Chapter 16

Protocols for Efficient Simulations of Long-Time Protein Dynamics Using Coarse-Grained CABS Model

Michal Jamroz, Andrzej Kolinski, and Sebastian Kmiecik

Abstract

Coarse-grained (CG) modeling is a well-acknowledged simulation approach for getting insight into long-time scale protein folding events at reasonable computational cost. Depending on the design of a CG model, the simulation protocols vary from highly case-specific—requiring user-defined assumptions about the folding scenario—to more sophisticated blind prediction methods for which only a protein sequence is required. Here we describe the framework protocol for the simulations of long-term dynamics of globular proteins, with the use of the CABS CG protein model and sequence data. The simulations can start from a random or a selected (e.g., native) structure. The described protocol has been validated using experimental data for protein folding model systems—the prediction results agreed well with the experimental results.

Key words Folding pathway, Folding mechanism, Protein dynamics, Protein folding, Coarse-grained modeling

1 Introduction

Protein folding events occur over a wide range of time scales: from picosecond (small fluctuations) to millisecond or longer (significant regrouping of thousands of atoms). No single experimental technique has yet presented a complete insight into the folding process due to the limitations in accessible time and resolution scales [1]. For small proteins, the 1 ms time scale has recently become accessible to atomic level molecular dynamics (MD) simulations run on special-purpose supercomputers [2]. Given the ambiguity of the experimental data, the major role of simulation techniques is to provide detailed structural models suitable for the experiment interpretation [3, 4].

The purpose of the described CABS software package is to perform long-time simulations of protein molecules including de novo folding from a random structure, near-native dynamics, unfolding processes, and long-time dynamics of unfolded structures.

The package simulation engine is the CABS protein model—a coarse-grained (CG) modeling tool—enabling an effective simulation of protein dynamics (at a much reduced computational cost compared to the most established simulation approach: an all-atom MD) and de novo prediction of protein structures. In the CASP experiments, the CABS-based prediction approach allowed for realistic de novo predictions of new folds for small proteins and an accurate modeling of large structures using various partial restraints derived from detected homologies with known structures (the approach was ranked first or second depending on the scoring system [5, 6]). The application of the CABS model to simulations of protein dynamics has been validated on experimental long-time scale (super-millisecond) data for protein folding model systems (perhaps the most extensively studied by experiment and theory): barnase [7], chymotrypsin inhibitor 2 [7], B1 domain of protein G [8], B domain of protein A [9, 10], and others [11]. The obtained simulation results concerning the folding mechanism or the denatured state properties agreed well with experimental data and other simulation findings (the review and comparison of the experimental, the CABS-predicted, and other simulation data for three protein folding model systems are presented in ref. 1). Another validation study included the comparison of the CABS dynamics with the results of MD simulations [12]. The test demonstrated that the consensus view of protein dynamics from short (10 ns)-time scale MD simulations (for different protein metafolds, using all-atom MD, explicit water, and four most popular force fields) is fairly consistent with the CABS dynamics. The CABS modeling approach has also been used in simulation studies of a chaperonin effect on folding mechanism (a simple chaperonin-like protocol was implemented within the CABS algorithm) [10].

Generally, in comparison with other simulation tools, the advantageous features of CABS include suitability for de novo prediction of small proteins, low computational cost of simulating significant conformational changes, and, in respect to other CG models, high resolution of coarse graining (physically realistic models can be obtained [9, 13]). The potential applications of the CABS model comprise structural characterizations of protein conformations along the folding pathway (denatured state ensembles, intermediates, and near-native ensembles) and thus the interpretation of the existing sparse experimental data. In all these prediction tasks, weak and/or fragmentary distance restraints (derived from sparse experimental data or from theoretical predictions of plausible structural biases) can be applied. Finally, the CABS-derived structures and trajectories can be used in multiscale modeling procedures, merging CG modeling with atomic level simulations (*see* the pipeline in Fig. 1).

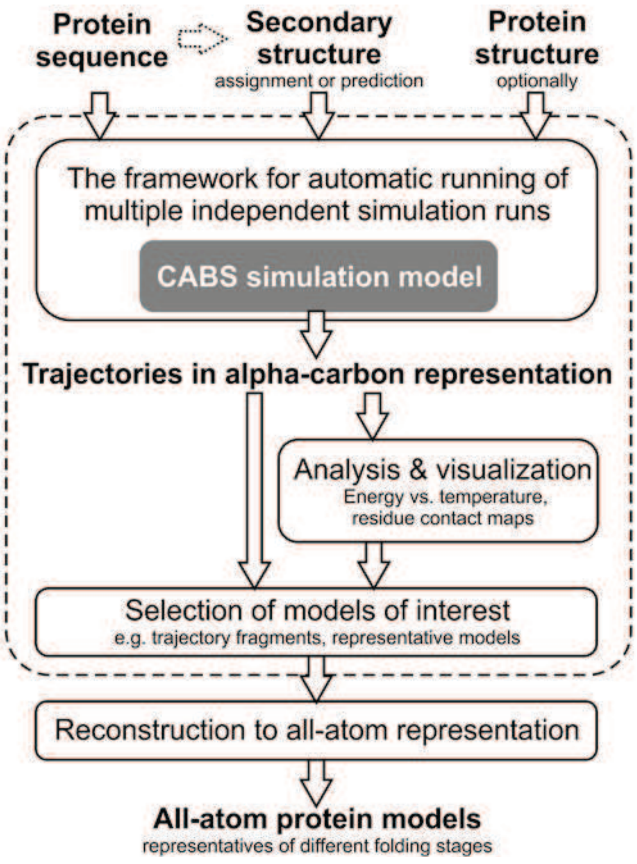
Protein sequence
Secondary structure
assignment or prediction
Protein structure
optionally
The framework for automatic running of multiple independent simulation runs
CABS simulation model
Trajectories in alpha-carbon representation
Analysis & visualization
Energy vs. temperature, residue contact maps
Selection of models of interest
e.g. trajectory fragments, representative models
Reconstruction to all-atom representation
All-atom protein models
representatives of different folding stages

Fig. 1 Multiscale characterization of protein dynamics pipeline with the use of the CABS model. The framework protocol for simulation and analysis described in this manuscript is marked with a *dashed line*

2 Materials

2.1 Input Data

The required input data are protein sequence and assigned (or predicted) secondary structure. The optional input is starting structure data (in PDB format; required for unfolding studies or RMSD-to-native analysis).

For barnase, the example protein studied in the Methods section, the following data have been used: sequence, structure (PDBID: 1BNR), and secondary structure assignment (by the DSSP method) [14]. For known protein structures, both PDB and DSSP files can be accessed from the PDB database (http://www.rcsb.org/), e.g., the files for 1BNR can be obtained using the following links:

http://www.rcsb.org/pdb/files/1bnr.pdb.

http://www.rcsb.org/pdb/files/1bnr.dssp.

2.2 Software

The required software modules are the CABS modeling package and the CABS Python wrapper (pyCABS) both available for download from http://biocomp.chem.uw.edu.pl/pycabs. For running the pyCABS, Python interface and necessary Python modules (listed in Subheading 3) are needed. For additional download and setup details *see* **Note 1**.

2.3 Skills

The user should have basic skills in Python language scripting as well as a basic knowledge of structural bioinformatics (particularly in the foundations of protein folding problems and the use of protein structural data).

2.4 Hardware

A computer running Linux/Unix with at least 3 GB of free hard-disk space for the output data. Since some of the protocols described here involve running multiple (up to one hundred) simulation runs, we recommend the usage of a multi-CPU workstation.

3 Methods

The details of the CABS protein model are described in ref. [15]. Below, step-by-step instructions are presented together with python script fragments (given in `Courier New` font style). The complete scripts are available from http://biocomp.chem.uw.edu.pl/pycabs.

3.1 Environment Preparation

Download the required software (for download instructions *see* **Note 1**). Next, the necessary Python modules need to be imported. Create a file with the *.py extension (e.g., `folding_pathway.py`) and type inside

```
#!/usr/bin/env python
import matplotlib as mmp
mmp.use('Agg')
import os, random, pylab, glob, pycabs, numpy as np,
multiprocessing as mp
```

The first line is the information for the system which interpreter should be used for running the script. The next two lines define the environment for creation of contact maps and standard deviation plots. The last line invokes imports of the multiprocessing module (for parallel execution of CABS software), pylab (for plotting the data), and pyCABS (for running CABS and processing CABS format files).

3.2 Running Isothermal Simulations

The following example describes how to run multiple simulations of protein folding dynamics, for the example protein barnase. The described simulation approach was used in the characterization of the barnase folding pathway in the work of Kmiecik and Kolinski [7].

Note that the results may vary in quantitative details due to possibly different simulation settings and/or later modifications of the CABS model.

It is recommended, but not required, to provide sequence and secondary structure information using the DSSP file format (for additional hints *see* **Note 2**):

```
sequence,secstr = pycabs.parseDSSPOutput("1bnr.dssp")
```

Alternatively, one can simply define it in `sequence` (protein sequence) and `secstr` (protein secondary structure) variables. The secondary structure should be defined for each amino acid in the three-letter code: H, a helix; E, an extended state; and C, a coil (less regular structures). In the case of secondary structure predictions, overpredictions of the regular secondary structure (H or E) are more dangerous for the quality of the results than underpredictions.

In previous works, as the first step in the characterization of long-term dynamics we found it convenient to execute multiple isothermal simulation runs in different temperatures. In the CABS algorithm, the temperature is the parameter controlling the acceptance ratio for new conformations (through an asymmetric Monte Carlo scheme).

To run simulations in a parallel fashion (one simulation on one thread), create a function definition (`runCABS`) for the multiprocessing threadpool:

```
name = "barnase"
template = ["/where/is/my/barnase/1bnr.pdb"]
independent_runs = 5
temp_from = 1.5
temp_to = 3.8
temp_interval = 0.05
temperatures = np.arange(temp_from,temp_to,temp_interval)

def runCABS(temperature):
   global name, sequence,secstr,template,independent_runs
   here = os.getcwd()
   for i in range(independent_runs):
      temp = "%06.3f" %(temperature)
      dir_name= name+"_"+str(i)+"_T"+temp
      a = pycabs.CABS(sequence,secstr,template,dir_name)
      a.rng_seed = random.randint(1,10000)
      a.createLatticeReplicas(replicas=1)
      a.modeling(Ltemp=temperature,Htemp=temperature,
phot=300,cycles=100,dynamics=True)
      os.chdir(here)

pool = mp.Pool()
pool.map(runCABS,temperatures)
```

The above code fragment contains a declaration of the `independent_runs` variable, which tells the script to start five independent (with a different pseudo-random number generator seed) simulations for each temperature, starting from 1bnr.pdb (the native structure). It also contains the `temperatures` variable, which is a list of temperatures in the range of 1.5–3.8 and interval 0.05. This gives a total number of $(3.8–1.5)/0.05 \times 5 = 230$ independent simulations (for additional details *see* **Note 3**). In order to start the simulations from extended random coil structures leave the `template` variable empty, i.e., `template=[]`. Doing so ensures that the simulation results are not biased from the starting structure. At elevated temperatures, due to the fast relaxation of the polypeptide chain, the simulation trajectory relatively quickly becomes independent from the starting structure.

The following parameters define the simulation length:

`cycles`—defines the number of CABS MC macrocycles [15] and determines the trajectory length (a number of trajectory snapshots is equal to `cycles` multiplied by 20, e.g., for `cycles=100` the resulting trajectory will have 2,000 snapshots).

`phot`—determines simulation length between the recorded snapshots.

The CABS-generated trajectories are produced in different output formats and representations: TRAF file (contains trajectory models in an alpha-carbon representation) and TRASG (contains trajectory models in a center-of-side-chain-mass representation). Both files are reformatted to a more popular PDB format. Additionally, each working directory contains an ENERGY file with CABS energy values for each model in a trajectory.

The CABS model (and the pyCABS module), developed primarily for protein structure prediction, enables application of distance restraints (derived from sparse experimental data or from theoretical predictions of plausible structural biases). For example instructions on running comparative modeling (with the use of structural template(s)), de novo modeling (template free), and modeling with the use of external distance constraints, *see* **Note 4**.

3.3 Calculating Simulation Statistics

Below are the instructions for the calculation of average CABS energy and standard deviation of energy values for the obtained trajectories. Both measures plotted in the function of temperature give an insight into the overall characteristics of the CABS energy landscape.

The standard deviation of energy (E) in function of temperature (T) is defined as

$$\sigma(T) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(E_i^T - \overline{E^T}\right)^2}$$

where N is the number of observables, and $\overline{E^T}$ is the mean in the given T.

To compute average energy and its standard deviation for each simulation, run the following code (`e_path` must be constructed identically to the `dir_name` variable in the `runCABS` procedure):

```
stdd = np.empty([independent_runs,len(temperatures)])
avgene = np.empty([independent_runs,len(temperatures)])
for j in range(independent_runs):
   for i in range(len(temperatures)):
     temp = "%06.3f" %(temperatures[i])
     e_path = os.path.join(name+'_'+str(j)+'_T'+temp,'ENERGY')
     energy = np.fromfile(e_path,sep='\n')[1000:]
     stdd[j][i] = np.std(energy)
     avgene[j][i] = np.mean(energy)
```

Adding the following commands

```
mean_sigma = np.mean(stdd,axis = 0)
stddev_sigma = np.std(stdd,axis = 0)
mean_ene = np.mean(avgene,axis = 0)
stddev_ene = np.std(avgene,axis = 0)
```

invokes computation of the average values from five independent simulations for each T value (*see* Fig. 2).

3.4 Plotting Simulation Statistics

To plot average CABS energy and standard deviation of energy values for the obtained trajectories (a single point denotes a single trajectory), users can apply the pylab module as in the code below

```
pylab.ylabel(r'Standard deviation of energy')
pylab.xlabel(r'Temperature, $T$')
pylab.xlim(temp_from,temp_to)
for i in range(independent_runs):
    pylab.plot(temperatures, stdd[i], '.')
pylab.errorbar(temperatures,mean_sigma,yerr=stddev_sigma,fmt='o-')
pylab.savefig("stdE_barnase.png",dpi=600)
pylab.close()
```

and analogously for the average energy plot (by changing `mean_sigma` to `mean_ene`, `stddev_sigma` to `stddev_ene` and `stdd` to `avgene`). The standard deviation of energy is written to `stdE_barnase.png` (upper panel in Fig. 2). The average energy plot is shown at the bottom of Fig. 2 (additional plotting options are given in **Note 5**).

3.5 Generating Contact Maps

Average contact maps (average for the entire isothermal trajectory or trajectory fragment of interest) provide a very informative insight into complex intramolecular interactions of highly diverse protein ensembles.

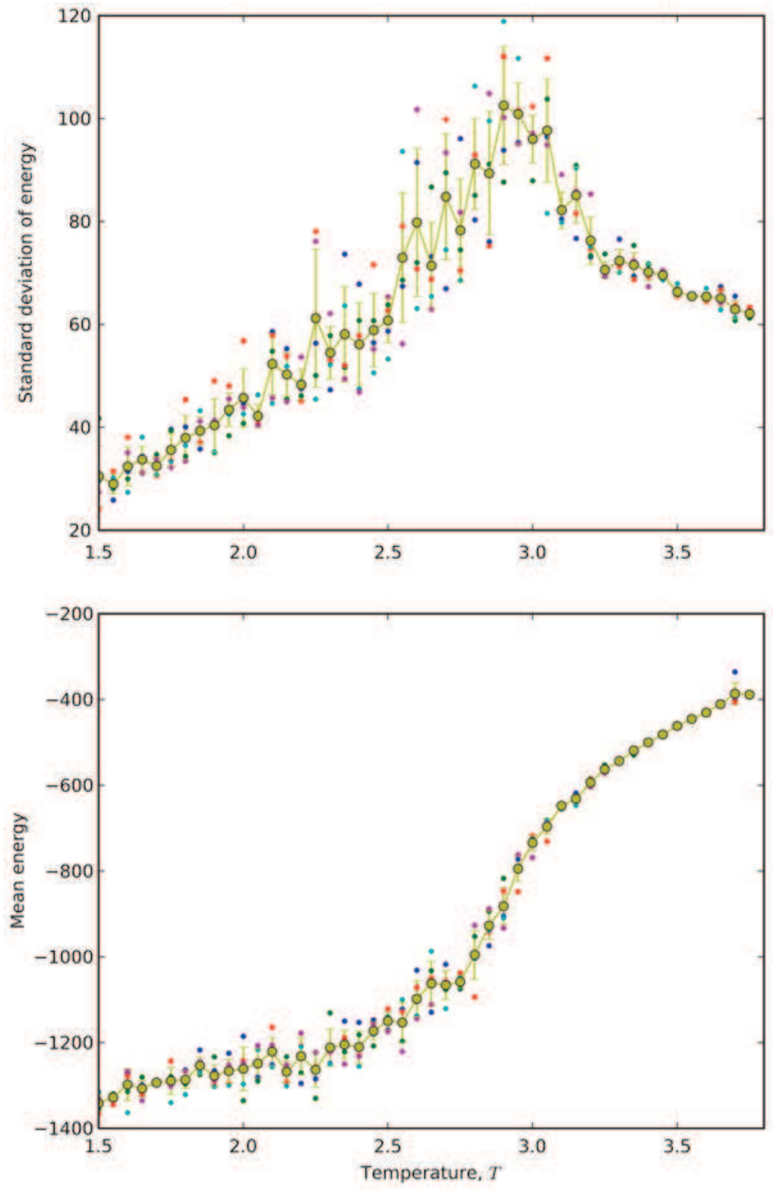

Fig. 2 CABS energy standard deviation (*above*) and CABS energy (*below*) as a function of temperature (*T*) for barnase (similar results were presented in ref. [7]). Variously colored small points represent individual isothermal simulations, while *larger yellow points* represent average value from five independent simulations in the given *T* value. The transition temperature (Tt) is identified by a steep drop of the energy and the peak of the energy standard deviation (heat capacity), here when $T = 2.9$. Tt cannot be strictly identified with the transition state of protein folding. Sometimes, as for chymotrypsin inhibitor (*see* ref. [7]), conformations observed at Tt may be relatively unstructured, with some features of a molten globule state. For a more exact estimation of the Tt value one can repeat the computations in a smaller range of temperatures, with a smaller `temp_interval` value

```
#!/usr/bin/env python
import matplotlib as mmp
mmp.use('Agg')
import pycabs,os,numpy as np
name = "barnase"
max_sd_temperature=2.9
independent_runs=5
trajectory = []
for j in range(independent_runs):
     temp = "%06.3f" %(max_sd_temperature)
     e_path = os.path.join(name+'_'+str(j)+'_T'+temp,'TRASG')
     trajectory += pycabs.loadSGCoordinates(e_path)[1000:]
```

The above code fragment loads the second half of trajectories in the center-of-side-chain-mass trace format from five independent simulations in the temperature 2.9. To calculate an average contact map (the contact map definition is given in **Note 6**) with the cutoff of 7.0 Å use

```
contact = pycabs.contact_map(trajectory,7.0)
```

and to write it to a file, use the pylab module (for the map coloring hint *see* **Note 7**):

```
from pylab import xlabel,ylabel,pcolor,colorbar,savefig,
xlim,ylim,cm
xlabel("Residue index")
ylabel("Residue index")
xlim(0,len(contact))
ylim(0,len(contact))
for k in range(len(contact)-3):
   for l in range(3):
         contact[k+l][k+l]=contact[k+l][k] = contact[k]
         [k+l] = contact[k][k]=0

pcolor(contact, cmap=cm.gnuplot2_r,vmax=0.6)
cb = colorbar()
cb.set_label("Fraction of contacts")
savefig("average_heatmap"+str(max_sd_temperature)+".png")
```

The example contact map, created as described above, is presented in Fig. 3.

Note that for generating contact map figures, instead of using the pylab module, one can use any specialized software for this purpose, e.g., Gnuplot program (for plotting instructions in Gnuplot *see* **Note 8**).

3.6 Selection of Models of Interest Using RMSD-to-Native and CABS Energy Measures

The resulted trajectories can be filtered and structurally analyzed using simple filters (for example CABS energy and RMSD-to-native cutoffs). More sophisticated structural analysis is perhaps most commonly performed with the use of clustering analysis [16] (like in the characterization of near-native ensemble in ref. [8] or transition state ensemble in ref. [9]) or principal component analysis [17].

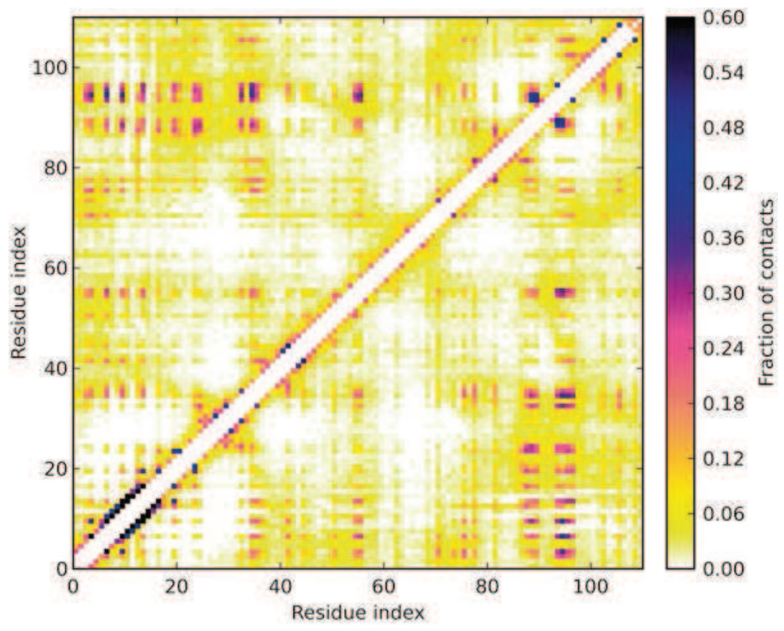

Fig. 3 Contact map for the intermediate (between fully denatured and near-native) state of barnase (similar results were presented in ref. [7]). The *colors* indicate the frequency of contacts. Short-range contacts are omitted for clarity

The simple filtering options are accessible from the provided modules. To filter out models dissimilar by RMSD to the native structure, one can use

```
#!/usr/bin/env python
import pycabs,os,numpy as np

name = "barnase"
rmsd_cutoff = 7.5
max_sd_temperature=2.9
independent_runs=5

native = pycabs.parsePDBfile("/path/to/barnase/1bnr.pdb")
trajectory = []
for j in range(independent_runs):
    temp = "%06.3f" %(max_sd_temperature)
    e_path = os.path.join(name+'_'+str(j)+'_T'+temp,'TRAF')
    for model in pycabs.loadTRAFCoordinates(e_path):
        if pycabs.rmsd(native,model) < rmsd_cutoff:
            trajectory += model
```

Note that each simulation directory contains an `ENERGY` file with the energy of each trajectory model. By reading it to memory (`numpy.fromfile("path/ENERGY",sep = "\n")`) the user can filter out models with a particular energy cutoff.

3.7 Reconstruction to All-Atom Representation

Selected individual models or consecutive trajectory fragments can be rebuilt to an all-atom representation. The task of reconstruction from alpha carbon trace is typically solved by a two-step procedure [9, 18]: backbone reconstruction from alpha carbon trace [19] followed by side-chain reconstruction [20] based on the position of backbone atoms. Note that models from the CABS method (in alpha carbon trace representation), as well as from the other CG modeling tools, are not free from unphysical local distortions. Therefore, building physically sound models from reduced models usually requires specialized reconstruction and refinement procedures [18].

4 Notes

1. All necessary applications can be downloaded from the following sources: Python (http://www.python.org), Pylab (http://www.scipy.org/PyLab), CABS/pycabs (http://biocomp.chem.uw.edu.pl), and GNUplot (http://www.gnuplot.info). All programs (except pyCABS and CABS) are available in most of the Linux distribution repositories.

   If one wants to compile the CABS software, use `g77 -O2 -static -ffloat-store -o cabs_dynamics CABS_dynamics.f` and move the `"cabs_dynamics"` file to the `FF` directory of pyCABS module.

   After downloading the pyCABS package, uncompress it into the working directory and modify the `pycabs.py` file by setting path to the FF directory. This can be done by changing the `self.FF` variable in the `__init__`method of the `CABS` class.
2. One can utilize secondary structure prediction software and write a predicted secondary structure (each residue in one-letter code: H—helix, E—extended, C—coil) in the `secstr` variable. Note that the Protein Data Bank does not contain DSSP files for all deposited proteins.
3. This task has taken about 28 h on 24 Intel® E5649 threads. That range of temperatures is typical for barnase; for other proteins it could be different. In order to roughly estimate the appropriate range, an initial simulation run can be performed with less computationally expensive operands: `temp_interval=1` and `independent_runs=1`. Note that `pool = mp.Pool()` uses all available CPUs by default, but the user can limit it, e.g., `pool = mp.Pool(4)`, to utilize only four CPUs.
4. Example instructions for running: comparative modeling (with the use of structural template(s)), de novo modeling (template free), and modeling with the use of external distance constraints.

The following is an example script for protein structure prediction by comparative modeling (with the use of secondary structure prediction (in the Porter method [21] format file) and three templates (e.g., from Pcons Structure Prediction Meta Server: pcons.net): t1.pdb, t2.pdb, t3.pdb). Residues in the template structure files have to be numbered according to the target sequence alignment:

```
#!/usr/bin/env python
import pycabs

sequence,sec_str = pycabs.parsePorterOutput("/absolute/
path/to/porter.ss")
working_dir = "prediction" # name of project
templates  =  ["/abs/path/to/t1.pdb","/abs/path/to/
t2.pdb","/abs/path/to/t3.pdb"]
a = pycabs.CABS(sequence,sec_str,templates,working_dir)
a.generateConstraints()
a.createLatticeReplicas(replicas = 10) # create start
models from templates
a.modeling(Htemp = 2.0,Ltemp = 1.0,cycles = 20,phot = 100)
```

The script presented above: (1) parses the secondary structure prediction file (one can directly define sequence and secondary structure in `sequence` and `sec_str` variables, respectively); (2) creates distance constraints from templates; (3) creates 10 starting structures projected on the CABS lattice (iteratively from each template), which can be viewed in PDB file format in the "`prediction`" directory; and (4) runs CABS simulation with REMC and simulated annealing in the temperature range from 2.0 to 1.0 (typical values for comparative modeling).

In order to run de novo modeling (without the use of templates/constraints) one needs to (1) specify the `sequence` and `sec_str` variables, (2) leave the templates empty (i.e., `templates = []`) and comment out the `a.generateConstraints()` line, and (3) run CABS simulation with REMC and simulated annealing in the temperature range from 3.5 to 1.0, `cycles=100, phot=100,` and `replicas=30,` which are typical settings for de novo modeling. Note that de novo modeling is an extremely difficult modeling task and the difficulty increases with the protein length. Thus, the procedure may be suitable for small proteins preferably not longer than 120 residues.

In order to introduce some external distance constraints (derived from sparse experimental data or from theoretical predictions of plausible structural biases), one can manually add the distances data before running the modeling procedure:

```
misc = []
misc.append((1, 40, 15.4, 16.6, 0.5))
a.generateConstraints(exclude_residues=range(1,1000),
other_constraints = misc)
```

The code fragment presented above (1) excludes all automatically generated constraints (`exclude_residues` for residues 1–999) and (2) adds user-provided constraint between the alpha carbon atoms of the residues No. 1 and No. 40 with the constraint range between 15.4 and 16.6 Å (the constraint range is a preferred distance between the selected alpha carbons) and the constraint force equal to 0.5. The variable `misc` is in the format of a list of tuples (`residue_i_index, residue_j_index, lower_distance, upper_distance, force`). If one needs to change the global force constraint, it is possible to do so by providing a new value for `constraints_force` (default 1.0) in the `modeling` method, i.e.,

```
a.modeling(Htemp = 2.0,Ltemp = 1.0,cycles = 20,phot = 100,
constraints_force = 2.0)
```

If the script is successfully terminated, the prediction results can be found in the "`prediction`" directory (TRAF.pdb file).

5. At the script level, one can define output plot parameters, e.g., label sizes, colors, and resolution (more visualization examples can be found at http://matplotlib.org).
6. Contact map C is a $N \times N$ matrix defined as

$$C(i,j) = C(j,i) = \begin{cases} 1 & \textit{if } d(x_i, x_j) < \textit{cutoff} \\ 0 & \textit{otherwise} \end{cases}$$

   where x_i is the position of the x-th atom (here the center of a mass of a side group of an i-th residue).
7. The `Pcolor` function of the pylab module has a `vmax` parameter which defines the maximum value of the colorbar scale. Manipulating the `vmax` value may be helpful for a proper visualization of contacts of interest.
8. Instead of using the pylab module, one can write text data to the output file. To write the contact array into a file formatted for GNUplot, write a file with three columns (i-th residue, j-th residue, contact fraction value) and leave a blank row each time before the i-th column changes its value:

```
fw = open("contact_map.dat","w")
for i in range(len(contact)):
    for j in range(len(contact)):
       fw.write("%5d %5d %7.5f\n" %(i+1,j+1,contact[i]
[j]))
    fw.write("\n")
 fw.close()
```

Note that in the example above, the script writes residue indexes starting from 1 (in pylab fragment it creates plots starting from 0).

Finally, plot `contact_map.dat` in GNUplot (and write to the postscript file with a font size suitable for presentation):

```
set terminal unknown
plot 'contact_map.dat' using 1:2:3
set xrange[GPVAL_DATA_X_MIN:GPVAL_DATA_X_MAX]
set yrange[GPVAL_DATA_Y_MIN:GPVAL_DATA_Y_MAX]

set terminal postscript eps enhanced color "Helvetica" 14
set output 'contact_map.eps'
set size ratio 1
unset key
set xlabel 'Residue index'
set ylabel 'Residue index'
set cbrange[:0.8]
set palette negative

plot 'contact_map.dat' with image
```

The first four lines of these GNUplot commands are responsible for the calculation of max/min values of axis data (1 to chain length); `set cbrange[:0.8]` sets the colorbar scale in the range of 0.0–0.8.

5 Case Studies

Below are brief descriptions of several applications of the CABS model, together with the post-processing analysis applied to the characterization of protein folding.

A staggering number of different protein conformations sampled during de novo simulations require post-processing strategies that reduce the vast conformational complexity into easy to understand and interpret data. The complex nature of intramolecular interactions of highly diverse ensembles can be relatively simply described by average contact maps (average for the entire isothermal trajectory or trajectory fragment of interest). As shown in the folding mechanism studies, the characterization of the appropriate protein ensembles in the form of the averaged residue contact maps (derived from the trajectories in CG representation), matched very well with the experimental data from protein engineering (phi value analysis) [7–10]. The relative contact frequencies from the CABS simulations were also shown to be in semiquantitative agreement with experimental data (phi value analysis, hydrogen-exchange protection factors) [8, 10, 11] and other theoretical predictions [8, 12]. In the case of the B1 domain of protein G folding studies [8], quantitative analysis of the clusters of the most persistent native long-range side-chain contacts and their evolvement from highly denaturing to native conditions allowed for a detailed (residue–residue contact level) description of the folding events. Apart from the contact-level description of the highly diverse ensembles, some persistent conformers appearing along the folding route can be structurally characterized through clustering analysis (as shown for the ensembles of the transition state of the B domain of protein A [9], and the native-like globule of the B1 domain of protein G [8]).

Acknowledgments

The authors acknowledge support from a TEAM project (TEAM/2011-7/6) co-financed by the EU European Regional Development Fund operated within the Innovative Economy Operational Program and from Polish National Science Center (Grant No. NN301071140) and from Polish Ministry of Science and Higher Education (Grant No. IP2011 024371).

References

1. Kmiecik S, Jamroz M, Kolinski A (2011) Multiscale approach to protein folding dynamics. In: Kolinski A (ed) Multiscale approaches to protein modeling. Springer, New York, pp 281–294
2. Lindorff-Larsen K, Piana S, Dror RO, Shaw DE (2011) How fast-folding proteins fold. Science 334:517–520
3. Schaeffer RD, Fersht A, Daggett V (2008) Combining experiment and simulation in protein folding: closing the gap for small model systems. Curr Opin Struct Biol 18:4–9
4. Rizzuti B, Daggett V (2013) Using simulations to provide the framework for experimental protein folding studies. Arch Biochem Biophys 531(1–2):128–135
5. Kolinski A, Bujnicki JM (2005) Generalized protein structure prediction based on combination of fold-recognition with de novo folding and evaluation of models. Proteins 61(Suppl 7):84–90
6. Debe DA, Danzer JF, Goddard WA, Poleksic A (2006) STRUCTFAST: protein sequence remote homology detection and alignment using novel dynamic programming and profile-profile scoring. Proteins 64:960–967
7. Kmiecik S, Kolinski A (2007) Characterization of protein-folding pathways by reduced-space modeling. Proc Natl Acad Sci USA 104: 12330–12335
8. Kmiecik S, Kolinski A (2008) Folding pathway of the B1 domain of protein G explored by multiscale modeling. Biophys J 94:726–736
9. Kmiecik S, Gront D, Kouza M, Kolinski A (2012) From coarse-grained to atomic-level characterization of protein dynamics: transition state for the folding of B domain of protein A. J Phys Chem B 116:7026–7032
10. Kmiecik S, Kolinski A (2011) Simulation of chaperonin effect on protein folding: a shift from nucleation-condensation to framework mechanism. J Am Chem Soc 133:10283–10289
11. Kmiecik S, Kurcinski M, Rutkowska A, Gront D, Kolinski A (2006) Denatured proteins and early folding intermediates simulated in a reduced conformational space. Acta Biochim Pol 53:131–144
12. Jamroz M, Orozco M, Kolinski A, Kmiecik S (2013) Consistent view of protein fluctuations from all-atom molecular dynamics and coarse-grained dynamics with knowledge-based force-field. J Chem Theory Comput 9:119–125
13. Kmiecik S, Gront D, Kolinski A (2007) Towards the high-resolution protein structure prediction. Fast refinement of reduced models with all-atom force field. BMC Struct Biol 7:43
14. Kabsch W, Sander C (1983) Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features. Biopolymers 22:2577–2637
15. Kolinski A (2004) Protein modeling and structure prediction with a reduced representation. Acta Biochim Pol 51:349–371

16. Jamroz M, Kolinski A (2013) ClusCo: clustering and comparison of protein models. BMC Bioinformatics 14:62
17. Maisuradze GG, Liwo A, Scheraga HA (2009) Principal component analysis for protein folding dynamics. J Mol Biol 385:312–329
18. Xu D, Zhang Y (2011) Improving the physical realism and structural accuracy of protein models by a two-step atomic-level energy minimization. Biophys J 101:2525–2534
19. Gront D, Kmiecik S, Kolinski A (2007) Backbone building from quadrilaterals: a fast and accurate algorithm for protein backbone reconstruction from alpha carbon coordinates. J Comput Chem 28: 1593–1597
20. Krivov GG, Shapovalov MV, Dunbrack RL Jr (2009) Improved prediction of protein side-chain conformations with SCWRL4. Proteins 77:778–795
21. Pollastri G, McLysaght A (2005) Porter: a new, accurate server for protein secondary structure prediction. Bioinformatics 21:1719–1720