Tutorial: Updates on *ncsu* Suite of *sander*

Mahmoud Moradi,[1, *] Christopher Roland,[2] and Celeste Sagui[2, †]

[1] *Beckman Institute, University of Illinois at Urbana-Champaign.*
[2] *Physics Department, North Carolina State University*

(Dated: July 4, 2015)

This tutorial explains additional unreleased features of *ncsu* suite of *sander*, *i.e.*, molecular dynamics (MD) engine of Amber package. The released version of the suite in Amber10-14 consists of four modules: `ncsu_abmd`, `ncsu_bbmd`, `ncsu_smd`, and `ncsu_pmd` allowing to perform algorithms such as adaptively biased MD (ABMD), steered MD (SMD), and umbrella sampling (US) with different single- or multiple-copy flavors such as multiple walkers (MW) and temperature/Hamiltonian replica exchange MD (REMD). The new unreleased features include: (1) ABMD with selection algorithm, (2) well-tempered ABMD, (3) driven ABMD, (4) string method with swarms of trajectories, and (5) new collective variables. (1) A selection algorithm, which is a bootstrapping-based resampling of conformations, is introduced to MW-ABMD (`ncsu_abmd` block) which is based on the method originally described in K. Minoukadeh *et al*, JCTC 2010, 6, 1008. The selection algorithm enhances the diffusion along the reaction coordinate such that innovative walkers are cloned, replacing less effective ones. (2) Well-tempered feature in ABMD is implemented (in both `ncsu_abmd` and `ncsu_bbmd` blocks) based on the idea introduced in A. Barducci *et al*, PRL 2008, 100, 020603. Well-tempered ABMD is associated with a smoother convergence as compared to conventional ABMD. The smoothness of the convergence can be tuned by using a temperature-like parameter. (3) Driven ABMD is based on driven adaptive-bias scheme introduced in M. Moradi *et al*, JPCL 2013, 4, 1882. Driven ABMD combines ABMD (either `ncsu_abmd` or `ncsu_bbmd`) and SMD (`ncsu_smd`) allowing for a uniform exploration of any desired segment of the configuration space, and is particularly useful when regular ABMD is too slow in exploring a region of interest in the reaction coordinate space. (4) String method with swarms of trajectories as implemented here is a parallel implementation of the method as described in A. C. Pan et al, JPCB 2008, 112, 3432. A new module, `ncsu_stsm` is added to the suite specifically for the swarms-of-trajectories string method (STSM). (5) Finally, three collective variables are added to the suite which can be employed along with the previously implemented collective variables in any of the *ncsu* modules.

I. METHODS

Adaptively biased MD or ABMD, along with other algorithms implemented in the *ncsu* suite of *sander*, allows for employing biasing potentials which enhance the sampling along user-defined collective variables. The original implementation of the suite in Amber10 [1] (also available in Amber11, Amber12 and Amber14) already includes a host of advanced free energy calculation methods, some of which are exclusive to this suite. The new features, which can be easily added to Amber12 or Amber14 using freely available patches `bugfix.amber12.ncsu` or `bugfix.amber14.ncsu`, include several distinct algorithms, most of which are used in conjunction with ABMD and SMD, and one of them is a new module that is a parallel variation of string method.

A. ABMD with Selection Algorithm

The multiple-walker ABMD method is based on a set of *non-interacting* walkers or replicas, all of which persist for the same duration. However, not all walkers are equally effective in sampling the configuration space. Often, these are “bunched-up” and/or clustered together, and kept near local metastable regions because of hidden barriers, which are along the degrees of freedom orthogonal to the reaction coordinate. For maximum efficiency, one would like to “encourage” walkers that are wandering in the under-explored regions, and penalize walkers in the oversampled regions, and achieve this “on-the-fly”. This requires a dynamic monitoring of the walkers, and their updating according to suitable, preselected fitness functions. A typical algorithm of this kind assigns the same weights to every walker, and lets them run for a preselected period of time. The “fitness” of each walker is then tested, which is a *de facto* measure of explored phase space overlap for each walker. Walkers who are wandering in under-sampled regions

* moradi@illinois.edu
† sagui@ncsu.edu

are given a stronger weight and replicated a number of times, while walkers found to be exploring oversampled regions are correspondingly killed. This kind of procedure is then repeated periodically, thereby accelerating the convergence to a uniform distribution of walkers in the reaction coordinate space. This "selection" algorithm is thus a bootstrapping-based resampling of conformations in that the walkers are bootstrapped periodically to flatten the free energy landscape more efficiently. We note that a variation of selection algorithm similar to the one implemented here has been introduced before for adaptive biasing force (ABF) method and implemented in NAMD (via scripts rather than internally in the NAMD code) [2].

The algorithm is performed at fixed time intervals of length τ. The weight w_i given to walker i at the n^{th} interval is proportional to $\exp(\int_{(n-1)\tau}^{n\tau} S(\zeta_t^i)\, dt)$ in which ζ_t^i is the instantaneous value of reaction coordinate vector at time t for walker i, $S(\zeta) = c\frac{\nabla^2\rho(\zeta)}{\rho(\zeta)}$, $\rho(\zeta)$ is the density of ζ, and c is a constant. The algorithm can be performed either during the entire simulation or until a stopping criterion is satisfied. The criterion is based on the entropy of weights $H = \sum_i w_i log(w_i)$. The selection algorithm will stop if $E_w = H - \log(\frac{1}{N})$ goes below $\epsilon \log(\frac{1}{N})$ in which N is the total number of walkers and $0 \leq \epsilon \leq 1$ is a constant. Note that $\log(\frac{1}{N})$ is the entropy of uniform weights. If $\epsilon = 0$, there will be no stop and if $\epsilon = 1$, the algorithm will stop no matter what values w_i's take.

Comparing to the original method [2], our implementation has the following novel features:

- Biasing potential is used to approximate the density (instead of histogram). Note that the biasing potential in ABMD is approximating the (unnormalized) density along the reaction coordinate.
- Laplacian is used as a generalization of second derivative for multi-dimensional cases (not addressed in the original implementation [2]).
- Since ABMD builds an analytic function for biasing potential (by using cubic B-spline curves/surfaces/hypersurfaces), the derivatives are calculated analytically (instead of using their difference quotient).

B. Well-tempered ABMD

In the original implementation of ABMD, a history-dependent biasing potential is used with a fixed rate of updating the biasing potential:

$$U_a(\zeta, t) = U^0(\zeta) + \int_0^t dt'\, \omega\, K(\zeta - \zeta^{t'}), \tag{1.1}$$

in which K is a kernel function, U^0 is an arbitrary function (*e.g.*, a flat function or an initial guess for the biasing potential), and ω is a constant, uniform energy rate. It has been shown that [3], once converged, $\left\langle U_a(\zeta, t \to \infty)\right\rangle_a \approx U_a^s(\zeta) + u(t)$ ($u(t)$ is an additive constant at each t) in which $U_a^s(\zeta)$ (*i.e.*, converged biasing potential) equals $-F(\zeta)$.

Using the same rate for updating the biasing potential during the simulation often results in quite rough convergence; rather than converging, the biasing potential fluctuates around $-F(\zeta)$ (with an amplitude which depends on ω). A solution to this problem has been proposed [4] by using a "well-tempered" ω:

$$U_a(\zeta, t) = U^0(\zeta) + \int_0^t dt'\, \omega(\zeta^{t'}, t')\, K(\zeta - \zeta^{t'}), \tag{1.2}$$

in which $\omega(\zeta, t)$ is a time-dependent, non-uniform energy rate $\omega_0 \mathrm{e}^{-\beta' U_a(\zeta,t)}$ ($1/\beta' = k_B T'$ and T' is a pseudo-temperature) that reduces to a constant ω_0 in the $\beta' \to 0$ limit (*i.e.*, conventional ABMD). It has been shown [4] that $\left\langle U_a(\zeta, t \to \infty)\right\rangle_a \approx U_a^s(\zeta) + u(t)$ ($u(t)$ is an additive constant at each t) in which $U_a^s(\zeta) = -(1 + \frac{\beta'}{\beta})^{-1} F(\zeta)$ or $F(\zeta) = -(1 + \frac{T}{T'}) U_a^s(\zeta)$.

C. Driven ABMD

ABMD and SMD schemes are both powerful nonequilibrium sampling methods; however, each comes with its own practical limitations. For instance, SMD is often associated with a very slow convergence if used for free energy calculations. However it can be used to explore the transition paths, at least qualitatively; an advantage over ABMD, in which the system starting from one end of the configuration space (the reactant) may take a long time to visit the other end (the product). SMD and ABMD schemes, however can be integrated into a novel driven adaptive-bias scheme, termed driven ABMD (D-ABMD) that takes advantage of both its driven and adaptive-bias components and is advantageous over both components in isolation. D-ABMD has an advantage over conventional (or well-tempered) ABMD in that it ensures the exploration of the transition pathway (from one end to the other) in the early stages of the simulation and gradually improves the estimate of the free energies almost uniformly along the reaction coordinate. D-ABMD has also an advantage over the conventional SMD in that the effective free energy surface gradually becomes smooth and flat such that the system can move along the reaction coordinate with progressively less amount of work. The D-ABMD method is similar to D-MetaD method, which was recently introduced in Ref.[5] as an example of driven, adaptive-bias schemes.

In order to combine the two schemes described above, we have developed a driven adaptive-bias scheme that adds an adaptive ($U_a(\zeta,t)$) and a driving ($U_d(\zeta,t)$) potential to the Hamiltonian. We use an iterative approach in which an independent simulation is performed from time $t=0$ to $t=T$ in the n^{th} iteration ($n=1,2,\ldots$), biased by the potential $U_d(\zeta,t)+U_a^n(\zeta,t)$ in which $U_d(\zeta,t)=\frac{k}{2}(\zeta-\eta(t))^2$ for all n ($\eta(t)$ is moving center of the SMD harmonic potential in the ζ space), and:

$$U_a^n(\zeta,t)=U^{n-1}(\zeta)+\int_0^t dt'\,\omega(\zeta^{t'},t')\,K(\zeta-\zeta^{t'})\,\mathrm{e}^{-\beta w^{t'}}, \tag{1.3}$$

in which w^t is either defined as the accumulated work or the transferred work. The accumulated and transferred works are defined as $w_{ac}^t=\int_0^t dt'\,\frac{\partial}{\partial t'}U_d(\zeta^{t'},t')$ and $w_{tr}^t=w_{ac}^t-U_d(\zeta^t,t)$ (note: this is actually transferred work shifted by $-U_d(\zeta^0,0)$). Theoretically the $\mathrm{e}^{-\beta w_{tr}^{t'}}$ factor or "constant weight" is more accurate but for practical reasons the $\mathrm{e}^{-\beta w_{ac}^{t'}}$ factor or "pulling wight" is preferred. For a discussion on "weight functions" used for the free energy estimation from conventional pulling experiments see Ref. [6]. Particularly, in our algorithm, the constant weight $\mathrm{e}^{-\beta w_{tr}^{t'}}=\mathrm{e}^{-\beta w_{ac}^{t'}}\mathrm{e}^{\beta U_d(\zeta^t,t)}$ may become instable for large biasing potentials. To avoid the instability in either case a cutoff for w^t is used (i.e., the algorithm will not be applied if w^t is smaller than the cutoff).

D. String Method

The swarms-of-trajectories string method [7] (STSM) is a path-finding algorithm that refines a putative transition pathway (or a string of conformations) iteratively until the pathway is converged. The string is defined by a number of nodes or images parameterized in a high-dimensional space of collective variables whose position is updated in each iteration. The drift applied to each image is estimated by averaging over the drifts of a swarm of short unbiased trajectories all starting at the current position of the image. The parallel variation of the method, which is considerably more efficient than the original variation, requires hundreds to thousands of replicas of the system to run simultaneously [8].

For a string of N nodes, each consisting of M copies, $N\times M$ replicas will be required. Each unbiased drifting iteration is followed by a biased equilibration stage to move the replicas to their updated position. Unlike the ABMD simulations, STSM simulations can easily employ many collective variables since the sampling always stays along a 1D curve. Therefore one may use all collective variables that are somewhat relevant to the transition of interest, particularly those associated with slow dynamics. The number of images used is important in the accuracy of the pathway and the number of copies used is important in determining the convergence behavior.

Our parallel implementation of STSM method is based on iterative unbiased and restrained MD simulations followed with reparametrization of the restraint centers defined in the multidimensional collective variable space of ζ. At iteration s, M copies of image i restrained around old center η_i^{s-1} equilibrate for τ_E steps ($t=0,\ldots,\tau_E$) such that $\zeta_{i,l}^t$ for each copy l of image i is expected to be close to η_i^{s-1} at $t=\tau_E$ assuming the restraint is stiff enough. The restraint is then released to allow drifting for τ_R steps. The drifted center η_i^s for image i is first determined by averaging over all drifted copies $\zeta_{i,l}^t$ at $t=\tau_E+\tau_R$. The resulting string of images is smoothed using the protocol proposed in Ref.[9]. The smoothing parameter $0\le\epsilon\le1$ determines the smoothness of the curve and is recommended to be on the order of $1/(N-1)$. The last step is the reparametrization which again follows the same protocol proposed in Ref.[9] to generate a set of N sequentially equidistant centers along the same string formed by the smoothed centers.

II. USAGE

How to install

If you have an unaltered version of Amber12 or Amber14—all you need in the latter case is AmberTools, which is freely available at ambermd.org—you can easily apply the patch provided along with this tutorial (bugfix.amberXX.ncsu patch for AmberXX or bugfix.ambertools15.ncsu patch for AmberTools15 of Amber14). Check if you have `patch_amber.py` or `update_amber` in your `$AMBERHOME`. Either one can be used to apply or reverse patches:

- `./patch_amber.py --apply-patch=PATH-TO-FILE/bugfix.amber12.ncsu`
- `./update_amber --apply-patch=PATH-TO-FILE/bugfix.amber14.ncsu`

in which PATH-TO-FILE is wherever the patch is located. You can simply reverse to the unmodified version by:

- `./patch_amber.py --reverse-patch=bugfix.amber12.ncsu`
- `./update_amber --reverse-patch=bugfix.amber14.ncsu`

Note that the `bugfix.amberXX.ncsu` or `bugfix.ambertools15.ncsu` patch modifies the following files in the `$AMBERHOME/src/sander` directory (for Amber12) or `$AMBERHOME/AmberTools/src/sander` directory (for Amber14):

- `Makefile`
- `ncsu-colvar.F90`
- `ncsu-colvar-type.F90`
- `ncsu-abmd-hooks.F90`
- `ncsu-bbmd-ctxt.F90`
- `ncsu-bbmd-hooks.F90`
- `ncsu-constants.F90`
- `ncsu-sander-hooks.F90`
- `ncsu-smd-hooks.F90`
- `ncsu-umbrella.F90`

and adds the following files:

- `ncsu-cv-PAIR_DIHEDRAL.F90`
- `ncsu-cv-PATTERN_DIHEDRAL.F90`
- `ncsu-cv-SIN_OF_DIHEDRAL.F90`
- `ncsu-stsm-hooks.F90`

Once you have the files updated, you need to go to the sander's source directory (`src/sander`) and use the `./makedepend > depend` script. Now you can go to $AMBERHOME and use the `make install` command or go to `src/sander` and use `make sander.MPI` or `make parallel` command (assuming the configuration is already done and is done with the "parallel" version). Note that the latter option will generate a new `sander.MPI` file in sander's source directory.

How to use

In the following, we assume the reader is familiar with the usage of `ncsu` suites of `sander` in Amber10-14.

1. *Selection Algorithm*

The selection algorithm currently works with `ncsu_abmd` block only. The algorithm should be used only within the multiple-walker scheme (*i.e.*, when command-line `-rem` flag is set to zero). There are three flags added to `ncsu_abmd` block in the `mdin` files (*i.e.*, *sander*'s *input* files):

- `selection_freq = INTEGER :` positive integer number that sets the frequency of the resampling algorithm (in MD steps). If `selection_freq` is not specified, the selection algorithm will not be used.
- `selection_constant = REAL :` positive real number that sets the parameter c. if `selection_freq` is specified, specifying `selection_constant` is required (no default value). Parameter c is to determine how strong the selection mechanism is. If c is too large, all the walkers will be replaced with the most dominant one. If c is too small, there will be no killing/duplicating. Note that since $S(\zeta) = c\frac{\nabla^2\rho(\zeta)}{\rho(\zeta)}$ is integrated over `selection_freq` number of steps, c should be adjusted accordingly. In other words, the larger `selection_freq`, the smaller the `selection_constant` should be to reproduce a similar weight distribution over the walkers.
- `selection_epsilon = REAL :` positive real number that sets the stopping criterion parameter ϵ. Parameter ϵ determines the threshold for stopping the selection algorithm. If `selection_epsilon` is not specified, there will be no stop to the algorithm. If `selection_epsilon` is equal or larger than one, the algorithm will be stopped after the first attempt. If `selection_epsilon` is too small, the algorithm may not stop (within the simulation time).

2. *Well-tempered ABMD*

The well-tempered flavor can be used within either `ncsu_abmd` or `ncsu_bbmd` block. There is only one flag relevant to the well-tempered feature. Thus, if the following flag is used the ABMD algorithm will use the well-tempered scheme:

- `wt_temperature = REAL :` positive real number that sets the pseudo-temperature T'. If this flag is not specified, conventional ABMD will be used (*i.e.*, $T' \to \infty$ or $\beta' \to 0$). The smaller the T'; the smoother/slower the convergence. Free energy estimated from the negative of the biasing potential needs to be scaled by $1 + (T/T')$ in which T is the reference temperature of the system (`temp0`).

3. *Driven ABMD*

Driven ABMD can be performed using `ncsu_smd` block (for the SMD part of the algorithm) along with either `ncsu_abmd` or `ncsu_bbmd` block (for the ABMD part of the algorithm). There is no additional flag for the `ncsu_smd` block relevant to the algorithm; however, there are two additional flags to ABMD relevant to the "driven" feature which can be used within either `ncsu_abmd` or `ncsu_bbmd` block.

- `driven_weight = STRING :` sets the weighting scheme. The default option (*i.e.*, not using the flag) is `NONE` which indicates no reweighting is used (NOT RECOMMENDED if SMD is performed along ABMD). Other options include `CONSTANT` and `PULLING` for constant and pulling reweighting protocols (see Methods).
- `driven_cutoff = REAL :` real number that sets a cutoff for work for applying the reweighting algorithm (default: 0.0). If the work (accumulated or transferred depending on the scheme) at any given time is lower than the cutoff, no reweighting is done at that particular time. If the cutoff is too small, it may result in instability of the algorithm.

4. *String Method*

A new block (`ncsu_stsm`) is defined specifically for STSM algorithm. `ncsu_stsm` block is somewhat similar to `ncsu_pmd` block which is used for umbrella sampling in Amber10-14. One needs $N \times M$ replicas (N and M being the number of images and copies (per image), respectively) of the system with a separate `mdin` file for each, similar to multiple-walker ABMD simulations with command-line `-rem` flag set to zero.

The `ncsu_stsm` block should be present in all `mdin` files. The format is similar to `ncsu_pmd` block which specifies the restraining protocol including collective variables (`variable`), harmonic constants (`anchor_strength`), and initial values of the centers (`anchor_position`) with the addition of following flags:

- `image` = `INTEGER` : positive integer number that sets the image id (between 1 and N).
- `num_copies` = `INTEGER` : positive integer that sets the number of copies for each image (*i.e.*, M). If specified, `image` will be automatically set for each replica to specify M copies to each image (unless `image` flag is specified which overrides `num_copies`).
- `equilibration` = `INTEGER` : non-negative integer number that sets the number of MD steps specified for biased equilibration (restraining) at each iteration.
- `release` = `INTEGER` : Number of MD steps specified for the release (drift) at each iteration. Note: The total number of iterations is determined by the total simulation time (`nstlim` flag in `mdin` file) divided by total time for each iteration given by `equilibration`+`release`.
- `smoothing` = `REAL` : positive number that sets the smoothing parameter for reparametrization (between 0 and 1). Smoothing parameter should be, preferably, on the order of $1/(N-1)$. If this flag is not used, no smoothing will be performed.
- `report_centers` = `STRING` : determines if drifted and/or smoothed and/or reparametrized centers will be reported. The default value is `NONE` and other available options include `ALL`, `DRIFT`, `SMOOTHED`, `REPARAMETRIZED`, `NO_DRIFT`, `NO_SMOOTHED`, `NO_REPARAMETRIZED`.

A. Dihedral-based Collective variables

Two reaction coordinates or collective variables based on dihedral angles are already available within Amber10-14 including `TORSION` and `COS_OF_DIHEDRAL`. Three additional collective variables are now available including:

- `type = SIN_OF_DIHEDRAL` : sum of sines of a list of dihedral angles formed by atoms with indices `i`. The number of atoms must be a multiple of four (See *Reaction Coordinates* in AMBER10-14 manual). For a list of dihedral angles such as $\{\alpha_1, \ldots, \alpha_N\}$, `SIN_OF_DIHEDRAL` is $\sum_{i=1}^{N} \sin(\alpha_i)$.
- `type = PAIR_DIHEDRAL` : sum of cosines of a list of angles each formed by summing two neighboring dihedral angles from a list formed by atoms with indices `i`. The number of atoms must be a multiple of four. For a list of dihedral angles such as $\{\alpha_1, \ldots, \alpha_N\}$, `PAIR_DIHEDRAL` is $\sum_{i=1}^{N-1} \cos(\alpha_i + \alpha_{i+1})$ which ranges between $-N+1$ and $N-1$.
- `type = PATTERN_DIHEDRAL` : a particular pattern-recognizing function defined on a list of dihedral angles formed by atoms with indices `i`. The number of atoms must be a multiple of four. The definition is particularly relevant for the dihedral angles with a binary-like behavior of being either around 0 or 180 (*e.g.*, ω backbone dihedral angle). For a list of dihedral angles such as $\{\alpha_1, \ldots, \alpha_N\}$, `PATTERN_DIHEDRAL` is $\sum_{i=1}^{N} \cos^2(\alpha_i/2) 2^{i-1}$ which ranges between 0 and $2^N - 1$.

III. EXAMPLES

The following examples illustrating the new features of `ncsu` suites are provided with a step-by-step instruction with all necessary input files and scripts in the `examples.tar.bz2` tarball. After extracting the files (using command `tar xvjf examples.tar.bz2`), you will be able to run the provided run and analysis scripts following the `README` file in each one of the four folders present.

A. Deca-alanine peptide in implicit water

In order to test the selection algorithm we use an ABMD simulation of deca-alanine peptide ($\mathrm{Ac-(ALA)_{10}-NH_2}$) in implicit solvent. The collective variable used in these simulations is the end-to-end distance (d_{end}) which measures the distance between the two carbon atoms at the termini. You can perform the simulations with and without the selection algorithm. All the following steps are scripted and all the scripts and input files are provided in the `A-deca-alanine` folder. These steps are as follows:

- Step 0: building the system using LEAP program (a stretched deca-alanine).
- Step 1: relaxing the system by minimization (needed prior to any MD simulation).
- Step 2: compressing the system using SMD to generate several conformations to be used for ABMD simulations.
- Step 3: running multple walker ABMD without selection. We use a 5 Å resolution with 4 walkers, each running for 1 ns.
- Step 4: running multple walker ABMD with selection. We use a 5 Å resolution with 4 walkers, each running for 1 ns.
- Step 5: running multple walker ABMD without selection. We use a 1 Å resolution with 8 walkers, each running for 1.4 ns.
- Step 6: running multple walker ABMD with selection. We use a 1 Å resolution with 8 walkers, each running for 1.4 ns.
- Step 7: generating Figures 1 to 7 from the results of the simulations using `fig.sh` script (`Gnuplot` program is needed).

In simulations that use the selection algorithm, selection constant c is set to 0.001 and the resampling is attempted every 10000 MD steps (10 ps). We do not use any stopping criterion in deca-alanine simulations. Following the `README` file, you will notice the analysis scripts are also provided for each step, allowing you to produce the data needed for the last step. You can then compare your results summarized in the figures generated in Step 7 with those shown in Section IV.

B. Di-alanine peptide

In order to show the workings of the selection algorithm in a 2D ABMD simulation and also test the stopping criterion that is also implemented in the code, we use a di-alanine peptide ($Ac-(ALA)_2-NH_2$) in implicit solvent. The collective variables used in these simulations are backbone dihedral angles ϕ and ψ. You can perform the simulations with and without the selection algorithm. In the latter case, a stopping criterion may or may not be used. All the steps are scripted in the example and can be performed simply by following the `README` file in the `B-di-alanine` directory. These steps are as follows:

- Step 0: building the system using LEAP program (a stretched deca-alanine).
- Step 1: relaxing the system by minimization (needed prior to any MD simulation).
- Step 2: pushing the system along ϕ using SMD to generate several conformations to be used for ABMD simulations.
- Step 3: running multple walker ABMD without selection.
- Step 4: running multple walker ABMD with selection but without a stopping criterion.
- Step 5: running multple walker ABMD with selection and with a stopping criterion ($\epsilon = 0.001$).
- Step 6: running multple walker ABMD with selection and with a stopping criterion ($\epsilon = 0.01$).
- Step 7: generating Figures 8 to 9 from the results of the simulations using `fig.sh` script.

Note that all ABMD simulations use 4 walkers, each running for 10 ns. In simulations that use the selection algorithm, selection constant c is set to 0.00001 and the resampling is attempted every 100000 MD steps (100 ps). We have also provided a gnuplot-based script to generate figures from the output files. The output files are also provided separately. Figures 8 to 9 summarize the results.

C. Deca-alanine *in vacuo*

In order to test all new ABMD features including well-tempered, driven, and selection algorithms, we use an ABMD simulation of deca-alanine peptide ($Ac-(ALA)_{10}-NH_2$) *in vacuo*. The collective variable used is the end-to-end distance (d_{end}) as defined above. The following steps are included with appropriate scripts and input files (see `README` file in `C-deca-alanine`):

- Step 0: building the system using LEAP program (a stretched deca-alanine).
- Step 1: relaxing the system by minimization (needed prior to any MD simulation).
- Step 2: compressing the system using SMD to generate several conformations to be used for ABMD simulations.
- Step 3: running multple walker well-tempered ABMD without selection. We use a 1 Å resolution with 4 walkers, each running for 10 ns. The pseudo-temperature is set to 10000 K.
- Step 4: running multple walker well-tempered ABMD with selection algorithm and with the same setting as in Step 3.
- Step 5: running multple walker well-tempered D-ABMD. Most ABMD parameters are set to the same values in Steps 3 and 4; however, the simulation is iterative with 100 iterations of 100-ps simulations. Unidirectional SMD is performed along with the ABMD at each iteration, stretching the compressed peptide from 10 to 35 Å thus only covering this range in free energy calculations.
- Step 6: generating Figures 10 to 11 from the results of the simulations using `fig.sh` script.

Note that the simulation time and/or number of iterations used is not enough for reliable convergence, thus the longer simulations, and/or more number of replicas, and/or more iterations are needed to reach convergence. Nonetheless, Figure 11 shows the smoother trend of convergence in D-ABMD as compared to non-driven ABMD.

D. String method applied to di-alanine peptide

In order to show the workings of our parallel implementation of string method with swarms of trajectories we use the same system used in Example B. The collective variables used are dihedral angles ϕ and ψ. The initial pathway is a linear curve of 8 images connecting $(-3, 3)$ and $(1, 0)$ points in the (ϕ, ψ) space. 16 copies are used for each image resulting in 128 replicas for the entire ensemble. 100 iterations of simulation time 1 ps are performed. You can perform the simulations and analyses simply by following the `README` file in the `D-di-alanine` directory. These steps are as follows:

- Step 0: steering the system to generate initial pathway. You will use SMD to generate 8 conformations along a linear path as described above. Note: for building the system, minimization, and some initial equilibration, you may use the protocol described in Example B. However, we have already provided the necessary input files generated by the same protocol.
- Step 1: equilibrating the initial conformations (with restarint). Restrained MD (or umbrella sampling) will be used to equilibrate the nonequilibrium conformations generated in step 0.
- Step 2: running string method with swarms of trajectories to optimize the pathway, followed by analysis and generating Fig. 12. The equilibration (or restraining) is 0.98 ps/iteration and release (or drifting) is 0.02 ps/iteration.
- Step 3: running string method with swarms of trajectories with longer drifting periods, followed by analysis and generating Fig. 13. The equilibration (or restraining) is 0.9 ps/iteration and release (or drifting) is 0.1 ps/iteration.

Steps 2 and 3 are performed with different drifting times. Note that the method is reliable only if short drifting periods are used. Therefore, the pathway in Fig. 12 more closely represents the most probable pathway (assuming (ϕ, ψ) is a good reaction coordinate), although the pathway in Fig. 13 is converged as well.

---

[1] D. A. Case, T. A. Darden, T. E. Cheatham III, C. L. Simmerling, J. Wang, R. E. Duke, R. Luo, M. Crowley, R. C. Walker, W. Zhang, K. M. Merz, B. Wang, S. Hayik, A. Roitberg, G. Seabra, I. Kolossvary, K. F. Wong, F. Paesani, J. Vanicek,

X. Wu, S. R. Brozell, T. Steinbrecher, H. Gohlke, L. Yang, C. T. J. Mongan, V. Hornak, G. Cui, D. H. Mathews, M. G. Seetin, C. Sagui, V. Babin, and P. A. Kollman, *"AMBER 10"* (University of California, San Francisco, 2008).
[2] K. Minoukadeh, C. Chipot, and T. Lelievre, **6**, 1008 (2010).
[3] G. Bussi, A. Laio, and M. Parrinello, Phys. Rev. Lett. **96**, 090601 (2006).
[4] A. Barducci, G. Bussi, and M. Parrinello, Phys. Rev. Lett. **100**, 020603 (2008).
[5] M. Moradi and E. Tajkhorshid, **4**, 1882 (2013).
[6] H. Oberhofer and C. Dellago, J. Comp. Chem. **30**, 1726 (2009).
[7] A. C. Pan, D. Sezer, and B. Roux, J. Phys. Chem. B **112**, 3432 (2008).
[8] W. Jiang, J. Phillips, L. Huang, M. Fajer, Y. Meng, J. Gumbart, Y. Luo, K. Schulten, and B. Roux, **185**, 908 (2014).
[9] L. Maragliano, A. Fischer, E. Vanden-Eijnden, and G. Ciccotti, J. Chem. Phys. **125**, 024106 (2006).

Appendix: FIGURES

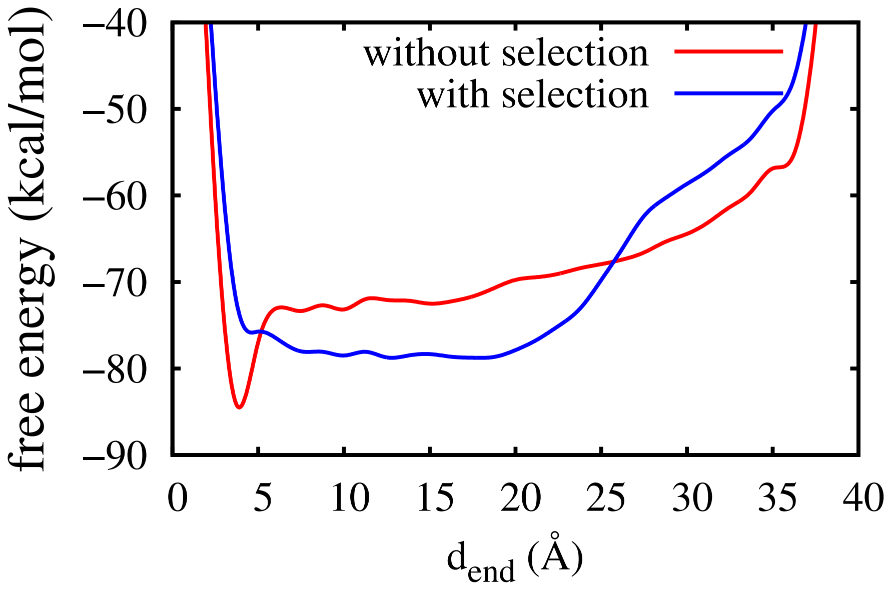


FIG. 1. **Example A:** (`fig5A.pdf`) Free energy of deca-alanine along d_{end} as obtained from ABMD simulations with (blue) and without (red) a selection algorithm (steps 3 an 4). The results are based on 4 walkers, each running for 1 ns. The resolution of 5 Å was used. This resolution is too large and often result in artifacts in ABMD simulations. The deep minimum around d_{end} =4 Å is one such artifact. Interestingly, this artifact is almost gone when the selection mechanism is turned on.

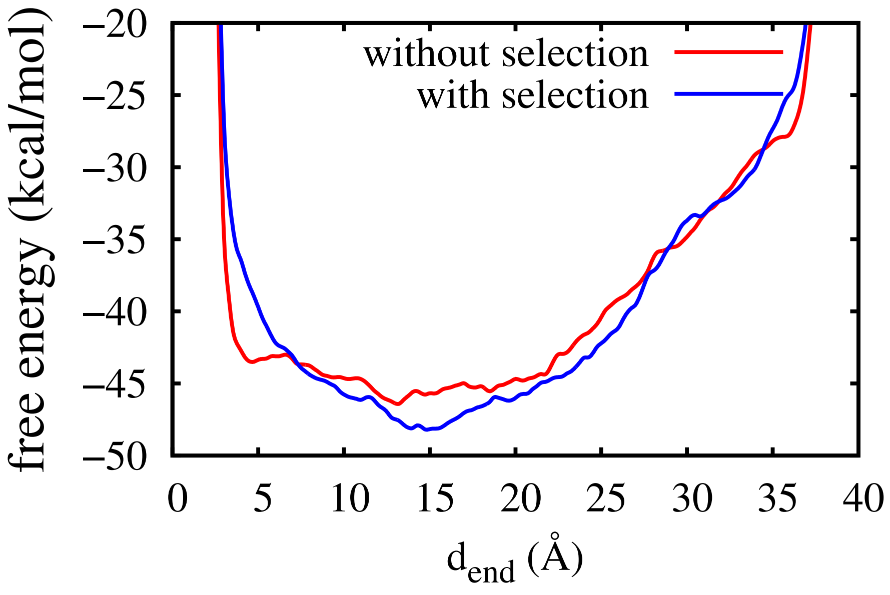


FIG. 2. **Example A:** (`fig1A.pdf`) Free energy of deca-alanine along d_{end} as obtained from ABMD simulations with (blue) and without (red) a selection algorithm (steps 5 an 6). The results are based on 8 walkers, each running for 1.4 ns. The resolution of 1 Å was used. Unlike Fig. 1, there is no deep minimum around d_{end} =4 Å in either case. However, the simulation without resampling results in a lower free energy around d_{end} =4 Å. The reason is the walkers around this minimum often get stuck unless they are killed by the selection mechanism.

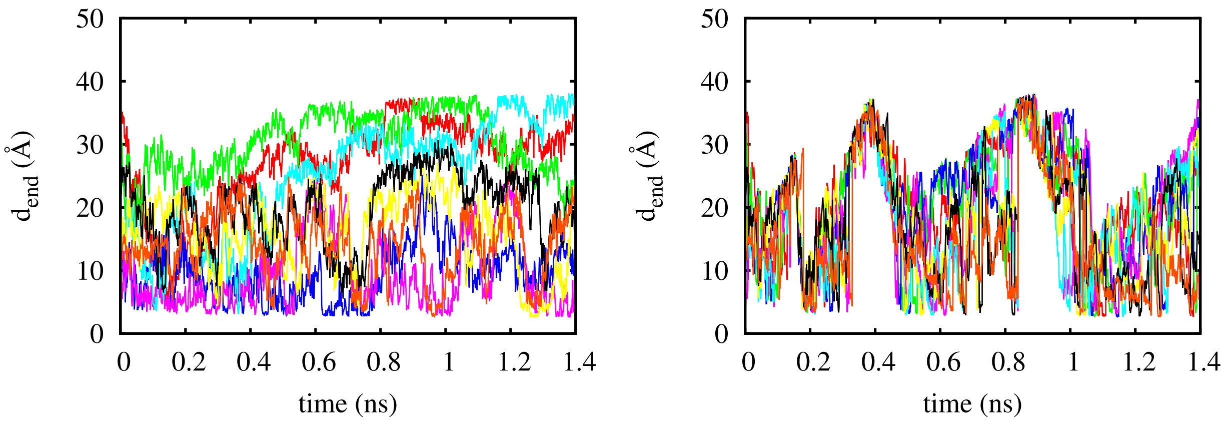


FIG. 3. **Example A:** (`figX1A.pdf`) The trajectory of deca-alanine walkers obtained from ABMD simulations with (right) and without (left) a selection algorithm (steps 5 an 6). The resampling mechanism (right panel) tends to occasionally move the walkers towards regions that are poorly sampled (larger d_{end}).

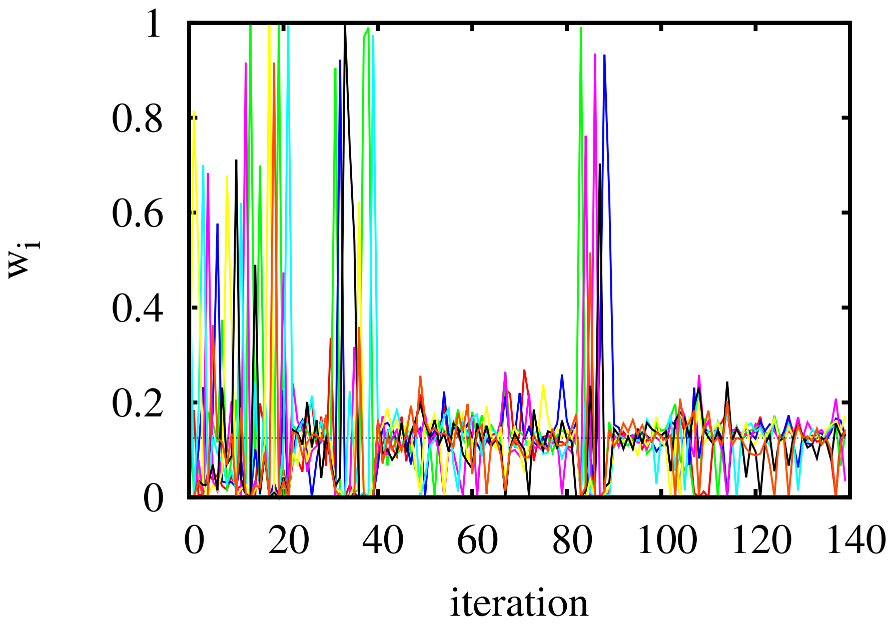


FIG. 4. **Example A:** (`figW1A.pdf`) The importance weights (w_i's) along the same trajectory shown in Fig. 3 (right panel) (steps 6). After about 40 iterations (400 ps) the weights stay around the uniform weight of 1/8 but there are occasional violations (*e.g.*, iterations 80-85).

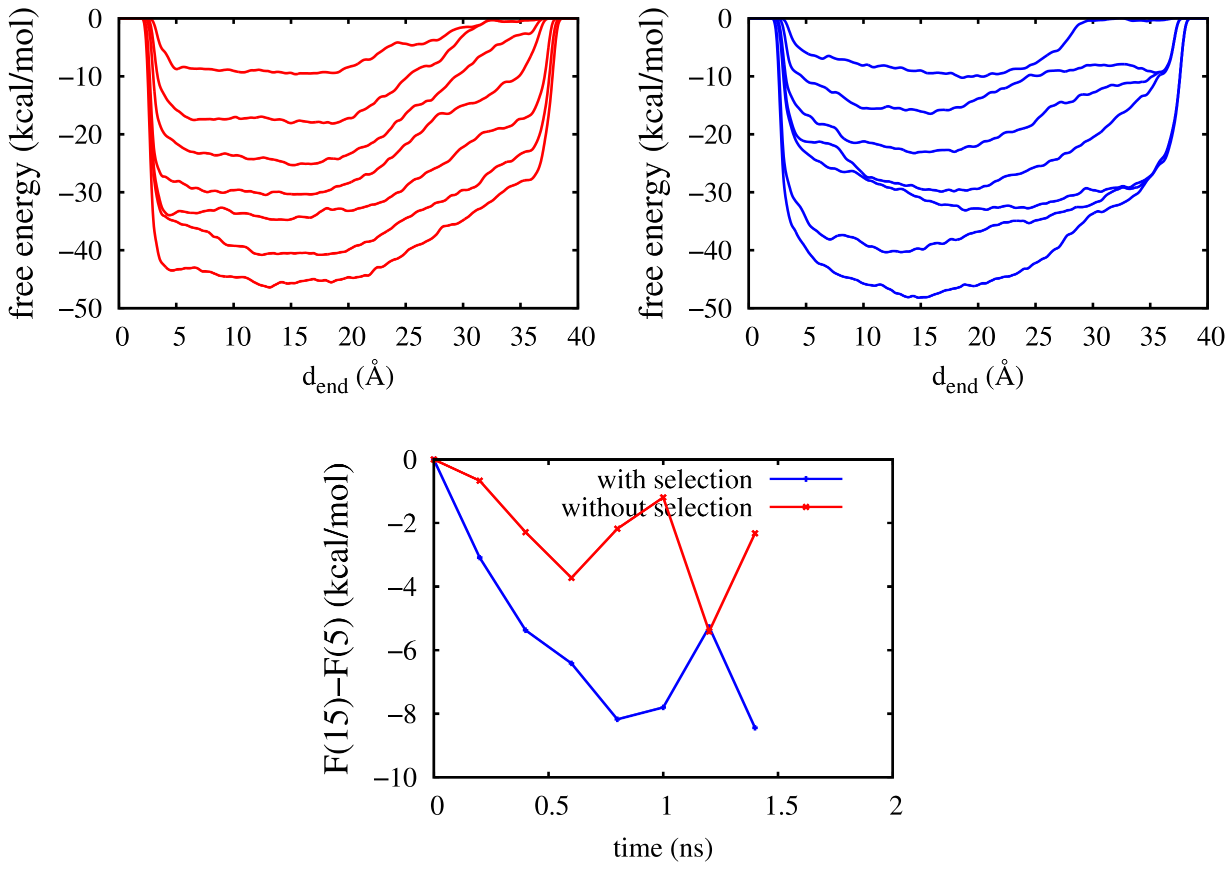


FIG. 5. **Example A:** (`fig8w.pdf`) (top panel) Evolution of free energy profile of deca-alanine along d_{end} as obtained from ABMD simulations with (blue) and without (red) a selection algorithm (steps 5 an 6). The snapshots are taken at 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, and 1.4 ns (*i.e.*, folders 00 to 06, respectively). (bottom panel) The free energy difference of $d_{end} = 15$ Å and $d_{end} = 5$ Å along these trajectories.

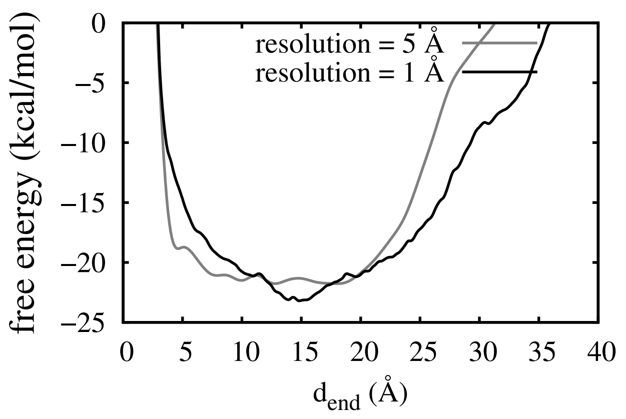


FIG. 6. **Example A:** (`figS.pdf`) Free energy of deca-alanine along d_{end} as obtained from ABMD simulations with a selection algorithm, using a resolution of 1 Å (black) and 5 Å (grey) (steps 4 an 6). The latter is more accurate since: (i) more walkers are used (8 vs 4), (ii) longer simulation time is used (1.4 vs 1 ns), and (3) smaller resolution is used (1 vs 5 Å).

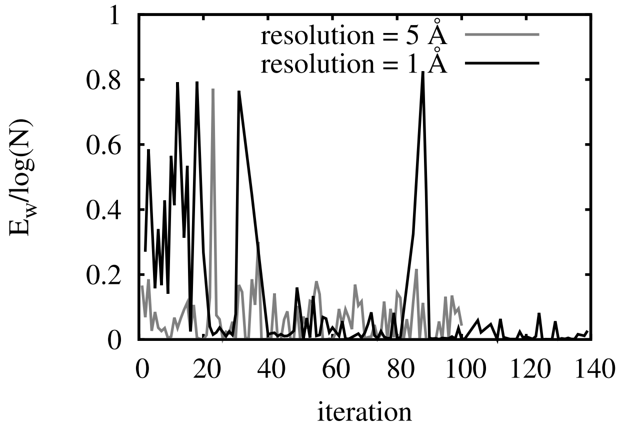


FIG. 7. **Example A:** (`figW.pdf`) The relative entropy $\frac{E_w}{\log(N)}$ along the ABMD trajectories with a selection algorithm, using a resolution of 1 Å (black) and 5 Å (grey) (steps 4 an 6).

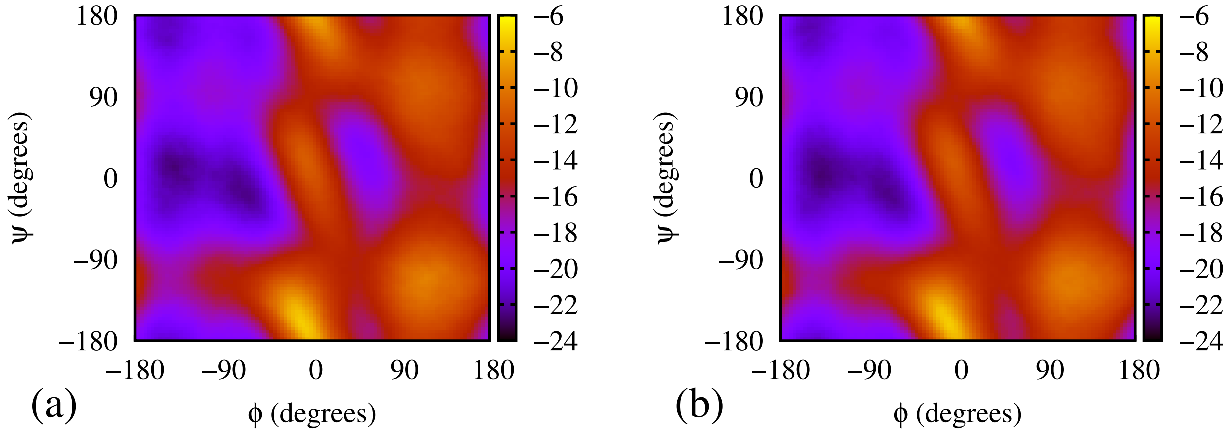


FIG. 8. **Example B:** (`fig-di-FE.pdf`) Free energy map of di-alanine peptide as obtained from ABMD simulations with (right) and without (left) a selection algorithm (steps 3 an 4). The results are based on 4 walkers, each running for 10 ns. Both simulations converge to the same free energy map in 10 ns. The selection mechanism is only helpful in speeding up the process.

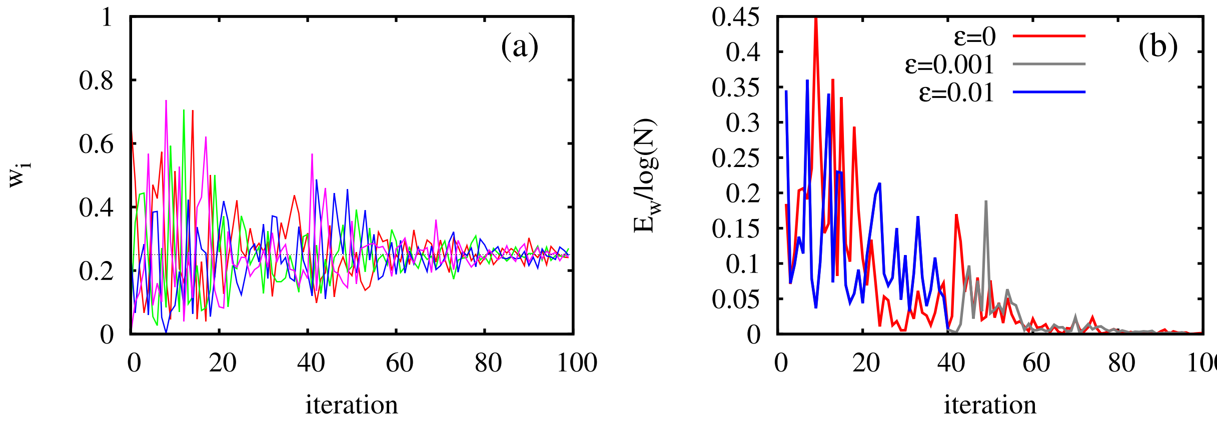


FIG. 9. **Example B:** (`fig-di-E.pdf`) (a) Resampling weights (w_i's) of di-alanine peptide as obtained from ABMD simulations with a selection algorithm and without a stopping criterion (step 4). (b) The relative entropy $\frac{E_w}{\log(N)}$ along the ABMD trajectories with a selection algorithm, using different stopping criterion conditions including $\epsilon = 0$ (no stop), 0.01, and 0.001.

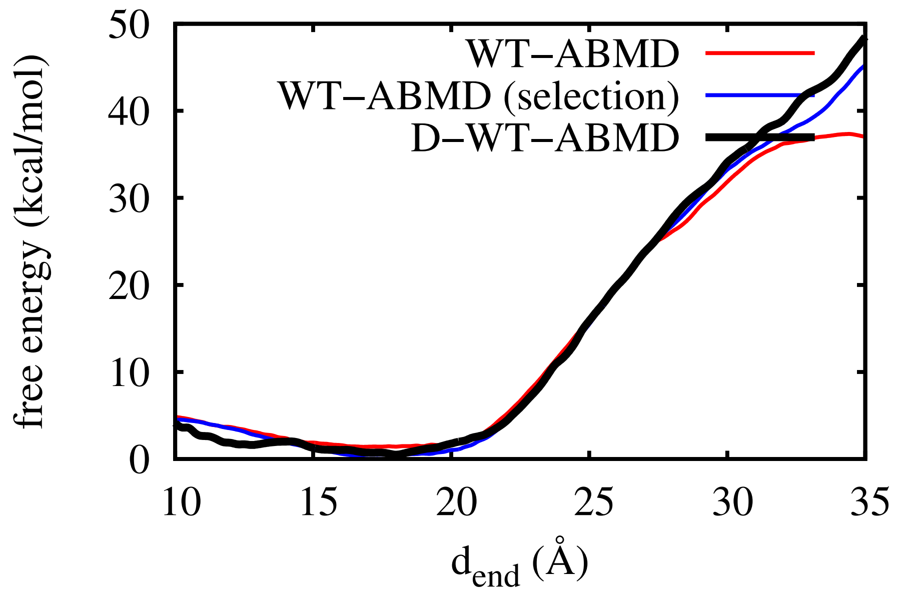


FIG. 10. **Example C:** (`fig-fe.pdf`) Free energy of deca-alanine *in vacuo* along d_{end} as obtained from well-tempered ABMD simulations without (red) and with (blue) selection algorithm (Steps 3 and 4), and with the driven well-tempered ABMD (black; Step 5).

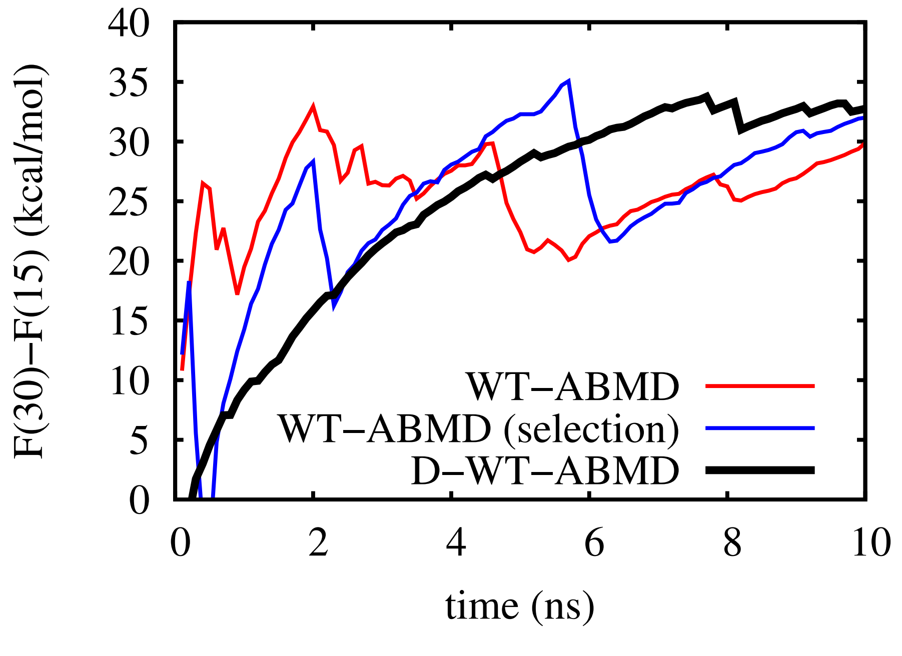


FIG. 11. **Example C:** (`fig-df.pdf`) Free energy difference of states associated with $d_{end} = 30\,\text{Å}$ and $d_{end} = 15\,\text{Å}$ as estimated along ABMD simulations of deca-alanine *in vacuo* with different flavors (See Fig. 10).

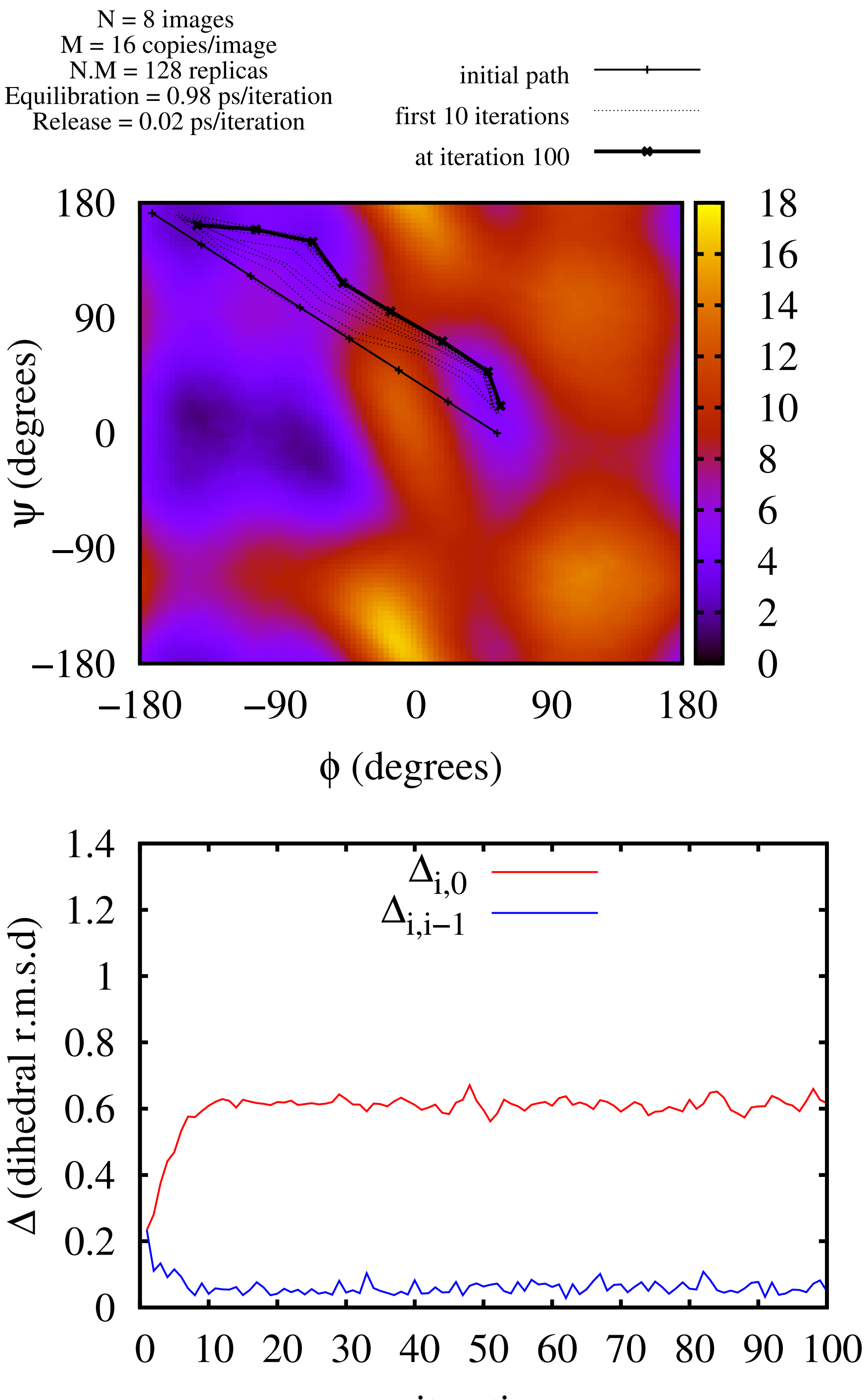
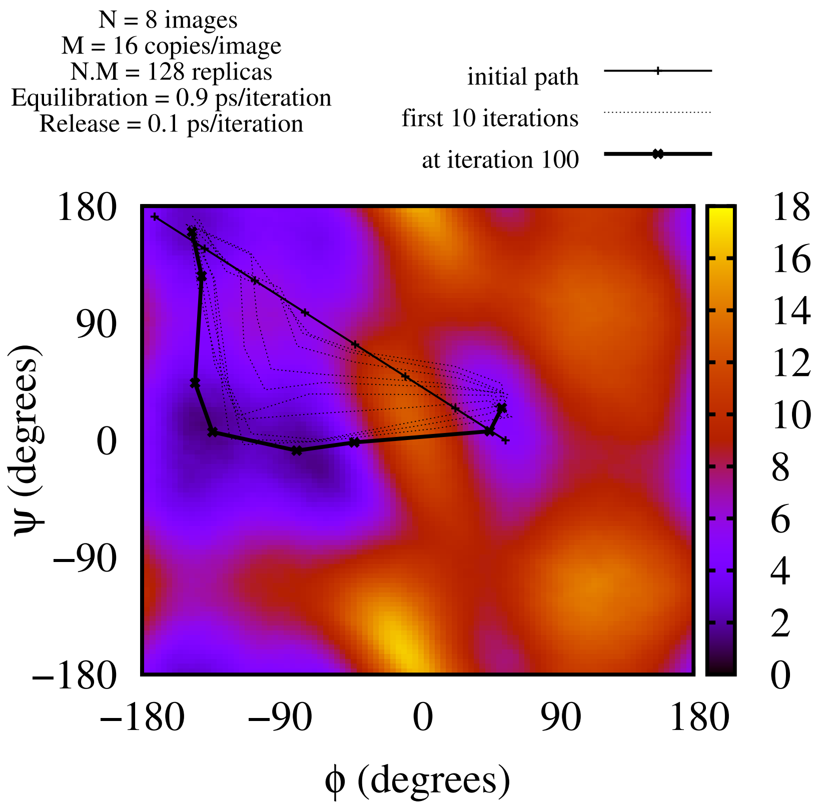


FIG. 12. **Example D:** (`string.pdf`) (top) The optimized transition pathway between the minima associated with β and α_L regions of the Ramachandran map of di-alanine peptide in implicit solvent as obtained from string method with short drifting times (Step 2). Free energy map is generated using regular ABMD (Step 3 of Example B-di-alanine; Fig.8a). (bottom) Time series of dihedral r.m.s.d of each string with respect to both initial pathway (red) and string at previous integration (blue). We define dihedral r.m.s.d of a string $\{(\phi_i, \psi_i)|i = 1, \dots, N\}$ from another string $\{(\phi'_i, \psi'_i)|i = 1, \dots, N\}$ by $\sqrt{\frac{1}{N}\sum_{i=1}^{N}(||\phi_i - \phi'_i||^2 + ||\psi_i - \psi'_i||^2)}$ in which $||a - b||$ is the angular difference between a and b (considering periodicity).



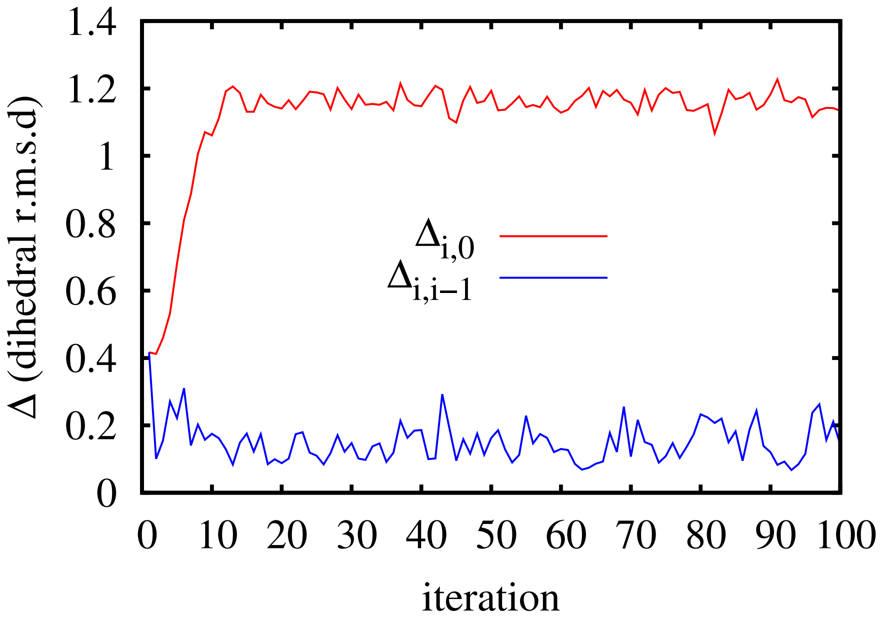


FIG. 13. **Example D:** (`string-long.pdf`) Similar to Fig. 12, resulting from Step 3 (long drifting).