Phaistos User Manual

Version 1.0

Wouter Boomsma, Sandro Bottaro, Thomas Hamelryck,
Jes Frellsen, Christian Andreetta, Mikael Borg, Tim
Harder, Kristoffer E. Johansson, Kasper Stovgaard,
Pengfei Tian

Contents

Chapter 1

Introduction

Phaistos is a molecular modelling software package for simulation of proteins. The distinctive features of Phaistos are the use of the Markov chain Monte Carlo (MCMC) approach to molecular simulation, and the use of sophisticated probabilistic models of protein structure in continuous space. Phaistos can be used for molecular simulation using empirical energy functions such as OPLS-AA or Profasi, or for probabilistic, Bayesian inference of protein structure based on a likelihood and conformational priors. The latter applications include for example the inference of protein structure using a likelihood that brings in NMR or SAXS data, and priors that takes into account the general features of protein structure. Phaistos covers a wide spectrum of applications, as it supports a variety of settings for the three key features of a Monte Carlo simulation: the move set, the acceptance criterion and the energy function.

Some of the features available in Phaistos include:

Probabilistic models of protein structure

- TorusDBN: a probabilistic model of the protein main chain [1].
- BASILISK: a probabilistic model of amino acid side chains [2].
- FB5HMM: a probabilistic model of a protein's $C\alpha$ trace [3].
- COMPAS: a probabilistic model of the center of mass of amino acid side chains [4].

Move set

- Pivot, semilocal and local backbone moves in internal coordinate space.
- Pivot, semilocal and local backbone moves in internal coordinate space from probabilistic models that capture the local structure of proteins [1].

- Rotamer moves, where tentative updates of one or more degrees of freedom are drawn from a uniform or Gaussian distribution.
- Rotamer moves from a probabilistic model that capture the side chain geometry of amino acids [2].

Simulation type

- Metropolis-Hastings.
- Greedy optimization.
- Simulated annealing.
- Generalized ensembles: multicanonical and $1/k$ [5].

Energy and likelihood functions

- The OPLS-AA force field [6] with GB/SA implicit solvent model [7].
- The Profasi force field [8].
- Likelihood functions for SAXS [9, 4] and NOE [10] data.
- Go-type potentials.

In addition, a number of built-in analysis routines are available (e.g fraction of native contacts, RMSD and helicity calculation, cluster analysis, etc.).

A website containing an online version of this manual and the source code of Phaistos can be found at *http://sourceforge.net/projects/phaistos/*. Bug reports, questions and comments regarding Phaistos should be done through the Sourceforge site.

1.1 Citing Phaistos

The main reference for Phaistos is:

W. Boomsma, J. Frellsen, T. Harder, S. Bottaro, K. E. Johansson, P. Tian, K. Stovgaard, C. Andreetta, S. Olsson, J. Valentin, L. D. Antonov, A. S. Christensen, M. Borg, J. H. Jensen, K. Lindorff-Larsen, J. Ferkinghoff-Borg, T. Hamelryck, J. Comput. Chem. 2013, DOI: 10.1002/jcc.23292.

Chapter 2

Installation

Phaistos requires a C++ compiler, a fortran compiler, the boost library[1], lapack[2] and CMake[3]. In addition, certain optional modules may have additional requirements.

2.1 Installation procedure

The recommended way to build Phaistos is out-of-source, meaning that the generated binaries will not clutter the source directory. This is done using the following sequence of commands:

```
$ mkdir build
$ cd build
$ cmake /path/to/phaistos/
$ make
```

CMake can be configured from the command line using the -D syntax. For instance, to specify a non-standard location of the boost directory, use:

```
$ cmake -DBOOST_ROOT:FILEPATH=directory ..
```

The CMake output will contain a list of packages that were successfully found and the corresponding CMake variables that can be modified if necessary. A full list of the available CMake variables can be obtained using:

```
$ cmake -LA ..
```

By default, CMake will hide the details of the compilation process. If you wish to see the executed commands, simply type:

```
$ make VERBOSE=1
```

[1] www.boost.org
[2] www.netlib.org/lapack/
[3] www.cmake.org

2.2 Build type

Phaistos is by default built in release-mode, meaning that the produced code is optimized for fast execution. During debugging, it is often an advantage to compile to non-optimized code. This can be done by modifying the BUILD_TYPE variable:

```
$ cmake -DCMAKE_BUILD_TYPE:STRING=Debug ..
```

If you are not satisfied with the default compiler settings, you can always override them yourself:

```
$ cmake -DCMAKE_CXX_FLAGS="-ggdb -Wall" ..
```

2.3 Modules

Phaistos has a module system, which makes it easy to write your own code (module) and compile it in with the main library. Parts of the Phaistos functionality is provided as modules, to allow the user to exclude it during compilation, which can speed up compilation times significantly. CMake will output a list of all detected modules:

```
-- Modules - phase: 1
--   Found modules:
--     git
--     muninn
--     pleiades
...
```

You can disable modules using the `PHAISTOS_MODULE_DISABLE` and `PHAISTOS_MODULE_ENABLE` options. Both take regular expressions specifying which modules to disable or enable, respectively. For instance, if we wish to include only the `git` module, this can be done by enabling it directly

```
$ cmake -DPHAISTOS_MODULE_ENABLE:STRING="git" ..
```

or by disabling everything else

```
$ cmake -DPHAISTOS_MODULE_DISABLE:STRING="muninn|pleiades|..." ..
```

In the list of detected modules, CMake will indicate which modules are currently disables:

```
-- Modules - phase: 1
--   Found modules:
--     git
--     muninn (disabled)
--     pleiades (disabled)
```

CMake variables are cached, so if you specify an option once it will be remembered in future executions of CMake. Therefore, remember to explicitly remove the PHAISTOS_MODULE_DISABLE or PHAISTOS_MODULE_ENABLE option value if you no longer need it. This can be done by setting it to the empty string:

```
$ cmake -DPHAISTOS_MODULE_DISABLE:STRING="" ..
```

2.4 Constructing a source tar-ball

You can create a tar-ball of all Phaistos source files by using the following command:

```
$ make package_source
```

This will create a file called phaistos-XXX.tar.gz where XXX, is the current version number.

2.5 Locating data files (PHAISTOS_ROOT)

Phaistos uses a number of data files, which are typically referred to using a path relative to the execution directory. This means that phaistos.cpp will be able to find these files when started from the bin directory in your build directory, but it might fail if you try to run it from another directory. This problem can be resolved by setting the PHAISTOS_ROOT environment variable to point to the root of your build directory. In the bash shell, this can be done using:

```
$ export PHAISTOS_ROOT='/path/to/phaistos/build/'
```

while tcsh requires a slightly different syntax:

```
> setenv PHAISTOS_ROOT '/path/to/phaistos/build/'
```

Chapter 3

Running simulations using `phaistos.cpp`

The main executable in the Phaistos package is called `phaistos.cpp`. This program is a front-end to basically all functionality in the library. Settings for `phaistos.cpp` are specified using either command line options or a configuration file. Most settings in Phaistos have default values, so typically, a simulation can be set up by specifying only a few parameters. As the simulation starts, it will output all used settings to the screen, allowing the user to verify that the current configuration is meaningful. This output is formatted so that it is compatible with the configuration file format, meaning that changes to the current setting can be made simply by copy-and-pasting from the initial output to a configuration file.

3.1 Command line options

`phaistos.cpp` uses long unix-style command line options of the form:

`--option-name` *`optional-value`*.

All available options for `phaistos.cpp`, along with their default values, can be obtained by using with the help options:

```
$ ./phaistos --help
```

For convenience, `phaistos.cpp` has shorthand options, which will automatically expand to their longer equivalents. The most important examples are the `move` and `energy` shorthands. These allow you to quickly specify sets of moves and energy terms. For instance

```
$ ./phaistos --energy clash-fast opls-angle-bend opls-vdw
```

corresponds to

```
$ ./phaistos --energy-clash-fast --energy-opls-angle-bend \
  --energy-opls-vdw
```

The individual energy terms might have options of their own. These can be specified in square brackets in the shorthand notation:

```
$ ./phaistos --energy clash-fast[debug:1,weight:10]
```

This will expand to

```
$ ./phaistos --energy-clash-fast --energy-clash-fast-debug 1 \
  --energy-clash-fast-weight 10
```

3.1.1 Procedures

The `phaistos.cpp` executable contains several main functions, which address different types of simulation problems. These main functions are called *procedures*, and can be selected from the command line using the `--procedure` option. The default procedure is `fold`, which runs a general protein simulation. There is a procedure called `compactify`, which rapidly produces compact structures, and debug procedure called `comparison`. The latter allows the user to easily compare results of two energy functions, for instance to verify that a cached version of an energy produces the same result as the non-cached version.

Procedures have sub-options which are relevant only to that particular procedure. For instance, the `fold` procedure has options to control whether debug information should be written out to screen during a simulation. This can be specified using the nested style introduced in the previous section:

```
$ ./phaistos --procedure fold[debug:true]
```

3.1.2 Modes

Modes are shortcuts that associate a name to a combination of specific simulation settings that are often used. Currently, there are only a few predefined modes, but more will be added in the future. A mode is selected using the `--mode` command-line option. One example is the `opls-mc-dynamics` mode, which will be illustrated in an example below (see section 3.3.1).

3.2 Configuration files

As the `phaistos.cpp` program is started, it will output a complete list of settings, including both the ones you specified from the command line, and default values for all other settings. This output, starting with

```
##################### PHAISTOS OPTIONS ####################
```

and ending in

```
################### PHAISTOS OPTIONS END ##################
```

is in a format that `phaistos.cpp` can read in as a configuration file. If you copy-and-paste these options into a file `filename`, and run Phaistos using

```
$ ./phaistos --config-file filename
```

it will use the same settings as those provided in the command line originally. This gives a convenient workflow for setting up a simulation: start by providing a few settings for the command line, copy-and-paste the resulting output to a configuration file, and then carefully read through (and modify) this configuration file to ensure that you have the desired settings.

3.3 Examples

We include a few examples of possible simulation configurations. This is mainly to give a broad overview of how simulations are set up. The move, energy, and simulation settings will be covered in detail in the next chapters.

3.3.1 Studying a native ensemble using OPLS

The `opls-mc-dynamics` mode provides a simple example of a simulation using the OPLS force field:

```
$ ./phaistos --mode opls-mc-dynamics --pdb-file 2gb1.pdb \
  --init-from-pdb
```

We explicitly state which PDB file we are using, and that you wish to initialize the simulation from this PDB file. The rest of the settings are given by the `opls-mc-dynamics` mode. This mode expands to settings for the complete OPLS forcefield, and a variety of moves that were found to be optimal in the context of exploring native ensembles [11]. In addition, the simulation type is set to Metropolis-Hastings. Note that both the `crisp` and `opls` modules must be enabled for this mode to work (which they are by default).

A mode simply provides default values for certain settings. Everything can be overridden by specifying additional options from the command line. For instance, you can adjust the number of iterations and dump pdb files at certain intervals by adding the following options to the command line:

```
--iterations 100000000 --observable pdb[register-interval:20000]
```

3.3.2 Generalized ensemble simulation using Profasi

Profasi is a highly efficient forcefield, that has been used in the literature both for folding and aggregation simulations [12]. This, in combination with the Muninn generalized ensemble framework [5], is a powerful tool for conducting rapid equilibrium folding simulations.

A simple version of such a simulation could look like this:

```
./phaistos --aa-file 1L2Y.aa --move pivot-uniform sidechain-uniform \
 --energy profasi-cached --threads 2 \
 --monte-carlo muninn[min-beta:0.05,max-beta:1.25]
```

In this case, we start from an amino acid sequence file, containing just a single line of amino acids:

```
NLYIQWLKDGGPSSGRPPPS
```

In addition, we specify that we wish to use uniform moves both for the protein backbone and sidechain, and that the simulation is done using the cached version of the profasi forcefield. We use the Muninn generalized ensemble framework for this simulation, and set up the temperature range (`min-beta` and `max-beta`). Finally, the `threads` option specifies that we wish to run with two threads. This has the effect that we will be conducting two independent simulations, which uses the same set of weights and both will contribute to the same histograms and entropy estimates in Muninn.

Chapter 4

Moves

Phaistos includes a range of different approaches to update the molecule chain in each iteration of the simulation. These are referred to as *moves*. Unless explicitly mentioned, all moves in Phaistos obey detailed balance. This means that if you simulate with for instance standard Metropolis-Hastings and no energy function, you can expect flat distributions over the dihedral angle degrees of freedom of your system. More generally, if you simulate with Metropolis-Hastings and a given *physical* energy function, you have the Boltzmann distribution as your target distribution, see section 5.1 for details.

4.1 General move settings

All moves share the following basic settings. The `weight` setting determines how often the move is used in the simulation, while the `regions` option allows you to specify where in the chain you want to apply the move. The `debug` flag can be used (if implemented by the author of the move) to output additional information about the move.

name	description
`weight`(=1.0)	Relative weight of the move (probability of choosing this move).
`debug`(=0)	Debug level.
`regions`(=[[0,chain_size]])	Regions affected by move.

4.2 Uniform pivot move (`pivot-uniform`)

Perhaps the most classic Monte Carlo move in molecular simulations, the pivot-uniform simply produces a random, uniformly distributed rotation of the dihedral angles (ϕ,ψ angles) in a single residue. The `max-delta` option determines the maximum size of the rotation. You can use the

`single-dof-only` option to limit yourself to a single degree of freedom. The `skip-proline-phi` option makes it possible to omit the ϕ dihedral in Prolines, which involves a change in bond length and should be avoided if no bond length term is included in the energy.

Settings

name	description
`move-length-min`(=1)	Minimum length of move.
`move-length-max`(=1)	Maximum length of move.
`single-dof-only`(=true)	Whether to update only one ϕ or ψ angle at a time (randomly chosen).
`skip-proline-phi`(=true)	Whether to sample angles in proline. Note, if `skip-proline-phi` is set to false, a change in proline bond length is introduced, which must be taken in account by an appropriate bond-stretch energy term.
`max-delta`(=π(rad))	Maximum change in angle value.

4.3 Backbone DBN move (`backbone-dbn`)

The Backbone DBN move is essentially a pivot move using a non-trivial probability distribution. More precisely, angular degrees of freedom are sampled from a probabilistic model of the protein backbone. Currently, two models are available: TorusDBN and Fb5Dbn [1, 3], using the full-atom and the C_α-only representation respectively. The relevant model is selected using the `backbone-dbn` option (see chapter 8). When using TorusDBN, the move basically resamples (ϕ,ψ) dihedral angles from a Ramachandran like distribution.

> IMPORTANT: This move introduces a bias (implicit energy) to the simulation that is possible to compensate by setting `implicit-energy`=false.

Settings

name	description
`move-length-min`(=1)	Minimum length of move.
`move-length-max`(=7)	Maximum length of move.
`resample-mode`(=RESAMPLE_ALL)	Which values to resample in the model.
`implicit-energy`(=true)	Whether the bias introduced by the dbn move should be included as an implicit energy.

dbn-consistency-window-size(=10)	Whenever a dbn move is combined with a different move, the dbn will need to be made consistent with the current state of the protein after every non-dbn move. Ideally, this requires a full-length resampling (corresponding to option value -1). However, since the dbn has a finite memory, a window of 10 is a reasonable approximation.
dbn-bias-window-size(=10)	To evaluate the bias introduced by a dbn move, ideally, a full-length forward calculation must be done. However, since the dbn has a finite memory, a window of 10 is a reasonable approximation.

4.4 Local pivot move (pivot-local)

The pivot-local move modifies (ϕ, ψ) angles and optionally non-standard degrees of freedom (e.g. bond angles or ω angles) in the backbone. Updates are proposed from a Gaussian distribution centered on the current value and with user-controlled variance.

Settings

name	description
move-length-min(=1)	Minimum length of move.
move-length-max(=1)	Maximum length of move.
sample-phi-psi-dofs(=true)	Whether to modify (ϕ, ψ) angles.
sample-bond-angle-dofs(=true)	Whether to modify bond angles.
sample-omega-dofs(=false)	Whether to modify omega angles.
std-dev-phi-psi(=4)	Parameter controlling the variance of the Gaussian distribution for (ϕ, ψ) angles (degrees).
std-dev-bond-angle(=0.8)	Parameter controlling the variance of the Gaussian distribution for bond angles (degrees).
std-dev-omega(=0.8)	Parameter controlling the variance of the Gaussian distribution for ω angles (degrees).

There is also a version of this move that includes an implicit DBN energy for dihedral angles and an implicit bond angle energy based on Engh-Huber statistics. This variant is called pivot-local-dbn-eh.

4.5 Uniform sidechain move (`sidechain-uniform`)

The `sidechain-uniform` move produces random, uniformly distributed rotations of the sidechain torsion angles (χ angles) in single residues.

Settings

name	description
`move-length-min`(=1)	Minimum length of move.
`move-length-max`(=1)	Maximum length of move.
`single-dof-only`(=true)	Whether to update only one χ angle at a time.
`skip-proline`(=true)	Whether to sample sidechain angles in proline. Note, if `skip-proline` is set to false a change in proline bond length is introduced, which must be taken in account by an appropriate bond-stretch energy term.

4.6 Rotamer sidechain move (`sidechain-rotamer`)

The rotamer move (`sidechain-rotamer`) modifies the χ-angles in the sidechain. New angles are proposed from the Gaussian distributions in the Dunbrack backbone-independent rotamer library [13].

> IMPORTANT: This move introduces a bias (implicit energy) to the simulation that is possible to compensate by setting `implicit-energy`=false.

Settings

name	description
`move-length-min`(=1)	Minimum length of move.
`move-length-max`(=1)	Maximum length of move.
`implicit-energy`(=true)	If implicit-energy is set to false, the bias introduced when proposing from the Dunbrack library is compensated for, ensuring that the uniform distribution is reached when simulating with no force-field.
`--sigma-scale-factor`(=1.0)	The width of the Dunbrack Gaussian distributions from which new angles are proposed can be scaled by this factor. Flattening the distribution can be useful to ensure ergodicity, as the Dunbrack library does not have support over the entire interval $[0, 2\pi[$.

`--rotamer-state-resample-frequency`(=1.0)	Frequency with which move will resample new rotamer states (g^+, t, g^-).
`--rotamer-skip-proline`(=true)	Whether to sample sidechain angles in proline. Note, if rotamer-skip-proline is set to false a change in proline bond length is introduced, which must be taken in account by an appropriate bond-stretch energy term.
`--sample-hydrogen-chis`(=true)	Whether to sample (uniformly) non-standard torsion angles in sidechain (e.g methyl groups, CB-S-H in Cys, etc...).

4.7 Local sidechain move (`sidechain-local`)

The `sidechain-local` move modifies χ angles and optionally non-standard degrees of freedom (e.g. bond angles) in the sidechain. Four different strategies can be employed:

simple Tentative updates are proposed for all the degrees of freedom in the sidechain from a Gaussian distribution centered on the current angle(s) and with user-controlled variance

scale-sigma Tentative updates are proposed for all the degrees of freedom in the sidechain from a Gaussian distribution centered on the current angle(s). The width of the Gaussians for the different angles in the sidechain is scaled linearly so that degrees of freedom further away from CA have larger variance.

select-dofs Tentative updates are proposed for one degree of freedom in the sidechain from a Gaussian distribution centered on the current angle and with user-controlled variance. The degree of freedom is chosen at random according to a weight given by its distance to the CA atoms, such that angles closer to the backbone are sampled less frequently.

constrain-endpoint Tentative updates are proposed for all the degrees of freedom in the sidechain from a multivariate Gaussian distribution with a bias towards small displacement of atoms involved as acceptors or donors in hydrogen bonds. This type of move is is useful to enable small adjustment of the sidechain without breaking the dense network of non-covalent interactions.

Settings

name	description
`move-length-min`(=1)	Minimum length of move.
`move-length-max`(=1)	Maximum length of move.

`sample-major-dofs`(=true)	Whether to sample χ angles.
`sigma-major-dofs`(=0.03490)	Width (in radians) of the Gaussian distribution for χ angles.
`sample-minor-dofs`(=true)	Whether to sample minor degrees of freedom (all bond-angles and non-χ dihedral angles).
`sigma-minor-dofs`(=0.00349)	Width (in radians) of the Gaussian distribution for minor dofs angles.
`mode`(=simple)	Four different move modes are available: simple, scale-sigma. select-dofs, constrain endpoint (described above).
`rotamer-lagrange-multiplier`(=200)	When using constrain-endpoint mode, this parameter controls the weight of the constraint.
`skip-proline`(=true)	Whether to sample sidechain angles in proline. Note, if `skip-proline` is set to false a change in proline bond length is introduced, which must be taken in account by an appropriate bond-stretch energy term.

4.8 Crankshaft move (`crankshaft`)

The CRANKSHAFT move consists of a rigid rotation of a chain segment about the axis defined by the C_α atoms delimiting the segment [14]. This move therefore alters only the backbone dihedral angles ϕ and ψ and the C_α bond angles of the segment ends. The move is implemented as described in reference [15]. The original procedure includes an optimized placement of C_β and H_α atoms, which does not fulfill detailed balance, which is therefore omitted in this implementation.

Settings

name	description
`move-length-min`(=2)	Minimum length of move.
`move-length-max`(=12)	Maximum length of move. The number of residues involved in a single update is randomly chosen in the interval [`move-length-min`,`move-length-max`].
`constrain-bond-angle`(=true)	Whether to apply a constraint on the bond angles variation.
`bond-angle-tolerance`(=15)	If constrain-bond-angle is true, this parameter sets the maximum variation in bond angles (in degrees) from the original structure.

only-internal-moves(=false)	Whether to perform only internal moves or both internal and end-moves. End-moves consist is small variations of bond- and torsion angles.

4.9 CRISP local move (crisp)

CRISP (Concerted Rotations Involving Self-consistent Proposals) is a Monte Carlo move that produces deformations in a small segment of the protein backbone, keeping the positions of all atoms outside the segment fixed. Compared to other local move approaches [14, 16], CRISP makes it possible to generate subtle and uniform tentative updates without disrupting the local geometry of the chain. For a complete description of the method, please refer to our recent paper on the move [11].

Settings

name	description
move-length-min(=5)	(see below)
move-length-max(=5)	By default a CRISP move modifies all the bond and torsion backbone angles of 5 consecutive residues at a time. The number of residues involved in a single update is randomly chosen in the interval [move-length-min,move-length-max].
std-dev-phi-psi(=5)	Parameter controlling the amplitude (in degrees) of ϕ and ψ angles variation. Large values correspond to large changes. The default value of this setting, together with the default value of std-dev-bond-angle and std-dev-omega are optimized for using OPLSaa potential in combination with GB/SA solvation energy.
std-dev-bond-angle(=0.8)	Parameter controlling the amplitude (in degrees) of bond angles variation. Large values correspond to large changes.
std-dev-omega(=0.8)	Parameter controlling the amplitude (in degrees) of omega angles variation. Large values correspond to large changes.
only-internal-moves(=false)	Whether to perform only internal moves or both internal and end-moves. End-moves consist of small variations of bond- and torsion angles.
sample-omega(=true)	Whether to modify omega angles, in addition to ϕ,ψ and bond angles during prerotation.

CRISP can also be used in combination with the TorusDBN model, using the move-crisp-dbn-eh version. In this case, a standard CRISP move is performed. Upon the update, the likelihood of the new structure is calculated according to the TorusDBN model (for torsion angles) and to the Engh-Huber values (for bond-angles) [17]. The approach is equivalent to proposing tentative structures according to the prior distribution governing bond and torsion degrees of freedom.

IMPORTANT: The dbn-eh version of this move introduces an implicit energy to the simulation.

Settings specific for dbn-eh version

name	description
`consistency-window-size`(=10)	Size of window used (to each side) when bringing the dbn back to consistency. A good value for the window size is > 7, and a negative window size means that the full hidden node sequence is resampled.
`bias-window-size`(=10)	Size of window used when calculating bias. Approximates the move bias as $P(X[i-w, j+w])/P(X'[i-w, j+w])$, where w in the window size and $[i, j]$ is the interval where angles have been changed. A good value for the window size is $w > 7$, and a negative window size means that the full bias is used.

4.10 Semi-local move (`semilocal`)

The BGS (Biased Gaussian Step) move is a semi-local move in which all the angular degrees of freedom in a small stretch of the chain are modified [18]. The atomic positions from the move-end to the end of the chain are rigidly updated. In the BGS move, tentative updates are drawn from a Gaussian distribution that favors approximately local deformations of the chain.

Settings

name	description
`move-length-min`(=4)	(see below)
`move-length-max`(=4)	By default a BGS move modifies 8 (ϕ, ψ) angles in 4 consecutive residues. The number of residues involved in a single update is randomly chosen in the interval [`move-length-min`,`move-length-max`].

only-internal-moves(=false)	Whether to perform only internal moves or both internal and end-moves. End-moves consist is small variations of bond- and torsion angles
sample-omega(=true)	Whether to modify omega angles, in addition to (ϕ, ψ) and bond angles.
sample-bond-angle(=false)	Whether to modify backbone bond angles, in addition to (ϕ, ψ) torsion angles.
sample-constraint-a(=300)	Global parameter controlling the size of the update. Large values correspond to small variations.
sample-constraint-b(=10)	Parameter controlling the locality constraint. Large values correspond to a heavy constraint on the end-point displacement.
omega-scaling(=8)	If sample-omega is true, the allowed omega angles variation is scaled down by this factor compared to (ϕ, ψ) angles.
bond-angle-scaling(=8)	If sample-bond-angle is true, the allowed bond-angles variation is scaled down by this factor compared to (ϕ, ψ) angles.

This move also has an dbn-eh version, which works in the same way as explained for the CRISP move.

IMPORTANT: The dbn-eh version of this move introduces an implicit energy to the simulation.

4.11 CRA local move (cra)

CRA (Concerted Rotations with Angles) is a Monte Carlo move that produces deformations in a small segment of the protein backbone, keeping the positions of all atoms outside the segment fixed. The method is constructed around the same idea as the BGS move: limiting the movement of the end-point of the pre-rotation, in order to increase the probability of finding a solution for the post-rotation [16].

Settings

name	description
move-length-min(=5)	(see below)
move-length-max(=5)	By default a CRA move modifies all the bond and torsion backbone angles of 5 consecutive residues at a time. The number of residues involved in a single update is randomly chosen in the interval [move-length-min,move-length-max]

`implicit-energy`(=false)	Whether to include an implicit energy for the bond angles. If set to true and no other moves that modifies bond angles are used, the bond angle term can be omitted in the energy function.

4.12 Sidechain DBN moves

The sidechain DBN moves (`sidechain-basilisk` and `sidechain-compas`) allow you to sample amino acid side chain conformations from one of the included probabilistic models of side chain geometry, namely COMPAS or BASILISK [2]. These moves will propose a new set of angles for a randomly selected stretch of residues. We recommend only using a stretch of length one, or in other words, to only resample a single side chain at a time. Changing multiple sidechains in one go will increase the likelihood of introducing clashes, leading to a significantly higher rejection rate.

Both moves use an external model file. Please remember to set the `PHAISTOS_ROOT` environmental variable to avoid problems accessing these files (see section 2.5).

4.12.1 General settings

The behavior of the two moves is identical in terms of parameters. However, the moves applies to different side chain representation: BASILISK requires full atomic details, whereas COMPAS only requires a pseudo atom at the side chain center of mass. Both sidechain DBN moves have the following options.

name	description
`move-length-min`(=1.)	Minimum move length.
`move-length-max`(=1.)	Maximum move length.
`mocapy-dbn-dir`(=../data/mocapy_dbns)	Path to mocapy model file directory.
`implicit-energy`(=*true*)	Whether the dbn bias (implicit energy) should be divided out (=false) or not (=true).
`ignore-bb`(=*false*)	Whether to use the backbone independent version instead.
`check-move`(=*true*)	Check that the move produced consisted atom positions.
`sample-hydrogen-chis`(=*false*)	Sample sidechain residual dofs.
`sample-hydrogen-chis-normal`(=1.)	Resampling constrained to a Gaussian around the current state.
`sample-hydrogen-chis-sigma`(=0.035)	Standard deviation of Gaussian for resampling residual dofs.

model-filename(=basilisk.dbn/compas.dbn)	Model filename.

IMPORTANT: Both variants of the move introduce a bias (implicit energy) to the simulation that is possible to compensate by setting implicit-energy=false.

4.12.2 BASILISK move (sidechain-basilisk)

BASILISK assumes a full atom representation of the protein sidechain. There are several variants of this move available:

sidechain-basilisk, sidechain-basilisk-local, and sidechain-basilisk-multi.

The local version samples the hidden node state based on the current angle values and subsequently samples angles corresponding to this state, which generally has the effect of small angle variations. The multi variant will do a sidechain move in two disjunct regions of the protein chain. See appendix G for details.

4.12.3 COMPAS move (sidechain-compas)

COMPAS uses a pseudo-sidechain representation, with one atom per sidechain. See appendix G for details.

4.13 None move (none)

The none move does no structural resampling. It is included because some energy terms are constructed to resample their own parameters, in which case it can be convenient to have a Monte Carlo iteration where the structure is not modified.

4.14 Selecting moves in phaistos.cpp

Moves can be selected from the phaistos.cpp command line either individually

```
$ ./phaistos --move-pivot-uniform --move-sidechain-uniform
```

or using a shorthand

```
$ ./phaistos --move pivot-uniform sidechain-uniform
```

When using the shorthand option, indivudial move options can be specified using square brackets as described in section 3.1.

Chapter 5

Energies

5.1 Phaistos energy convention

To ensure consistency between probabilistic terms and physical energy terms, energies in Phaistos, E_ϕ, are reported as the negative logarithm of the probability of observing a structure. This means that

$$E_\phi(x) = -\log P(x) \ , \tag{5.1}$$

where $P(x)$ is the probability of observing the structure x according to the energy term, and $E_\phi(x)$ is the corresponding Phaistos energy. As a consequence Phaistos energies are unitless.

For classical physics based energy terms, this means that the Phaistos energy becomes $E_\phi(x) \propto \beta E(x)$, where E the physical energy function and $\beta = (kT)^{-1}$ is the thermodynamic beta. Here, k is the Boltzmann constant and T is the temperature. The β weight is specified using the `weight` setting of the individual energy terms. Physical energies will often be carrying a default weight of 1.67857 at 300 K if the physical energy is measured in kcal/mol. This weight corresponds to the value of $(kT)^{-1}$ for $T = 300\,\mathrm{K}$ and $k = 1.9872159 \cdot 10^{-3} \frac{\mathrm{kcal}}{\mathrm{mol\,K}}$.

5.2 General energy settings

The following settings are shared by all energy terms. The `weight` is multiplied onto the energy by the energy term itself and consequently before the summation of energy terms when calculating the total energy of a structure. Note, the energy values outputted during a simulation is alway multiplied by the weight associated with the energy term. For physical energy terms, the weight corresponds to usual $1/kT$ factor (see previous section). Just as for the moves, all energies also have a `debug` flag, which indicates that the energy evaluation should execute in debug mode.

name	description
`weight`(=1.0)	Absolute weight used when evaluating the weighted energy used for summing energy terms.
`debug`(=0)	Debug level.

5.3 Clash detection (`clash-fast`)

Phaistos employs a ChainTree data structure to quickly determine interacting atoms within a given distance [19]. The `clash-fast` energy function uses this data structure to do a coarse-grained self-collision check, returning infinity if atoms are found to be within the given cutoffs. This makes it useful as a first energy term in an energy function, acting as a quick filter. This works because remaining energy terms are skipped if the energy becomes infinite during the evaluation.

Settings

name	description
`only-modified-pairs`(=true)	Specifies whether the energy should consider only pairs modified by the last move. If a clashfree structure is maintained at all times, this is sufficient to detect all clashes.
`boolean-mode`(=true)	Specifies whether the energy should work in clash/non-clash mode and return infinity/0 (true) or count all the clashes and return the number of clashes (false).
`minimum-residue-distance`(=2)	Minimum distance along chain (measured in number of residues) before pair is taken into account by the energy (in angstrom).
`clash-distance-h`(=1.5)	Distance within which pairs of hydrogens are considered to be clashing (in angstrom).
`clash-distance-no`(=2.3)	Distance within which pairs of Nitrogen-Oxygen pairs are considered to be clashing (in angstrom).
`clash-distance-ps`(=1.9)	Distance within which pairs of pseudo-sidechain atom pairs are considered to be clashing (in angstrom).
`clash-distance-sg`(=1.8)	Distance within which pairs of SG atom pairs are considered to be clashing (in angstrom).
`clash-distance-any-pair`(=2.3)	Default distance within which pairs of atom pairs of all other types are considered to be clashing (in angstrom).

5.4 Backbone DBN energy (`backbone-dbn`)

This is a backbone energy that uses one of the probabilistic models described in chapter 8. The model itself is selected using the `backbone-dbn` option (see chapter 8).

In principle, when another move modifies the backbone angles of a segment of the protein backbone, this affects the distribution over all hidden nodes in the sequence according to the probabilistic model. In practice, we know that the models have a finite memory, and it is sufficient to recalculate only a window around the changed segment. The size of this window is controlled using the `window-size` parameter.

Settings

name	description
`enable-dbn-update`(=true)	Whether to update the angles in the DBN from the chain when necessary.
`always-full-update`(=false)	Whether to always force update of all angles in the DBN from the chain.
`window-size`(=-1)	Size of window used when calculating the energy. A good value for the window size is > 7, and a negative window size means that the full bias is used.

5.5 MuMu: Non-local multibody energy

The multibody multinomial (MuMu) model is a coarse grained probabilistic model of non-local interactions in proteins [20].

The main usage of this module is for residue probability annotation of PDB files. This is easily done with the following line:

```
./evaluate_observable --pdb-file xxxx.pdb --observable-mumu
output-target=pdb-b-factor
```

Note that the current implementation does not check for missing atoms or modified residues. The module will use the atoms provided by the Phaistos PDB parser. Missing or unknown atoms will result in erroneous evaluations.

The usage as a potential of mean force in a folding simulation will require a reference model which is not provided. However, with a weight of 2-5 the energy will stabilize most structures.

Settings

name	description
`use-ratio`(=0)	Use the ratio model. This is not available.

`model-filename`(=default)	Name of the Mocapy *.dbn model file. If set to 'default' the file name will be read from mumu_data_miner.h
`reference-model-filename`(=default)	Name of the Mocapy *.dbn reference model file. If set to 'default' the file name will be read from mumu_data_miner.h. Not provided.

5.6 Implicit solvent model

Phaistos includes an implementation of the Generalized Born/solvent accessible surface area (GBSA) implicit solvent model [7]. It exists in two versions: a basic implementation and an implementation that uses the Phaistos' ChainTree datastructure [19] to speed up the calculation and cache contributions between iterations.

5.6.1 GBSA standard implementation (`gbsa`)

This standard implementation is extremely computationally intensive, and is primarily intended for debugging purposes.

5.6.2 GBSA cached implementation (`gbsa-cached`)

This version has been optimized so that it exploits the ChainTree data structure both when determining Born radii and for the subsequent calculation of solvation energies. Due to the inherent structure of the GBSA expression, it is difficult to obtain great speedups using caching. However, this version is typically around a factor 10 faster than the standard version mentioned above. The parameters below determine how aggressively to reuse calculations between iterations.

Settings

name	description
`maximum-deviation-cutoff`(=0.1)	Maximum deviation allowed in born radii in two subtrees of the chaintree before it is recalculated.
`cutoff-distance-phase1`(=inf)	Distance beyond which contributions are set to zero in phase1.
`cached-cutoff-distance-phase2`(=inf)	Distance beyond which contributions are set to zero in phase2.

5.7 OPLS all-atom energy

This implementation of the OPLS-AA/L all atom force field [6] is based on the TINKER molecular modeling suite version 4.2 [21].

The OPLS-AA/L energy is composed of 6 terms: A Lennard-Jones potential term (attractive van der Waals forces plus steric repulsion), a partial charge interaction term, a torsional angle energy term, a bond angle bending energy term, a bond stretching energy term and a term for out-of-plane bending of trivalent atoms (improper torsion). The non-bonded terms have been merged into a single loop for optimized performance. This non-bonded term has furthermore been implemented in a cached version that uses the Phaistos ChainTree to rapidly locate the parts of the chain that needs updating (see section 5.7.7).

To include all the OPLS and GBSA energy terms in a simulation, you can use the short-hand command line options:

`--energy opls` or `--energy opls-cached`

5.7.1 Torsional angle energy (`opls-torsion | opls-torsion-cached`)

This term calculates the energy contribution from the local backbone configuration (dihedral angles). The cached version only recalculates the segment that has been resampled in a given iteration.

5.7.2 Angle bend energy (`opls-angle-bend | opls-angle-bend-cached`)

This term calculates the contribution from bond angles fluctuations. The cached version only recalculates the segment that has been resampled in a given iteration.

5.7.3 Bond stretch energy (`opls-bond-stretch`)

This term evaluates the deviation of bond lengths from their ideal values. Note that this is rarely a degree of freedom in Monte Carlo moves.

5.7.4 Improper torsion energy (`opls-improper-torsion`)

This term implements an expression for out-of-plane bending of trivalent atoms. Note that this is rarely a degree of freedom in Monte Carlo moves.

5.7.5 Partial charge interaction energy (`opls-charge | opls-charge-cached`)

This term evaluates partial charge interactions. The cached version uses the ChainTree to cache previously calculated contributions.

Settings (for the cached version)

name	description
`cutoff-distance`(=inf)	Distance beyond which contributions are set to zero.

5.7.6 van der Waals interaction energy (`opls-vdw | opls-vdw-cached`)

Van der Waals interactions (Lennard-Jones potential) are evaluated by this term. The cached version uses the ChainTree to cache previously calculated contributions.

Settings (for the cached version)

name	description
`cutoff-distance`(=inf)	Distance beyond which contributions are set to zero.

5.7.7 Non-bonded interaction energy (`opls-non-bonded | opls-non-bonded-cached`)

This term gathers all non-bonded OPLS terms and the GBSA solvent calculation into a single term, thereby speeding up calculations. The cached version of this term is the recommended way to run OPLS+GBSA simulations.

Settings (for the cached version)

name	description
`vdw-cutoff-distance`(=inf)	Distance beyond which vdw contributions are set to zero.
`charge-cutoff-distance`(=inf)	Distance beyond which vdw contributions are set to zero.
`gbsa-maximum-deviation-cutoff`(=0.1)	Maximum deviation allowed in born radii in two subtrees of the chaintree.
`gbsa-cutoff-distance-phase1`(=inf)	Distance beyond which gbsa contributions are set to zero in phase1.
`gbsa-cutoff-distance-phase1`(=inf)	Distance beyond which gbsa contributions are set to zero in phase2.

5.8 PROFASI energy

The PROFASI force field implemented in Phaistos is an implicit solvent all-atom potential for simulations of protein folding and aggregation. The potential is developed through studies of structural and thermodynamic properties of 17 peptides with diverse secondary structure [12]. It is composed of four terms

$$E = E_{\text{loc}} + E_{\text{ev}} + E_{\text{hb}} + E_{\text{sc}} \ , \tag{5.2}$$

where E_{loc} is a local electrostatic potential between atoms separated by a few covalent bonds, E_{ev} is a potential of the excluded volume effect, E_{hb} is a hydrogen bond potential, and E_{sc} represents interactions between sidechains. These four terms are divided into seven parts in the implementation for the sake of optimization. The individual terms are briefly discussed below. There are various versions of each term. Typically, either the `cached` or the `improved` versions are faster than the standard implementation. Which of the two are fastest depends on the length of the protein. To give an idea of the speeds, we include speedup factors for a particular protein (PDB Code 1lq7). In a given simulation scenario, the user is recommended to try both variants to see which one is most efficient.

5.8.1 Profasi local energy (`profasi-local | profasi-local-cached`)

This term calculates the E_{loc} along the backbone, including the Coulomb interactions between partial charged atoms on the neighbouring peptide along the backbone, and in addition, the repulsion of the HH atoms and the OO atoms between neighbouring peptides.

The cached version of the profasi local energy term, uses the ChainTree data structure, which will give the same profasi local energy value with around 10 times faster calculation speed.

5.8.2 Profasi local sidechain energy (`profasi-local-sidechain | profasi-local-sidechain-cached`)

This term calculates the E_{loc} along the sidechains, which is the explicit torsional angle potential.

The cached version of the profasi local sidechain energy term uses the ChainTree data structure, which will give the same energy value with around 6 times faster calculation speed.

5.8.3 Profasi excluded volume energy (`profasi-excluded-volume | profasi-excluded-volume-cached`)

This term calculates the non-local part of the E_{ev}, by summing over all the excluded volume potential between pairs of atoms that are separated by non-constant disance and more than three convalent bonds.

The cached version of the profasi non-local excluded volume energy term uses the ChainTree data structure, which will give the same energy value with around 15 times faster calculation speed.

5.8.4 Profasi local excluded volume energy (`profasi-excluded-volume-local | profasi-excluded-volume-local-cached`)

This term calculates the local part of the E_{ev} that sums over all the excluded volume potential between pairs of atoms that are separated by non-constant distance and three convalent bonds.

The cached version of the profasi local excluded volume energy term uses the ChainTree data structure, which will give identical energy value to local excluded volume with around 13 times faster calculation speed.

5.8.5 Profasi hydrogen bond energy (`profasi-hydrogen-bond | profasi-hydrogen-bond-cached | hydrogen-bond-improved`)

This term calculates the E_{hb} by summing up backbone-backbone and side-chain-backbone hydrogen bonding interactions.

The cached version of the profasi hydrogen bond energy term uses the ChainTree data structure, which will give identical energy value to hydrogen bond energy with around 1.8 times faster calculation speed.

The improved version of the profasi hydrogen bond energy term uses a term-specific optimization technique, which will give identical energy value to hydrogen bond energy term with around 4.4 times faster calculation speed.

5.8.6 Profasi hydrophobicity energy (`profasi-hydrophobicity | profasi-hydrophobicity-cached | profasi-hydrophobicity-improved`)

This term calculates the hydrophobic part of E_{sc}.

The cached version of the profasi hydrophobicity energy term is using the ChainTree method, which will give identical energy value to hydrophobicity energy with around 1.7 times faster calculation speed.

The improved version of the profasi hydrophobicity energy term uses a term-specific optimization method, which will give identical energy value to hydrophobicity energy term with around 1.1 times faster calculation speed.

5.8.7 Profasi sidechain charge energy (profasi-sidechain-charge | profasi-sidechain-charge-cached | profasi-sidechain-charge-improved)

This term calculates the part of E_{sc} which is due to sidechain charge-charge interaction.

The cached version of the profasi sidechain charge energy term uses the ChainTree method, which will give identical energy value to sidechain charge energy with around 1.5 times faster calculation speed.

The improved version of the profasi sidechain charge energy term uses a term-specific optimization method, which will give identical energy value to sidechain charge energy term with around 1.4 times faster calculation speed.

5.9 SAXS energy (saxs-debye)

The SAXS energy term provides support for sampling under restraints from Small Angle X-ray Scattering (SAXS) data. It is described in detail in Appendix F.

Settings

name	description
debug(=0)	Print debugging information. 5: Basic information on loading the data bases. 10: Preparation of the error model. Energy is printed for every evaluation. 50: Filling the form factors for the molecule in exam. Sine lookup table extensive validation: computes the output of the sin() function, and warns the user if the absolute discrepancy with the lookup table is above 10^{-3}. 100: Full internal debug information.

`saxs-intensities-filename`	The reference experimental reading. It should be discretized at the same q bins as the form factors database (see next option). The discretization is performed at $q = 0.015\text{Å}^{-1}$.
`saxs-form-factors-filename`	The form factors database, as available as open data from our article [9]. It should be discretized at the same q bins as the experimental reading.
`exp-errors-alpha`(=0.15)	Influences the amplitude of the scattering error, as per ref. [9].
`exp-errors-beta`(=0.30)	Influences the amplitude of the scattering error, as per ref. [9].
`q-bins`(=51)	Sets the total number of elements read from the `saxs_intensities_filename` and `saxs_form_factors_filename` data bases.
`q-bins-first`(=0)	Sets the initial element of the `saxs_intensities_filename` and `saxs_form_factors_filename` data bases (can be used to skip the initial bins of the profile).
`one-body-model`(=false)	Use the one- or two-body model per amino acid. The two-body model is slower but significantly more accurate at higher q-values (see ref. [9]).
`sine-lookup-table`(=true)	Use a pre-computed sine table in the Debye formula. Depending on your hardware, this can improve the speed of the Debye computation up to 3x.

5.10 BASILISK explicit energy (`basilisk`)

This sidechain DBN energy terms allows you to use the BASILISK model of amino acid sidechain geometry as an *explicit* energy (rather than the *implicit* term that arises when sampling from the model and not dividing out the bias). This can of interest when one wishes to evaluate (or compensate for) the bias introduced by sampling from the model. It may also be used to evaluate the likelihood (score) of the side chain geometry. The returned energy is the negative log likelihood as calculated from the side chain models.

The BASILISK energy term can be used on side chain conformations in full atomic detail. Using the `ignore-bb` option, it is possible to turn off the backbone dependency of the energy.

Both moves use an external model file. Please remember to set the `PHAISTOS_ROOT` environmental variable to avoid problems accessing these files (see section 2.5)..

For more information on the BASILISK model please consult our BASILISK publication [2].

Settings

name	description
`mocapy-dbn-dir`(=../data/mocapy_dbns)	Path to mocapy model file directory.
`model-filename`(=basilisk.dbn)	Filename of the mocapy model.
`use-cache`(=*true*)	Use the cached version of the term (should be faster).
`ignore-bb`(=*false*)	Whether to omit the backbone conformation (backbone-independent mode)

5.11 COMPAS explicit energy (`compas`)

The COMPAS energy term is the pseudo-sidechain atom equivalent to the full-atom `basilisk` term. It makes it possible to evaluate an explicit energy for the COMPAS model (which is normally included implicitly through the use of the COMPAS move). The COMPAS model is described in detail in reference [4].

Settings

name	description
`mocapy-dbn-dir`(=../data/mocapy_dbns)	Path to mocapy model file directory.
`model-filename`(=compas.dbn)	Filename of the mocapy model.
`use-cache`(=*true*)	Use the cached version of the term (should be faster).
`ignore-bb`(=*false*)	Include the backbone conformation for more accurate evaluation.

5.12 Distance constraints (`constrain-distances`)

The `constrain-distances` term can be used to restraint distances between arbitrary atom in a simulation. Usually starting from a compact, near native conformation, this term allows to set up a network of distance restraints that will then be kept fixed over the course of the simulation. The individual distances are modeled by Gaussian distributions.

Settings

name	description
`network-filename`(=optional)	Restore network from a file.
`pdb-file`	Path to PDB input file (see below).

`include-bb-hbond`(=true)	Include H bonds in the network.
`include-sc-hbond`(=true)	Include sidechain-sidechain hydrogen bonds in the network.
`include-bb-sc-hbond`(=true)	Include sidechain-sidechain hydrogen bonds in the network.
`include-ca-contacts`(=true)	Include CA-CA contacts in the network.
`include-fixed-point-contacts`(=false)	Include contacts between atoms and fixed points in space.
`include-ss-bond`(=true)	Include disulfide bonds (SS bonds) in the network.
`init-hbond-from-native`(=false)	Whether to initialize ideal H bond distances from PDB file (rather than averages from the Top500 database).
`prune-network`(=true)	Whether to prune/optimize the hbond network.
`dehydron-bb-cutoff`(=14)	Cutoff below which backbone-backbone hydrogen bonds are considered as dehydrated aka weak/broken.
`dehydron-bb-sc-cutoff`(=9)	Cutoff below which backbone-sidechain hydrogen bonds are considered as dehydrated aka weak/broken.
`dehydron-sc-cutoff`(=7)	Cutoff below which sidechain-sidechain hydrogen bonds are considered as dehydrated aka weak/broken.
`ca-distance`(=5)	Default CA-CA cutoff distance.
`ss-distance`(=3)	Default CYS(SG)-CYS(SG) cutoff distance.
`ca-skip`(=4)	How many residues to skip along the chain before considering a Calpha contact.
`distances-bb-hbond-skip`(=1)	How many residues to skip along the chain before considering a backbone-backbone hydrogen bond contact.
`distances-sc-hbond-skip`(=1)	How many residues to skip along the chain before considering a sidechain-sidechain hydrogen bond contact.
`distances-bb-sc-hbond-skip`(=1)	How many residues to skip along the chain before considering a backbone-sidechain hydrogen bond contact.
`ss-skip`(=1)	How many residues to skip along the chain before considering a disulfide contact.
`generate-pymol`(=false)	Whether to generate a Python script that will visualize the network in PyMOL.
`use-caching`(=true)	Whether to cache interations (recommended).
`verbose`(=false)	Whether to print out additional information (recommended).

5.13 Distance disulfide constraints (constrain-disulfide-bonds)

This is a simplified front-end to the `constrain-distances` energy term. It is an easy way to fix (as in unbreakable) the disulfide bonds in a structure during a simulation.

Settings

name	description
`network-filename`(=optional)	Restore network from a file.
`include-ss-bond`(=true)	Include disulfide bonds (SS bonds) in the network.
`ss-distance`(=3)	Default CYS(SG)-CYS(SG) cutoff distance.
`ss-skip`(=1)	How many residues to skip along the chain before considering a disulfide contact.
`generate-pymol`(=false)	Whether to generate a Python script that will visualize the network in PyMOL.
`use-caching`(=true)	Whether to cache interations (recommended).
`verbose`(=false)	Whether to print out additional information (recommended).

5.14 Visible volume

The visible volume measure of L. L. Conte and T. F. Smith [22] is calculated using an algorithm by K. E. Johansson. Two additional measures based on this algorithm, called angular exposure and first shell coordination number, are also implemented. All three measures can be dumped as observables and a simple solvation energy can be calculated using the angular exposure in combination with hydrophobic solvation parameters from FoldX [23].

5.14.1 Visible volume

Visible volume is a residual measure of the volume that is accessible for a side chain. Observing from the side chain geometrical center, the volume behind neighboring atoms is excluded as invisible and the remaining volume gives the space available for the side chain. A sphere of radius `sphere-radius` is considered and this setting determines the maximum volume available. The algorithm probes the space in a number of descrete *angle points* dependent on the setting of `min-sphere-points`.

5.14.2 Angular exposure

The fraction of the sphere surface not covered by neighbors is calculated as a measure of the solvent exposure of the side chain. This measure is highly correlated with relative solvent accessible surface area (SASA).

5.14.3 First shell coordination number

The conventional coordination number used in the context of protein structure counts the number of atoms within some cutoff distance of a central atom. The visible volume algorithm gives information on which neighbors within the sphere are visible and which are occluded by closer neighbors.

The visible neighbors constitutes the first shell of coordinated atoms. However, the first shell of neighbors does not constitute a solid shade and more distant neighbors may be visible between them. Because of this, it is often useful to set a threshold for a first shell neighbor. The algorithm considers angular space in discrete points and the option `fscn-threshold` sets the number minimum number of visible angle points from a single neighbor to define it as coordinated.

5.14.4 Options

All three measures can be dumped to stdout using

```
./evaluate_observable --pdb-file xxxx.pdb
--observable-visible-volume volume=1 angexp=1 fscn=1
```

or to the B-factor field in a PDB file by adding `output-target=pdb-b-factor`. The latter usage requires that the user select a single measure. When used as an energy, only the angular exposure is used and the others may be disabled for enhanced performance. Residual energies cannot be outputted. When printed to a text file, all enabled measures are printed to a single line.

Settings

name	description
`sphere-radius`(=11)	Neighbors within this distance are considered (angstroms)
`min-sphere-points`(=1000)	Minimum number of angle points used in the algorithm. The actual number of angle points used by the algorithm is rounded up to constitute a regular mesh.
`atom-radius`(=1.8)	Radius of the disk representing atom neighbors in angstroms.
`volume`(=1)	Toggle calculation of visible volume when used as observable.
`angexp`(=0)	Toggle calculation of angular exposure when used as observable.

fscn(=0)	Toggle calculation of first shell coordination number when used as observable.
fscn-threshold(=35)	Threshold value used in the calculation of first shell coordination number. At least this many angle points must be visible to consider a neighbor as coordinated. This should be adjusted to the total number of angle points used in the algorithm (dependent on min-sphere-points).

5.15 Contact-map energy (contact-map)

This is an artificial energy term that can be used to drive proteins to the correct fold by providing native contact information. There are two supported functional forms: $w \exp \frac{-(d-d_0)^2}{\sigma^2}$ (default) and $w \frac{(d-d_0)^2}{\sigma^2}$ (selected by setting dist-squared=false), where σ is called the contact width and w is the contact weight. The default value for the contact width can be set by the option default-width, whereas the default value for the contact weight is 1. The contact map can be defined with a contact map file. The input file takes one contact per line, in the following format:

```
residueno1 atomname1 residueno2 atomname2 idealdistance [width [weight]]
```

In this format, the width and the weight is optional. The input format could for example look like:

```
5 CA 1 CA 13.029
6 CA 2 CA 13.168 2.0 1.5
```

Settings

name	description
contact-map-file	Contact map input file.
contact-map-string	Contact map input string.
pdb-file	Path to PDB file for constructing contact map.
cutoff(=inf)	contact cutoff distance when constructing map from PDB.
minimum-residue-distance(=7)	Minimum residue-separation in a pair.
iteration-type(=ALL)	Which atoms to include when constructing contactmap from chain.
dist-squared(=false)	Whether to use the squared distance version of the energy.
potential-width(=1.0)	Width of energy potential.

5.16 Selecting energies in `phaistos.cpp`

In the `phaistos.cpp` command line, energy terms can either be specified one by one

```
$ ./phaistos --energy-clash-fast --energy-opls-angle-bend
```

or using a shorthand

```
$ ./phaistos --energy clash-fast opls-angle-bend
```

When using the shorthand option, indivudial energy options can be specified using square brackets as described in section 3.1.

Chapter 6

Observables

An observable is an energy term with some additional functionality, allowing the user to specify how the value should be outputted and how frequently. All energy terms can be used as observables, and conversely, all observables are based on an underlying energy term.

6.1 General observable settings

In addition to the general settings of an energy term (see section 5.2), all observables have the following settings.

name	description
`register-interval`(=5000)	Interval (number of iterations) with which observable should be registered.
`register-burnin`(=0)	Interval (number of iterations) before observable should start registering.
`output-target`(=observables_%p_%t.dat)	Where and how the observable should be dumped (see text).
`output-interval`(=1)	Interval (number of iterations) with which observable should be outputted. This value is ignored if it is lower than register-interval.

The `output-target` setting can take various predefined values.

`stdout`|`cout` Output to `stdout`.

`stderr`|`cerr` Output to `stderr`.

`pdb-header` Output to the header in dumped PDB files.

`pdb-b-factor` Output to the b-factors in dumped PDB files.

Any other values will be regarded as a filename specification. The strings `%p` and `%t` will be replaced by the current process id and thread number,

respectively. If a `%i` substring is found in the filename, it will be replaced by the current iteration of the simulation (meaning that there will be a new log-file for each iteration in which an observation is made).

It is possible to specify several output targets for one observable. This is done by separating them with a `+`. For instance, this command line will output both to a logfile and to stdout:

```
$ ./phaistos --observable \
  helix-content[output-target:stdout+logfile.dat]
```

6.1.1 Output modes

There are a number of different output modes available:

`verbose`, `verbose-with-id`, `verbose-with-time`, `compact`, `minimal`.

These are normally chosen automatically based on the output-target. However, they can be overridden by providing an *`#output-mode`* in an output-target. For instance

```
$ ./phaistos --observable \
  helix-content[output-target:logfile.dat#verbose]
```

will force the observable to be written to the `logfile.dat` file in `verbose` mode.

6.2 Radius of gyration (`rg`)

The radius of gyration energy term is mostly used as an observable, and is therefore described in this section, although it could equally well have been included in chapter 5. It calculates the root mean square distance of the atoms to the center of mass of the protein.

6.3 Root mean square deviation (`rmsd`)

This is another energy term that is mostly used as an observable, but can serve as an artificial energy function as well. It returns the Root Mean Square Deviation between the current structure and a reference structure.

Settings

name	description
`reference-pdb-file`	Path to a PDB file to which comparison should be done.
`ca-only`(=true)	Whether only to include CA atoms in the calculation.

6.4 Helix content (helix-content)

This observable measure the percentage of residues that are in a helical conformation. The boundaries of what is considered a helix can be adjusted using the four angle settings. The `per-residue` option allows you to output the results in a vector format, which can be read by the `pdb-b-factor` output mode described above.

Settings

name	description
`min-angle1`(=-1.57)	Minimum angle1 boundary.
`max-angle1`(=-0.52)	Maximum angle1 boundary.
`min-angle2`(=-1.34)	Minimum angle2 boundary.
`max-angle2`(=-0.30)	Maximum angle2 boundary.
`per-residue`(=false)	Output for each residue individually.

6.5 PDB (pdb)

Observables can also be used to extract complete structural states from a simulation. One examples is the PDB observable, which will dump pdb files are regular intervals. If the output filename contains special characters `%e` (energy value) and/or `%i` (iteration number), the structures will be dumped to individual files. If not, they will be dumped as models in a single pdb file.

Settings

name	description
`minimum-energy-mode`(=false)	Whether only to dump minimum energy structures.
`minimum-energy-fraction`(=0.1)	How far from the minimum energy a structure must be before being dumped.
`minimum-energy-interval`(=500)	Minimum interval between dumped minimum energy structures.

6.6 GIT (git)

It is also possible to dump structural states in GIT vectors. As for the PDB observable, the vectors can either be dumped to a single file or to individual files, depending on whether the `%e` or `%i` special characters are included in the output file name.

6.7 XTC trajectory (`xtc-trajectory`)

Dumping structural states as PDB files can be quite space consuming. As an alternative, PHAISTOS includes support for dumping structural states in Gromacs' XTC file format. The default output file for this observable is `trajectory_%p_%t.xtc`, where `%p` and and `%t` are replaced by the process id and thread number, respectively.

6.8 Angle histograms (`angle-histogram`)

The angle histogram observable maintains statistics on the distribution of angles in the protein. Several of the general observable settings are important when using this observable: The statistics will be updated at intervals given by the `register-interval` option and dumped at intervals given by the `output-interval` option. Optionally, a `register-burnin` can be set to skip a number of iterations at the beginning.

6.9 Selecting observables in `phaistos.cpp`

In order to use an energy term as an observable, use the `observable` keyword, rather than the `energy` that is normally used for specifying energies. Just as for moves and energies, observables can be specified individually

```
$ ./phaistos --observable-helix-content --observable-rg
```

or by using a shorthand

```
$ ./phaistos --observable helix-content rg
```

Any specific settings can be specified using square brackets as described in section 3.1. Here is a simple example using the angle histogram observable:

```
$ ./phaistos --observable \
  angle-histogram[output-target:output_%i.txt,\
  register-interval:1,output-interval:10]
```

6.9.1 Special observables: `@energy-sum` and `@energy-terms`

It is natural to have both the total energy sum and the individual energy terms as part of your observable. This is supported through two special observable options: `@energy-sum` and `@energy-terms`. These observables work by looking up the previously calculated values in the main energy term – no recalculation of energies is done.

An additional benefit of this approach is that observable settings can be provided for the whole group of energies in one go:

```
$ ./phaistos --energy profasi \
  --observable @energy-terms[register-interval:1]
```

6.10 evaluate_observable.cpp

The `evaluate_observable` program is located in the `applications` module. It makes it possible to easily evaluate energies/observables for a given set of PDB files. It follows the syntax of the `phaistos.cpp` command line. To evaluate for instance the helix content of the structure in a PDB file, one would write:

```
$ ./evaluate_observable --observable helix-content \
   --pdb-file 1enh.pdb
# ID    helix-content
1enh.pdb  0.71153846153846156
```

One distinction from the `phaistos.cpp` executable is that `evaluate_observable` can take many PDB-files at once. This is done using the `pdb-files` option:

```
$ ./evaluate_observable --observable helix-content \
  --pdb-files *.pdb
# ID    helix-content
sample1.pdb 0.80769230769230771
sample2.pdb 0.78846153846153844
...
```

Of course, the whole observable machinery is available from `evaluate_observable.cpp`, making it possible to determine where and how often results are outputted. Using the `pdb-header` and `pdb-b-factor` output targets described above, it is also possible to use `evaluate_observable` to annotate a given set of PDB files with energy information.

The `evaluate_observable.cpp` program also has an `energy` option, which works the same way as for `phaistos.cpp`. The output of energies is controlled by the `@energy-sum` and `@energy-terms` observables introduced in the last section. The default setting for `--observable` is to include both `@energy-sum` and `@energy-terms`, which means that if you specify energies and no observables, the sum and individual terms will be included:

```
$ ./evaluate_observable --pdb-file 1enh.pdb --energy opls
# ID    @energy-sum     clash-fast      opls-angle-bend ...
/PATH/1enh.pdb -3627.4051175360833 0 139.3105497686366 ...
```

Note that if observables are specified, the default will be overridden, and the energy evaluations might become invisible

```
$ ... --energy opls --observable rg
# ID    rg
/PATH/1enh.pdb  10.088942787694526
```

in which case you need to specify the special observables explicitly

```
$ ... --energy opls --observable @energy-sum rg
# ID    @energy-sum     rg
/PATH/1enh.pdb  -3627.4051175360833 10.088942787694526
```

Chapter 7

Monte Carlo: Simulation and Optimization

Phaistos is a Monte Carlo simulation framework. In its broadest sense, the term Monte Carlo describes any stochastic (non-deterministic) algorithm. In the context of protein simulations, it is used both to describe *simulation*, where one wishes to sample from a given probability distribution (typically the Boltzmann distribution), and stochastic *optimization*, where the task is merely to find the structure with lowest energy (highest probability). Simulations are typically based on a specific MC technique called *Markov Chain Monte Carlo* (MCMC), where each new state is generated using only information about the previous state, and where the property of *detailed balance* ensures that the simulation will converge to a given target distribution. In some fields, the Markov Chain Monte Carlo algorithm is referred to simply by the term Monte Carlo.

7.1 MCMC simulations: detailed balance and ergodicity

Markov Chain Monte Carlo is a technique that makes it possible to draw samples from a complex distribution π (target distribution), for which only the density function is known. The idea is to construct a simulation where each new state is dependent only on the previous state. If new states are accepted/rejected in accordance with the *detailed balance* equation, it can be shown that this Markov Chain will converge to the target distribution of interest.

The *detailed balance* property entails that the probability of being in a state x multiplied by the probability of going from state x to state x', is equal to the probability of being in state x' and moving to x

$$\pi(x)P(x \to x') = \pi(x')P(x' \to x) \ . \tag{7.1}$$

It can be convenient to factor the transition probability $P(x \to x')$ into a *proposal* probability P_p and an *acceptance* probability P_a, that is

$$P(x \to x') = P_p(x \to x')P_a(x \to x') \ . \tag{7.2}$$

In many cases, the moves used in a simulation are symmetric, $P_p(x \to x') = P_p(x' \to x)$. When this is not the case, the move is said to be *biased*. In such case, the acceptance probability factors must be chosen in a way to compensate for this bias. The Metropolis-Hastings method described below corresponds to one choice of these acceptance probabilities.

It should be noted that the detailed balance equation alone is not sufficient to guarantee convergence. The set of moves used in the simulation must also be chosen to ensure *ergodicity*, which means that there is a non-zero probability of moving from any state to any other state in a system.

7.2 General Monte Carlo settings

In the next sections, the different available Monte Carlo methods in Phaistos will be described in detail, along with a list of any available settings. A number of common options are shared between all Monte Carlo types. To avoid repetition, these are listed here.

name	description
`debug`(=0)	Debug level.
`declash-on-reinitialize`(=true)	Whether to remove self-collisions in the chain when reinitializing the structure.
`maximum-declash-attempt`(=20000)	The maximum number of moves to allow before aborting the declash procedure.
`reinitialization-interval`(=0)	How often to reinitialize the simulation (default: 0 (never)).
`consistency-check-interval`(=10000)	How often to check whether the simulated system is internally consistent.

7.3 Metropolis-Hastings (`metropolis-hastings`)

From expression (7.1) and (7.2), we can write the detailed balance requirement as

$$\frac{P_a(x \to x')}{P_a(x' \to x)} = \frac{\pi(x')P_p(x' \to x)}{\pi(x)P_p(x \to x')} \ . \tag{7.3}$$

The Metropolis-Hastings method [24] specifies that a move from state x to x' should be accepted with the probability

$$P_a(x \to x') = \min\left(1, \frac{\pi(x')P_p(x' \to x)}{\pi(x)P_p(x \to x')}\right) \ . \tag{7.4}$$

In other words, assuming an unbiased move set, the move is always accepted if the probability according to the target distribution is increased ($\pi(x') > \pi(x)$). If the probability decreases, the acceptance probability depends on how much worse the probability of the new structure is compared to the previous one. It can be seen that (7.4) is a solution to (7.3). Other solutions exist, leading to different convergence properties of the algorithm. However, the Metropolis-Hastings criterion is the most frequently used in the literature.

7.4 Greedy Optimization (`greedy-optimization`)

The simplest Monte Carlo type in Phaistos is greedy optimization. As in the case for Metropolis-Hastings, a sequence of new states are generated. However, in this case, only structures improving the energy are accepted. This can be useful for quickly relaxing a structure.

7.5 Simulated Annealing (`simulated-annealing`)

The simulated annealing method [25] lies somewhere between Metropolis-Hastings and greedy optimization. It uses a Metropolis-Hastings acceptance criterion, but rather than using a fixed temperature, the simulation will gradually move from a high to a low temperature. This has the effect that initially, the simulation will accept nearly all generated candidate structures, both with higher and lower energies. As the temperature decreases, the simulation will increasingly appear as a greedy optimization, accepting fewer structures that would increase the energy.

Settings

name	description
`e-min`(=-inf)	Lower bound on energy.
`e-max`(=inf)	Upper bound on energy.
`temperature-start`(=inf)	Starting temperature (must be finite).
`temperature-end`(=inf)	Ending temperature (must be finite).

7.6 Generalized Ensembles: Muninn (`muninn`)

The Metropolis-Hastings method sometimes suffers from long convergence time. This is partly due to the fact that the forcefields used for simulations correspond to very rugged energy landscapes. The simulation might spend a large part of its time on irrelevant local minima. Frequently, simulations get stuck for such extended periods that it is questionable whether ergodicity is obtained.

A potential solution to this problem is to sample from a different distribution with fewer ergodicity problems while keeping track of enough information to make it possible to reweight the produced samples to correspond to samples from the original target distribution.

Muninn is a framework to conduct these types of simulations. In the default case, Muninn will sample from the *multicanonical* distribution [26], which corresponds to a uniform distribution over energies. The method approximates the density of states using histograms over energy bins. Using the density of states, the ensemble produced by a Muninn simulation can be transformed back into a canonical ensemble at any specified temperature. Phaistos includes a number of scripts to facilitate this reweigthing (See chapter C).

There are a substantial number of settings in Muninn. In most cases, the default values will produce good results. The most important settings are probably the `min-beta` and `max-beta`, which will set a limit to the range of temperatures explored by Muninn during the simulation.

It should be emphasized that in Phaistos, the Muninn beta values are unitless. This means that the Muninn beta value β_m corresponds to the target distribution

$$P_{\beta_\mathrm{m}}(x) \propto \exp(-\beta_\mathrm{m} E_\phi(x)) \ , \tag{7.5}$$

where $E_\phi(x)$ is the Phaistos energy for the structure x as describe in section 5.1. For a physical energy function the Phaistos energy is $E_\phi(x) \propto \beta_\mathrm{p} E(x)$, where $E(x)$ is the physical energy for the structure x, $\beta_\mathrm{p} = (kT)^{-1}$ is the thermodynamic beta (specified by the `weight` option for the energy) and the temperature T always is a fixed value. This means that the distribution can be expressed as

$$P_{\beta_\mathrm{m}}(x) \propto \exp(-\beta_\mathrm{m}\beta_\mathrm{p} E_\phi(x)) \ . \tag{7.6}$$

Clearly, this expression is just a normal Boltzmann distribution $P_{\beta'} = \exp(-\beta' E_\phi(x))$ with $\beta' = (kT')^{-1} = \beta_\mathrm{m}\beta_\mathrm{p}$. This means that P_{β_m} is equal to the Boltzmann distribution $P_{\beta'}$ where $T' = \frac{T}{\beta_\mathrm{m}}$, and consequently we have the relation

$$\beta_\mathrm{m} = \frac{T}{T'} \ . \tag{7.7}$$

As an example, assume that we want to explorer the temperature range between 273 K and 1000 K. If the weight for the physical energy function is set a weight corresponding to $T = 300$ K then `min-beta` and `max-beta`

should be set as

$$\texttt{max-beta} = \frac{300\,\mathrm{K}}{273\,\mathrm{K}} \approx 1.09890 \tag{7.8}$$

$$\texttt{min-beta} = \frac{300\,\mathrm{K}}{1000\,\mathrm{K}} = 0.3\ . \tag{7.9}$$

Settings

name	description
`energy-min`(=-inf)	Lower bound on energy.
`energy-max`(=inf)	Upper bound on energy.
`histograms-per-reinit`(=4)	Number of histogram updates between every reinitialization (zero or negative means no reinitialization will be done).
`burnin-fraction`(=2)	The fraction of `initial-max` used for burn-in (in each thread).
`use-energy2`(=false)	Use the secondary energy as an additional energy in muninn - not used to estimate histograms.
`weight-scheme`(=multicanonical)	Which weight scheme to use: `invk` or `multicanonical`.
`slope-factor-up`(=0.3)	Slope factor use for the linear extrapolation of the weights, when the weights are increasing in the direction away from the main area of support.
`slope-factor-down`(=3)	Slope factor use for the linear extrapolation of the weights, when the weights are decreasing in the direction away from the main area of support.
`min-beta`(=-inf)	The minimal beta value to be used based on thermodynamics and in the extrapolation.
`max-beta`(=inf)	The maximal beta value to be used based on thermodynamics.
`initial-beta`(=uninitialized)	The initial beta value to be used (if uninitialized it takes the same value as `min-beta`).
`resolution`(=0.2)	The resolution for the non-uniform binner.
`initial-width-is-max-left`(=true)	Use the initial bin with as maximal bin width, when expanding to the left.
`initial-width-is-max-right`(=false)	Use the initial bin with as maximal bin width, when expanding to the right.
`statistics-log-filename`(=muninn.txt)	The filename (including full path for the Muninn statistics logfile (default is `Muninn_[PID].txt` and is obtained by removing this option from the config file or setting it to "")).

`read-statistics-log-filename`	The filename for reading the Muninn statistics logfile. If the value difference from the empty string (""), this log file is read and the history is set based on the content.
`log-mode`(=all)	The log mode of Muninn: `current` or `all`.
`read-statistics-log-filename`	The filename for reading the Muninn statistics logfile. If the value difference from the empty string (""), this log file is read and the history is set based on the content.
`read-fixed-weights-filename`	The filename for reading a set of fixed weights for a given region. If the value difference from the empty string (""), this file is read and the weights are fixed in the given region.
`initial-max`(=5000)	Number of iterations used in first round of sampling.
`memory`(=40)	The number of consecutive histograms to keep in memory.
`min-count`(=30)	The minimal number of counts in a bin in order to have support in that bin.
`restricted-individual-support`(=false)	Restrict the support of the individual histograms to only cover the support for the given histogram.
`dynamic-binning`(=true)	Use dynamic binning.
`max-number-of-bins`(=1000000)	The maximal number of bins allowed to be used by the binner
`bin-width`(=0.1)	Bin width used for non-dynamic binning.
`verbose`(=3)	The verbose level of Muninn

7.6.1 Setting the beta values

As mentioned, the most important settings for Muninn are probably the beta values: `min-beta`, `max-beta` and `initial-beta`. Convergence of the ensemble (`multicanonical` or `invk`) rely highly on the choice of these settings.

The `max-beta` should in general be set to or just above the beta value of interest, which usually is 1. The `min-beta` value should on the other hand be less than the beta value that corresponds to the critical temperature, since the critical temperature is considered to be the boundary where the protein goes from an unfolded to a folded state. However, the critical temperature is rarely known, and in that case `min-beta` value should be set close to zero. However, the lower `min-beta` is the more time the system will spend in the unfolded state.

The `initial-beta` value is by default set equal to `min-beta`. During

the burnin, Muninn samples from the distribution with $\beta_{\mathrm{m}} =$ `initial-beta`. This means that `initial-beta` should be set so that during the burin time, the Markov chain should be able to reach a state which is representable for this distribution.

The `min-beta` and `init-beta` values are highly system dependent.

7.7 Selecting Monte Carlo type in `phaistos.cpp`

The type of Monte Carlo algorithm used by `phaistos.cpp` is set using the `--monte-carlo` option. As for energies and moves, there is a convenient command line shorthand to quickly choose both the type of Monte Carlo run and the detailed settings:

```
$ ./phaistos --monte-carlo metropolis-hastings[debug:1]
```

If no Monte Carlo type is specified, `phaistos.cpp` will use Metropolis Hastings.

Chapter 8

Probabilistic Models

Phaistos contains implementations of a number of probabilistic models, each modeling distinct aspects of protein geometry. In this chapter, we will focus on the generative models, which are used both for proposing new structures, and as energy functions. Other models, which are used solely as energy functions, are described in separate appendices.

The generative models are divided into backbone models and sidechain models. For historical reasons, the backbone models are built into the core of Phaistos, while the sidechain models are available as a module.

8.1 Backbone models

The backbone models control the angular degrees of freedom of the protein main chain. There are several variants: FB5HMM (`fb5`), TorusDBN (`torus`), and TorusCsDBN (`torus-cs`). The FB5HMM model is concerned with the C_α trace of a protein, while the TorusDB describes the backbone in atomic detail (N-CA-C, CB, O). Both the FB5HMM and TorusDB models have been described in detail in the literature [3, 1]. The last model, TorusCsDBN, is an extension of the TorusDBN that includes chemical shift information. In cases where chemical shift data are available for a system, this can significantly improve speed and accuracy of a simulation.

8.1.1 Using backbone models in `phaistos.cpp`

By default, `phaistos.cpp` will choose a model depending on your chain type: FB5HMM for C_α chains and TorusDBN for full atom chains. You can override this behaviour, or change the settings of your model using the backbone-dbn option:

```
$ ./phaistos --backbone-dbn torus-cs
```

The different models have different associated settings, depending on the available node types. However, all models have a `initial-aa-file` option, an `initial-ss-file` option, and an `initial-pdb-file` option, which allows you to initialize your model from an amino acid sequence file, a secondary structure file or a PDB file. In the following command `torus-cs` is initialized from the PDB file `file.pdb`:

```
$ ./phaistos --backbone-dbn torus-cs[initial-pdb-file:file.pdb]
```

If no input settings are explicitly specified for the model, `phaistos.cpp` will use the global `pdb-file`, `aa-file` and `ss-file` options. The last example is therefore equivalent to:

```
$ ./phaistos --pdb-file file.pdb --backbone-dbn torus-cs
```

For a complete list of all available options, please see the Phaistos help output:

```
$ ./phaistos --help
```

8.2 Sidechain models

As for the backbone case, the two available sidechain models correspond to two different representations of protein sidechains. BASILISK is a probabilistic model that represents sidechains in full atomic detail and in continuous space, modeling all the individual χ angles in each sidechain [2]. COMPAS uses a more coarse-grained representation, where sidechains are represented by a single pseudo-atom. These models are described in greater detail in appendix G. The models are set up through the energies and moves that use them.

Appendix A

BackboneDBN applications module (`applications_backbone_dbn`)

The `applications_backbone_dbn` module contains a number of small applications that interface to the various backbone probabilistic models available in Phaistos (see chapter 8). The programs follow the command line syntax of the `phaistos.cpp` program, and models are set up using the `backbone-dbn` option in the same way as described in chapter 8.

A.1 Sampler (`sampler.cpp`)

This program provides functionality to draw samples from a model. This involves a forward-backtrack sampling in the range specified (using the `start-index`, `end-index` options), and a subsequent sampling of values for the output nodes. From the command line, it is possible to specify which output nodes you want to sample (`sample-aa`, `sample-ss`, and `sample-angles`). The `iterations` option specifies how many samples to generate.

A.2 Predictor (`predictor.cpp`)

The `predictor.cpp` program uses either Viterbi decoding or Posterior decoding to *predict* a sequence of output values. This choice is made using the `mode` option. The Viterbi mode works by finding the most probably hidden node sequence path through the model, and then outputting the corresponding emission probabilities for each position. Posterior decoding works by choosing the state with maximum probability at each position in the sequence, and then outputting the corresponding emission probabilities.

Using the `output-mode` option, you can choose to predict either the amino acid sequence (`aa`) or the secondary structure sequence (`ss`).

A.3 Likelihood (`likelihood.cpp`)

This program simply calculates the probability of the input given the model. You can specify the range of the sequence to evaluate (`start-index` and `end-index`). Optionally, you can choose to subtract the likelihood of the amino acid sequence from the total likelihood – this will give you the likelihood *conditioned* on the amino acid sequence.

A.4 Entropy (`entropy.cpp`)

The `entropy.cpp` calculates the structural entropy for a given amino acid sequence. This can be viewed as a measure of the flexibility of a sequence, or the entropy of the unfolded state of a sequence. This is carried out by estimating $\langle P \ln P \rangle$ through resampling local structures conditioned on the amino acid sequence. Note that only local structure is taken into account, effects such as steric repulsion are not.

A.5 Check mutations (`check_mutations.cpp`)

The `check_mutations.cpp` program calculates the change in log likelihood of the local structure for every possible single mutation of a protein structure. In other words, the changes reflect whether a given point mutation is compatible with the local structure of the WT structure.

Appendix B

GIT module (`git`)

B.1 PDB file to GIT vectors (`pdb2git`)

The `pdb2git` program is a convenient way to transform a set of regular PDB files into a GIT vector file. The conversion itself only takes fractions of a second, but due to I/O and the fact that one often want to convert the several thousand decoys generated by a long simulation, the conversion may take minutes.

Command line options

name	description
`git-file`(=*required*)	The outputfile to store the generated vectors in. Lines will be appended to an existing file.
`verbose`(=false)	Whether produce a verbose output (recommended if you want to see what is going on).
`debug`(=false)	Whether to print debugging information.
`input-file`	Calculate the GIT vector for a specified file only.
`input-dir`	Calculate the GIT vectors for all (`*.pdb`|`*.ent`) files in the directory `path`. The directory `path` should with a "/".

B.1.1 Examples (cookbook style)

The main application of the `pdb2git` is to transform entire directories of decoy structures into GIT vectors. The list of options for `pdb2git` can be obtained using the command:

```
$ ./pdb2git --help
```

In order to transform an entire directory of decoy structures into git vectors use the `input-dir` option. Is it also necessary to specify a target file, where all the generated vectors will be stored:

```
$ ./pdb2git --git-file my-git-file.git \
  --input-dir ../folding-simulation/samples/ --verbose
```

B.1.2 Frequently asked questions (FAQ)

What is GIT? GIT is a description of the overall protein fold using Gauss integrals. It was developed by Røgen and Fain, and the distances between the vectors has been shown to correlate well with standard measures like RMSD [27].

Appendix C

Muninn module (`muninn`)

This module is an interface to the Muninn generalized ensemble framework. For details, we refer to the main publication [5].

C.1 Scripts

The module includes a number of Phaistos-specific Muninn scripts, facilitating the analysis of simulations conducted through Muninn.

C.1.1 1D plot (`plot_1d.py`)

This script allows you to plot the free energy as a function of a given reaction coordinate. The script takes its input in a column format, where the relevant columns are specified as parameters. By default, the program assumes that column two holds the total energy, and column three the reaction coordinate. This is compatible with the observable output in Phaistos (where column one contains an ID).

Command line options

name	description
`muninn-log-file`	Path to Muninn log file.
`observable-file`	Path to data file in column format.
`column-index-energy`(=1)	Column index at which energies are found (zero-based indexing).
`column-index-x`(=2)	Column index at which x-values are found (zero-based indexing).

C.1.2 2D plot (`plot_2d.py`)

This script is similar to the 1D case, but creates a color-coded 2D plot of the free energy as function of two reaction coordinates. As for the 1D case, input is assumed to be in column format, and the default columns used

are column two (energy), three (reaction coordinate 1) and four (reaction coordinate 2).

Command line options

name	description
`muninn-log-file`	Path to Muninn log file.
`observable-file`	Path to data file in column format.
`column-index-energy`(=1)	Column index at which energies are found (zero-based indexing).
`column-index-x`(=1)	Column index at which x-values are found (zero-based indexing).
`column-index-y`(=2)	Column index at which y-values are found (zero-based indexing).

C.1.3 Assign weights (`assign_weights.py`)

For further processing, it can be convenient to calculate the Muninn weights for a set of PDB files. This script will return a list of (ID, weight) pairs given an input file containing IDs and energies. By default the input format is assumed to have IDs in the first column and energy in the second.

Command line options

name	description
`muninn-log-file`	Path to Muninn log file.
`observable-file`	Path to data file in column format.
`column-index-id`(=0)	Column index at which IDs are found (zero-based indexing).
`column-index-energy`(=1)	Column index at which energies are found (zero-based indexing).

Appendix D

OPLS module (`opls`)

D.1 Testing OPLS (`test_opls.cpp`)

The OPLS module features a test_opls program to test and profile the energy terms. In `/opls/test/CMakeLists.txt` there is a line that adds the `-pg` tag when uncommented. It is intended for expert users only.

It is build in the following way:

```
phaistos/build$ make test_opls
phaistos/build$ cd test
phaistos/build/test$ ./test_opls 2gb1.pdb
```

Appendix E

PLEIADES module (`pleiades`)

E.1 The PLEIADES program `pleiades.cpp`

PLEIADES is a protein structure clustering program that allows to quickly cluster very large amounts of decoy structures.

E.1.1 Command line options

name	description
`git-file`(=*required*)	Input file containing all the vectors to be clustered. See `pdb2git` if you want to convert pdb files into GIT vectors.
`out-file`(=out.cluster)	File to store the final cluster in.
`verbose`(=false)	Produce a verbose output (recommended if you want to see what is going on).
`debug`(=false)	Print debugging information.
`iterations`(=100)	How many iterations to maximally run before slopping. Since the convergence is not always detected automatically you should set a reasonable amount of iterations after which the algorithm terminates.
`k-means`(=true)	Flag indicating to perform k-means clustering.
`k`(=10)	Number of K cluster being used.
`w-k-means`(=false)	Flag indicating to perform weighted k-means clustering based on weights derived from the Muninn bin weights. See section 7.6 and chapter C for details on Muninn.
`beta`(=1.0)	Inverse temperature for the weighted clustering.

`long-output`(=false)	Prints all members of each cluster (per default only the 5 members closest to the centroid are printed).
`smart-seed`(=false)	Use the kmeans++ algorithm to set the initial cluster seeds. This will choose new centroids with a likelihood proportional to the distance to the closest existing cluster centriod. This will lead to more equally distributed cluster seeds [28].
`muninn-log`	Filepath to the muninn logfile to be used for the weight determination in the weighted clustering.
`scale-weights`(=false)	Allows to scale the weights to the interval [0,1[. Use this option if you find that the weights do contain zeros. Zero-weights will otherwise mess up the clustering.
`rmsd-native-pdb`	It is possible calculated the RMSD of each structure of a cluster to a native, target structure after the clustering procedure. This option allows to set the filepath to the native structure to be used.
`rmsd-decoy-prefix`	In order too be able to calculate the RMSD to the native structure for each decoy, we need the decoy structures themself. The filename is extracted from the first column of the GIT file. Please adjust this prefix, so that both combined points to the correct files.
`guess-k`(=false)	Experimental procedure, trying to guess a good K by using a cutoff based clustering approach.
`guess-k-threshold`(=30)	Threshold for the experimental detection of K.

E.1.2 Examples (cookbook style)

In this section we are going to go over a few application scenarios and how those can be handled by the PLEIADES program.

In order to get the list for options for the `pleiades` program use the commend:

```
$ ./pleiades -h
```

To then start a clustering run, you need all the structures in GIT vector format. The GIT vectors can be obtained using the program `pdb2git`, see section B.1.

The following command will the start a clustering run for a maximum of 500 iterations sorting the structures into $k = 20$ cluster. The option

`--long-output` indicates that all members of the clusters will be printed and the output-file will be named `samples.cluster`.

```
$ ./pleiades -git-file ~/data/samples/samples.git --verbose -k 20 \
  --iterations 500 --long-output -o samples.cluster
```

E.1.3 Frequently asked questions (FAQ)

What is GIT? GIT is a description of the overall protein fold using Gauss integrals. It was developed by Røgen and Fain, and the distances between the vectors has been shown to correlate well with standard measures like RMSD [27].

How can I calculate GIT vectors? I only have PDB files The PHAISTOS package also contains a convenient conversion tool to transform PDB structures into the according GIT vectors called `pdb2git`. See section B.1 for more information.

How can I cite PLEIADES? We are currently in the process of writing up a paper on PLEIADES. We will update this section once the article is published.

Appendix F

SAXS module (saxs)

Small angle X-ray scattering (SAXS) is an experimental technique which can readily provide low resolution structural information on proteins in solution. The experiment requires relatively low amount of sample: typical quantities are a volume of $\sim 15\,\mu l$, and a sample concentration of $\sim$1.0 mg/ml [29].

SAXS was originally developed for structure analysis of condensed matter in isotropic environments and the first applications date back to the 1930s, with the publication of studies on metal alloys [30]. The method was later extended to other materials, included biological macromolecules in solution [31].

The information that can be obtained directly from a SAXS experiment includes the radius of gyration, volume, molecular mass (provided a known reference is available) and possibly a coarse-grained description of the electron distribution in the sample. These parameters are inferred from inverse Fourier analysis of the SAXS experiment, and are limited by the noise level of the instrumentation. In particular, the determination of the electronic distribution function $p(r)$ requires a number of regularization steps whose relative weight is unclear. Manual tuning is therefore typically required to produce satisfactory results [32].

Since the 1980s, improvement in the instrumentation has led to reliable analyses of polymeric systems, and increase in computation power during the 1990s allowed for Bayesian regularization of the inverse Fourier analysis, providing better estimators and more rigorous error analysis [33]. The latter is unfortunately still not routinely applied.

Historically, analysis of obtained SAXS curves was limited to a comparison to scattering curves describing simple geometrical bodies with known analytical solutions. As sufficient computational power became available, this analysis was extended with the possibility of evaluating forward models of SAXS data from a hypothetical experiment. These structural proposals are typically compared to the experimental result in order to validate one or more structural models [34, 31, 29].

There are two main approaches for calculating a theoretical SAXS curve from the structure (the forward model). One is based the approximation of the electronic inhomogeneity via spherical harmonics expansion [35], the other evaluates the pairwise interactions occurring between elementary scatterers by means of direct Fourier transforms [36]. The spherical harmonics approach is computationally attractive, but it is also quite sensitive to the correct determination of the atomic volume of each individual scatterer. Further, the algorithm enforces a compact, quasi-globular shape, and the evaluation has to be completely recomputed even in case of a minor update in the proposal. For this reason, Monte Carlo protocols do not fully benefit of this method.

The SAXS module in Phaistos follows the description using pairwise interactions. This is based on the Debye formula [37]

$$I(q) = \sum_{i=1}^{M} \sum_{j=1}^{M} F_i(q) F_j(q) \frac{\sin(q \cdot r_{ij})}{q \cdot r_{ij}} \ , \tag{F.1}$$

where the experimental SAXS curve $I(q)$ is defined as the sum of the pairwise interactions between the M bodies in the system. These are individually described by the product of the form factors, $F_i(q)$, with the structure factors depending on the distance between them (r_{ij}) and the scattering momentum $q = \frac{4\pi}{\lambda} \sin\theta$. Here, $\theta/2$ is the scattering angle between the incoming beams, and λ the X-ray wavelength. This approach is computationally more expensive than the spherical harmonics evaluation ($O(n^2)$) but allows for partial updates in the structure and hence caching. Importantly, it is not limited by assumptions on the protein shape.

An attractive performance can be achieved by reducing the number of elementary scatterers. By grouping neighboring atoms in a set of *dummy bodies*, the $I(q)$ profile can be conveniently approximated at least to the current experimental resolutions [9]. The Phaistos SAXS module includes parameters for two levels of coarse-graining. The one-body model describes each protein residue a single scattering body, whereas the two-body model used one (common) dummy body for the backbone part of the residue and one for the side chain scattering. The two-body model is significantly more accurate for resolutions above $q \approx 0.2\text{Å}^{-1}$ while the one-body model evaluates around three times faster. A detailed description of the coarse-grained models can be found in the publication [9].

The SAXS module is implemented as an energy term for Phaistos and the configuration is described in Section 5.9.

Appendix G

Sidechain-dbns module (`sidechain-dbns`)

G.1 BASILISK

BASILISK is a probabilistic model of the conformational space of amino acid side chains in proteins. Unlike rotamer libraries, BASILISK models the χ-angles in continuous space, including the influence of the protein's backbone.

The BASILISK model and certain applications are incorporated into the Phaistos framework, providing both a move (section 4.12.2) and an energy term (section 5.10). If you are interested in the model itself and its applications outside the Phaistos framework, we refer to the separately released software package:

`http://sourceforge.net/projects/basilisk-dbn/`

If you use BASILISK in your work, please cite reference [2].

G.2 COMPAS

COMPAS models protein sidechains using a coarse-grained representation. Each sidechain is represented using a single pseudo atom, parameterized using a set of angles and a distance. As for the BASILISK model, the COMPAS model is available both as a move (section 4.12.3) and an energy term (section 5.11). The model is described in some detail in a recent paper on simulations under SAXS restraints [4].

Appendix H

Typhon module (`typhon`)

TYPHON is a novel approach to exploring the conformational space of proteins based on probabilistic models of protein structure. The incorporated models, namely TorusDBN [1] and BASILISK [2], ensure that the samples are protein–like on a local length scale. In order to control the overall structure of the protein, different types of distance based restraints can be defined. Over the course of the simulation, the probabilistic models will thus explore the conformational space of the protein that is allowed by the given restraints.

There are four types of interactions currently supported in TYPHON:

Hydrogen Bonds All sorts of intra-protein hydrogen bonds are supported and detected (using the DSSP HBond score). The bond is defined by four distances: N–O, N–C, H–O and H–C. The specific parameters where obtain from high resolution structures.

Disulfide bridges Disulfide bridges are detected based on their geometry. Additionally to the sulfur-sulfur bond, the bond geometry is stabilized by three additional Gaussian contacts.

C_α-C_α **contacts** Per default C_α-C_α contacts of residues that are far in the amino acid chain (5 or more residue) but close in space (6 Ångstrom or less) are included in the network. To ignore those please use the `--ignore-gaussians` option.

Arbitrary atom-atom interaction One can specify any atom pair interaction by adding it to the network file. This can for example be used to *pull* certain parts of the structure closer together or make sure that they stay at a certain distance.

H.1 The TYPHON program (typhon.cpp)

The TYPHON distance-restraint functionality is available as an energy term for the Phaistos framework, described in section 5.12. For convenience, we also provide a `typhon` executable specifically tailored for the Typhon functionality.

Common options

name	description
`verbose`(=false)	Produce a verbose output (highly recommended if you want to see what is going on).
`debug`(=false)	Print debugging information.
`analyze`(=false)	Analyze and print the restraint network only, no sampling will be performed. This options is useful when initially analyzing and modifying the restrain network.
`iterations`(=10,000,000)	For how many iterations to run the simulation.
`burnin`(=1,000)	How many iterations to skip before starting to sample.
`reinitialize-interval`(=0)	How often to reinitialize to the native structure, in number of iterations.
`seed`(=*current timestamp*)	It is possible to use specific seeds for the random number generator. Usually you should leave this empty to generate a seed based on the current system time. This will ensure different seeds for all runs started. The seed is a natural number.

Input options

name	description
`pdb-file`(=*required*)	Protein structure to start from, in the PDB file format.
`ss-file`	Optional input of the expected secondary structure in the Phaistos secondary structure file format, which in short is a one lined file with a C,E or H for each residue position in the structure. Where C would be a coil conformation, E an extended strand and H a helical conformation.
`restore`	To restore a network from a previous run or a modified network. Expects the path to the network file as an argument.

Output options

name	description
`output-directory`(=./samples/)	Where to store the generated structures.
`dump-interval`(=10000)	How often should a structure be saved, in number of iterations.
`dump-git`(=false)	Activates the output in the form of GIT vectors as well.
`create-phaistos-structure`(=false)	Create a "native" structure with all atoms at the beginning of the run. This option can be useful for visualization purposes and when investigating and/or modifying the restraint network. Note: Typhon may add atoms to the input structure (i.e. missing hydrogens).

Network options

name	description
`no-prune`(=false)	Do not prune/optimize the hydrogen bond network. Per default hbonds with multiple partners are removed along with very weak bonds due to high solvent accessibility.
`ignore-hbonds`(=false)	Disregard any hydrogen bonds found (overwrites the restore network option, meaning that no hydrogen bonds will be present in the network, even if they were specified in the network file).
`ignore-sc-hbonds`(=false)	Disregard any side chain - side chain hydrogen bonds found (overwrites the restore network option).
`ignore-bbsc-hbonds`(=false)	Disregard any backbone - side chain hydrogen bonds found (overwrites the restore network option).
`ignore-ssbonds`(=false)	Disregard any disulfide bridges found (overwrites the restore network option).
`ignore-gaussians`(=false)	Disregard any Gaussian contacts found (overwrites the restore network option).
`init-hbond-from-native`(=false)	Per default we use parameters estimated from the top 500 set to describe the bond geometry. Set this flag to initialize hydrogen bond distances from the native structure.
`ca-cutoff`(=6)	Cutoff distance in Ångstrom to be considered a C_α contact.

`ca-skip`(=5)	How many residues to skip along the chain before considering a C_α contact. Default is set to 5.

Path options

If you specify the `PHAISTOS_ROOT` environmental variable pointing to your Phaistos build directory you should not have to worry about these options, see section 2.5 for details.

name	description
`dbn-parameter-dir`	In which directory can the TorusDBN parameter set be found.
`dbn-parameter-file`	Which TorusDBN parameter set to use.
`sc-dbn-file`	Which BASILISK DBN to use for the side chain movements.

Move options

name	description
`sc-move-weight`(=5)	How many sidechain moves per backbone (CRISP) move. Per default we recommend doing five moves, since this will amount in roughly an equal amount of motion in the backbone (CRISP 5 residues) and side chains (BASILISK 1 residue).

Sampling options

name	description
`ignore-ss`(=false)	Allows to sample backbone conformations without secondary structure input. This will generally lead to a broader sampling of the backbone conformational space.

Dehydron options

For more information and details on the dehydron concept see reference [38].

name	description
`dehydron-bb-cutoff`(=14)	Consider backbone hydrogen bonds below (d <= `cutoff`) as dehydrated aka weak/broken. All weak hydrogen bonds will be removed.

`dehydron-bbsc-cutoff`(=9)	Consider backbone-side chain hydrogen bonds below (d <= `cutoff`) as dehydrated aka weak/broken.
`dehydron-sc-cutoff`(=7)	Consider side chain hydrogen bonds below (d <= `cutoff`) as dehydrated aka weak/broken.

H.1.1 Examples (cookbook style)

In this section we discuss a few application scenarios and how those can be handled in TYPHON. We are not discussing the installation process again as this was covered in chapter 2. However, we would like to stress the advantage of setting an environmental variable called `PHAISTOS_ROOT` pointing to your Phaistos build directory, see section 2.5. With the `PHAISTOS_ROOT` setup, you should be able to run TYPHON and obtain the help message:

```
$ ./typhon -h
```

Before starting a simulations it is probably a good idea to inspect the network as it was detected by TYPHON. Per default TYPHON adds all hydrogen bonds it finds to the network along with a number of long range C_α-C_α contacts.

```
$ ./typhon --pdb-file 2GB1.pdb --analyze --verbose \\
   --generate-pymol
```

This command will do several things. First specifying the `--analyze` flag, we are signaling TYPHON to only calculate the interaction network, print the result and finish without running any simulation. The `--verbose` flag indicates, that all output is printed and `--generate-pymol` triggers the output of a PyMOL script visualizing the restraint network. The following command will launch PyMOL and load the network as detected by TYPHON:

```
$ pymol load_pymol_network.py
```

From here the network can be manipulated with the expert knowledge available through you. You know a certain bond has been shown to be weak or especially important. You can modify the network at will and use the `--restore` option to load the optimized network for your simulation.

All that is really required for a dynamics simulation is a PDB file describing the protein structure. We do recommend to use the verbose option in order to receive a more detailed, step by step output:

```
$ ./typhon --pdb-file 2GB1.pdb --verbose
```

Of course, in a more realistic application scenario you will want to specify more parameters, such as where to store the generated samples, how long to run the simulation, and which network to use:

```
$ ./typhon --pdb-file 2GB1.pdb --verbose --iterations 50000000 \
  --output-directory /my/storage/typhon/runID/samples \
  --restore optimized-network.net
```

H.1.2 Frequently asked questions (FAQ)

What is a dehydron? The concept of dehydrons is based on the observation that hydrogen bonds are less stable if they are exposed to the solvent, since water molecules will be competing for the charge. Fernádez and coworkers found an easy way to determine the solvent accessibility of a hydrogen bond, by counting the unpolar carbon atoms within a certain sphere around the bond [38]. In TYPHON, all weakly shielded hydrogen bonds are removed from the restraint network. You can include all hydrogen bond by either decreasing the cutoff or using the `--no-prune` option.

The program cannot find the DBN models The DBN files are required to load the structural prior models or their parameters to be a little more specific. Those files can be found in your Phaistos build directory in the subfolder `./data`. You can either specify the paths directly (see command-line options above) or, a little easier, specify the `PHAISTOS_ROOT` environmental variable to point to your Phaistos build directory.

How can I cite TYPHON? Harder, T., Borg, M., Boomsma, W., Røgen, P., Hamelryck, T. (2011) Fast large-scale clustering of protein structures using Gauss integrals. Bioinformatics. Accepted.

H.2 Typhon in `phaistos.cpp`

The main functional component in Typhon is an energy term called `constrain-distances`. As described in chapter 5, this energy term can be included in any `phaistos.cpp` simulation. For convenience, the set of moves and energies used in the `typhon.cpp` executable are grouped together in a Phaistos mode called `typhon`. As an example, the typhon simulation corresponding with the command

```
$ ./typhon --verbose --pdb-file 1enh.pdb --ss-file 1enh.dssp
```

can be run through `phaistos.cpp` using

```
$ ./phaistos --pdb-file 1enh.pdb --init-from-pdb \
  --mode typhon --ss-file 1enh.dssp
```

This makes it easy to experiment with different move sets and energies.

Bibliography

[1] W. Boomsma, K.V. Mardia, C.C. Taylor, J. Ferkinghoff-Borg, A. Krogh, and T. Hamelryck. A generative, probabilistic model of local protein structure. *Proc. Natl. Acad. Sci. U S A*, 105(26):8932–8937, 2008.

[2] T. Harder, W. Boomsma, M. Paluszewski, J. Frellsen, K.E. Johansson, and T. Hamelryck. Beyond rotamers: a generative, probabilistic model of side chains in proteins. *BMC bioinformatics*, 11(1):306, 2010.

[3] T. Hamelryck, J.T. Kent, and A. Krogh. Sampling realistic protein conformations using local structural bias. *PLoS Comput. Biol.*, 2(9):e131, 2006.

[4] K. Stovgaard, C. Andreetta, N. Clark, T. Harder, J. Ferkinghoff-Borg, W. Boomsma, M. Borg, J. Frellsen, V. Receveur-Bréchot, Luger K., and T Hamelryck. Inferential structure determination of flexible multi-domain proteins from small-angle X-ray scattering data. Submitted, 2011.

[5] J. Frellsen and J. Ferkinghoff-Borg. Muninn - An automated method for Monte Carlo simulations in generalized ensembles. In preparation, 2012.

[6] W.L. Jorgensen, D.S. Maxwell, and J. Tirado-Rives. Development and testing of the OPLS all-atom force field on conformational energetics and properties of organic liquids. *J. Am. Chem. Soc.*, 118(45):11225–11236, 1996.

[7] D. Qiu, P. S. Shenkin, F. P. Hollinger, and W. C. Still. The GB/SA continuum model for solvation. A fast analytical method for the calculation of approximate Born radii. *J. Phys. Chem. A*, 101(16):3005–3014, 1997.

[8] A. Irbäck and S. Mohanty. Profasi: a Monte Carlo simulation package for protein folding and aggregation. *Journal of computational chemistry*, 27(13):1548–1555, 2006.

[9] K. Stovgaard, C. Andreetta, J. Ferkinghoff-Borg, and T. Hamelryck. Calculation of accurate small angle X-ray scattering curves from coarse-grained protein models. *BMC bioinformatics*, 11(1):429, 2010.

[10] S. Olsson, W. Boomsma, J. Frellsen, S. Bottaro, T. Harder, J. Ferkinghoff-Borg, and T. Hamelryck. Generative probabilistic models extend the scope of inferential structure determination. *J. Magn. Reson.*, 213:182–186, 2011.

[11] Sandro Bottaro, Wouter Boomsma, Kristoffer E. Johansson, Christian Andreetta, Thomas Hamelryck, and Jesper Ferkinghoff-Borg. Subtle Monte Carlo updates in dense molecular systems. *J. Chem. Theory Comput.*, 8(2):695–702, 2012.

[12] A. Irbäck, S. Mitternacht, and S. Mohanty. An effective all-atom potential for proteins. *PMC Biophys.*, 2(1):2, 2009.

[13] R.L. Dunbrack Jr and F.E. Cohen. Bayesian statistical analysis of protein side-chain rotamer preferences. *Protein Sci.*, 6(8):1661–1681, 1997.

[14] M.R. Betancourt. Efficient Monte Carlo trial moves for polypeptide simulations. *J. Chem. Phys.*, 123:174905, 2005.

[15] C.A. Smith and T. Kortemme. Backrub-like backbone simulation recapitulates natural protein conformational variability and improves mutant side-chain prediction. *J. Mol. Biol.*, 380(4):742–756, 2008.

[16] J.P. Ulmschneider and W.L. Jorgensen. Monte Carlo backbone sampling for polypeptides with variable bond angles and dihedral angles using concerted rotations and a Gaussian bias. *J. Chem. Phys.*, 118:4261, 2003.

[17] R.A. Engh and R. Huber. Accurate bond and angle parameters for X-ray protein structure refinement. *Acta Crystallogr. A*, 47(4):392–400, 1991.

[18] G. Favrin, A. Irbäck, and F. Sjunnesson. Monte Carlo update for chain molecules: biased Gaussian steps in torsional space. *J. Chem. Phys.*, 114:8154, 2001.

[19] I. Lotan, F. Schwarzer, D. Halperin, and J.C. Latombe. Algorithm and data structures for efficient energy maintenance during Monte Carlo simulation of proteins. *J. Comput. Biol.*, 11(5):902–932, 2004.

[20] Kristoffer E. Johansson and Thomas Hamelryck. A simple probabilistic model of multibody interactions in proteins. Submitted manuscript, 2013.

[21] J.W. Ponder, C. Wu, P. Ren, V.S. Pande, J.D. Chodera, M.J. Schnieders, I. Haque, D.L. Mobley, D.S. Lambrecht, R.A. Jr. DiStasio, M. Head-Gordon, G.N.I. Clark, M.E. Johnson, and T. Head-Gordon. Current status of the AMOEBA polarizable force field. *J. Phys. Chem. B*, 114(8):2549–2564, 2010.

[22] Loredana Lo Conte and Temple F. Smith. Visible volume: A robust measure for protein structure characterization. *J Mol Biol*, 273(1):338 – 348, 1997.

[23] Joost Schymkowitz, Jesper Ferkinghoff-Borg, Francois Stricher, Robby Nys, Frederic Rousseau, and Luis Serrano. The foldx web server: an online force field. *Nucleic Acids Res*, 33(suppl 2):W382 – W388, 2005.

[24] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, E. Teller, et al. Equation of state calculations by fast computing machines. *J. Chem. Phys.*, 21(6):1087, 1953.

[25] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671, 1983.

[26] B.A. Berg and T. Neuhaus. Multicanonical ensemble: A new approach to simulate first-order phase transitions. *Physical Review Letters*, 68(1):9, 1992.

[27] P. Røgen and B. Fain. Automatic classification of protein structure by using Gauss integrals. *Proc. Natl. Acad. Sci. U S A*, 100(1):119, 2003.

[28] D. Arthur and S. Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[29] D.I. Svergun, L.A. Feigin, and G.W. Taylor. *Structure analysis by small angle X-ray and neutron scattering*. Plenum Press, 1987.

[30] A. Guinier. Diffraction of X-rays of very small angles-application to the study of ultramicroscopic phenomena. *Ann. Phys.(Paris)*, 12:161–237, 1939.

[31] O. Kratky and O. Glatter. *Small angle X-ray scattering*. Academic New York, Jan 1982.

[32] A.V. Semenyuk and D.I. Svergun. GNOM-a program package for small-angle scattering data processing. *Journal of Applied Crystallography*, 24(5):537–540, 1991.

[33] S. Hansen. Calculation of small-angle scattering profiles using Monte Carlo simulation. *Journal of applied crystallography*, 23(4):344–346, 1990.

[34] M.H.J. Koch, P. Vachette, and D.I. Svergun. Small-angle scattering: a view on the properties, structures and structural changes of biological macromolecules in solution. *Q Rev Biophys*, 36(2):147–227, Jan 2003.

[35] D. Svergun, C. Barberato, and MHJ Koch. CRYSOL-a program to evaluate X-ray solution scattering of biological macromolecules from atomic coordinates. *J. Appl. Cryst.*, 28(6):768–773, 1995.

[36] O. Glatter and O. Kratky. *Small angle X-ray scattering*, volume 102. Academic Press London, 1982.

[37] P. Debye. Zerstreuung von rontgenstrahlen. *Ann Phys*, 351(6):809–823, January 1915.

[38] A. Fernández and R. Scott. Dehydron: a structurally encoded signal for protein interaction. *Biophys. J.*, 85(3):1914–1928, 2003.