General parameters**
   In the ITERATION tag one can control the size of the gene pool (the number of chromosomes used) using the size attribute and the number of the evolutionary algorithm iterations performed in the number tag.

2. **Elitism**
   The ELITE tag controls the process of moving the chromosomes with the best scoring phenotypes intact to the new iteration (i.e., without applying crossing–over and mutations). Our previous studies show that a small number of elite members stabilizes the population and leads to a better final result. [14] The number of elite chromosomes is set by the size attribute. Setting this number does not alter the total number of chromosomes, the more the elite members, the less chromosomes will be generated by the standard procedure. Since the elite chromosomes are by default moved intact, their chromosome is also copied. However, the user may request to recalculate them by using the recalc="true" attribute, e.g.:

   ```
   <ELITE size="4" recalc="true" />
   ```

   In this case the elite chromosome score will be recalculated and combined with the existing score, providing better phenotype estimation for elite members.

3. **Mating**
   The MATING tag controls the chromosome mating process. Three selection methods are available. The default one is the tournament selection: two chromosomes are randomly picked and the one with a better phenotype is selected (this method can be used with a probabilistic comparison described in Sec. 2.4.1).

   One can also use the fitness choice:

   ```
   <MATING fitness="true" />
   ```

   where the chromosome for mating is randomly selected from the gene pool with the selection probability proportional to the phenotype.

   Similar is the rank choice:

   ```
   <MATING rank="true" />
   ```

   but the probability is proportional to the rank in the phenotype list, not to the actual score. If the phenotype scores are unevenly distributed among the gene pool, the rank selection can do a better job in distinguishing the chromosomes.

4. **Crossing–over**
   The way two chromosomes are combined is defined in the CROSSOVER tag. When the *parent* chromosomes, $(x_1, x_2, ..., x_n)$ and $(y_1, y_2, ..., y_n)$, were selected in mating, they are next mixed

in a crossing–over operation. The simplest way is a single point cross–over; a single gene is selected and all genes following the selected one are flipped:

$$(x_1, x_2, ..., x_{p-1}, x_p - \beta(y_p - x_p), y_{p+1}, ...y_n)\ , \tag{12}$$

$$(y_1, y_2, ..., y_{p-1}, y_p - \beta(x_p - y_p), x_{p+1}, ...x_n)\ , \tag{13}$$

where β is a random number ranging from 0.0 to 1.0.

Alternatively one can select the blending crossing–over:

```
<CROSSOVER blending="1.0" />
```

where the number in the `blending` attribute corresponds to the probability of using this type of cross–over (as in RedMDStream one can use all three crossing–over methods simultaneously with different probabilities). In the blending cross–over new solutions are created by "blending" parent gene values:

$$(x_1 - \beta * (y_1 - x_1), x_2 - \beta * (y_2 - x_2), ..., x_n - \beta * (y_n - x_n))\ , \tag{14}$$

$$(y_1 - \beta * (x_1 - y_1), y_2 - \beta * (x_2 - y_2), ..., y_n - \beta * (x_n - y_n))\ , \tag{15}$$

where β is a random number ranging from 0.0 to 1.0.

Finally, the blending extrapolating cross–over is available:

```
<CROSSOVER blending="0.5" extrapolate="0.2"
           extrapolateFactor="0.5" />
```

where the number in `extrapolate` represents the probability of using extrapolate crossing–over (the remaining probability is applied to a single point cross–over). A blending cross–over can also extend the gene pool by creating three children:

$$(0.5x_1 + 0.5y_1, 0.5x_2 + 0.5y_2, ..., 0.5x_n + 0.5y_n)\ , \tag{16}$$

$$((1.0 + \alpha)x_1 - \alpha y_1, (1.0 + \alpha)x_2 - \alpha y_2, ..., (1.0 + \alpha)x_n - \alpha y_n)\ , \tag{17}$$

$$(-\alpha x_1 + (1.0 + \alpha)y_1, -\alpha x_2 + (1.0 + \alpha)y_2, ..., -\alpha x_n + (1.0 + \alpha)y_n)\ . \tag{18}$$

where α is set by the `extrapolateFactor` attribute.

5. **Mutation**
Random point mutations are controlled by the `MUTATION` tag. Mutations are applied after generating the chromosomes with crossing–over. They are applied with a probability set by the `probability` attribute:

```
<MUTATION probability="0.1" />
```

By default a gene selected for mutation has a new value assigned from the available spectrum, without any preference from the old value.

One can also use Gaussian mutations, where a new value is found according to the Gaussian function probability around the old value:

```
<MUTATION probability="0.1" gauss="true" tolerance="0.1" />
```

Since in this case a new value can be found outside the allowed spectrum, one can use the `tolerance` value to set the allowed margin outside the minimal and maximal gene value.

6. **Input/output files**
 The `FILE` tag specifies how to load and save the data during optimization:

```
<FILE save="optimize.result"
      saveTxt="optimize.result.csv" />
<FILE load="old.optimize.result"
      save="optimize.result"
      saveTxt="optimize.result.csv" />
<FILE loadTxt="old.optimize.result.csv"
      save="optimize.result"
      saveTxt="optimize.result.csv"
      separator="," />
```

One way is to use binary data – a more computer friendly storage (more effective space and information). The `load` attribute points to a binary file that should be loaded with previous results. The `save` attribute points to a binary file to save the results after each generation.

The other way are the text files, which are human readable. The `loadTxt` attribute points to a text file that should be loaded with previous results. The `saveTxt` attribute points to a text file to save the results after each generation. These files can be opened and analysed by a spreadsheet like LibreOffice Calc or Microsoft Excel. Please see Sec. 2.4.8 for tips and tools on analyzing the files. One can specify the field separator for the CSV format (space by default) using the `separator` attribute.

Note, that there is an important difference between loading the binary and text files. Binary files are used to resume an ongoing simulation and contain the whole generation, with scores, etc. To run properly they require exactly the same optimization setup. Text files on the other hand do not contain all the necessary information[14] and are merely used to create a new generation by taking the best solutions found in the text files. Optimization results saved in a text format might be imported in RedMDStream between different optimization types. They can also come from RedMDStream optimization with other parameters (parameters with the same names are copied and the ones missing in the loaded text file are randomly generated) or other score observables (only the final score is used).

[14]E.g., text files contain only the average score and its standard deviation, not the scores of all respective runs, therefore other trials cannot be added with additional recalculations for elite members.

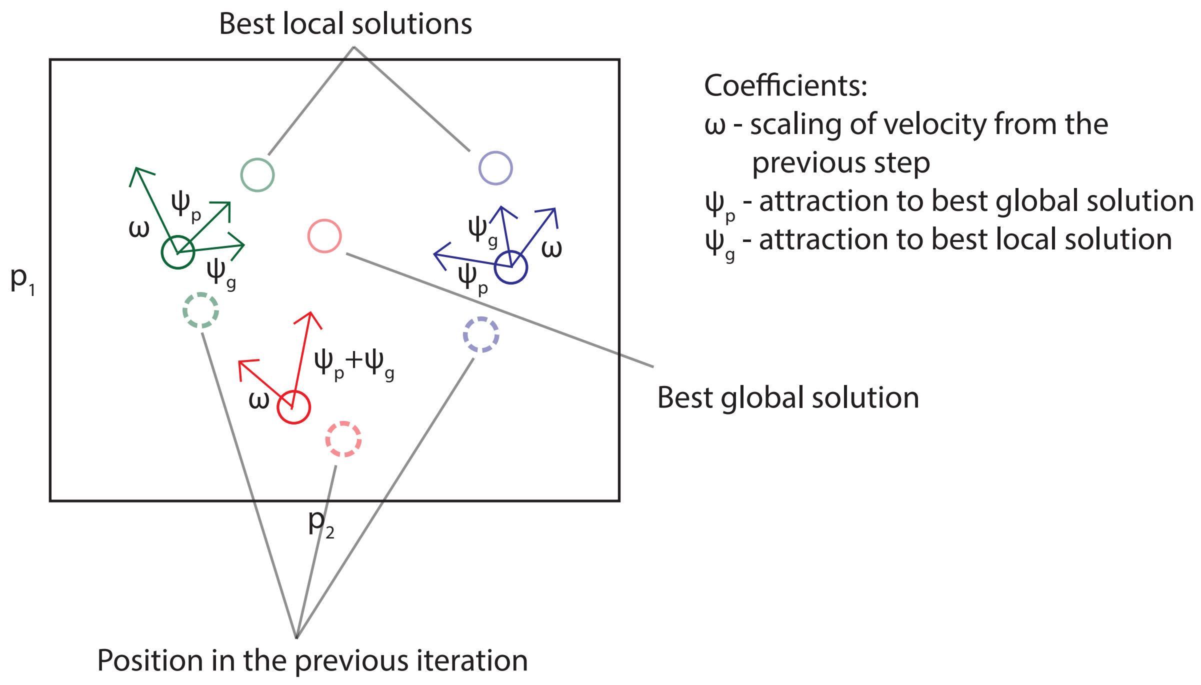

Best local solutions
Coefficients:
ω - scaling of velocity from the previous step
ψp - attraction to best global solution
ψg - attraction to best local solution
p1
p2
ω
ψp
ψg
ψp+ψg
Best global solution
Position in the previous iteration

Figure 7: Schematic description of the particle swarm optimization method.

2.4.3 Particle Swarm Optimization

Particle swarm optimization [13, 10] is a second metaheuristic optimization method implemented in RedMDStream. In this method the optimized models mimic a *swarm* of interacting particles. The main advantage of the algorithm lies in its simplicity since it is controlled by only three parameters ω, ψ_p and ψ_g in contrast to a much large number of parameters and options in the evolutionary algorithm.

In the particle swarm optimization algorithm the arguments x_i of the function of Eq. 11 are considered the *coordinates* of a *particle*. Each particle besides coordinates has a set *velocity*, which is randomly assigned in the beginning. In each iteration the new velocity $\mathbf{v_1}$ and position $\mathbf{r_1}$ of the particle are calculated:

$$\mathbf{v_1} = \omega\ \mathbf{v_0} + \psi_p\ \alpha_p(\mathbf{r_{global}} - \mathbf{r_0}) + \psi_g\ \alpha_g(\mathbf{r_{local}} - \mathbf{r_0})\ , \tag{19}$$

$$\mathbf{r_1} = \mathbf{r_0} + \mathbf{v}\ , \tag{20}$$

where α_p, α_g are random numbers in the range from 0 to 1. $\mathbf{r_{global}}$ is a position of the best scoring particle updated during the algorithm execution and $\mathbf{r_{local}}$ is a position of a specific particle with the best score (set to initial position during initialization). The ω parameter controls the effect of decreasing velocity of the molecule, where ψ_p controls the attraction to swarm–wise best scoring position and ψ_g controls the attraction to particle–wise best scoring position. Please see Fig. 7 for a schematic drawing of the algorithm.

The optimization XML file for the particle swarm optimization should be formatted as:

```
<OPTIMIZATION method="pso">
   <PROTOCOL ... />
   <DATA>
      ...
   </DATA>
   <OPTIONS>
      <FILE save="optimize.result"
            saveTxt="optimize.result.csv" />
      <ITERATION size="16" number="128"/>
      <OMEGA val="0.9" />
      <PSI_P val="0.1" />
      <PSI_G val="0.1" />
   </OPTIONS>
</OPTIMIZATION>
```

where the `PROTOCOL` and `DATA` tags were already described in Sec. 2.4.1. `ITERATION` tag controls the size of the swarm (`size` attribute) and number of iterations (`number` attribute). `OMEGA`, `PSI_P` and `PSI_G` define the values of parameters ω, ψ_p and ψ_g specified in subsequent paragraph. For the treatment of input/output files (`FILE`) see the notes for the evolutionary algorithm on page 6 since they apply in full to the particle swarm optimization (although the binary files are not interchangeable between the evolutionary algorithm and particle swarm optimization).

2.4.4 Simplex Optimization

The simplex algorithm [18] is a local optimization method. The method contains a set of rules that lead to finding the best solution in the vicinity of a provided started point. The simplex algorithm does not involve any random number generator based decisions. This is one of a few local optimization methods that does not involve gradient calculation and for this reason it was implemented in RedMDStream.

If the scoring function (Eq. 11) takes n arguments then at every iteration of the algorithm it needs $n + 1$ models. The parameters of these models form a n dimensional simplex and this is where the algorithm name comes from. The algorithm scheme is as follows (see also Fig. 8 for reference):

1. Rank models according to their score.
2. Select the worst model and reflect it around the simplex (Fig. 8A).
3. If step 2 resulted in a better solution, try to expand further reflection (Fig. 8B).
   If this move improved over the solution from point 2, exchange the worst node to the expanded node and finish the iteration.
   If this move did not improve over the solution from point 2, exchange the worst node to the reflected one and finish the iteration.
4. If step 2 did not improve the solution, try to contract the simplex edge (Fig. 8C). If the node gave a better solution than the original worst node, exchange it with the contracted node and finish the iteration.

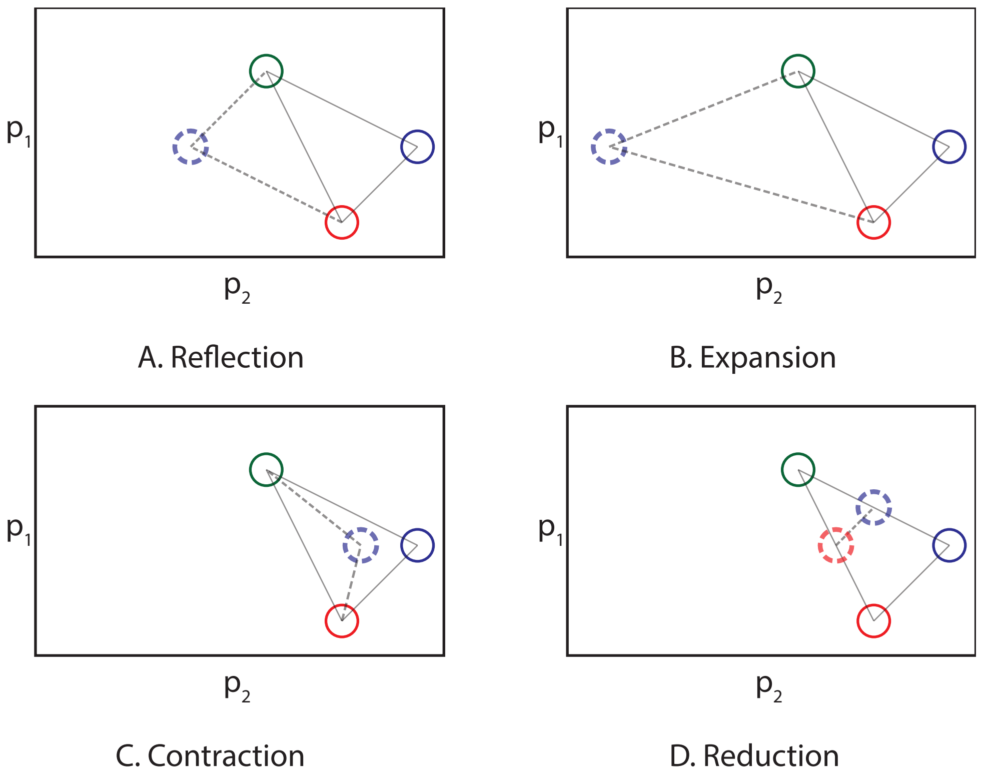


Figure 8: Simplex optimization schematics for a 2D optimization.

5. Reduce the simplex, leaving only the best solution in place (Fig. 8D).

The optimization XML file for the simplex method should be formatted as:

```
<OPTIMIZATION method="simplex">
   <PROTOCOL ... />
   <DATA>
      ...
   </DATA>
   <OPTIONS>
      <FILE saveTxt="optimize.result.csv" />
      <ITERATION number="128"/>
   </OPTIONS>
</OPTIMIZATION>
```

where the `PROTOCOL` and `DATA` tags were already described in Sec. 2.4.1. The simplex optimization method implemented is unconstrained so we do not guarantee that the parameter values will remain in the ranges specified inside the `DATA` section. The `ITERATION` tag controls the number of simplex algorithm iterations (the `number` attribute). One can alter the default behavior of the simplex operations by using the following tags inside the `OPTIONS` section:

1. **Initial simplex formation**
   The simplex algorithm starts with a single point, which can be either randomly selected in allowed range:

```
<RANDOM_START />
```

or placed in a certain part of the allowed spectrum:

```
<START val ="0.25" />
```

where the `val` attribute controls the percentage between the min and max parameter value starting point (default is 25%).

From the starting point simplex is being built in each direction. The length of each direction can be either set directly:

```
<UNITDIST val="0.5" />
```

with a fixed percentage of the range between the minimal and maximal parameter value or randomly:

```
<RANDOM_UNITDIST />
```

An alternative way to initialize the simplex algorithm is to seed it with the results of other RedMDStream optimization:

```
<ITERATION number="128" size="5" />
<FILE loadTxt="optimize.results.csv"
      saveTxt="simplex.csv" />
```

In this case the initial starting points for the simplex algorithm will be loaded from a text file specified in the `loadTxt` attribute. The `ITERATION` tag `size` algorithm controls how many points are loaded. Each starting point is connected with a separate optimization. These optimizations are performed in parallel if OpenMP is enabled. The results are saved to a file mentioned in the `saveTxt` attribute.

2. **Reflection**

   ```
   <ALPHA val="1.0" />
   ```

   The `ALPHA` parameter controls how far the worst simplex node is reflected. Default is 1.0.

3. **Expansion**

   ```
   <GAMMA val="2.0" />
   ```

   The `GAMMA` parameter controls how far the worst simplex node is expanded. Default is 2.0.

4. **Contraction**

   ```
   <RHO val="-0.5" />
   ```

   The `RHO` parameter controls how far the worst simplex node is contracted. Default is -0.5.

5. **Reduction**

   ```
   <SIGMA val="0.5" />
   ```

The SIGMA parameter controls how far the worst simplex node is contracted. Default is 0.5.

The simplex mode does not allow loading or saving binary files. To save the text files one should use the following option:

```
<FILE saveTxt="simplex.csv" />
```

See Sec. 2.4.8 for tips and tools on analyzing the output text file.

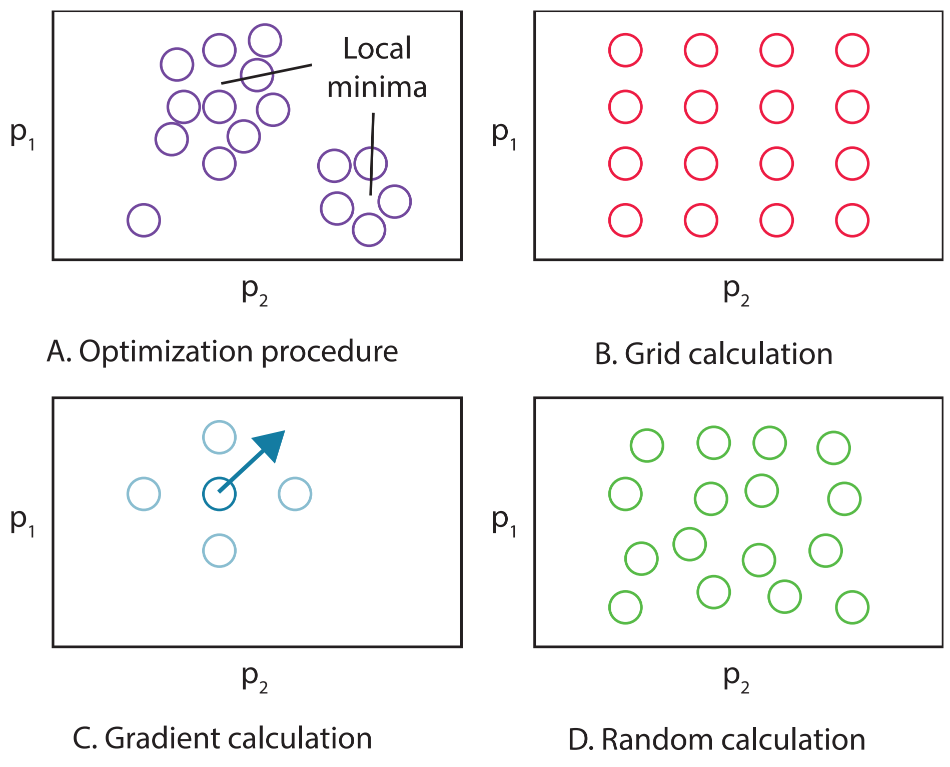


Figure 9: Schemes of different calculation modes.

2.4.5 Grid Parameter Calculation

Grid calculations can be used to find a general landscape of a scoring function (Eq. 11) in a desired parameter range. In this mode the function is calculated for every point on a multidimensional grid, each dimension corresponding to a scoring function argument. See Fig. 9B for a two dimensional grid example. In this mode the optimization XML file has the following structure:

```
<OPTIMIZATION method="grid">
   <PROTOCOL ... />
   <DATA>
      ...
   </DATA>
   <OPTIONS>
      <STEPS val="8" />
      <FILE saveTxt="optimize.result.csv" />
   </OPTIONS>
</OPTIMIZATION>
```

The `PROTOCOL` and `DATA` tags were already described in Sec. 2.4.1. The `val` attribute of the `STEPS` tag corresponds to the number of points p calculated for each dimension, without counting the points corresponding to the minimum and maximum value of the function. $(p+2)^n$ points will be calculated, where n is the number of dimensions, therefore this mode should be used for low number of independent function arguments. The results are saved to a file defined in the `saveTxt` file of FILE tag. Please see Sec. 2.4.8 for tips and tools on analyzing the output file. In this mode there is no option to load and save data in a binary format.

2.4.6 Gradient and Hessian Calculations

The gradient calculation provides a way to analyze the local landscape of the scoring function (Eq. 11). In this mode a particular set of arguments is selected and then a gradient vector or hessian matrix are calculated (see Fig. 9C). Calculating these entities can help (i) understand the effect of changing the parameters and (ii) answer the question if one should use the gradient based optimization methods (the well-established gradient and hessian) or gradient–free methods (with large error on calculating the gradient and hessian). Please note that at the moment RedMDStream can not use the calculated gradient/hessian in the actual optimization.

The derivatives used to find the gradient or hessian are numerically approximated. Instead of an infinitesimally small value $\mathrm{d}x_i$, a small, but noticeable, difference Δ_i is used. To account for this substitution and for possible imperfections of calculating $f(x_1, ..., x_n)$ (Eq. 11), a numerical derivative is found by using 2 points (default)[15]:

$$\frac{\mathrm{d}g(x)}{\mathrm{d}x} \approx \frac{g(x+\Delta) - g(x-\Delta)}{2\Delta} \,, \tag{21}$$

or 4 points:

$$\frac{\mathrm{d}g(x)}{\mathrm{d}x} \approx \frac{-g(x+2\Delta) + 8g(x+\Delta) - 8g(x-\Delta) + g(x-2\Delta)}{12\Delta} \,, \tag{22}$$

or 6 points:

$$\frac{\mathrm{d}g(x)}{\mathrm{d}x} \approx \frac{-g(x+3\Delta) + 9g(x+2\Delta) - 45g(x+\Delta) + 45g(x-\Delta) - 9g(x-2\Delta) + g(x-3\Delta)}{60\Delta} \,. \tag{23}$$

For the gradient calculation the optimization XML file should have the following form:

```
<OPTIMIZATION method="gradient">
   <PROTOCOL ... />
   <DATA>
      ...
   </DATA>
   <OPTIONS>
      <DIFF val="0.001" />
      <FILE save="gradient.csv"
            saveTxt="optimize.result.csv" />
   </OPTIONS>
</OPTIMIZATION>
```

The PROTOCOL and DATA tags were already described in Sec. 2.4.1. The starting point is taken as the average of the lower and higher bounds of each argument present in the DATA section. For the hessian calculation in the first line one should exchange the method attribute to method="hessian". The c coefficient to calculate the derivative is defined in the val attribute of the DIFF tag. The

[15]For clarity of the presentation numerical derivatives are shown for a single argument function, however this also applies to partial derivatives used to find the hessian/gradient.

difference is calculated for each argument separately, as $\Delta_i = c(u_i - l_i)$, where l_i and u_i are, respectively, the lower and upper bound of the i-th argument of the scoring function. In the `OPTIONS` section one may also add the `ACCURACY4` or `ACCURACY6` tag to perform numerical differentiation using the formulas of Eqs. 22 or 23. The gradient calculation mode does not support reading of any file. However, the results of calculations can be stored in a text file name according to the `saveTxt` attribute of the `FILE` tag. Please see Sec. 2.4.8 for tips and tools on analyzing the output text file. One can also find the actual gradient or hessian value, without the details of the performed CG MD simulations, in a file specified in the `save` attribute of the `FILE` tag.

2.4.7 Random Model Calculation

The random test provides a similar role as the grid test (Sec. 2.4.5), but the solutions are randomly picked from the arguments range and not according to any spacing rules (see Fig. 9D). The optimization XML file should be prepared according to the following general format:

```
<OPTIMIZATION method="random">
   <PROTOCOL ... />
   <DATA>
      ...
   </DATA>
   <OPTIONS>
      <ITERATION size="1000" />
      <FILE saveTxt="optimize.result.csv" />
   </OPTIONS>
</OPTIMIZATION>
```

where the `ITERATION` tag corresponds to the number of models to test the (`size` attribute and the `FILE` tag contains information on a text file used to save results. Please see Sec. 2.4.8 for tips and tools on analyzing the output file. The `PROTOCOL` and `DATA` tags were already described in Sec. 2.4.1.

2.4.8 Analysis of the Output File

The optimization output text file can be imported into major spreadsheet, statistical or graph plotting software. The RedMDStream application also contains a program to analyze and aggregate such data (see Sec 2.4.9). For better compatibility a proper selection of the field separator (the `separator` attribute) might be necessary, e.g. for some spreadsheets ";" or "," could be the best choice, but for the Gnuplot software " " (space) will be the most natural choice. The columns are organized in the following order:

- Identification of the model: the iteration number and the solution number inside the iteration. In evolutionary algorithm there is information if a solution is the elite one (if no `recalc="true"` option was set, the elite solutions were moved from the previous generation without recalculation and are redundant). In the output of the simplex algorithm there is an additional column describing which simplex operation was used to generate this solution.
- Force field parameters.

- Average total score and its standard deviation, based on multiple recalculation of the same force field with different random number generator seed.
- Average score and its standard deviation for each scoring function term, based on multiple recalculations of the same force field with different random number generator seed.

The format of the file is also presented in Fig. 10.

The analysis of the results could provide much more information than just the optimal force field parameter set. For example, calculating the correlations of parameters with their score (using the Pearson pairwise correlation coefficient) or plotting and eye inspecting the results, might provide insight which force field parameters are crucial and which do not affect significantly the final results. Also, one can calculate the Pearson pairwise correlation coefficient between the parameter values in a selection of highest–scoring solutions to find if certain parameters are dependent on each other, which might affect the Boltzmann inversion optimizations by different software.

2.4.9 Analyzer of the Optimization File

To help obtain statistics, we provide a `RedMDStream.optAnalyze` tool which allows aggregating and comparing the information from one or multiple optimization runs. Please see an example in Sec. 3.3.6. `RedMDStream.optAnalyze` should be executed in the following way:

```
   RedMDStream.optAnalyze <mode options> <other options>
<list of all CSV text files with optimization outputs>
```

The mode can be set as follows:

- `--all` Generates all statistics described below.
- `--best` Prints solutions according to their total score.
- `--fileCount` Counts the number of solutions in each file.
- `--fileBest` Prints solutions according to their total score, separately for each file.
- `--fileAvg` Prints the average solution for each file.
- `--fileBest` Prints the best solution in each iteration for each file.
- `--fileAvg` Prints the average solution in each iteration for each file.
- `--bestSubscore` Prints the best value of each score subcategory in each file.
- `--paramFunc` For each parameter creates a relationship between its value and the scoring function term. The parameter space is divided into bins and for each bin the best score is found.
- `--paramFunc=<paramName>` The same as above, but only for the parameter termed `paramName`.

Other options:

a.

```
<OPTIMIZATION method="evo">
  <DATA>
    <PARAMETER name="compl1K" min="0.0" max="50.0" />
    <PARAMETER name="compl2K" min="0.0" max="50.0" />
    <SCORE name="RMSD" val="rmsd" max="10.0" weight="4.0" />
    <SCORE name="Compl1_KS" val="ks_compl1" max="1.0" weight="3.0" />
    <SCORE name="Compl2_KS" val="ks_compl2" max="1.0" weight="3.0" />
  </DATA>
  <PROTOCOL runs="5" file="../protocol3.xml" />
  <OPTIONS>
    <ITERATION size="7" number="2" />
    <ELITE val="3" />
    <FILE saveTxt="evo.csv" />
    ...
  </OPTIONS>
</OPTIMIZATION>
```

b.

```
Iter Memb Elite compl1K compl2K Total TotalStd RMSD RMSDStd Compl1_KS Compl1_KSStd Compl2_KS Compl2_KSStd
0 0 0 11.90 10.48 0.44 0.03 4.35 0.30 0.42 0.03 0.47 0.04
0 1 0 26.69 7.34 0.45 0.02 4.53 0.28 0.41 0.01 0.48 0.03
0 2 0 0.52 31.43 0.40 0.02 4.64 0.16 0.23 0.02 0.49 0.02
0 3 0 15.32 0.41 0.41 0.03 4.58 0.17 0.40 0.03 0.35 0.04
```

c. First columns in the optimization output identify solution (algorithm iteration number, member number in iteration, elite status, etc.).

Scoring function paramter values are provided, with name of the parameter as a table heading.

Total score of the solution and its standard deviation is positioned after parameter values and before scoring function term values.

Individual scoring function terms are provided, with name of the term listed in the heading. Values are presented before further normalization. Score values are followed by standard deviation, which describes how well multiple runs of the same protocol agree in score.

Iter	Memb	Elite	compl1K	compl2K	Total	TotalStd	RMSD	RMSDStd	Compl1_KS	Compl1_KSStd	Compl2_KS	Compl2_KSStd
0	0	0	11.90	10.48	0.44	0.03	4.35	0.30	0.42	0.03	0.47	0.04
0	1	0	26.69	7.34	0.45	0.02	4.53	0.28	0.41	0.01	0.48	0.03
0	2	0	0.52	31.43	0.40	0.02	4.64	0.16	0.23	0.02	0.49	0.02
0	3	0	15.32	0.41	0.41	0.03	4.58	0.17	0.40	0.03	0.35	0.04
0	4	0	4.27	1.43	0.40	0.02	4.62	0.14	0.36	0.01	0.37	0.03
0	5	0	46.15	1.18	0.41	0.02	4.57	0.28	0.44	0.01	0.33	0.03
0	6	0	5.32	4.80	0.42	0.02	4.42	0.11	0.37	0.03	0.43	0.02
1	0	1	0.52	31.43	0.40	0.02	4.64	0.16	0.23	0.02	0.49	0.02
1	1	1	4.27	1.43	0.40	0.02	4.62	0.14	0.36	0.01	0.37	0.03
1	2	1	15.32	0.41	0.41	0.03	4.58	0.17	0.40	0.03	0.35	0.04
1	3	0	6.14	0.60	0.42	0.04	4.77	0.46	0.38	0.02	0.38	0.04
1	4	0	0.48	9.26	0.42	0.02	4.77	0.25	0.25	0.02	0.52	0.02
1	5	0	11.52	33.86	0.44	0.02	4.38	0.22	0.40	0.02	0.49	0.02
1	6	0	46.34	32.96	0.46	0.03	4.45	0.16	0.43	0.03	0.49	0.04

Figure 10: Example of an output file from the RedMDStream optimization run. In (**a**) a sample optimization XML file is presented. Parts of the file directly connected with the output file structure are colored. In (**b**) a fragment of the text file is presented with (**c**) a table generated from this data with a spreadsheet software, along with the description and position of fields.

- `-o <filename>` or `--output=<filename>` Specifies the output file. **Compulsory.** (if the `--all` or `--paramFunc` mode is used then the output specifies only a prefix)
- `-s<sep. char>` Field separator has to be consistent with the RedMDStream one.
- `-i <num>` or `--iter=<num>` Read only `<num>` iterations from the files.
- `--minIter=<num>` Read only iterations starting from `<num>` from the files.
- `--maxIter=<num>` Read only iterations up to `<num>` from the files.
- `-r <num>` or `--read=<num>` Read only first `<num>` from the files.
- `-p <num>` or `--print=<num>` Analyze and print only first `<num>` best solutions.
- `-t <num>` or `--threshold=<num>` Analyze only solutions with the total score lower than `<num>`.
- `--paramMin=<num>` `<num>` is used as the lower bound for the parameter value in the `--paramFunc` mode.
- `--paramMax=<num>` `<num>` is used as the upper bound for the parameter value in the `--paramFunc` mode.
- `-b <num>` or `--bins=<num>` `<num>` is taken as the number of bins that divide the parameter space from the lower to the upper bound in the `--paramFunc` mode.

3 Examples

The examples for RedMDStream are in the `Examples` subdirectory and can be found after unpacking the archive with the program. They are divided into categories corresponding to different classes of input files for RedMD.

3.1 Topology XML Files

The topology XML file example folder contains four different examples of force fields. It is out of the scope of this manual to explain these force fields in detail, please refer to a review paper [15] or publications cited as a reference.

3.1.1 One Bead per Nucleotide Ribosome Model

In this example we present the implementation of a one-bead per nucleotide model of the ribosome by Trylska et al. [22]. In this model ribosomal RNA is represented as a set of beads positioned on phosphorus atoms and ribosomal proteins as a set of beads positioned on C_α carbons. Beads are connected by (a) unbreakable harmonic bonds based on their sequence in the chain (1–2, 1–3 and 1–4 bonds), (b) unbreakable harmonic bonds based on their secondary structure (only in RNA) and (c) ”elastic network” interactions, where the beads close to each other in the reference structure are connected by breakable bonds (see Fig. 4). See Ref. [22] or [15] for details.

To run the example enter the `Examples/Topology/Rib_1Bead` directory and run RedMD-Stream:

```
RedMDStream protocol.xml
```

RedMDStream will load the *E. coli* 70S ribosome structure present in `3DF3.xml.gz` and `3DF4.xml.gz` PDBML files. Next, the program will apply the coarse–grained topology described in the `topology.xml` file. The RedMD SXML `ribosome.sxml` output file will be created, containing the coarse–grained structure with all the interactions, as well as `ribosome.cg.pdb` with a coarse–grained structure. To better visualize the interactions additional PDB files are generated, each applying a different bond rule:

- `ribosome.cg.Pbond.pdb` (RNA 1–2 sequence interactions), `ribosome.cg.CAbond.pdb` (protein 1–2 sequence interactions),
- `ribosome.cg.Psec.pdb` (RNA secondary structure interactions),
- `ribosome.cg.Pdist.pdb` (RNA ”elastic network” interactions),
- `ribosome.cg.CAdist.pdb` (protein ”elastic network” interactions) and
- `ribosome.cg.PCAdist.pdb` (RNA–protein ”elastic network” interactions).

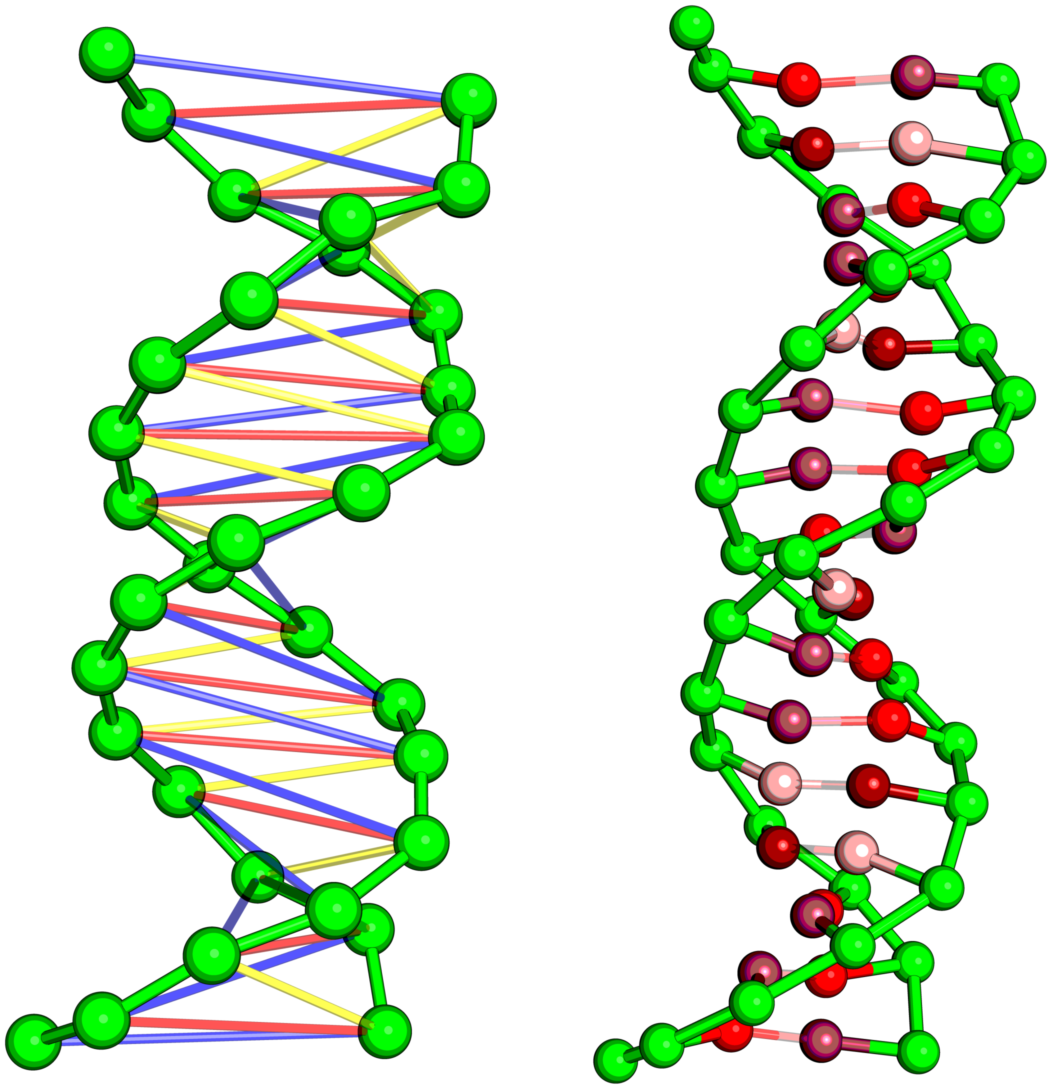


Figure 11: 16-base pair B–DNA helix (PDB code: 3BSE) in (a) a one bead per nucleotide DNA model and (b) two beads per nucleotide model. In the first model (a) the green beads are centered on phosphorus atoms of the original structure, the green bonds go along the P-backbone, the blue bonds are the *i:j* interactions, red *i:(j+1)* and yellow *(i+1):(j+1)*. In the second model (b) the green beads represent a backbone group, the red beads represent a base (different shades of red represent different nucleotides). The interactions plotted are backbone–backbone bonds, backbone–base and base–base bonds (complementary interactions, transparent due to their breakable nature).

3.1.2 One Bead per Nucleotide DNA Model

In this example we present the implementation of a one-bead per nucleotide model for DNA, including entire plasmids, by Trovato et al. [21]. In this model nucleotides are represented as single beads centered on phosphorus atoms. Beads from one chain are connected by pseudo–bonds, pseudo–angles and pseudo–dihedrals. Beads connected by complementary pairing are connected with 3 bonds (*i:j*, *i:(j+1)*, *(i+1):(j+1)*). For details please refer to Ref. [21] or [15] and Fig. 11a.

To run the example enter the `Examples/Topology/DNA_1Bead` directory and run RedMD-Stream:

```
RedMDStream protocol.xml
```

RedMDStream will load 16–base–pair B-DNA structure stored in the `3BSE.xml` PDBML file and apply coarse–grained topology described in the `topology.xml` file. The RedMD SXML `3BSE.sxml` output file will be created, containing the coarse–grained structure, along with tabulated potentials and all interactions, as well as `3BSE.cg.pdb` with a coarse–grained structure. To better visualize the interactions additional PDB files are generated, each applying a different bond rule:

- `3BSE.cg.bond.pdb` (RNA sequential pseudo–bonds),
- `3BSE.cg.ij.pdb` (RNA *i:j* complementary pairing),
- `3BSE.cg.ij1.pdb` (RNA *i:(j+1)* complementary pairing) and
- `3BSE.cg.i1j1.pdb` (RNA *(i+1):(j+1)* complementary pairing).

3.1.3 One Bead per Nucleotide RNA Model

In this example we present a one-bead model of RNA optimized using the evolutionary algorithm by Leonarski et al. [14]. The interaction pattern in the model is based on the DNA model of Trovato et al. [21] presented in the previous paragraph. There are two optimized potential force field parameter sets. One set is for equilibrium dynamics and the other one for structure prediction. As noted in [14], the force field has to be parameterized for a specific purpose.

To run the example enter the `Examples/Topology/RNA_1Bead` directory and run RedMD-Stream:

```
RedMDStream protocol.EqDyn.xml
```

```
RedMDStream protocol.StrPred.xml
```

The first protocol XML file creates two files: RedMD SXML `6TNA.EqDyn.sxml` and PDB `6TNA.EqDyn.pdb`. The second protocol XML file creates files for the structure prediction model and in this case produces the RedMD SXML file `6TNA.StrPred.sxml`. Along with this file a set of PDB files is outputted corresponding to a reference structure and unfolded circle structure:

- structure without any bonds annotated
  (`6TNA.StrPred.pdb` and `6TNA.StrPred.unfolded.pdb`),
- structure with sequential pseudo–bonds
  (`6TNA.StrPred.PPbond.pdb` and `6TNA.StrPred.unfolded.PPbond.pdb`)
- complementary *i:j* pairing
  (`6TNA.StrPred.ij.pdb` and `6TNA.StrPred.unfolded.ij.pdb`),
- complementary *i:(j+1)* pairing
  (`6TNA.StrPred.ij1.pdb` and `6TNA.StrPred.unfolded.ij1.pdb`) and
- complementary *(i+1):(j+1)* pairing
  (`6TNA.StrPred.i1j1.pdb` and `6TNA.StrPred.unfolded.i1j1.pdb`).

3.1.4 Two Beads per Nucleotide DNA Model

In this example we present the implementation of two-bead per nucleotide model for DNA designed by Drukker et al. [7]. In the model each nucleotide is represented by two beads: one in the center of mass of the phosphate and sugar (the backbone bead) and the second one in the center of mass of the base. The backbone beads are the same for every nucleotide and the base beads are nucleotide–type specific. The interactions in the model include the bonded interactions: backbone–backbone, backbone–base, base–backbone–backbone angle, base–backbone–backbone–base dihedral and complementary pairing. The complementary pairing potential energy is implemented as a product of an angular term (backbone$_1$–base$_1$–base$_2$) and distance term (base$_1$ and base$_2$). See Ref. [7] or [15] and Fig. 11b for details.

To run the example enter `Examples/Topology/DNA_2Bead` directory and run RedMDStream:

```
RedMDStream protocol.xml
```

RedMDStream will load 16–base–pair B-DNA structure stored in `3BSE.xml` PDBML file and apply coarse–grained topology described in the `topology.xml` file. The RedMD SXML `3BSE.sxml` file will be created, containing the coarse–grained structure, along with tabulated potentials and all interactions, as well as `3BSE.cg.pdb` with a coarse–grained structure. To visualize the interactions additional PDB files are generated, each applying a different bond rule:

- `3BSE.cg.bond.pdb` (RNA sequential pseudo–bonds),
- `3BSE.cg.ij.pdb` (RNA *i:j* complementary pairing),
- `3BSE.cg.ij1.pdb` (RNA *i:(j+1)* complementary pairing) and
- `3BSE.cg.i1j1.pdb` (RNA *(i+1):(j+1)* complementary pairing).

3.2 Protocol XML Files

The protocol XML files contain three examples to test different features of RedMDStream: distance distribution calculations, MD simulation, and structure prediction. One-bead per nucleotide RNA representation, with beads centered on phosphorus, is used in the following examples, along with Leonarski et al. force field. [14]

3.2.1 Distribution calculation

In this example the calculation of distance distribution is presented from both a set of structures and trajectory.

To run the example enter the `Examples/Protocol/Distribution` directory and run RedMDStream:

```
RedMDStream protocol.xml
```

RedMDStream will calculate the distributions for structures found in the `rRNA` subdirectory. First, the distribution of all P–P distances will be calculated, divided by the $d_0(r) = r^2$ reference function and saved to the `rRNA.all.txt` file. Second, the distribution of the distances between the first–neighbor pseudo–bond connected beads will be calculated and saved internally as `PneighDistr` and in a text file `rRNA.neigh.txt`. Finally, a similar sequential pseudo–bond distribution will be calculated for a trajectory in `Trajectory`. This distribution will be stored in a file `traj.neigh.txt` and compared with a saved rRNA distribution using Kolmogorov–Smirnov test. The distribution-containing text files can be opened with gnuplot or other plotting tool for inspection.

3.2.2 Simulation

In this example we present the files used to run a short 2 ns simulation of a 35 base-pair RNA helix. The simulation is carried out using: (a) Langevin dynamics, (b) NVT ensemble with Berendsen thermostat, (c) Brownian dynamics and (d) NVE ensemble in 310 K.
To run the example enter `Examples/Protocol/Simulation` directory and run RedMDStream:

```
RedMDStream protocol.langevin.xml
RedMDStream protocol.berendsen.xml
RedMDStream protocol.brownian.xml
RedMDStream protocol.nve.xml
```

Simulation trajectory will be saved in a DCD file and RMSD value will be printed as standard output. One RedMDStream protocol XML file can include simulations in different ensembles. The four tests mentioned above can be executed with a single file in the same directory as above:

```
RedMDStream protocol.xml
```

A `helix35.cg.pdb` will be created with a CG representation of the simulated helix. For visualization in VMD, PyMol or other software this PDB should be loaded as a template prior to loading the DCD files. DCD files contain only coordinates in a binary format and do not carry information about atom and residue names and types.

3.2.3 Structure Prediction Simulation

In this example a one-bead per nucleotide structure prediction model for RNA [14] is applied to predict the structure of a specifier domain of the *B. subtilis* tyrS T box leader hairpin.
To run the example enter the `Examples/Protocol/StrPred` directory and run RedMDStream:

```
RedMDStream protocol.xml
```

RedMDStream loads the structure, unfolds it to a circular conformation, and simulates it for 20 ns using the Berendsen thermostat in 310 K. The average RMSD over 1000 frames is printed out to a standard output and a trajectory to the `2KHY.StrPred.dcd` file.

3.3 Optimization XML Files

Finally, we provide the optimization XML example files. These examples should be considered as a toy model. In general, the optimization of the model and force field parameters requires tedious preparation and testing of parameter ranges, scoring function and protocol. So usage of high performance computing resources is advised. Here, we provide the examples which can be calculated on a desktop computer in a reasonable time and provide a basic introduction to the optimization procedure using RedMDStream. Three example are presented:

1. 1 ns simulation of a folded 35 base-pair A-RNA helix with 4 varying parameters — RMSD minimization.
2. 100 ps simulation of an unfolded 35 base-pair A-RNA helix with 10 varying parameters — RMSD minimization.
3. 500 ps simulation of a folded 35 base-pair A-RNA helix with 6 varying parameters — RMSD and distribution similarity minimization.

The last subsection provides short instructions on how to analyze the output data.

3.3.1 Evolutionary Algorithm

To run the example of the evolutionary algorithm enter the `Examples/Optimization/Evo` directory and run RedMDStream:

```
RedMDStream optimize1.xml
RedMDStream optimize2.xml
RedMDStream optimize3.xml
```

RedMDStream will perform optimization and generate output files: `evo1.csv`, `evo2.csv`, `evo3.csv`.

See Sec. 2.4.8 and 2.4.9, as well as an example in Sec. 3.3.6 for instructions how to analyze the output files.

3.3.2 Particle Swarm Optimization

To run the example of the particle swarm optimization enter the `Examples/Optimization/PSO` directory and run RedMDStream:

```
RedMDStream optimize1.xml
RedMDStream optimize2.xml
RedMDStream optimize3.xml
```

RedMDStream will perform optimization and generate output files: `pso1.csv`, `pso2.csv` and `pso3.csv`. See Sec. 2.4.8 and 2.4.9, as well as an example in Sec. 3.3.6 for instructions how to analyze the output files.

3.3.3 Simplex Optimization

To run the example of the simplex optimization enter the `Examples/Optimization/Simplex` directory and run RedMDStream:

```
RedMDStream optimize1.xml
RedMDStream optimize2.xml
RedMDStream optimize3.xml
```

RedMDStream will perform optimization and generate output files: `simplex1.csv`, `simplex2.csv` and `simplex3.csv`. See Sec. 2.4.8 and 2.4.9, as well as an example in Sec. 3.3.6 for instructions how to analyze the output files.

In addition we provide an example showing how the evolutionary algorithm results stored in a file
`Examples/Optimization/Evo/evo1.csv` and `Examples/Optimization/Evo/evo2.csv` are re-optimized by the simplex algorithm using the scoring function from example no 1. The evolutionary algorithm example has to be executed before executing the commands below. The example runs as follows:

```
RedMDStream optimize1.evo1.xml
RedMDStream optimize1.evo2.xml
```

RedMDStream will select the best models from the evolutionary algorithm run file, as a starting point for the simplex method. The output files `simplex1.evo1.csv` and `simplex1.evo2.csv` will be generated.

3.3.4 Grid Parameter Calculation

To run the particle grid parameter calculation enter the `Examples/Optimization/Evo` directory and run RedMDStream:

```
RedMDStream grid1.xml
```

RedMDStream performs the optimization and generates the output `grid1.csv` file. Due to high cost of increasing the number of dimensions in a grid calculation, only example 1 is suitable for this type of calculations.

3.3.5 Gradient Calculation

To run the gradient optimization example enter the `Examples/Optimization/Evo` directory and run RedMDStream:

```
RedMDStream gradient1.xml
RedMDStream gradient2.xml
RedMDStream gradient3.xml
```

RedMDStream performs optimization and generates the following output files: `gradient1.csv`, `gradient2.csv`, `gradient3.csv` (with the results of simulations used to find gradients), and `gradient1.val.csv`,
`gradient2.val.csv`, `gradient3.val.csv` (with the actual gradient values). See Sec. 2.4.8 and 2.4.9, as well as an example in Sec. 3.3.6 for instructions how to analyze the output files.

3.3.6 Analysis of subsequent runs

To compare the quality of the optimization methods the output files can be analyzed using the `RedMDStream.optAnalyze` tool (see Sec. 2.4.9). To obtain the best potential with the scores in each method write:

```
RedMDStream.optAnalyze --fileBest simplex1.csv evo1.csv pso1.xml -o best.csv
```

The output file `best.csv` will print the best potential in each file — its parameters, the total score (with the st. deviation) and the scoring function terms. In a similar way one may obtain the average score in each file:

```
RedMDStream.optAnalyze --fileAvg simplex1.csv evo1.csv pso1.xml -o avg.csv
```

One may also aggregate the three files and use them to find the relationship between the parameter value and total scoring function:

```
RedMDStream.optAnalyze --paramFunc simplex1.csv evo1.csv pso1.xml -o func
```

This command will generate many files, all starting with the `func` prefix, followed by a dot and name of the parameter and the `csv` extension. For example, in the `func.bondR0.csv` one will find the `bondR0` values (the distance of bonded interaction), along with the lowest (best) total score obtained with a particular `bondR0`[16] and with any value of other parameters. Note that such statistics requires a large amount of data and even then the obtained line will be noisy, requiring smoothing with the spline function. Even though this kind of statistical analysis should be taken with caution, the general relationship *parameter–score* can be inferred.

[16]The parameter range is divided into bins and the smallest (best) score is chosen for the parameters inside each bin. The number of bins is controlled with the `-b <num>` option and bin range with the `--paramMin=<num>` or `--paramMax=<num>` options to `RedMDStream.optAnalyze`

References

[1] Andersen, H. Rattle: a velocity version of the shake algorithm for molecular dynamics calculations. *J. Comput. Phys. 52* (1983), 24–34.

[2] Baker, N. A., Sept, D., Joseph, S., Holst, M. J., and McCammon, J. A. Electrostatics of nanosystems: application to microtubules and the ribosome. *Proc. Natl. Acad. Sci. USA 98* (2001), 10037–10041.

[3] Bernstein, F. C., Koetzle, T. F., Williams, G. J. B., Meyer, J. E. F., Brice, M. D., Rodgers, J. R., O. Kennard, T. S., and Tasumi, M. The protein data bank: A computer-based archival file for macromolecular structures. *J. Mol. Biol. 112* (1977), 535–542.

[4] Bourne, P. E., Berman, H. M., McMahon, B., Watenpaugh, K. D., Westbrook, J., and Fitzgerald, P. M. The macromolecular crystallographic information file (mmCIF). *Methods Enzymol. 277* (1997), 571–590.

[5] Bray, T., Maler, E., Paoli, J., Yergeau, F., and Sperberg-McQueen, M. Extensible markup language (XML) 1.0 (fifth edition). W3C recommendation, W3C, Nov. 2008. `http://www.w3.org/TR/2008/REC-xml-20081126/`.

[6] Brooks, B. R., Bruccoleri, R. E., Olafson, B. D., States, D. J., Swaminathan, S., and Karplus, M. CHARMM: a program for macromolecular energy, minimization, and dynamics calculations. *J. Comp. Chem. 4(2)* (1983), 187–217.

[7] Drukker, K., Wu, G., and Schatz, G. C. Model simulations of dna denaturation dynamics. *J. Chem. Phys. 114*, 1 (2001), 579–590.

[8] Górecki, A., Szypowski, M., Długosz, M., and Trylska, J. RedMD – reduced molecular dynamics package. *J. Comput. Chem. 30* (2009), 2364–2373.

[9] Górska, A., Jasiński, M., and Trylska, J. MINT: Motif identier for nucleic acids trajectory. to be submitted, `http://mint.cent.uw.edu.pl`.

[10] Haupt, R. L., and Haupt, S. E. *Practical Genetic Algorithms*, 2 ed. Wiley-Interscience, May 2004.

[11] Hodgkin, E., and Richards, W. Molecular similarity based on electrostatic potential and electric field. *Quantum Biol. Symp.*, 14 (1987), 105–110.

[12] Kabsch, W. A solution for the best rotation to relate two sets of vectors. *Acta Crystallogr., Sect. A: Cryst. Phys., Diffr., Theor. Gen. Crystallogr. 32*, 5 (Sep 1976), 922–923.

[13] Kennedy, J., and Eberhart, R. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks* (Nov 1995), vol. 4, pp. 1942–1948 vol.4.

[14] Leonarski, F., Trovato, F., Tozzini, V., Leś, A., and Trylska, J. Evolutionary algorithm in the optimization of a coarse–grained force field. *J. Chem. Theory Comput. 9* (2013), 4874–4889.

[15] LEONARSKI, F., AND TRYLSKA, J. Modeling nucleic acids at the residue–level resolution. In *Computational methods to study the structure and dynamics of biomolecules and biomolecular processes - from bioinformatics to molecular quantum mechanics*, A. Liwo, Ed. Springer, 2013, pp. 109–149.

[16] LEONTIS, N., AND WESTHOF, E. Geometric nomenclature and classification of rna base pairs. *RNA 7* (2001), 499–512.

[17] MATHEWS, D. H., AND TURNER, D. H. Prediction of RNA secondary structure by free energy minimization. *Curr. Opin. Struct. Biol. 16*, 3 (June 2006), 270–278.

[18] NELDER, J. A., AND MEAD, R. A simplex method for function minimization. *Comp. J. 7*, 4 (1965), 308–313.

[19] PRESS, W. H., TEUKOLSKY, S. A., VETTERLING, W. T., AND FLANNERY, B. P. *Numerical Recipes 3rd Edition: The Art of Scientific Computing.* Cambridge University Press New York, NY, USA, 2007.

[20] RYCKAERT, J. P., CICCOTTI, G., AND BERENDSEN, H. J. C. Numerical integration of the cartesian equations of motion of a system with constraints: Molecular dynamics of n-alkanes. *J. Comput. Phys. 23* (1977), 327–341.

[21] TROVATO, F., AND TOZZINI, V. Supercoiling and local denaturation of plasmids with a minimalist DNA model. *J. Phys. Chem. Lett. 112* (2008), 13197–13200.

[22] TRYLSKA, J., TOZZINI, V., AND MCCAMMON, J. A. Exploring global motions and correlations in the ribosome. *Biophys. J. 89* (2005), 1455–1463.

[23] WESTBROOK, J., ITO, N., NAKAMURA, H., HENRICK, K., AND BERMAN, H. M. PDBML: the representation of archival macromolecular structure data in XML. *Bioinformatics 21* (2005), 988–992.

[24] WILKINSON, K. A., MERINO, E. J., AND WEEKS, K. M. Selective 2-hydroxyl acylation analyzed by primer extension (SHAPE): quantitative RNA structure analysis at single nucleotide resolution. *Nat. Protoc. 1* (2006), 1610–1616.

[25] YANG, H., JOSSINET, F., LEONTIS, N., CHEN, L., WESTBROOK, J., BERMAN, H., AND WESTHOF, E. Tools for the automatic identification and classification of RNA base pairs. *Nucl. Acids Res. 31* (2003), 3450–3460.