A Suite of Tutorials for the WESTPA 2.0 Rare-Events Sampling Software [Article v2.0]

Anthony T. Bogetti[1,†], Jeremy M. G. Leung[1,†], John D. Russo[2,†], She Zhang[3,†], Jeff P. Thompson[3,†], Ali S. Saglam[4,†], Dhiman Ray[5,†], Barmak Mostofian[2], AJ Pratt[1], Rhea C. Abraham[1], Page O. Harrison[1], Max Dudek[1], Paul A. Torrillo[1], Alex J. DeGrave[1], Upendra Adhikari[2], James R. Faeder[4], Ioan Andricioaei[5], Joshua L. Adelman[4], Matthew C. Zwier[6], David N. LeBard[3], Daniel M. Zuckerman[2], Lillian T. Chong[1*]

[1]Department of Chemistry, University of Pittsburgh, Pittsburgh, PA; [2]Department of Biomedical Engineering, Oregon Health and Science University, Portland, OR; [3]OpenEye Scientific, Santa Fe, NM; [4]Department of Biological Sciences, University of Pittsburgh, Pittsburgh, PA; [5]Department of Chemistry, University of California Irvine, Irvine, CA; [6]Department of Chemistry, Drake University, Des Moines, IA

This LiveCoMS document is maintained online on GitHub at https://github.com/westpa/tutorials; to provide feedback, suggestions, or help improve it, please visit the GitHub repository and participate via the issue tracker.

This version dated March 30, 2023

Abstract The weighted ensemble (WE) strategy has been demonstrated to be highly efficient in generating pathways and rate constants for rare events such as protein folding and protein binding using atomistic molecular dynamics simulations. Here we present two sets of tutorials instructing users in the best practices for preparing, carrying out, and analyzing WE simulations for various applications using the WESTPA software. The first set of more basic tutorials describes a range of simulation types, from a molecular association process in explicit solvent to more complex processes such as host-guest association, peptide conformational sampling, and protein folding. The second set ecompasses six advanced tutorials instructing users in the best practices of using key new features and plugins/extensions of the WESTPA 2.0 software package, which consists of major upgrades for larger systems and/or slower processes. The advanced tutorials demonstrate the use of the following key features: (i) a generalized resampler module for the creation of "binless" schemes, (ii) a minimal adaptive binning scheme for more efficient surmounting of free energy barriers, (iii) streamlined handling of large simulation datasets using an HDF5 framework, (iv) two different schemes for more efficient rate-constant estimation, (v) a Python API for simplified analysis of WE simulations, and (vi) plugins/extensions for Markovian Weighted Ensemble Milestoning and WE rule-based modeling for systems biology models. Applications of the advanced tutorials include atomistic and non-spatial models, and consist of complex processes such as protein folding and the membrane permeability of a drug-like molecule. Users are expected to already have significant experience with running conventional molecular dynamics or systems biology simulations.

***For correspondence:**
ltchong@pitt.edu (LTC)

[†]These authors contributed equally to this work

Received: 18 May 2022
Accepted: 14 January 2023

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

Contents

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

1 Introduction and Scope of Tutorials

The WESTPA (Weighted Ensemble Simulation Toolkit with Parallelization and Analysis) software package [1, 2] is a highly scalable implementation of the weighted ensemble (WE) path sampling strategy [3, 4] that has helped transform what is feasible for molecular simulations in the generation of pathways for long-timescale processes (> µs) with rigorous kinetics. Among these simulations are atomically detailed simulations of protein folding [5], protein-protein binding [6], protein-ligand unbinding [7], and the large-scale opening of the SARS-CoV-2 spike protein [8]. The latter involved the slowest process (seconds-timescale) yet studied for a massive system (one million atoms) using WE simulations. As a "bleeding edge" application, these efforts have motivated major upgrades to WESTPA (version 2.0) that enable the sampling of processes at even longer timescales and more streamlined handling of large datasets [2]. Like its predecessor, WESTPA 2.0 is a Python package that is (i) interoperable, enabling the use of any type of stochastic dynamics simulation (e.g., MD or Monte Carlo simulations) and any model resolution (e.g., atomistic, coarse-grained, non-spatial or spatially resolved systems biology models) [9, 10]; and (ii) extensible, making it straightforward to modify existing modules or create plug-ins in order to support new scientific efforts.

Here we present a suite of tutorials organized into two groups. The first four tutorials, presented in the original version of this paper, range in order of difficulty from basic to intermediate, including a tutorial involving the suite of analysis tools.

The second group of tutorials addresses new features in the major upgrades appearing in the WESTPA 2.0 software package. Among these tutorials is one involving the Markovian Weighted Ensemble Milestoning (M-WEM) approach [11], which interfaces the WE strategy with another path sampling method called milestoning [12, 13]. In the final tutorial, we broaden the scope of path sampling to a systems biology application involving a WESTPA plugin for enhancing the efficiency of Monte Carlo simulations using the BioNetGen systems biology package [14, 15].

For general prerequisites to attempting these tutorials, please see **Section** 1.3 below. All files for the tutorials can be found online in the WESTPA Tutorials GitHub repository https://github.com/westpa/tutorials. In each tutorial, we outline learning objectives and expected outcomes.

1.1 Learning objectives

After completing the **Basic Tutorial** 7.1 involving the simulation of Na^+/Cl^- association, the user should be able to:

1. Understand the main simulation directory layout
2. Choose a progress coordinate
3. Choose an appropriate binning scheme
4. Prepare input files
5. Monitor a simulation

After completing the **Intermediate Tutorial** 7.2 involving the conformation sampling of a p53 peptide fragment, the user should be able to:

1. Set up a two-dimensional progress coordinate
2. Monitor this coordinate as the simulation progresses
3. Evaluate whether the binning scheme is effective
4. Combine and create bins "on-the-fly"
5. Store and access auxiliary data

After completing **Intermediate Tutorial** 7.3 involving the folding/unfolding of the chignolin mini-protein the user should be able to:

1. Use brute force simulations to identify appropriate initial and/or target states
2. Obtain the probability flux into the target state of a WESTPA simulation, convert it to a mean rate constant, and interpret the results
3. Approach larger, more biologically relevant events (like protein folding) with a WE-oriented mindset

After completing the **Analysis Tutorials** 7.4, the user should be able to:

1. Calculate progress coordinates using an external analysis suite (MDTraj or MDAnalysis)
2. Automate analysis and interactively explore WE simulation data using the `w_ipa` tool

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

3. Create a movie of how a probability distribution evolves with time

After completing **Advanced Tutorial** 7.5, which involves the simulation of Na^+/Cl^- association, the user should be able to:

1. Create a customized binless resampler scheme for splitting and merging trajectories based on by k-means clustering using the `BinlessMapper` resampler module;
2. Initiate a WE simulation from multiple starting conformations;
3. Combine multiple WE simulations for analysis using the `w_multi_west` multitool;
4. Perform post-simulation analysis using the `w_crawl` tool.

After completing **Advanced Tutorial** 7.6 involving the simulation of drug membrane permeation, the user should be able to:

1. Set up a double membrane bilayer system for permeability studies;
2. Use the highly scalable HDF5 framework for more efficient restarting, storage, and analysis of simulations;
3. Apply the minimal adaptive binning (MAB) scheme.

After completing **Advanced Tutorial** 7.7 involving the simulation of ms-timescale protein folding, the user should be able to:

1. Apply the haMSM plugin for periodic reweighting of simulations;
2. Use the `msm_we` package to build an haMSM from WE data;
3. Estimate the distribution of first passage times.

After completing **Advanced Tutorial** 7.8 involving the creation of custom analysis routines and calculation of rate constants, the user should be able to:

1. Access simulation data in a `west.h5` file using the high-level `Run` interface of the `westpa.analysis` Python API and retrieve trajectory data using the `BasicMDTrajectory` and `HDF5MDTrajectory` readers;
2. Access steady-state populations and fluxes from the `assign.h5` and `direct.h5` data files, convert fluxes to rate constants, and plot the rate constants using an appropriate averaging scheme;
3. Apply the RED analysis scheme to estimate rate constants from shorter trajectories;

After completing **Advanced Tutorial** 7.9 involving simulations of alanine dipeptide using the M-WEM method, the user should be able to:

1. Install the M-WEM software and perform a M-WEM simulation;
2. Create milestones to define the M-WEM progress coordinate;
3. Analyze an M-WEM simulation to compute the mean first passage time, committor, and free energy landscape.

After completing **Advanced Tutorial** 7.10 involving rule-based modeling of a gene switch motif using the WESTPA/BNG plugin, the user should be able to:

1. Install the WESTPA/BNG plugin and set up a WESTPA/BNG simulation;
2. Apply adaptive Voronoi binning, which can be used for both non-spatial and molecular systems;
3. Run basic analyses tailored for high-dimensional WESTPA simulations.

The tutorials will use an array of different dynamics packages to showcase WESTPA's interoperability. In each tutorial, all of the required software, including the dynamics engine and analysis tools, are freely available with easily-accessible online documentation. Please note the version of each software package listed in the **Computational Requirements** section of each tutorial.

1.2 The Weighted Ensemble Strategy

WE is a highly-parallel path sampling strategy for generating rare events, for studying non-equilibrium steady states, and less commonly, for studying equilibrium properties. At heart, it is a simple and flexible strategy which is agnostic to system type and which therefore lends itself to numerous applications and optimizations. The properties of WE, including strengths and limitations, have been reviewed in detail before [4], although improvements continue to be developed [17–21]. Here, we briefly review key aspects of WE.

The Basic WE Procedure. See **Figure 1**. WE orchestrates multiple trajectories—each assigned a weight—run in parallel by stopping them at regular time intervals of length τ (typically a large multiple of the underlying simulation time step), examining the trajectories, and restarting a new set of trajectories. The new trajectories are always continuations of the existing set, but some trajectories may not be continued (they are "pruned") and others may be replicated. Discontinued trajectories result from probabilistic "merge" events where a continued trajectory absorbs the weight of one that is pruned. Replicated trajectories are said to be "split" with the original weight shared equally among the copies. Usually bins in configuration space are used to guide split and merge events based on a target number of trajectories for each bin, but any protocol—including binless strategies highlighted below—may be used for this purpose. Regardless of

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

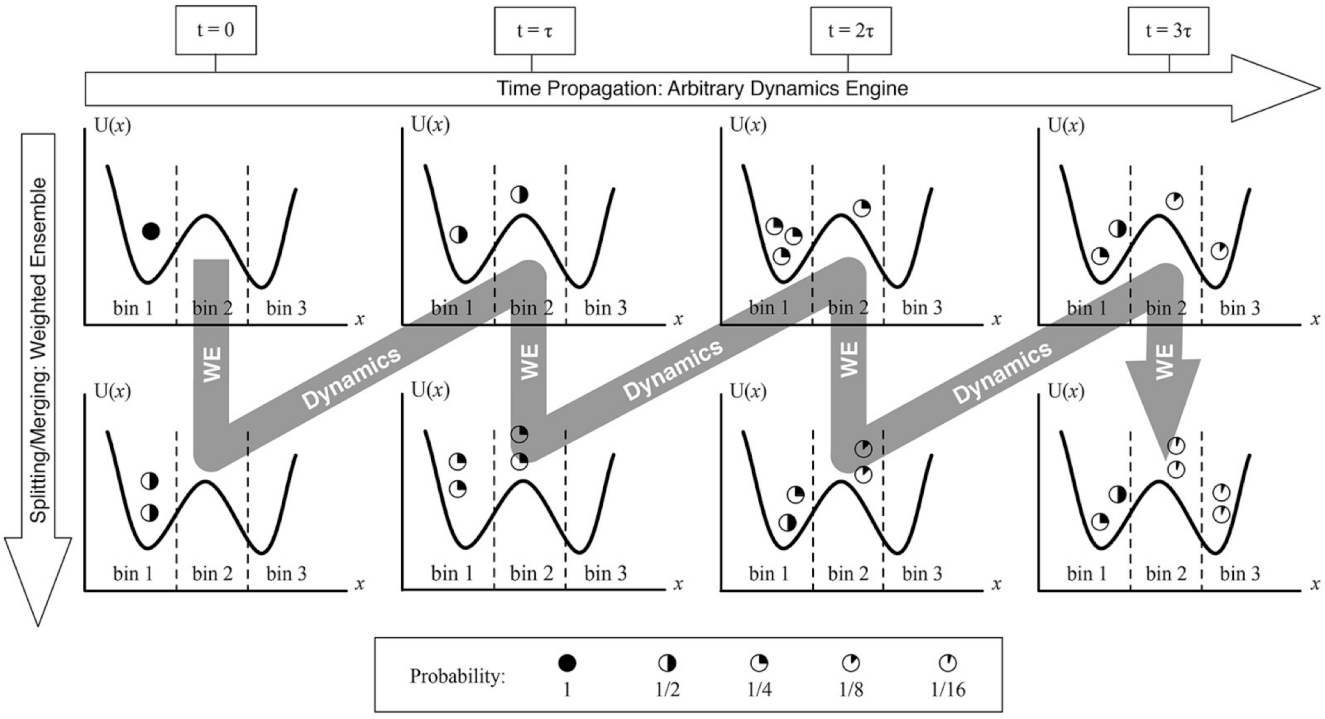

Figure 1. Overview of the weighted ensemble (WE) strategy [10]. WE typically employs bins, demarcated here by dashed vertical lines, to guide a set of trajectories to sample throughout configuration space. Using only ordinary dynamics—without biasing forces—WE replicates ("splits") trajectories in unoccupied or under-occupied regions of space and prunes ("merges") trajectories in over-occupied regions, according to the user-specified allocation scheme which here is a target of two trajectories per bin. Throughout the process, weights (partially filled circles) are tracked by statistical rules of inheritance that ensure that the overall ensemble dynamics are consistent with non-equilibrium statistical mechanics [16]. Figure adapted with permission from [9].

the resampling protocol, a "recycling" protocol often is used whereby events reaching a user-specified target state are re-initiated according to a specified distribution of start states [22]. This recycling protocol focuses all sampling on a single direction of a process of interest and has valuable properties as noted below.

WE is Resampling, and Hence Unbiased. The simple steps defining WE simulations stem from its basis as a statistical "resampling" procedure [16]. The split/merge steps generate a statistically equivalent (re)sample of an initial trajectory set by increasing/reducing trajectory density in some regions of configuration space at a given time, using weight adjustments to maintain the underlying trajectory distribution. The trajectory set is therefore unbiased at all times, i.e., average time-dependent observables derived from many WE runs will match the average of a large number of conventional simulations without splitting or merging events [16]. Furthermore, the distributions of transition path times ("barrier crossing times") from WE runs match those from converged conventional simulations, and can be generated in orders of magnitude less computing time [23, 24]. The lack of bias in the dynamics of WE runs holds true regardless of whether recycling is employed.

Observables and Ensembles Sampled by WE. WE can yield transient and/or steady-state observables. When recycling is not used, WE provides pathways, i.e., sequences of conformations in a transition and the frequencies of those sequences, in addition to time-dependent observables as the system relaxes to equilibrium, e.g., the probability of a given event at a given time after initiation in the chosen starting state. Complex systems are unlikely to relax fully to equilibrium during a WE simulation. With a recycling protocol, the system will not relax to equilibrium but instead to a non-equilibrium steady state (NESS) that has steady probability flow from initial to target state. If reached, the NESS provides a simple mechanism for computing rate constants via the Hill relation [22]. However, although relaxation to a NESS can be considerably faster than relaxation to equilibrium [25, 26], the process may be too slow for WE to reach NESS on practical timescales, motivating the haMSM approach [5, 18] described below.

Resampling Introduces Correlations, which Increase Variance. WE has intrinsic limitations, like any method [27], and it is essential to understand them. Most fundamentally, splitting and merging introduce correlations into the sampled trajectory ensemble that could decrease its

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

information content. These stem primarily from splitting events: multiple trajectories share an identical history up to the time of the split event and hence do not contribute fully independent information to any observable. These correlations, in turn, can lead to large run-to-run variance [5] because the trajectory ensemble in each WE run results from a relatively small number of "parent" trajectories which have been split repeatedly. This variance is addressed to some extent by the iterative haMSM protocol described below, and more directly by ongoing mathematical optimizations noted below. Importantly, correlations within WE ensembles lead to significant challenges in quantifying uncertainty [4, 28].

Ongiong Efforts at Optimization and Variance Reduction. Because WE is unbiased so long as a correct resampling protocol is used [16], there is an opportunity to reduce the run-to-run variance noted above by improved resampling procedures. In the context of binned WE simulations, both the construction of bins and the number of trajectories per bin can be optimized based on a recently developed mathematical formulation [21, 29] or based on heuristics [19]. Bins do not need to be kept static over time [16, 19]. Optimization approaches are actively being studied and incorporated into WESTPA as appropriate.

WE Cannot Solve Every Problem. Despite its great strengths and highly notable achievements [5, 7, 8], users should not assume WE can tackle any problem. Independent of the correlation/variance issues noted above, certain systems will remain too complex for WE given current hardware and algorithms. In every system, there is a minimum transition path time t_{TP} (also called t_b) [30, 31] for physically realistic events which sets an absolute requirement on sampling required: in a WE run, a set of trajectories exceeding the minimum t_{TP} must be generated, which may be a prohibitive cost. Additionally, even if the necessary computing resources are available, current binning and resampling strategies might not be sufficient to generate events of interest. And finally, even if events of interest are generated, the sampled trajectories may be insufficient for producing observables of interest such as a reliable estimate of the rate constant.

1.3 Prerequisites and Computing Requirements

Background Knowledge and Experience. The WESTPA software is not intended for total beginners in molecular simulation. Users should already have extensive experience running conventional simulations using the underlying dynamics engine of interest (Amber [32], OpenMM [33], BioNetGen [14], etc.). Prior to applying the WE strategy to their own systems, we suggest that users run multiple conventional simulations to (i) ensure that the preparation of the system and propagation of dynamics is according to best practices (e.g., see [34]), (ii) identify potential progress coordinates and initially define the target state, and (iii) estimate the ns/day on a single CPU/GPU for your system and storage needs for the full-scale WE simulation. We highly recommend that new WESTPA users read this review article [4] and this introduction to non-equilibrium physics of trajectories [35]. It is also important to identify sources of validation for your simulation (e.g., from experiment and/or standard simulations) and to be familiar with the estimation of statistical uncertainty in the computed observables, including those used for validation [36].

Software Requirements. The WESTPA 2.0 software is a standard Python package that can be used on any Unix operating system. The software requires Python versions $\geq$3.8 and a number of standard Python scientific computing packages. We recommend installing WESTPA either as a PyPI or conda package using miniconda. Both packages provide all required software dependencies and can be installed using one-line commands: (1) `python -m pip install westpa` or (2) `conda install -c conda-forge westpa`. Note that it is a best practice to install WESTPA into an isolated virtual or conda environment, along with the dependencies specific to your project. Due to the use of the MDTraj Python library with the WESTPA 2.0 HDF5 framework, certain modifications to the installation procedure are required for running WESTPA 2.0 on ppc64le architectures (e.g., TACC Longhorn or ORNL Summit supercomputers; see https://github.com/westpa/westpa/wiki/Alternate-Installation-Instructions).

WESTPA 2.0 is designed to be interoperable with any dynamics engine, requiring an external dynamics engine to propagate the dynamics in a WE simulation. Please see the prerequisite sections of each tutorial for additional software requirements that are specific to that tutorial.

Hardware Requirements. Like its predecessor, WESTPA 2.0 is highly-scalable on CPUs/GPUs, making optimal use of high-performance computing (HPC) clusters available at academic institutions or supercomputing centers. Memory requirements are dependent on the underlying dynamics engine, e.g., ~1 GB per CPU core (or GPU) for atomistic MD simulations. Users should refer to the best practices of their dynamics engine of choice to determine the optimal allocation of resources for each CPU/GPU. The most efficient way to run WESTPA is to use a computing resource that provides the user with a number of CPUs/GPUs—all the same processor speed—that either matches the number of trajectories per WE iteration or a number by which the number of trajectories at any point in time is evenly divisible. WE can nevertheless run on heterogeneous hardware (different processor or memory bus speeds) or with trajectory counts that do not

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

Table 1. WE parameters used for notable applications in the literature. The asterisk (*) indicates an application with I/O operations that is too frequent for supercomputers and gaming GPUs.

Rare-event process	System and size	WE Parameters	Suitable computing resources
millisecond protein folding [28]	NTL9 protein in generalized Born (GB) implicit solvent with low and high solvent viscosity (collision frequency of 5 ps^{-1} and 80 ps^{-1}, respectively): 627 atoms	1D progress coordinate: C_α RMSD from the folded structure. Binning: 53 bins, that are finely spaced for near-folded structures (35 bins for 1.0 Å < RMSD < 4.4 Å) and more coarsely spaced for more unfolded structures: (12 bins for 4.4 Å < RMSD < 6.6 Å and 5 bins for 6.6 Å < RMSD < 10.2 Å). τ = 10 ps; 1200 WE iterations; 4 trajectories/bin	Professional-graphics-programming GPUs* (e.g. NVIDIA Quadro RTX 5000)
peptide-protein association [37]	p53 peptide/MDM2 protein in GB/SA implicit solvent: 1685 atoms	2D progress coordinate: heavy-atom RMSD of p53 peptide relative to its MDM2-bound conformation following alignment on (i) MDM2 and (ii) itself. Binning: 16 bins with 0.5 Å widths along the p53-aligned RMSD and widths ranging from 0.2 to 2 Å for the MDM2-aligned RMSD. τ = 50 ps; 396 WE iterations; 8 trajectories/bin	1600 CPU cores on a supercomputer (e.g. PSC's Bridges-2) or 16 GPUs (e.g. NVIDIA A100 GPUs)
protein-protein association [6]	barnase/barstar proteins in explicit solvent: >100,000 atoms	2D progress coordinate: (i) heavy-atom RMSD of barstar residues D35 and D39 after alignment on barnase, and (ii) minimum protein-protein separation distance. D35 and D39 are the barstar residues that become the most buried upon binding barnase. τ = 20 ps; 650 WE iterations Binning: 72 bins with coarsely spaced bins every 1 Å from 10 to 60 Å and more finely spaced bins every 0.5 Å from 0 to 10 Å along the RMSD coordinate; two bins along the distance coordinate separated by a bin boundary at 5 Å; fixed total number of trajectories (1600)	1600 CPU cores on a supercomputer (e.g. PSC's Bridges-2) or 16 GPUs (e.g. NVIDIA A100 GPUs)

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

Table 2. $WEST_SIM_ROOT organization and file explanations

`bstates/`	directory containing basis states
`env.sh`	set environment variables
`init.sh`	initialize the WESTPA simulation
`common_files/`	directory containing files for dynamics (i.e. topologies)
`run.sh`	run the WESTPA simulation
`tstate.file`	define the target state (for steady state simulations only)
`west.cfg`	specify main WE simulation parameters
`westpa_scripts/`	directory containing essential scripts
`system.py`	a separate script to define functions or parameters (optional)
`reference/`	directory containing reference files for calculations (optional)

divide evenly onto CPUs/GPUs, but this scenario decreases efficiency as some processors are inevitably idle for at least a portion of the overall runtime.

Users can estimate the approximate storage space required for their project by taking the product of the following: (i) amount of disk space required for storing data from one trajectory segment of length τ, (ii) the maximum number of trajectories per WE iteration, and (iii) the total number of WE iterations required to generate a reasonable maximum trajectory length. To optimize the use of storage space, we recommend that users tar up trajectory files into a single file for each WE iteration and remove coordinates of the system that are not of primary interest (e.g., solvent coordinates for certain processes). We note that the WESTPA 2.0 HDF5 framework dramatically reduces the storage space required for trajectory coordinates by consolidating the data from millions of small trajectory files into a relatively small number of larger HDF5 files, reducing the large overhead from the file system that results from the storage of numerous small trajectory files. By doing so, the HDF5 framework also alleviates potential I/O bottlenecks when a large amount of simulation files are written after each WE iteration.

2 Workflow of Running a WE Simulation

An overview of the workflow for running a WE simulation using WESTPA is detailed below. This workflow is only meant to give a sense of the mechanics and flow of using WESTPA once your system and WE parameters have already been carefully chosen. See **Table 2** for a summary of all files mentioned in this workflow.

Overall Flow

Ready: The purpose of this step is to ensure that the chosen WE parameters are correctly specified in the proper places and that all environment variables are correctly set. Most of the WE parameters (such as the number of WE iterations, binning scheme etc.) and auxiliary datasets (auxdata; see **Section 7.2**) are specified in the `west.cfg` file. You can view an example of this file in any of the tutorials below; labels exist directing where to specify each parameter. More complex binning schemes (such as recursive schemes or schemes involving functional bin mappers) can be specified in an external file called `system.py`. A user may also choose to write functions to this file. Usually, these functions will calculate progress coordinate or auxiliary data and are more complex than usual.

The environment is set up in the `env.sh` file. The location of the main WESTPA simulation directory (`$WEST_SIM_ROOT`) and the location of dynamics/analysis programs are placed in your system path. When setting up WESTPA on a cluster, program modules will be loaded in the `runwe.slurm` file instead of the `env.sh` file (see **Section 7.1.3** and view the cluster-specific `runwe.slurm` file). It is a best practice to define variables in `env.sh` for each program that will be called. These variables should contain the full path to that program (such as `CPPTRAJ=$(which cpptraj)`, see **Section 5.1** for more information). Always source `env.sh` before trying to run WESTPA just to see if any errors appear relating to programs not being found. If errors are present, edit `env.sh` to specify the proper locations of programs and try to source it again. The goal of this action is to make sure that any issues with your environment are fixed before continuing so that troubleshooting becomes much easier later on.

Set: After setting up the system environment and specifying the WE parameters, users will need to initialize the simulation. This involves running the `init.sh` script, which will take an initial structure (or structures), calculate a progress coordinate (pcoord for short, this is also the name used in WESTPA datasets pertaining to the progress coordinate) value for that structure and then place that structure in the appropriate bin. The `init.sh` file is also the location where users can specify whether the simulation will be run under equilibrium or steady-state conditions.

Place the starting structure(s) in the `bstates/` directory. The structure should be a coordinate file giving the starting configuration of your system (e.g. Amber restart file). The `bstate.file` tells WESTPA which structure to use as the initial structure for the simulation. If you have only one structure, this file will contain the name of that structure only; if you have more than one structure, `bstate.file` should list each structure along with its associated statistical weight. An ex-

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

ample of the latter is a representative ensemble of unbound protein conformations in a binding process that could be generated using a prior equilibrium WE simulation [6, 37].

Next, specify whether the simulation will be run under equilibrium or steady-state conditions. This specification is made in the `init.sh` file. Including a `$TSTATE_ARGS` argument for `w_init` will signal for WESTPA to run under steady state conditions. The `tstate.txt` file in the main simulation directory is where the progress coordinate value of the target state is specified. If the `$TSTATE_ARGS` argument is absent, the simulation will be run under equilibrium conditions. See the tutorials in **Sections 7.1 and 7.2** below for examples of how `init.sh` will change from running a steady-state simulation versus an equilibrium simulation (respectively).

Running `init.sh` will cause WESTPA to execute `get_pcoord.sh`, which is a script located in `westpa_scripts/`. This script will give an initial progress-coordinate value for the basis state(s) (located in `bstates/`) to WESTPA.

Users will need to modify `get_pcoord.sh` to either read or calculate the progress coordinate for their particular simulation. For instance, in the **Basic Tutorial 7.1**, the distance between the Na^+ and Cl^- ions is used as the progress coordinate. The `get_pcoord.sh` file for that tutorial simply prints the contents of an already-existing file (`pcoord.init`, which already contains the calculated value) and passes that value to WESTPA. However, `get_pcoord.sh` can also perform the calculation for the basis state, as in the **Intermediate Tutorial 7.2**. However this is done, a value (or values) for that progress coordinate should be echoed into `$WEST_PCOORD_RETURN`, a WESTPA variable containing all of the progress coordinate values for the entire simulation (see **Section 7.2** for the added considerations if a two-dimensional progress coordinate is used).

If errors appear while trying to initialize the simulation, the following troubleshooting methods are recommended. First, make sure that the command entered in `get_pcoord.sh` properly calculates the progress coordinate. Copy the initial structure from the `bstates/` directory to another directory and run the command. If the command does not work, make sure the proper atoms and residues are selected and then try running the command again. If the command works, make sure that the calculated value is being successfully echoed into `$WEST_PCOORD_RETURN`.

To make troubleshooting easier, turn on logging for the `get_pcoord` step in the `west.cfg` file. By setting the location of the standard output (stdout) and/or standard error (stderr) to `$WEST_SIM_ROOT/get_pcoord.log`, you can more closely monitor the output of the `get_pcoord.sh` script to try to find out where things are not working.

Go: Running the `run.sh` script will start a WESTPA simulation. If `init.sh` was just run, a new simulation will begin and continue until the number of WE iterations specified in `west.cfg` have been completed. If the simulation was stopped after previously running, `run.sh` will continue the simulation from the point at which it was stopped. If WESTPA is being run on a cluster, then this script will take the form of a Slurm or other submission script (such as `runwe.slurm`, see the **Basic Tutorial 7.1** for an example). WESTPA will propagate dynamics for one trajectory segment (of length τ) and calculate progress coordinate values (and all auxiliary data) for the propagated structure(s). After completing a trajectory segment, WESTPA will combine and replicate trajectories to maintain the target number of trajectories per bin (as specified in the `west.cfg` file). One cycle of dynamics and combination/replication is referred to as a single WE iteration. The number of iterations is repeated until the observable of interest (e.g. rate constant) is reasonably converged.

Running `run.sh` will cause WESTPA to execute `runseg.sh`, which is a script similar to `get_pcoord.sh`, located in `westpa_scripts/`. Users will need to modify `runseg.sh` to call the dynamics engine and calculate the appropriate progress coordinate (and auxiliary data) value(s). Refer to the `runseg.sh` file in the **Basic Tutorial 7.1** as an example. This particular simulation uses Amber's `pmemd` program for dynamics propagation. Running this program requires a certain input/output syntax that is specific to the dynamics engine (such as Gromacs or OpenMM). The section of this file that calculates the progress coordinate will be identical to that in the `get_pcoord.sh` file. If a user is collecting auxiliary data (as specified in the `west.cfg` file), those values will need to be calculated after calculating the progress coordinate value (see **Intermediate Tutorial 7.2**).

Since `runseg.sh` will cause many different files to be generated, it is important to consider how WESTPA is handling these files, especially when using a shared file space such as on a cluster. The methods used in the example `runseg.sh` files that have been provided in the tutorials below are sufficient in most cases, but please refer to **Section 5.1** for a discussion on file management and network traffic.

If there are any errors in the WESTPA setup (e.g. incorrect number of elements in the pcoord array, misplaced input files), the simulation will not proceed past the first WE iteration. If this is the case, check the `west.log` file to see if there is a good reason for why the simulation is failing. Usually, however, detailed logging of any errors is available in the `seg_logs/` directory for each segment of each iteration. View the segment log for a particular segment to see if the dynamics are completing successfully and that the progress coordinate (and auxdata) values are being calculated and passed to the appropriate variables (such as `$WEST_PCOORD_RETURN`).

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

If the dynamics fail to start, copy all necessary input files into an empty directory and run the dynamics manually. If no errors appear, make sure that your progress coordinate consists of the proper number of datapoints (as specified in the `west.cfg` file). Also remember that you must include the parent coordinate file as your first data point when storing the progress coordinate data. This will ensure that the analysis tools work properly.

This is determined by the frequency at which the progress coordinate is being calculated. For example, if WESTPA expects 50 progress coordinate values per τ and only receives 10 values, the simulation will fail after the first WE iteration. Check the dynamics input file (`md.in` in the **Basic and Intermediate Tutorials** 7.1-7.2) to make sure that the coordinates of your system are being saved at a frequency that matches the number of specified progress coordinate values.

If the simulation proceeds to the second iteration, there should not be any errors in the WESTPA setup. To monitor the progress of the WE simulation, use `w_pdist` to generate probability distributions as a function of your progress coordinate and WE iteration. WESTPA's `plothist` command will allow you to visualize these probabity distributions with a few different visualization options (see **Tutorials** 7.1-7.2).

Analyze: All data generated from the simulation is contained in one place: the `west.h5 file`. From this data, users can track the evolution of progress coordinate values, calculate fluxes into certain bins or states (see the `w_ipa` analysis tutorial in **Section** 7.4.2) and view other statistics pertaining to the simulation. To visualize a completed trajectory, refer to **Basic Tutorial** 7.1 and the **Analysis Tutorial** 7.4 involving the visualization of trajectories (**Section** 7.4.3).

To assess the convergence of the simulation, a user might want to monitor the evolution of the flux into a target state as a function of the number of WE iterations by using the hdfview program to plot the `target_flux_evolution` dataset in the `direct.h5` file generated by `w_ipa` (see **Tutorial** 7.1).

3 Additional Simulation Workflow with WESTPA 2.0 Upgrades

Given the major upgrades in the WESTPA 2.0 software package [2], we recommend the three-stage simulation workflow illustrated in **Figure** 2. These workflow details are meant to supplement the simulation workflow outlined above. Details of the particular schemes mentioned are provided in the relevant **Advanced tutorials** 7.5-7.10.

In the first ("Ready") stage, if one uses a binned resampling scheme, we recommend using one of the two adaptive binning schemes available in WESTPA 2.0: the minimal adaptive binning (MAB) scheme or adaptive Voronoi binning scheme. These adaptive binning schemes enable quicker explorations of the chosen progress coordinate than manual, fixed binning schemes. The MAB scheme is effective at surmounting barriers in a direction of interest [19] while the adaptive Voronoi binning scheme [16] is ideal for enhanced sampling in high-dimensional space (more than three dimensions) when all parts of the progress-coordinate space are potentially important. However, if progress-coordinate space includes, for example, undesirable unfolded protein conformations, adaptive Voronoi binning might allocate bins and computing resources to those regions. Besides the adaptive binning schemes, one can opt for a "binless" resampling scheme by defining a grouping function as described in **Advanced Tutorial** 7.5 below. The choice of τ (the resampling interval used for your WE simulation) should also be made on a system-by-system basis, with a sufficiently long time interval to capture relevant motions of interest but not so long that no net progress is made toward the target state. Examples of τ values used for various systems in previous WE studies are provided in the first suite of WESTPA tutorials [38], but some trial-and-error will likely be necessary. Convergence of a WE simulation will ultimately depend on the overall goal of running the simulation, but will most likely involve the time-evolution of an observable of interest leveling off over time (e.g., trajectory flux into a user-defined target state). If the convergence criterion is not met, a WE simulation using WESTPA can be resumed by simply running the `run.sh` script or resubmitting the job if it was originally run with slurm. Even if changes were made to the progress coordinate or binning, WESTPA will incorporate those changes and resume the simulation accordingly.

In the second ("Set") stage, we recommend starting the WE simulation from multiple, pre-equilibrated starting conformations that are representative of the initial stable state (at least one "basis state" for each trajectory walker) to improve the sampling of the initial state and diversity of generated pathways to the target state. Initial structures for a WE simulation should be chosen on a system-by-system basis, but in general, more starting structures (each with a slightly different initial configuration) should provide you with a more diverse trajectory ensemble. When the initial state of interest is well-defined, only a small number of structures may be necessary, but a truly heterogeneous initial state such as the unbound or unfolded states will require more structures to be representative of the intrinsic diversity. As discussed in **Section** 7.5, these "basis" structures will govern the recycling process (if used), so care should be exercised in choosing them.

In the third ("Go/Analyze") stage, we recommend applying one of the following three options to further accelerate convergence to a steady state once successful pathways are generated. The Rates from Event Durations (RED) analysis

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

scheme [20] estimates rate constants more efficiently than the original WE scheme [3] by exploiting information in the transient region of the simulation. Another option is the haMSM plugin, which employs a fine-grained "microbin" analysis and can be used to not only estimate rate constants following WESTPA simulation (e.g., for the seconds-timescale coronavirus spike opening process [39]), but to also restart trajectories with their weights adjusted for a steady state [2]. Because restarts in the haMSM plugin are initiated from configurations occurring throughout previously run trajectories, the continuity of the generated pathways is broken. The third option is the weighted ensemble steady state (WESS) plugin [22], which uses the less fine-grained WE bins to estimate steady state but preserves the continuity of pathways, restarting from only the final points of trajectories. While all trajectory files of the chosen dynamics engine are saved by default, we recommend storing the trajectory coordinates using the WESTPA 2.0 HDF5 framework, which greatly facilitates the restarting, storage efficiency, and analysis of WE simulations. When possible, users should run multiple WE simulations, which provides a greater number of independent pathways and enables straightforward estimation of error using the Bayesian bootstrap method [28] (see https://github.com/ZuckermanLab/BayesianBootstrap).

Random Seeds for Simulations. For WE trajectories to diverge from one another after a splitting event, a stochastic thermostat is required for MD simulations. Furthermore, the random number seeds for such thermostats must be sufficiently random (uncorrelated) to avoid undesired bias of the dynamics when trajectories are restarted at short time intervals (e.g., in the case of WE simulations) [40]. To avoid such bias, we strongly recommend using WESTPA's system entropy-seeded random number facility instead of any time-seeded random number generator of the chosen dynamics engine. To use this facility, we first set the random seed to `RAND` in the dynamics input file (e.g., `ig=RAND` in the AMBER `md.in` file) and then specify this input file in `runseg.sh`, which will replace the `RAND` string with the WESTPA random number seed.

Extremely Low Trajectory Weights. While it is possible to set a minimum threshold weight (e.g., 10^{-100}) for trajectories to be considered for splitting, the generation of trajectories with extremely low weights (e.g., $<10^{-100}$) is a potential warning sign that the division of configurational space is not capturing all relevant free energy barriers. If a WE simulation yields such trajectories, we strongly suggest re-evaluating the choice of progress coordinate and/or restricting the binning to a carefully chosen subset of configurational space that would avoid generating such trajectories. For example of the latter, see **Advanced Tutorial 7.6**.

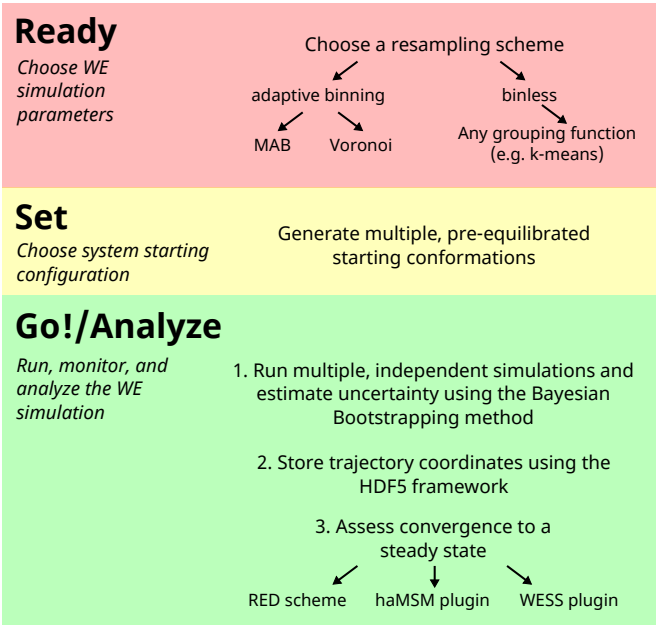

Figure 2. Recommended simulation workflow that makes use of major upgrades in the WESTPA 2.0 software.

4 General Guidelines for Choosing WE Parameters

Suitable WE parameters such as the progress coordinate, binning scheme, and resampling interval τ depend on the particular system under investigation and the particular process of interest. Note that all of these WE parameters are tightly coupled to one another. Below are general recommendations that aim to assist in choosing these parameters. See **Table 2** for examples from the literature. Currently, choosing WE parameters is something of an art, although the hope is to automate some aspects of parameter selection in the future. For now, we suggest what may be considered a semi-systematic, trial-and-error procedure:

1. Initially, choose the simplest 1D coordinate that would be expected to capture the slowest relevant motion along with initial bin spacings, τ value, and number of trajectories/bin. Choose these initial parameters following examples in the tutorials and/or literature, bearing in mind they likely will require modification.
2. The τ value should be sufficiently long such that at least one trajectory progresses to the next bin. In addition, a code scaling test (plot of the time required to complete a WE iteration vs. τ value) should be carried out for a range of potential τ values on the intended computer hardware to identify a τ value that yields reasonable linear scaling.

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

3. If your system stops advancing along your progress coordinate, consider reducing the τ value, increasing the number of trajectories/bin, and/or using a finer bin spacing in that region of the progress coordinate while combining bins from higher probability regions. Note that bin spacings are arbitrary in WESTPA and the most efficient bin sizes likely are not exactly equal. Details for combining and creating bins "on-the-fly" are provided below in the **Intermediate Tutorial** 7.2.
4. If none of the above efforts in step 3 are effective based on a one-dimensional progress coordinate, your progress coordinate may be missing orthogonal and relevant slow degrees of freedom. To address this issue, consider using a two-dimensional progress coordinate (**Section** 7.2) [[6, 37]] or a "nested" coordinate in which the progress coordinate switches to monitoring another observable once a particular value for the initial observable is reached. Note that additional dimensions in the progress coordinate greatly increase the number of bins and hence the cost of the WE run, which is the motivation for nesting an additional coordinate in only a subset of the initial bins. You might also consider binning strategies that are not based on user-defined coordinates, but instead employ Voronoi cells potentially in conjunction with a string method. The WESTPA community will continue researching the important topic of self-adjusting adaptive bins. If all of your best efforts fail to generate transitions, consider simplifying your system (e.g. coarse-graining the model) and/or applying methods that involve the introduction of external forces (e.g. umbrella sampling) to generate initial transitions that can further inform the choice of progress coordinate.

5 Cluster-Specific Considerations

To take full advantage of WESTPA's scaling and parallelizability, users may seek to run the software on HPC clusters. The tutorials included herein are written with the goal of teaching new and relatively inexperienced users the basics of using the software and therefore do not focus on optimizations pertaining to the code. We recommend that users become familiar with running WESTPA on a cluster, especially the cluster-specific issues and considerations that may arise.

5.1 Minimizing the Number of Output Files

It is advisable to minimize the number of output files generated by your simulation as this reduces the I/O overhead and will therefore be less taxing on the filesystem of the computing cluster. We recommend saving only the restart files that are necessary for continuing trajectories and analysis of the simulation. If the user needs additional information (e.g. coordinates that have been saved at a greater frequency than the τ value) contained in certain output files, those files should of course be kept. To further reduce the number of files, we suggest separately tarring up the files for each WE iteration. The resulting tarballs will also facilitate any transferring of your simulation data to another location.

In some cases such as WE simulations that are run using GPUs, trajectory segments can complete too quickly, leading to a bottleneck where the transfer of files over the network to the local storage of the node is too slow or there are too many transfers over the network. In such cases, copy over the data of the entire previous WE iteration as a tarball to the local storage of the node, run the entire iteration from this local storage, and copy back the results to the scratch space in a single tarball. While these transfers over the network will add some overhead to each WE iteration, they will avoid the network bottleneck.

5.2 Data Management

A single WE simulation may generate multiple terabytes of data, presenting a challenge for storage and retrieval of data. Moreover, using short trajectory segments in WE simulations commonly results in a large numbers of small files, which are managed more slowly on some file systems than a smaller number of large files with the same overall disk size. To alleviate these potential issues, we recommend the following:

1. Perform an initial run to monitor data storage and retrieval. Note that the initial number of trajectory segments may be a small fraction of the amount that would be generated in the eventual production run.
2. Delete unnecessary files as each trajectory segment is simulated (see example `runseg.sh` files in the **Basic and Intermediate Tutorials** 7.1-7.3). Unnecessary files may include input files, log files from analysis tools, and raw text output files from analysis tools. Often, useful data from log files (e.g., temperature from an MD simulation) may be extracted from the log files and saved as auxiliary data to the WESTPA data file (`west.h5` file), which stores data more efficiently than raw text.
3. Tar and optionally compress data from each WE iteration. This strikes a balance between excessive file count and excessive file size, either of which is typically suboptimal for long term storage, especially on tape systems that may not guarantee the integrity of large files.
4. Consider saving coordinates for only the solute atoms of your system to an H5 file.

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

5.3 Minimizing Network Traffic Across Multiple Computing Nodes

Given the large scale of a WESTPA simulation, it is advisable to limit the number and frequency of network operations (e.g. I/O operations and file transfers from the local disk to the global filesystem). We recommend the following strategies for reducing network traffic:

1. Perform a code scaling test to identify an appropriate τ value (see **Section 4** above).
2. Set environment variables to the full pathnames of repeatedly used programs (e.g. analysis tools used to calculate progress coordinates; see **Basic Tutorial 7.1**).
3. Copy repeatedly accessed files (e.g. reference structures and analysis scripts) to local scratch space and temporarily write the output files to this scratch space. After each trajectory segment of length τ completes, tar the output files, and copy the tarred files to the globally accessible filesystem using `rsync`.

5.4 Advice when Using GPUs

If your WE simulation has extremely frequent starting up of simulation segments, your simulation may overheat gaming GPUs and potentially damage the hardware. For example, folding simulations of the NTL9 protein in implicit solvent with a τ value of 15 ps resulted in such issues on gaming GPUs (i.e. NVIDIA GTX 1080Ti GPUs) while the same simulations have no such issues on professional-graphics-programming GPUs. Coarse-grained simulations (residue-level models and coarse-grained) with high I/O are also problematic on gaming GPUs.

6 Uncertainty Quantification and Monitoring of Convergence

Although they can report on much longer timescales, WE calculations still have limitations analogous to those of conventional MD simulations – namely, force field inaccuracy and inadequate sampling. Assessing convergence requires care, as noted below. Even if sampling is adequate, as with any simulation result, error bars are required to set the results in context because there is always a finite range of results which are predicted in any stochastic calculation [36]. Error analysis is particularly challenging because WE results ultimately depend on a large number of trajectories which typically are significantly correlated with one another due to repeated replication ("splitting") events. Over the years, different error analyses have been employed [28, 31, 37]. Here we give a brief overview of current practice.

The primary recommendation is to perform multiple, fully independent WE simulations when possible. To understand the variation intrinsic to WE sampling, we suggest performing these runs from identical starting states. The data from these runs will not go to waste, as it can be combined for estimating observables, convergence, and error bars. When multiple runs are not feasible for a large-scale application, a sufficiently large number of trajectories/bin (at least 4 trajectories/bin) should be used to increase the chances of obtaining a diverse ensemble of pathways. To further enhance the diversity of the pathways, we recommend starting the simulation from multiple starting states when that is physically appropriate such as in protein binding. We note that a single run with a large number of trajectories/bin (4-50 trajectories/bin) has been shown to be more efficient in calculating rate constants than multiple runs with a small number of trajectories/bin (i.e. < 4 trajectories/bin) for molecular association/dissociation systems [41].

We focus here on understanding uncertainty in rate-constant estimation. First, there is the issue of "convergence": how much time is required to obtain a result without systematic bias that is governed only by statistical noise? In a typical simulation started in a single state (A), the rate constant into a target state B is estimated by the steady-state probability flux into B – i.e., the amount of probability arriving per unit time as sketched in **Figure 3**. However, there is a transient regime before the flux levels off to its steady value, and it is unknown in advance how long the transient will last. Of course, one should examine the time-dependence of the average flux (averaged over all WE runs) by eye, but this is unlikely to be sufficient. In addition, one can plot the flux as a function of some continuous coordinate which progresses from A to B: in steady state, the flux will be constant along any such coordinate [18]. Finally, we recommend using a "history augmented" Markov state model (haMSM) employing very fine bins/microstates, which can be built from the WE data as a different means for estimating steady-state flux values which can be compared to those measured directly in WE simulation [18]. Alternatively, the impact of transient effects on rate-constant estimation can be reduced by incorporating the distribution of event durations (excluding dwell time in the initial stable state) that correspond to pathways captured by the simulation. This strategy has been shown to yield rate constants using a fraction of the simulation time required by the original WE method [20].

Once the transient has completed, if multiple runs were performed, it is necessary to estimate the uncertainty in the rate constant based on the group of independent WE runs. The flux curves from the individual runs, plotted as a function of molecular time, may vary significantly as sketched in **Figure 3**. This large variation invalidates typical uncertainty estimation schemes based on the standard error of the mean,

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

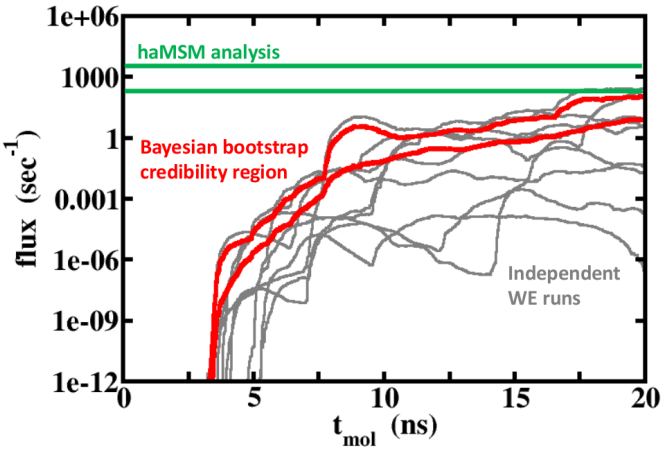

Figure 3. Convergence assessment and error analysis in the face of large run-to-run variation. The flux of probability into the target state B computed as a function of continuous molecular time, t_{mol}, is shown for several independent WE runs (grey). The large variation among individual runs makes it challenging both to assess whether the transient period has ended and to construct reliable error bars (see text). The history augmented Markov State Model (haMSM) analysis (green lines) provides an estimate of the long-time behavior, and the Bayesian bootstrap credibility region (red lines) estimates the average transient behavior.

and we therefore recommend employing a Bayesian bootstrapping procedure [28]. This approach appears to be better than alternative approaches for handling estimates which vary over orders of magnitude, but we emphasize that the nominal 95% "credibility regions" produced are overly optimistic and only cover the true mean a much smaller percentage of the time [28].

7 Tutorials

7.1 Basic Tutorial: Na^+/Cl^- Association

7.1.1 Introduction

This tutorial involves carrying out a WE simulation of a molecular association process: Na^+/Cl^- association. After completing this tutorial, a user should be able to set up a simple WE simulation using the WESTPA software and develop an intuition for how changes in the WE parameters will influence the efficiency of sampling a process of interest, thus allowing the user to choose appropriate parameters for that process.

Learning Objectives. Though we strive to make the WESTPA software as user-friendly as possible, there are many system-specific parameters that must be carefully specified. The purpose of this basic tutorial is to introduce a new user to WESTPA and have that user become familiar with the flow of setting up and running a WE simulation. Specific learning objectives are:

1. Become familiar with the main simulation directory layout
2. Choose a progress coordinate
3. Choose an appropriate binning scheme
4. Prepare input files
5. Monitor a simulation

7.1.2 Prerequisites

Users should install the latest version of the WESTPA software package through Conda. Installation instructions can be found on our Github wiki (https://github.com/westpa/westpa/wiki/Installing-WESTPA). For analysis of simulation data, the `hdfview` software greatly facilitates the visualization of large datasets. We will make use of that program in the Analysis Tutorials (**Section 7.4**).

Users should have basic knowledge of command line usage and the Python programming language. Since WESTPA is designed to conveniently interface with any external dynamics engine, users will also need to have experience using an MD engine (Amber, Gromacs, etc.). This tutorial will not provide instructions on how to use those engines; only how to interface the engines with WESTPA. In addition, a knowledge of analysis programs (such as Amber's `cpptraj` program or the MDAnalysis software) is necessary and will not be covered here. This tutorial will go over examples of the various input files that are necessary for interfacing with WESTPA. This tutorial also assumes the user has some knowledge of the WE strategy, as its basic theory is not discussed herein.

Computational Requirements. A user should set aside at least 18 GB of disk space. This simulation took ~50 hrs to complete using 1 Intel Xenon 3.50 GHz CPU core.

This tutorial uses OpenMM version 7.3 for dynamics propagation (http://openmm.org/) and MDTraj 1.9.3 for progress coordinate calculations (http://mdtraj.org/1.9.3/index.html). System setup and equilibration was performed separately in OpenMM. A minimum version of 3.1.0 for HDFView is required for H5 file analysis.

7.1.3 Setting up a WE Simulation Using WESTPA

Overview. WESTPA is run by calling the `w_run` program from the command line with the appropriate options. This is normally done by running the `run.sh` script from the main simulation directory. The simulation will then run until it has either completed the number of iterations specified by the user or has run out of time. Both of these parameters can be adjusted. Before a simulation can be run, however, the system must be initialized by calling the `w_init` program from the command line with the appropriate options. This is normally done by running the `init.sh` script from the main simulation directory.

Therefore, assuming the system is set up properly and all parameters have been properly specified, the WESTPA simulation can be run with the following at the command line

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

(throughout our suite of tutorials, the command prompt is indicated with $, which itself is not part of the commands that should be entered by the user):

```
$ ./init.sh
$ ./run.sh
```

Data from a WESTPA simulation will be stored in a file called `west.h5`, which is an H5 file that can be opened with Python's `h5py` package or with a graphical interface such as `hdfview`.

To monitor the simulation's progress, we will use the `w_pdist` program of WESTPA. This will generate probability distributions (histograms) as a function of the progress coordinate and will enable the user to view those histograms with the plothist program.

A WESTPA simulation, even after the requested number of iterations, may not be "complete." Completion is assessed by whether some observable has converged to an expected or steady value. The choice of this observable is up to the user. To obtain these observables (such as the flux or rate constant), one will have to access the data in the H5 file and plot it using Python's `matplotlib` package (or another equivalent package).

Once a simulation is deemed complete, users can make use of the WESTPA analysis tools suite of programs, specifically `w_ipa` in order to extract relevant data from the H5 file.

The System. To obtain a basic understanding of WESTPA's parameters and learn how the software works, we will begin by studying the molecular association of Na^+ and Cl^- ions. Our system will consist of a single Na^+ cation along with a single Cl^- anion modeled with Joung and Cheatham parameters [42] and solvated in a box of TIP3P water molecules [43]. These ions are initially dissociated at a separation distance of 12 Å. The system was prepared using OpenMM and the appropriate input files are provided under "westpa/tutorials" on GitHub, where you will also find a copy of this tutorial's simulation directory (`basic_nacl`). We will not cover how the input files were generated or the rationale behind choices made when setting up the system (e.g. force field, water model etc.).

Choosing an Initial State. In looking at the association of two entities, especially thinking about how to extensively sample this process, there are some things we want to consider before we begin WE. The first is how our initial state should look. If we choose to place the ions too close together, we may only observe one "type" of binding pathway, since the ions will not have as much time to orient themselves before binding. In reality, ions are symmetrical and we will not need this consideration but this would be an issue when determining how far apart to space, say, a drug and protein system or two protein binding partners. We also do not want to space the ions too far apart, as that would unnecessarily increase the time needed to observe binding events. We will therefore choose a generous distance of 12 Å.

The coordinates (and velocities) of this starting structure, `bstate.xml`, are placed in the `bstates/` directory. This is an OpenMM save-state file, which was saved after equilibration. This is the file needed to directly resume dynamics. Depending on the dynamics engine you are using, this file will be different but will have the same function (for instance, an Amber restart file would be placed here if one were using `sander` to run dynamics). Also in this directory is a file named `bstates.txt`. This file contains the name of our basis state structure and the probability of it being chosen if we want to sample a variety of initial structures (since we are preparing only one basis state, that probability is just 1). To more fully sample the configurational space of some process, it is often prudent to include more than one initial structure. In that case, all of those structure files can be placed in this directory with their file names and probabilities included in the `bstates.txt` file.

Files for Dynamic Propagation. Also necessary for running an Amber simulation are the topology and simulation input files. Those two files (`bstate.pdb` and `nacl_prod.py`) are placed in the `common_files/` directory. This is a catch-all folder for any files needed while running dynamics. Notice that our τ value is defined in the `nacl_prod.py` file, which is a Python script that runs OpenMM. This is the length of each WE iteration; so if the MD input script will run dynamics for, say, 10 ps then your τ value is 10 ps. This number needs to be carefully chosen depending on your system of interest. For this simulation, we will use a τ value of 50 ps.

Preparing the System Environment. Next, we will want to make sure that WESTPA can properly access the MD engine we want to use and set up our simulation environment properly. These variables are all defined in the `env.sh` file. You will need to open that in `vim` or another text editor and make sure that your WESTPA environment is being sourced correctly (only if you are not using the Conda environment) and that your dynamics environment is being sourced correctly. It is also advised to set the runtime command variables for more efficient system calls, if applicable.

Equilibrium vs Steady State WE. Now, let's examine the `init.sh` file, which initializes the simulation. In this file, we can specify whether to run an equilibrium or steady state simulation. The file in the tutorial directory is set up to run a steady state simulation. This is specified with the definition of the `$TSTATE_ARGS` variable and its use in the `w_init` command. To run an equilibrium simulation, simply delete those two lines.

The choice of whether to run an equilibrium vs steady state simulation will depend on the research question being

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

asked. Where do we want the system to go? Equilibrium simulations can be efficient in exploring configurational space and sampling ensembles of conformations. On the other hand, steady state simulations, where trajectories that reach some target state are recycled back to the initial state (along with their trajectory weights), can be more efficient in generating rate constants, and for exploring pathways towards some known target state [41].

In our simulation, we do have a specific target state in mind and we know exactly what it looks like: Na^+ and Cl^- interacting ionically at a close distance. We will therefore prepare to run a steady state simulation.

Progress Coordinate, Binning Scheme and τ value. For any WE simulation, we recommend choosing a progress coordinate that monitors the slowest relevant motion(s) such that faster motions will "go along for the ride." The efficiency of generating pathways is tightly coupled to the choice of progress coordinate, along with how you choose to divide up that coordinate into bins. For the molecular association process involving the Na^+ and Cl^- ions, a logical choice of progress coordinate would simply be the distance between the two ions, assuming that the surrounding solvent molecules respond relatively quickly to the positions of the ions. In other words, we can measure the simulation's "progress" by how close the ions are to each other in a particular trajectory. This will turn out to be a good choice for our system, but for systems in which the binding partners involve ensembles of conformations, a pure distance-based progress coordinate will not be adequate and must be combined with a second dimension of the progress coordinate that tracks some other motion of the system.

Now that we have chosen a progress coordinate, we will need to consider our binning scheme. Imagine a space that contains all of the possible values of our progress coordinate. A good place to start is to perhaps define our progress coordinate as ranging from your initial state (basis state) to a preliminary definition of your target state and divide up this coordinate into 1 Å-wide bins. One way to obtain a preliminary definition of the target state for the Na^+/Cl^- association process is to subject a model of the associated Na^+ and Cl^- ions to energy minimization using the same force field that will be used during the WE simulation and calculate the resulting distance between the ions using `cpptraj`. This distance ended up being 2.6124 Å, so we will set 2.60 Å as our preliminary definition of the target state. We recommend choosing the most strict definition possible for the target state for the recycling of trajectories in a steady state WE simulation to enable the use of more lenient definitions after the completion of the simulation. Make sure to add this number to `tstate.file` in the main simulation directory, where your steady-state target state definition should always be placed.

Back to our bin definitions. If we choose to space our bins by ones from 2.6 to 12 Å by 1 Å's (or some similar increment), this can lead to your simulation stalling. If trajectories cannot move to the next bin before a round of combination and replication occurs, the bins may be too large with respect to the chosen τ value or progress coordinate. It is a good idea, therefore, to run a short (10-20 iterations) WESTPA equilibrium simulation to see how your trajectories are progressing with the WE parameters you have set. If necessary, adjust the binning or include an additional dimension to your progress coordinate.

Here is the preliminary binning scheme we will employ, which is defined in the `west.cfg` file:

```
[0.00, 2.60, 2.80, 3.00, 3.20, 3.40, 3.60, 3.80,
 4.00, 4.50, 5.00, 5.50, 6.0, 7.0, 8.0, 9.0,
 10.0, 11.0, 12.0, 13.0, 14.0, 15.0, 'inf']
```

Notice how we start at 15 Å (a little bit beyond our initial value of 12 Å) and increment by ones, but as we get closer to our preliminary state of 2.60 Å, we start incrementing more finely. This finer binning will help to collect probability closer to our target state and promote more binding events.

Other WE Parameters. The following WE parameters are discussed along with where they are specified in the parameter files. First, make sure you have chosen an appropriate τ value (see **Section 4**) and that it is properly specified in your dynamics input file. As mentioned above, the τ value, along with the number of trajectories per bin, is coupled to the choice of progress coordinate and binning scheme. We recommend starting with ~4-5 trajectories/bin. This value is specified in the `west.cfg` file as `bin_target_counts`. Make sure that the frequency at which conformations are saved in your trajectories (as indicated in your dynamics input file, e.g. `md.in` for Amber) matches the number of elements in the pcoord array of the `west.cfg` file. We recommend running the simulation for a short time to test the effectiveness of the WE parameters, setting `max_total_iterations` to 10 in the `west.cfg` file before letting the simulation run to a full 100 iterations.

Trajectory Imaging. Since the replication and combination of trajectories in a WE simulation depends on the values of the progress coordinate, trajectories that are carried out with periodic boundary conditions should be imaged before calculating the progress coordinate (e.g., after completing each trajectory segment of length τ). Otherwise, erroneous values of the progress coordinate may result from parts of the simulation system drifting outside of the periodic box. MDTraj, which is used to calculate the distance in this tutorial, is able to only calculate distances for nearest-image ion pairs (essentially what Amber does with the `autoimage` command in AmberTools' `cpptraj` program.)

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

7.1.4 Initializing the WE Simulation

To initialize the simulation, run the `init.sh` script as mentioned before. You will see a body of text output indicating that the initialization has completed successfully. We will briefly present the key features of this script.

As mentioned before, `init.sh` calls the `w_init` program, which in turn, runs a script in the `westpa_scripts/` directory called `get_pcoord.sh`. This script, in this tutorial, is very simple. It prints the contents of a file, `pcoord.init`, and gives that to `$WEST_PCOORD_RETURN`. The `pcoord.init` file contains the progress coordinate value of the basis state, and so this operation essentially tells WESTPA which bin your basis state falls into. The `pcoord.init` file is generated by running the `get_distance.py` script in `common_files/` on `bstate.xml` and redirecting the output into a file named `pcoord.init`. Initializing your system this way is often a good idea, as it allows you to test out your particular method of progress coordinate calculation. However, `get_pcoord.sh` can calculate the progress coordinate directly (see **Intermediate Tutorial** 7.2), or run whatever script you need to do so. In fact, `get_pcoord.sh` can include any additional commands; this built-in flexibility allows you to perform operations on your basis states before beginning the WESTPA simulation.

7.1.5 Running the WE Simulation

To carry out the simulation, run the `run.sh` script as mentioned before. You will not see any output. What `run.sh` does is call `w_run` which, among other things, runs the `runseg.sh` script that is in the `westpa_scripts/` directory. This script will run dynamics each iteration, calculate a progress-coordinate value for the updated structure and then return that value to `$WEST_PCOORD_RETURN`.

In this tutorial, OpenMM is used to run dynamics (by running the `nacl_prod.py` script) and MDTraj is used to calculate the progress coordinate (by running the `get_distance.py` script). If a user wishes to change either the dynamics or analysis programs, these are the two locations where it will need to be done.

For an example script for using Slurm to run a job on a computing cluster, see `runwe.slurm`. You can adapt this template script to run WESTPA on your desired cluster.

7.1.6 Monitoring the WE Simulation

We recommend checking the progress of your WE simulation every 10 iterations or so. This can be done with the `w_pdist` program. To use this program, first stop the simulation (it can be started easily from the point it left off by running `run.sh` again) and then call `w_pdist`:

```
$ w_pdist
```

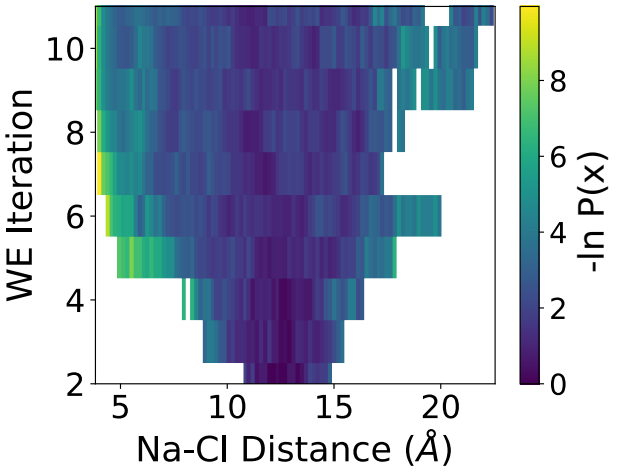

Figure 4. Probabiity evolution of Na^+/Cl^- association as a function of interatomic distance and WE iteration. The distribution from your particular simulation may look slightly different. Observe that at the beginning of the simulation, the probability is centered around 12 Å (the initial distance).

This will produce a new H5 file called `pdist.h5`. To see how our progress coordinate is evolving over time, we can use the plothist program with the evolution option:

```
$ plothist evolution pdist.h5
```

This will produce a pdf file called `hist.pdf`. Open this file, the contents of which are displayed in **Figure** 4.

As expected, most of the probability at the start of our simulation is concentrated around the progress coordinate value for our initial state (10 Å). As our simulation progresses, the probabilities fan out in both directions, with most of the probabilities moving towards larger values and some of the probabilities nearing our target value of 2.6 Å. To see if your simulation has generated some successful binding events after only 10 iterations, run the following:

```
$ w_succ
```

The example simulation had its first successful event after 14 iterations. The output will show (if a successful event occured) the iteration and segment number in which the first event occurred (e.g. iteration 14, segment 2).

You can trace this successful trajectory back to the basis state to obtain a complete trajectory with the `w_trace` command. You will need to provide the iteration and segment of the successful trajectory as options separated by a colon:

```
$ w_trace 14:2
```

The output will be written to the file `traj_14_2_trace.txt`. That file contains the parents of the successful trajectory all the way back to the basis state.

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

7.1.7 Analyzing the WE Simulation

One way to assess the convergence of our simulation is to determine when the primary observable of interest (i.e. the flux into the target state) levels off. To monitor the flux, we will first need to prepare our `west.cfg` file to analyze the simulation. This is normally done by adding an analysis module to the end, which is already present in this tutorial's files. Use this as a template for future analyses.

You will see that we create an analysis instance called `TEST` and then define bins and states for this scheme. These bins are strictly for analysis and have nothing to do with our progress coordinate bins defined earlier. Since we only need to designate the bound and unbound states here, we define three bins:

```
[0.0, 2.6, 10.0, 'inf']
```

The way that state definitions work is that you provide a progress coordinate in the configurational space and whichever analysis bin that coordinate is in becomes that state. For instance, our bound state definition is given by `[0]`, so whichever bin above that the value 0 falls into will be our "bound" state. This is the bin from 0 to 2.6. The same goes for the unbound state (10.0 to infinity). The intermediate state (2.6 to 10.0) does not need to be defined.

With these states defined we can now analyze how much probability, in the form of trajectory weight, is entering or leaving each state using the `w_ipa` program, which will run two separate WESTPA tools, `w_assign` and `w_direct`. To generate the H5 files needed to analyze the fluxes, run the following from the main simulation directory:

```
$ w_ipa -ao
```

You will see that a new directory titled `ANALYSIS` has been created, inside of which is a subdirectory corresponding to our `TEST` analysis scheme that was defined in the `west.cfg` file. Inside of this subdirectory are our `assign.h5` and `direct.h5` files. The `direct.h5` file is where the fluxes are stored. We can open it up with `hdfview` and view all of the datasets.

The `target_flux_evolution` dataset gives the flux over time (number of WE iterations) into each state we defined earlier. To view this dataset, double click on it. The 0th column corresponds to the flux into state 0, which we defined as our target state. The iter stop is at the beginning of that iteration, so if you had a binding event by iteration 10, observe the flux into our target state. Highlight the "expected" column and click the plotting button in the upper-left hand corner to view the flux evolution as a function of 0-indexed iteration.

By iteration 10, the flux has most likely not levelled off, so our simulation cannot be considered converged. Let's continue the simulation for a total of 100 WE iterations and analyze the resulting dataset. A completed H5 file is included in the `for_analysis/` directory for your convenience. Your plot should look something similar to **Figure 5**, which was generated in `matplotlib`.

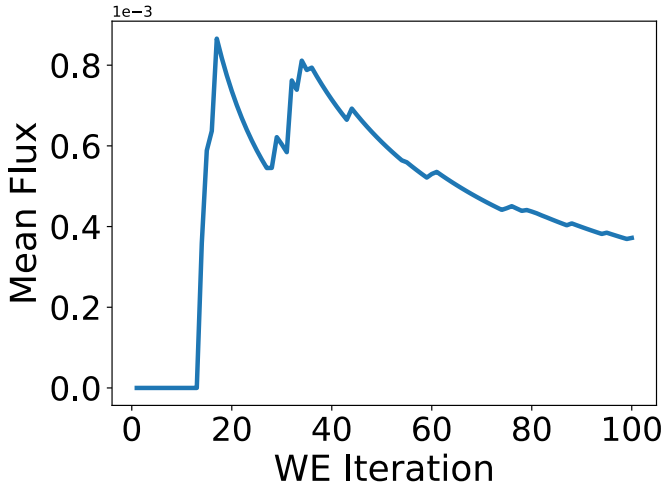

Figure 5. Mean flux evolution of Na^+/Cl^- association as a function of WE iteration. The mean flux alternatively rises sharply and then relaxes. These "peaks" correspond to probability crossing into the target state. Your plot may still not be completely converged after 100 WE iterations.

While the flux into the target state has not completely levelled off, it is much more steady than previously, so we can stop the simulation here and consider how much longer we should extend the simulation. For other systems, you may want to run the simulation longer for better convergence. You may also want to have additional criteria for convergence.

To visualize a trajectory, one must first identify a continuous series of trajectory segments in each iteration from the basis state to the target state. This will be given in the `w_succ` output along with `w_trace`, as we have done previously. However, you will also need to retrieve the trajectory file from each of those segments and combine them using `cpptraj`. To automate this process, we have provided the `amberTraj.sh` script, which can be adapted for other systems. This script uses the `cpptraj` program available in AmberTools to extract the binding trajectory of a successful event. The resulting trajectory file can be loaded along with the system topology into the VMD visualization software to generate a movie of the association process.

7.1.8 Conclusion

Hopefully by this point you have gained a good idea of the work flow required to set up, run, and analyze a WESTPA simulation using a simple progress coordinate. If you desire more complex options for your simulations (e.g. multi-

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

dimensional progress coordinates) and further discussion of how to choose various simulation parameters, we highly suggest going through the other tutorials to get a sense of how that can be done.

7.2 Intermediate Tutorial: P53 Peptide Conformational Sampling

7.2.1 Introduction

Since the WE algorithm aims to fill empty bins in configurational space, WE simulations can be effective in the enhancement of conformational sampling [1] as well as the generation of pathways and rate constants for rare events. This tutorial will focus on the conformational sampling of a peptide and instruct users on how to set up and analyze a simulation involving a two-dimensional progress coordinate. In addition, we will go over how the binning scheme can be chosen and adjusted in order to balance efficiency and performance.

Learning Objectives. This tutorial will help users develop a sense for which progress coordinates may be effective for conformational sampling of a peptide and how to bin along those progress coordinates.

Specific learning objectives include:

1. How to set up a two-dimensional progress coordinate
2. How to monitor this coordinate as the simulation progresses
3. How to evaluate whether the binning scheme is effective
4. Combining and creating bins "on-the-fly"
5. Storing and accessing auxiliary data

7.2.2 Prerequisites

Users should have completed the **Basic Tutorial 7.1** and have a potential progress coordinate in mind for their system of interest.

Computational Requirements. This simulation required at least 10 GB of disk space and ~36 hours to complete (40 iterations) on a 12-core, 2.6 GHz Intel Xeon node. This tutorial uses AmberTools19's `sander` package for dynamics propagation and the `cpptraj` package for progress coordinate calculations (http://ambermd.org/AmberTools.php). AmberTools is available free of charge.

7.2.3 Adding Another Dimension to the Progress Coordinate

While a one-dimensional progress coordinate can be effective for molecular association processes (e.g. Na^+/Cl^- in the **Basic Tutorial**), a two-dimensional coordinate may be necessary for more complex processes such as peptide/protein conformational transitions. To add another dimension to the progress coordinate, we first specify the progress coordinate dimensionality as "2" in the `west.cfg` file. Next, we calculate the values corresponding to each dimension of the progress coordinate and pass the resulting two values at the same time to `$WEST_PCOORD_RETURN` in both the `get_pcoord.sh` and `runseg.sh` scripts. For example, if the first dimension of the progress coordinate has a value of 1 and the second dimension has a value of 5, (1 5) must be passed at the same time to `$WEST_PCOORD_RETURN` instead of sequentially as 1 and then 5. This can be done with the `paste` command in bash (see example `get_pcoord.sh` and `runseg.sh` files). In addition, the bins will need to be specified as two lists, one for each of the two dimensions. This is done by adding dashed entries (one underneath the other) in the `west.cfg` section for bin definitions. A user may alternatively choose to define a two-dimensional binning scheme in a `system.py` file.

7.2.4 Preparing the WE System

The System. We will focus on the conformational sampling of a 15-residue, N-terminal peptide fragment of tumor suppressor p53 that has been thought to be disordered in its unbound state and adopts an α-helical conformation upon binding the MDM2 protein. Simulations were run at 275 K using the Amber ff14SBonlysc force field [44] and generalized Born implicit solvent [45]. As in the Basic Tutorial, we will not go into detail about how the files were generated in Amber or the decisions made in setting up the system with Amber.

Choosing an Initial State. Our WE simulation will be started from the MDM2-bound conformation of the p53 peptide. In particular, coordinates for the peptide conformation will be extracted from the crystal structure of the MDM2-p53 peptide complex [46]. This α-helical conformation of the peptide will then be energy-minimized and equilibrated before subjecting the resulting, solvated system to a WE simulation.

Files for Dynamics. The topology file (`P53.MDM2.prmtop`) and dynamics input file (`md.in`) can be found in the `common_files/` directory. In the `md.in` file, it should be specified that the trajectory segment will be run for a length that corresponds to a τ value of 50 ps.

Preparing the Simulation Environment. See the corresponding subsection in the **Basic Tutorial 7.1**.

Equilibrium vs Steady State WE. In the `init.sh` file, observe that all lines mentioning `TSTATE_ARGS` have been removed. This signals WESTPA to run an equilibrium WE simulation in which we do not have a set target state. This is a good option when the goal of your process is to generate as many configurations as possible and you have no set target state in mind.

Progress Coordinate, Binning Scheme and τ Value. To extensively sample the conformations of the peptide, we might define a progress coordinate that monitors the extent

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

of "unfoldedness" in the peptide using the RMSD of a given conformation from the initial structure. However, RMSD cannot differentiate among conformations that have the same large RMSD values. To further differentiate between such conformations, we can include another orthogonal measure of unfoldedness such as the end-to-end distance of the peptide.

To determine a suitable binning scheme, we will start with an upper limit of 10 Å for the heavy-atom RMSD dimension of the progress coordinate. Spacing the bins along this dimension by 1's may be too large for any transitions to occur between bins so we opt for a finer bin spacing:

```
[0.0, 0.2, 0.4, 0.6, 0.8, 1.0, 1.4, 1.8,
 2.2, 2.6, 3.0, 3.5, 4.0, 5.0, 6.0, 7.0,
 8.0, 9.0, 10.0, 'inf']
```

We will see how the trajectories progress and adjust accordingly. Notice that a bin spacing of 0.2 is not maintained for the entire length, as 50 bins even along one dimension would result in a very large number of total trajectories (4 trajectories per bin would yield a total of 200 trajectories if all of the bins are occupied). Furthermore, care must be exercised in the addition of bins along a second dimension as the total number of trajectories can "blow up" to an enormous number of trajectory segments (e.g. 10,000).

To get a feel for how the end-to-end distance evolves in the simulation, let's expand out from the initial distance of 28.5 Å with 0.5-Å wide bins in either direction:

```
[0, 20, 20.5, 21, 21.5, 22, 22.5, 23, 23.5,
 24, 24.5, 25, 25.5, 26, 26.5, 27, 27.5, 28,
 28.5, 29, 29.5, 30, 30.5, 31, 31.5, 32, 32.5,
 33, 33.5, 34, 34.5, 35, 35.5, 36, 'inf']
```

Our τ value should allow for successful transitions between bins of this spacing.

Other WE Parameters. Let's run our WE simulation with 4 trajectories/bin for 40 iterations. Since the goal here is the conformational sampling of a peptide and we are running an equilibrium WE simulation, we do not need to define a target state.

7.2.5 Tracking the Auxiliary Data

While it is possible to go back after a simulation has run and calculate some value you wish you had kept track of, it can be tricky to do so (though possible with a tool called `w_crawl` which is not discussed in this guide). We strongly recommend conducting all relevant analysis during the simulation and storing the resulting data as auxdata in the H5 file. In our case, we will calculate and store the ϕ/ψ backbone dihedral angles of the peptide as auxdata for each of the sampled conformations.

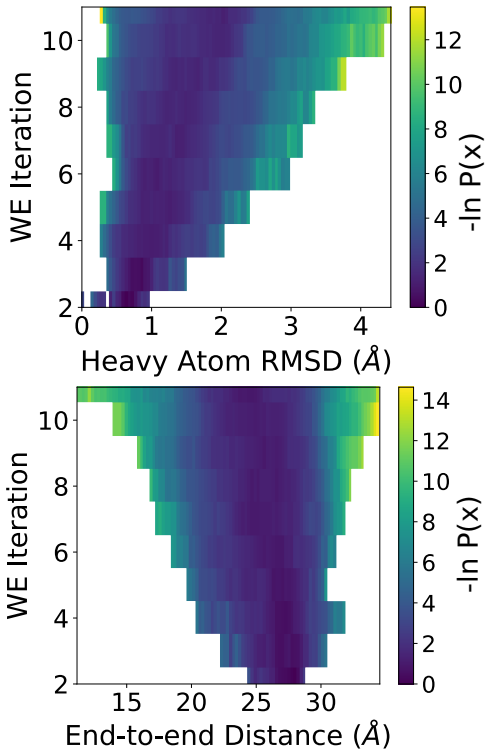

Figure 6. Probability distributions for each of the two progress coordinate dimensions versus WE iteration for the p53 system. The simulation was analyzed after 10 WE iterations.

To signal for WESTPA to collect auxdata, you will need to add an auxiliary dataset into the `west.cfg` file and make sure it is enabled. See the `west.cfg` file in the tutorial directory for an example of how this might look. You can name the dataset whatever you would like.

Once you have specified the datasets and named them, you will need to add in commands to `runseg.sh` that calculate those values and pass them to WESTPA system variables. The variables will be named `$WEST_XYZ_RETURN` where "xyz" is the name given to the dataset in the `west.cfg` file. This can be treated analogously to the pcoord value and `$WEST_PCORD_RETURN`.

7.2.6 Initializing and Running the WE Simulation

Make sure that your `get_pcoord.sh` and `runseg.sh` files are calculating the RMSD and end-to-end distance and returning these values to `$WEST_PCOORD_RETURN`. The `get_pcoord.sh` script will calculate the initial progress coordinates using AmberTools' `cpptraj` program from within the script, as opposed to reading the value from an external file as in the **Basic Tutorial 7.1**. The `runseg.sh` uses AmberTools' `sander` program for dynamics propagation and does so within the script.

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

7.2.7 Monitoring the WE Simulation (10 Iterations)

Once the simulation has run for about 10-20 iterations, copy the H5 file and run `w_pdist` with the copied file. You can then use `plothist` to view each dimension of the progress coordinates separately as the values evolve over the course of those few iterations:

```
$ plothist evolution pdist.h5 0 -o hist_dim0.pdf
$ plothist evolution pdist.h5 1 -o hist_dim1.pdf
```

Where the "0" or "1" after the `plothist` command is the progress coordinate dimension (zero indexed). Observe the two probability distributions in **Figure 6**.

7.2.8 Adjusting Bin Spacings "On the Fly"

The RMSD has reached a value of 4-5 Å and the end-to-end distance has reached ~10 Å, which is encouraging progress for only 10 iterations. Note that most of the probability (and therefore most of the computation) is still stalled in the initial states of 1-2 Å RMSD and 20-25 Å end-to-end distance. We can help focus the computing power on the more interesting "edge" conformations by modifying the binning scheme before continuing the simulation.

In WESTPA, the binning scheme can be updated at any time since the trajectory weights are independent of the bins (and progress coordinate). To do so, first stop the simulation and then adjust the bins in your `west.cfg` file. Re-start your simulation by running the `run.sh` script again and the simulation will continue from where it left off. At the start of the next iteration, the new bins will have been implemented.

In our case, I would like to focus sampling on higher RMSD values (3-4 Å) instead of those ~1-2 Å. To do this, I will collapse the bins from 0 to 1.8 Å and define some more bins past 10 Å:

```
[0.0, 1.8, 2.2, 2.6, 3.0, 3.5, 4.0, 5.0, 6.0, 7.0
 8.0, 9.0, 10.0, 11, 12, 14, 16, 18, 20, 'inf']
```

For the end-to-end distance, I will add more bins for the lower distances and collapse bins over 26 Å. We would normally want to keep these bins over 26 Å but having fewer will shorten the runtime of this tutorial.

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 21,
 22, 23, 24, 25, 26, 'inf']
```

The reason we eliminated the initial 0.5 Å spacings is that this degree of freedom is readily explored in the system.

7.2.9 Monitoring the WE Simulation (40 Iterations)

After running the simulation for another 30 iterations (for a total of 40), we obtained the following updated probability distributions displayed in **Figure 7**. The completed H5 file is included in `for_analysis/` for your convenience.

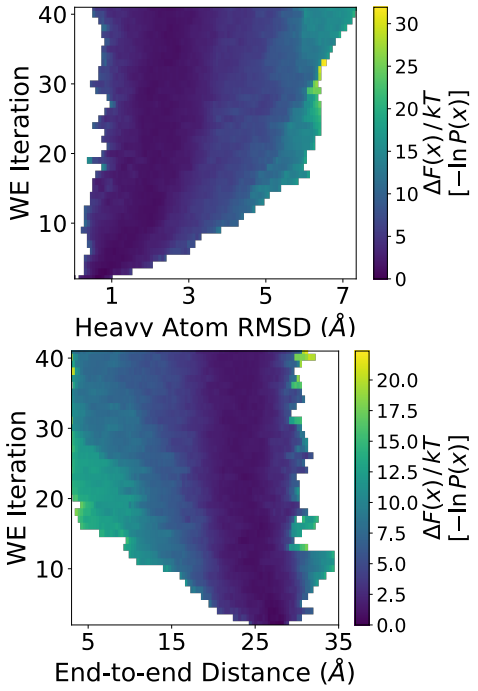

Figure 7. Probability distributions for each of the two progress coordinate dimensions versus WE iteration for the p53 system. The simulation was re-analyzed after 40 WE iterations

The effects of the bin-modifications can clearly be seen in the case of the end-to-end distribution. No more trajectories with an end-to-end distance >30 Å can be seen after iteration 10, a result of the choice not to bin over 26 Å in that dimension.

The end-to-end distance seems to have reached 2-3 Å around iteration 20. The RMSD plateaued a bit from iterations 20-30 but then proceeded to values around 7 Å. Two lessons can be learned from these observations. First, if you do not have bins in a particular direction, you may not see sampling in that direction. Second, even though the RMSD coordinate appeared to have stalled around iteration 20-30, it eventually was able to surmount whatever barrier existed and attain some higher RMSD values. Patience is key, as a single trajectory may replicate to become many trajectories if it crosses into a new bin.

7.2.10 Accessing Auxiliary Data

To access the auxdata from the H5 file, you can open `west.h5` in `hdfview` but this will not allow you really use the data. To plot all of the dihedrals as a Ramachandran plot in `matplotlib` as shown in **Figure 8** (actually, we just did so for the second dihedral, but you could extend it to all if you so desire), you will need to utilize the `h5py` package in Python to extract the auxdata values from the `west.h5` file and then plot them. The plotting script is included in the tutorial directory.

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

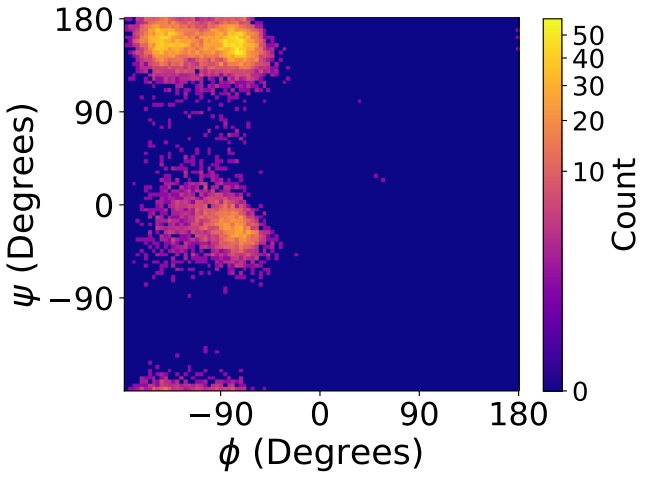

Figure 8. Ramachandran plot showing the occurance of ϕ/ψ angles of the second peptide bond of the p53 peptide, for each WE segment throughout the course of the simiulation.

7.2.11 Conclusion

Users should now be familiar with setting up a two-dimensional progress coordinate and working with auxiliary data. These two "tools" will help to expand your repertoire of WESTPA simulation techniques and give you access to more complex and informative simulations. Users should also now be familiar with changing bin spacings “on-the-fly” as well.

7.3 Intermediate Tutorial: Folding of Chignolin Mini-Protein

7.3.1 Introduction

Protein folding processes have been challenging to simulate due to the relatively long time scales involved. In this tutorial, we will use WESTPA to simulate the folding and unfolding of the chignolin mini-protein and to calculate the corresponding rate constants. We will run steady-state WE simulations of chignolin folding and unfolding processes separately. We will also compare the results of these simulations with those from brute force MD simulations, demonstrating the correctness and potential usefulness of the WE strategy.

Learning Objectives. This tutorial demonstrates how steady state WE simulations can be used to generate pathways and rate constants for both protein folding and unfolding processes.

Specific learning objectives include:

1. How to use brute force simulations to identify appropriate initial and/or a target states
2. How to obtain the probability flux into the target state of a WESTPA simulation, how to convert it to a mean rate constant, and how to interpret the results

Prerequisites. Users should have completed the **Basic Tutorial** 7.1.

Computational Requirements. We note that significantly more computing time is required for the folding simulations to yield converged rate constants and hence we suggest the user should start with the unfolding simulations. In particular, the WE unfolding simulation required ~53 hours for 1000 iterations on 32 CPU cores of 2.6 GHz Intel Xeon processors (~5 GB of disk space) while the WE folding simulation required ~8 days for 10,000 iterations (200 ns of molecular time) using the same resource (~50 GB of disk space). To become familiar with setting up and running the WE simulations, the users can carry out several iterations. Also, the brute-force simulation described below can be performed for tens of ns, as we benchmarked this system to produce ~150 ns per day on one of the above-mentioned CPUs. Output files for 1000 iterations of the WE unfolding and 10000 iterations of the WE folding simulations (as well as for 4 us of the brute-force simulation) can be found in the corresponding subdirectories. These files should be used for the analysis procedures outlined below. This tutorial uses AmberTools19’s `sander` package for dynamics propagation and the `cpptraj` package for progress coordinate calculations (http://ambermd.org/AmberTools.php). AmberTools is available free of charge.

The System. The chignolin mini-protein with the sequence GYDPETGTWG forms a β-hairpin and folds/unfolds on a timescale that is accessible to brute force simulations, which provide a reference data set for comparison with WESTPA results. The folded chignolin structure (PDB code: 1UAO, [47]), serves as the starting structure for both the brute-force and WE unfolding simulations. Both dynamics propagation and simulation analysis are carried out using the Amber software package. Simulations were run at 275 K using the Amber ff14SBonlysc force field [44] and generalized Born implicit solvent [45].

7.3.2 Brute Force Simulations

Overview. As mentioned in **Section** 1.3, it is important to run multiple, short, brute force simulations prior to using WESTPA. In the case of chignolin, which both folds and unfolds on timescales accessible to brute force simulation, brute force simulations can provide information on defining the unfolded and folded states.

Running and Analyzing the Brute Force Simulation. We perform a 4-μs brute force simulation of chignolin and write out coordinates every 20 ps. All files can be found in the `brute_force/` directory. The user can change these parameters in the MD config file `md.in`. The simulation can be submitted with the following command:

```
$ ./run.sh
```


https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

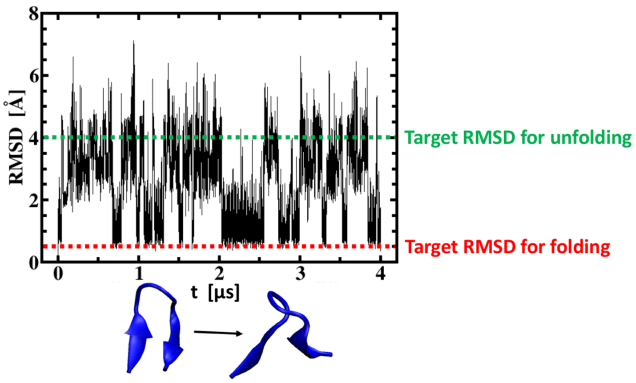

Figure 9. C_α RMSD vs simulation time for the brute force simulation of chignolin.

This submission script may have to be adjusted to the user's computing platform.

The chignolin C_α RMSD can be computed in the following way:

```
$ cpptraj chignolin.prmtop < get_rmsd.in
```

This command assumes the brute force simulation trajectory as well as the chignolin parameter topology and folded structure pdb files are all in the current directory.

The output RMSD data file, `rmsd.dat`, lists the time evolution of the chignolin C_α RMSD over the course of the simulation (each line corresponds to a frame).

Figure 9 shows the C_α RMSD over simulation time for a brute-force simulation that started from the folded β-hairpin, revealing several unfolding and refolding events within 4 μs. The unfolded and folded states are defined by visual inspection of the RMSD plot and simulated conformations, which show a fully formed β-sheet and native hydrogen bonds at RMSD < 0.5 Å and a disrupted β-sheet with broken native hydrogen bonds at RMSD > 4 Å (this pair of RMSD values will also be used later to define target states in WESTPA simulations). Note that the (un)folding rate constants will be sensitive to the state definitions, and defining states is a challenging process beyond the scope of this tutorial. Our state definitions are designed to avoid potential recrossing artifacts in rate calculations: once a trajectory reaches a state it should tend to remain there, rather than immediately returning to the previous state.

According to the Hill relation [48], the rate constant is exactly the inverse mean first-passage time (MFPT) of the underlying process, where, for instance, the FPT for unfolding is the time required to reach the unfolded state (RMSD > 4 Å) after first folding (RMSD < 0.5 Å). The user can run the following to obtain the MFPTs for both the folding and unfolding processes:

```
$ python get_mfpt.py rmsd.dat 20e-12 0.5 4.0
```

The command-line arguments are the RMSD data file, time interval at which the RMSD values are calculated in seconds, and threshold RMSD values for the folded and unfolded states in Angstroms. The rate constant of unfolding is estimated to be 0.13 x 108 s^{-1} (confidence interval: 0.09 x 108 s^{-1} – 0.18 x 108 s^{-1}) and that of folding is estimated to be 0.71 x 107 s^{-1} (confidence interval: 0.44 x 107 s^{-1} – 1.24 x 107 s^{-1}). Confidence intervals are derived from a Bayesian bootstrapping procedure [28].

7.3.3 Using WESTPA

Overview. We will carry out separate steady-state WE simulations for the unfolding and folding processes. This strategy is not only more efficient than equilibrium WE simulations in estimating rate constants (see **Section 7.1.3**), but enables us to set WE parameters for each process (e.g. bin spacing) in a more process-specific way if needed. The target state of the folding simulation will be used as the initial state of the unfolding simulation and vice versa.

Choosing an Initial State. As done for the brute force simulations, WE simulations of the unfolding process will be started from the NMR structure of chignolin. WE simulations of the folding process will be started from an unfolded conformation of chignolin (RMSD > 4 Å) that has been generated by the above brute force simulations.

Files for Dynamics. All files are in the `common_files/` subdirectory of either the `WE_folding/` or the `WE_unfolding/` directory.

Preparing the Simulation Environment. See the corresponding subsection in **Basic Tutorial 7.1**.

Equilibrium vs Steady State WE. Here we will run separate steady state WE simulations of the folding and unfolding processes, defining a target state (`TSTATE_ARGS`) in the `init.sh` files.

Progress Coordinate, Binning Scheme and τ value. As mentioned above, we will use a one-dimensional progress coordinate consisting of the C_α RMSD from the folded structure of chignolin. Although the RMSD with respect to a single reference structure may not be an ideal coordinate for distinguishing between various conformation, it proves sufficient for our example. Folded and unfolded states are defined based on maximum and minimum RMSD values, respectively, that have been sampled by the above brute force simulations. We will use a bin spacing of 0.2 Å and a τ value of 20 ps. However, the very first bin for the unfolding simulations is larger than the regular bin width with RMSD = [0 Å, 0.5 Å] because any structure with RMSD < 0.5 Å is considered to be in the folded initial state. Analogously, for the folding simulations, the very last bin is larger than the regular bin width of 0.2 Å.

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

Other WE Parameters. As done in the previous tutorials, our WE simulations were carried out using 4 trajectories/bin. The unfolding and folding simulations were run for 1000 and 10,000 WE iterations, respectively, in order to reach a steady value of the corresponding rate constants.

Initializing and Running the WE simulations. The `init.sh` and `run.sh` files can be found in the corresponding directories for both WESTPA simulations. The RMSD progress coordinate is calculated and its values returned to `$WEST_PCOORD_RETURN`.

Monitoring and Analyzing the WE Simulations. To compute the rate constant for the folding or unfolding process, we first calculate the mean probability flux into the target state by running the following WESTPA analysis tool:

```
$ w_fluxanl
```

The output is the H5 file `fluxanl.h5`, which contains the instantaneous probability flux into the target state for each iteration. The following Python script calculates, for any WE iteration, the average rate constant based on the corresponding probability flux arriving in the target state over a preceding window of molecular simulation times (e.g., over 1 ns):

```
$ python get_mean_rate.py 20e-12  1e-9
```

The command-line arguments are the τ value and the time width for window-averaging. Both arguments are in units of seconds.

Figure 10 shows the evolution of the average unfolding rate constant of chignolin as a function of molecular time for three independent WE simulations. After a few ns, the average rate constants for all of these simulations have leveled off and are roughly comparable to that derived from brute force simulations. One difference between the WE and brute force simulations is that the former estimates the MFPT based on the chosen initial structure(s) which may not correspond precisely to the ensemble of starting structures implicit in extracting first-passage events from brute force simulations. Note that a three-fold difference in the rate constants among the three WE simulations amounts to only ~0.6 kcal/mol difference in the effective free energy barrier to unfolding (at the simulation temperature of 275 K).

Figure 11 shows the evolution of the average folding rate constant for chignolin as a function of molecular time for three independent WE simulations. Compared with unfolding simulations, the folding simulations require much longer to reach a converged average rate constant that is in rough agreement with that from the brute force simulations; we note that the average rate constant is dominated by the largest flux. In addition, the folding rate constant exhibits significantly larger fluctuations, even after the apparent transient period of the first ~100 ns, indicating that the chosen bins are less suited for the folding process. During the

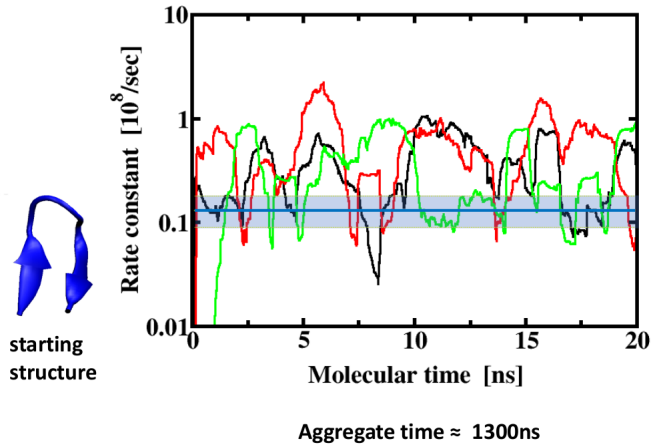

Figure 10. Estimating the unfolding rate constant of chignolin. The 1 ns window-averaged unfolding rate constant is shown in a semi-logarithmic plot for three independent WE simulations (black, red, and green) that were started from the same folded starting structure (see lower left). The corresponding unfolding rate constant from the brute force simulation is indicated by the horizontal blue line and its confidence interval by the shaded region. The molecular time is the time elapsed, N_τ where N is the number of WE iterations that each have a length of τ. The aggregate simulation time was on average, ~1.3 μs for each simulation.

folding process, distinct hydrogen bonds must be formed between the neighboring anti-parallel strands, and possibly in a specific order, to eventually reach an RMSD < 0.5 Å. In contrast, the unfolding process results in faster convergence of the corresponding rate constant and likely involves the simultaneous breaking of hydrogen bonds in order to reach an RMSD > 4 Å.

The resulting WE simulations consist of multiple continuous unfolding or folding pathways that may cover different regions of configurational space at any given time. To select for particular pathways (trajectories), we can run the following:

```
$ python get_target_trajs.py  1  10000
```

The command-line arguments indicate the first and last iteration number to be considered. The output file `target_trajs.dat` has two columns: one with the iteration number and one with the segment number of the trajectory that has reached the target state at that iteration. Thus, the number of rows indicates the total number of generated events. The iteration and segment numbers can be used by `w_trace` to obtain the full path of a particular folding or unfolding event (see **Section 7.1.6**).

7.3.4 Conclusion

In this tutorial, you have learned how to apply the WE strategy to simulate a protein folding process under steady

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

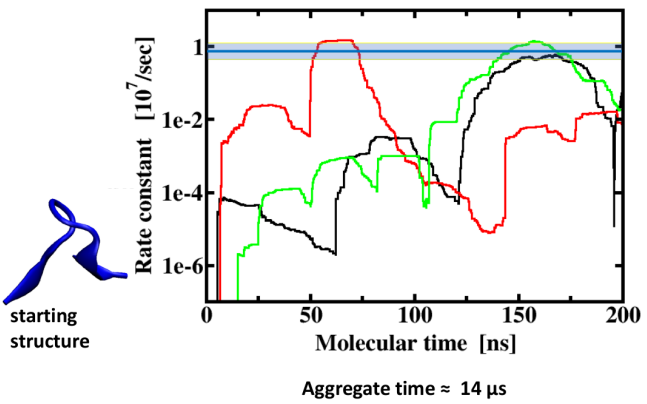

Figure 11. Estimating the folding rate constant of chignolin. The 20-ns window-averaged folding rate constant is shown in a semi-logarithmic plot for three independent WESTPA simulations (black, red, and green profiles) with the same unfolded starting structure (see lower left). Note the significantly longer molecular and aggregate simulation times for each simulation to obtain converged rate constants of folding compared to unfolding (see **Figure** 10). The corresponding rate constant from the brute force simulation is indicated by the horizontal blue line and its confidence interval by the shaded region.

state conditions. The recycling of trajectories at a target state allows the generation of a non-equilibrium steady state, to which the trajectory ensemble converges faster compared to an equilibrium ensemble of trajectories. Such steady states trajectories enable the direct computation of rate constants as described in this tutorial.

7.4 Analysis Tutorials

In this tutorial, we will go over how to calculate progress coordinates using external analysis suites, automate analysis of a WE simulation using the WESTPA `w_ipa` tool and visualize the evolution of WE datasets with time. We focus on the p53 peptide system described above in the **Intermediate Tutorial** 7.2 in which the progress coordinate is the C_α RMSD of the peptide from its folded, α-helical conformation.

7.4.1 Calculating Progress Coordinates Using External Analysis Suites

Introduction. Here we will demonstrate how to write scripts for calculating custom progress coordinates for WESTPA simulations using the external analysis suites MDAnalysis and MDTraj [49–51]. A prerequisite to this tutorial is completion of the **Basic Tutorial** 7.1. You will also need to install the MDAnalysis or MDTraj analysis suites. Other required files are provided on GitHub.

Learning Objectives. The specific learning objective of this tutorial is to calculate progress coordinates using an external analysis suite (MDAnalysis or MDTraj).

Explanation of Files and Scripts. The master configuration file for the simulation, `west.cfg`, specifies the dimensionality of the progress coordinate (`pcoord_ndim`), as well as how many progress coordinate data points should be returned from each segment (`pcoord_len`) (it specifies many other things but these are of primary interest for this tutorial as they specify the shape of the progress coordinate).

The script `rmsd.py` is responsible for using MDTraj or MDAnalysis to calculate the RMSD values during the simulation. Read the comments in the script to understand its setup for each package (there is a unique version for both).

Two scripts are responsible for calling `rmsd.py` at different points in the simulation (both found in `westpa_scripts/`):

- `get_pcoord.sh` calculates the progress coordinate during the initialization of the system. Because dynamics have not been run yet, WESTPA only needs a single point progress coordinate, rather than an array. This difference is controlled by the FORM argument, explained in the `rmsd.py` script.
- `runseg.sh` calculates the progress coordinate during dynamics propagation. It passes each segment's trajectory file as input to the custom progress coordinate loader, `rmsd.py`.

There are slight differences in these files for the MDAnalysis and MDTraj setups, explained in the comments of each script.

Files in amber_config/ directory:

- `P53.MDM2.prmtop` - The topology file.
- `md.in` - The input file which specifies conditions for dynamics propagation.

The other files needed for the simulation are found in the bstates folder, and are explained in the MDAnalysis/MDTraj specific sections below.

Running the Simulation. Before running the simulation, you may want to change the binning scheme, the number of iterations, or other parameters, which can be found in `west.cfg`.

To run the simulation, only two scripts must be executed.
To initialize the system:

```
$ ./init.sh
```

To run the simulation in the background:

```
$ ./run.sh &
```

To monitor the progress of the simulation:

```
$ tail -f west.log
```

The rest of the tutorial is specific to the software package used. See below for specifics involving the MDAnalysis and MDTraj analysis suites.

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

Using the MDAnalysis Analysis Suite
Files in `bstates/` directory:

- `P53.MDM2.ncrst` - Used as initial crystal structure to compare to the trajectory when calculating the RMSD and to start new trajectories in `runseg.sh`.
- `bstates.txt` - specify restart file `P53.MDM2.ncrst`.

Using the MDTraj Analysis Suite
Files in `bstates/` directory:

- `P53.MDM2.nc` - because MDTraj does not support restart files, this file is used in `get_pcoord.sh` to calculate the initial progress coordinate. It is also used by `runseg.sh` as an initial crystal structure to compare to the trajectory when calculating the RMSD.
- `P53.MDM2.ncrst` - Used to start new trajectories in `runseg.sh`.
- `bstates.txt` - specify restart file `P53.MDM2.ncrst`.

Conclusion. You have learned in this tutorial the basic structure of a Python script to calculate progress coordinates for WESTPA using the MDAnalysis and MDTraj analysis suites. There are two scripts run by WESTPA which call `pcoord_loader.py`, triggering the calculation of progress coordinates. The bash script, `get_pcoord.sh`, triggers the calculation of only a single progress coordinate, while `runseg.sh` triggers the calculation of the progress coordinate at multiple points in a trajectory, as defined in `west.cfg`. It is important to include the last line of the Python scripts, setting `segment.pcoord` equal to the progress coordinate array, so that the progress coordinate may be used to further the simulation.

7.4.2 The w_ipa Analysis Tool

Introduction. The `w_ipa` analysis tool is designed to facilitate analysis of WESTPA simulation datasets through a single interface (Jupyter Notebooks or the command line). In particular, `w_ipa` automates analysis routines, ensures data consistency through the use of automatically updated "analysis schemes", enables a user to easily view a particular dataset or trajectory segment in the H5 file, and monitors the progress of the simulation (e.g. trajectory weights, progress coordinates, and other properties of interest).

Learning Objectives. The specific learning objectives of this tutorial are to use the `w_ipa` analysis tool to:

1. Calculate rate constants
2. Trace and analyze trajectory segments (weight, pcoord, auxdata)
3. Plot datasets

Setting Up. Using `w_ipa` is straightforward. The `west.cfg` file, which specifies most of the simulation parameters, also specifies the analysis parameters under the `Analysis` heading.

The general format of the analysis section can be seen in the included `west.cfg` file. More detailed examples are available in the **Basic and Intermediate Tutorials** (**Sections 7.1-7.3**).

In order to run `w_ipa`, there must be at least a single analysis scheme specified. This scheme does not have to consist of the bins and/or state definitions used during the simulation. Less physically relevant schemes may be employed. Any changes made to analysis schemes in the `west.cfg` file will be actualized the next time `w_ipa` is run. The user is therefore guaranteed to never wonder whether the analysis files are up to date.

The `assign.h5`, `reweight.h5`, and `direct.h5` files are stored under `ANALYSIS/SCHEME_NAME`. The optional arguments that can be passed to `w_assign`, `w_direct`, and `w_reweight` can be specified by creating a section with the tool name and using the value pairs argument.

The Interface. To run `w_ipa` from the command line, enter the command `w_ipa` after having sourced `westpa.sh` (if not already sourced). To run `w_ipa` in a Jupyter notebook enter the command `w_jupyter` from the command line. When you create a new Jupyter notebook, there are some basic Python commands that must be executed:

```
import w_ipa
w = w_ipa.WIPI()
# At startup, it will load or run the analysis
 schemes specified in the configuration file
 (typically west.cfg)
w.main()
w.interface = 'matplotlib'
```

The Python kernel must be launched with the use of `w_jupyter`, or otherwise, the `$PYTHON_PATH` variable must be set to include the WESTPA directories. The command `w_env`, which ships with WESTPA, is responsible for setting environment variables and can be used with the Jupyter notebook command to ensure `w_ipa` is importable.

All commands are applicable from both the command line and Jupyter notebook interface; if plotting functions are called from the command line, the plot will appear within the console (it can be configured to use matplotlib if desired; this requires an active, available X session).

All of the variables are now accessible from the `w` object.

Changing Schemes and Accessing Datasets. A typical analysis routine begins by selecting an appropriate analysis scheme that may consist of multiple state definitions, averaging options, or reweighting parameters that are appropriate for the simulation. Most of the datasets are presented from the current, "active" state, although access to other datasets

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

is conveniently available. All numerical datasets are given as NumPy arrays, allowing for easy analysis of data.

To see what schemes are available, run the following command:

```
$ w.list_schemes
```

To change schemes, you may set the `w.scheme` variable to a string or integer value (corresponding to the index of the scheme). For instance, suppose you have the following two schemes: "`EXAMPLE`", and "`ALTERNATE`", and the current scheme is "`EXAMPLE`". To access the properties of the current iteration in the current scheme (explained in more detail below), you would type the following:

```
$ w.current
```

However, to access the alternate scheme, you would run the following command:

```
$ w.schemes.ALTERNATE.current
```

Where "`ALTERNATE`" corresponds to the scheme name written in the `west.cfg` file.

The `w_ipa` tool works by presenting an iteration and all its data as a single object. Each iteration object contains numerous datasets and helper functions designed to ease analysis. After loading, `w_ipa` defaults to the final iteration. You can change the iteration by using the following command:

```
$ w.iteration = 39
```

At any time, we have three iteration objects available in the object `w`: current, past, future. The past and future datasets are keyed to the parents and children of the segments in the current dataset. For instance, if you are analyzing segment 200 in the current iteration and wish to analyze the parent segment it came from, you could access the two datasets using the following iteration objects:

```
$ w.current[200]
$ w.past[200]
```

Even though it is very unlikely that the actual segment ID of the parent of segment 200 is 200, it is mapped correctly to enable convenient analysis. To obtain the actual segment ID, just run:

```
$ w.past[200].seg_id
```

OR

```
w.past[200]['seg_id']
```

As indicated above, objects in `w_ipa` can be called either as Python dictionaries or as attributes on the object. These can be listed by calling the print method on the parent object. In addition, as `w_ipa` is using iPython under the hood, tab completion works as when using the command-line interface (CLI).

To access the main datasets of interest, `pcoord` and `auxdata`, type the following:

```
$ w.current.pcoord
$ w.current.auxdata
```

These commands will output the full datasets, which can be useful for calculating properties on all trajectory segments at once. But what if we are only interested in looking at the properties of particular segments?

You could manually find a segment of interest, but `w_ipa` includes a few convenient properties that return certain segments. In particular, `w.current` provides the following:

```
maxweight
minweight
successful_trajectories
```

The `maxweight` and `minweight` properties return objects which contain data about the segments that carry the highest and lowest weights in the current iteration, respectively. The `successful_trajectories` property returns the IDs of the segments that successfully transitioned between states (the states are defined in your `west.cfg`). Calling these functions on an iteration object yields all datasets pertaining to the segment with the desired property. In this WE simulation, each trajectory contains 101 timepoints. Therefore, the `maxweight` segment (seg_id 177) in iteration 49 has (101,2) pcoord values, 101 auxdata values, and it can switch bins and states 101 times. You can see this by running `w.current.maxweight`.

The auxdata dataset is unique in that the simulations can contain any number of auxiliary datasets with any unique name. Here, they are returned as a dictionary where the key is the dataset name defined in `west.cfg` and the value is a NumPy array containing the actual dataset.

Segment 177 above was in state 1 during the entire iteration. But what is state 1? It is defined in `west.cfg`, but we do not have to go back to `west.cfg` to look it up. Simply run:

```
$ w.state_labels
```

It is also in bin 0 the entire time (note that these are the bins defined in `west.cfg` for this analysis scheme and not the bins used in the simulation). What is the pcoord value of that bin? Run:

```
$ w.bin_labels
```

To track the immediate parent and children of a segment, we can use `w.past` and `w.future`. These iteration objects are similar to `w.current`, but keyed to give information about the segments in `w.current`. For instance, to look up the weight of segment 177's parent, run the following:

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

```
$ w.past[177].weights
```

Likewise, to see whether the same segment had any children, run:

```
$ w.future[177]
```

Segments always have a past, but do not always have a future. They may also produce multiple children, so the values returned by `w.future[seg_id]` are usually more complicated. Rather than being given the datasets directly, `w.future` returns a list of the datasets.

To determine the properties of a complete trajectory (that is, the string of segments going back to the first iteration), `w_ipa` includes a fast trace function. To trace segment 177 in iteration 39 (current iteration), run the following:

```
$ s = w.trace(177)
```

It returns an object similar to `w.current[177]`, except that it also contains all historical information. The `auxdata`, `bins`, `pcoord`, and `states` datasets are all going to be very large; their shape should be the product of the number of time-points per iteration and the trajectory length. As we are at iteration 39, and have 101 time points per τ value, we should have 3939 values in each dataset!

Plotting. Rather than visually inspecting each value, let us just plot it. Run the following:

```
$ clear
$ s.weights.plot()
$ clear
$ s.pcoord.plot()
$ clear
```

Many datasets, such as `weight`, default to a logscale; others, such as `pcoord`, use a linear scale. By default, the 0th dimension of `pcoord` is plotted. When the plotting function is called via the CLI, a rough estimate of how the trajectory's pcoord has evolved is plotted in the terminal.

The `w.current` iteration object contains information about the rate constants that were calculated in the active analysis scheme. To view an array containing the rate constants along with the upper and lower confidence intervals, run the following (do not forget about tab completion):

```
$ w.current.direct.rate_evolution
```

OR

```
$ w.current.rate_evolution.direct
```

To view a plot of their evolution, run the following:

```
$ w.current.direct.rate_evolution.plot()
```

The `w_ipa` tool displays the upper and lower confidence intervals on the plot as well.

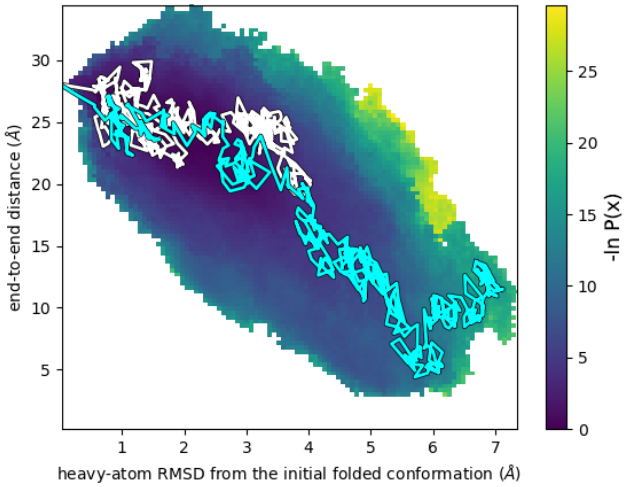

Figure 12. Two-dimensional probability distribution as a function of the progress coordinate. Two representative, continuous trajectories that originate from distinct initial states are traced in cyan and white, respectively.

7.4.3 Visualization of WE Datasets

In addition to generating probability distributions as a function of the progress coordinate (or other observables of interest), it can be helpful to examine movies of how the distributions evolve with time. Such movies can be used to determine the optimal number of trajectories per bin in a particular region of the progress coordinate by tracking how the probability distribution evolves with the number of trajectories that region.

Learning Objectives. The specific learning objectives of this tutorial are:

1. Create a movie of how a probability distribution evolves with time.
2. Trace representative trajectories over this probability distribution.

Here, we will create a movie of how a two-dimensional probability distribution (**Figure 12**) evolves with time. This movie-making feature is currently carried out using a bash script (`pdist_evol.sh`) and will eventually be added to the WESTPA `plothist` tool.

The bash script involves the following three steps: (1) run the `w_pdist` tool on the `west.h5` file to generate probability distributions in a specified folder that will also contain the eventual movie of how the distributions evolve with time, (2) generate a plot of a two-dimensional probability distribution for each iteration as a cumulative moving average from iteration 1 to 40 and (3) create the movie from the 40 generated frames of the probability distributions. The most important part of this script is the `--postprocess-function` option of `plothist` that is defined in `postprocess.py`. This function re-

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

quires a basic knowledge of Python and `matplotlib`, and can be used to modify features of the plot (e.g. adjustment of axis labels, tick marks, titles, and lines) via the `matplotlib` interface. In addition, external files from various analyses can be uploaded and overlaid on the plot as demonstrated in this example.

Here, we will select two trajectories from the last WE iteration and overlay their pathways on the probability distribution of the overall simulation as a function of progress coordinate. First, we will use the `trace_walker` function to determine the segment number of the selected trajectories in each WE iteration going all the way back to the corresponding conformation of the initial state ensemble. This process of tracing can also be accomplished by using WESTPA tools `w_ipa` and `w_trace`. After the segment numbers are obtained, the `get_pcoords` function loads in 10 progress coordinate values per iteration for the trajectories. Finally, a movie-making tool (here, we use `mencoder`) creates a movie from the 40 frames of probability distributions.

7.5 Advanced Tutorial: Creating "Binless" Resampling Schemes: Na^+/Cl^- Association Simulations

7.5.1 Introduction

The non-linearity of certain progress coordinates (e.g., those identified by machine learning tools) requires the creation of "binless" rather than binned resampling schemes for rare-event sampling (**Figure 13**). In addition, binless schemes can be useful in grouping trajectories by a feature of interest for resampling. For example, trajectories could be grouped by history (sharing the same parent structure) to improve the diversity of trajectories that successfully reach the target state, or by a simple k-means clustering. This tutorial builds upon the **Basic Tutorial** (Na^+/Cl^- association simulations, **Section 7.1**) by introducing users to running and analyzing a WESTPA simulation that employs "binless" resampling schemes. This tutorial is a prerequisite for **Advanced Tutorials 7.6-7.8**.

Learning Objectives. This tutorial introduces users to the generalized resampler module in the WESTPA 2.0 software package that allows for the creation of either binned or binless resampling schemes.

The tutorial also instructs users on how to initiate a WE simulation from multiple representative conformations of the starting state and how to apply two key post-simulation analysis tools. Specific learning objectives are:

1. How to create a binless scheme for splitting and merging trajectories based on k-means clustering using the `BinlessMapper` resampler module;
2. How to initiate a WE simulation from multiple starting conformations;
3. How to combine multiple WE simulations for analysis using the `w_multi_west` tool;
4. How to perform post-simulation analysis using the `w_crawl` tool.

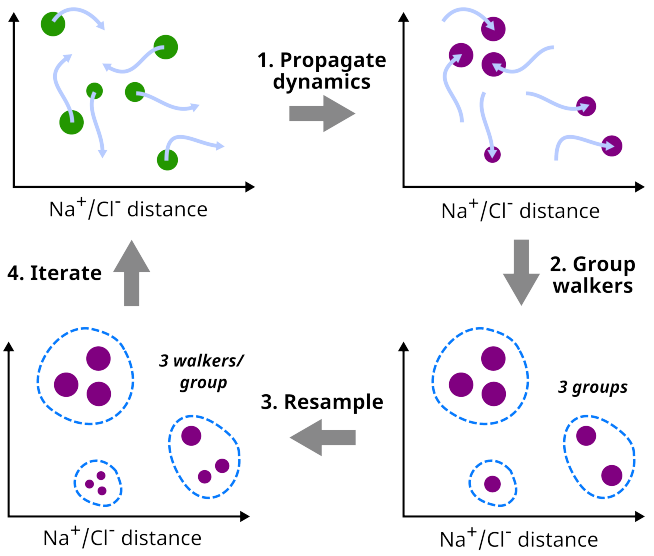

Figure 13. Flow chart for simulating Na+/Cl- association using the new binless resampler module. After dynamics propagation in step 1, trajectories in each bin are grouped with the user-specified group.py function. In this example, trajectory walkers are grouped using k-means clustering.

7.5.2 Prerequisites

Computational Requirements. This simulation can be completed in 5 hrs using 8 Intel Xeon W3550 3.07 GHz CPU cores, generating 1 GB of data using the HDF5 framework of the WESTPA 2.0 software package. Two independent `west.h5` files, each containing 100 WE iterations of simulation data, are provided for the analysis portions of this tutorial in the `for_analysis/` directory. This tutorial uses the OpenMM 7.6 software package for dynamics propagation (https://openmm.org) and the MDTraj 1.9.5 analysis suite (https://www.mdtraj.org/1.9.5/index.html) for calculations of the WE progress coordinate. The scikit-learn 1.1.0 package (https://scikit-learn.org) is used to identify the "binless" groups.

Jupyter Notebook. Sample code for running and analyzing a WESTPA simulation according to "best practices" is made available in a Jupyter notebook. For the visualization portions of the notebook, nglview, matplotlib, and ipympl are required.

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

Quick Start for this Tutorial. Users can run the following command within the `tutorial7.5-hdf5/` directory to install all the software dependencies for this tutorial to an existing conda environment:

```
$ conda env update --name <your WESTPA conda env>
                                --file environment.yml
```

7.5.3 Setting up the WE Simulation

This simulation uses the same WE parameters (τ, number of trajectories per bin, etc.) as the **Basic Tutorial** (Na^+/Cl^- association simulations, **Section 7.1**). The following are major differences from the **Basic Tutorial** that highlight more advanced features of WESTPA simulation setup.

Binless Resampler Module. The binless resampler module can be accessed from the `west.cfg` file in the `system_options` section where binned and binless schemes are all defined. If the `BinlessMapper` is used by itself, the entirety of configurational space will be binless. To recycle trajectories while using a binless framework, we will need to place a `BinlessMapper` inside of a `RecursiveBinMapper` bin and demarcate the target state in a separate `RecursiveBinMapper` bin. This framework is identical to the one used for the MAB scheme with recycling.

The `BinlessMapper` takes three mandatory arguments. The first, `ngroups`, specifies the total number of groups to assign trajectory walkers in the binless space. The second, `ndims`, specifies the dimensionality of the progress coordinate and is limited to either 1 or 2 at this time. The `ndims` parameter specifies the dimensionality of clustering, e.g., 2 for generating clusters in two-dimensional space (each dimension will not be grouped separately as is typically the case for binned resampling schemes). The clustering of trajectories enables sampling of high-dimensional space without an exponential increase in the number of walkers. The final argument is `group_function`, which specifies the function for grouping trajectories in an external file (here, `group.py`) and will take as input the progress coordinates and the `ngroups` values. We provide a general example of using this function for a one-dimensional progress coordinate using k-means clustering. Additional keyword options can be specified under `group_arguments` in the `west.cfg` file. An example of using a recursive binless scheme is shown in the `west.cfg` snippet below:

```
west:
  system:
    system_options:
      bins:
        type: RecursiveBinMapper
        base:
          type: RectilinearBinMapper
          boundaries:  # Under base:
            - [0, 2.60, 'inf']
          mappers:
          - type: BinlessMapper
            ngroups: [5] # Number of groups
            ndims: [1] # Number of grouping
                       # dimensions
            group_function: group.kmeans
            at: [5]  # Location of binless mapper
                     # relative to base mapper
```

Initiating a WE Simulation from Multiple Structures. Ideally, a WE simulation is initiated from multiple pre-equilibrated structures that are representative of the initial state for the rare-event process of interest, e.g. using a conventional simulation or a separate WE simulation of the initial state. Within the WESTPA framework, we refer to these structures as "basis states". If the simulation is run under non-equilibrium steady-state conditions, trajectories that reach the target state are “recycled” by terminating that trajectory and initiating a new trajectory with the same statistical weight from one of the basis states. Structural files (XML files in this tutorial) for basis states contain the coordinates and velocities, and are placed in separate, numbered folders within the `bstates/` directory. Accompanying each XML file is a `pcoord.init` text file which contains the progress coordinate value of that basis state. These progress coordinates are saved to the HDF5 file during the initialization process. The `bstates/` directory also contains a reference file (`bstates.txt`) that lists all of the available basis states. The `bstates.txt` file is formatted with three columns, corresponding to the basis states’ names, associated probabilities, and folder name, respectively. Additional basis states can be added as separate, additional lines at the end of the `bstates.txt` file. The probability over all basis states must sum to one, and will be normalized by WESTPA during the initialization process to sum to one if the condition is not met. Compared to the **Basic tutorial 7.1** involving Na^+/Cl^- association [38], the `get_pcoord.sh` and `runseg.sh` files are also modified such that `$WEST_DATA_STRUCT_REF` now corresponds to the directory for each basis state and not the xml file itself.

7.5.4 Running the WE Simulation

As with previous versions of WESTPA, the simulation can be initialized using `./init.sh` and run using `./run.sh`. Alternatively, both steps could be executed consecutively using the new Python API by running the following command:

```
$ python init_and_run.py
```

The `init_and_run.py` script will print out simulation updates to the console in real-time. An example `runwe.slurm`

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

file with commands for both methods of execution is provided for use with SLURM-like workload managers. We also provide a Jupyter notebook that demonstrates the steps for cleaning up, initializing, and running the WE simulation.

Note that this tutorial is using the new HDF5 trajectory storage framework, which will be explained further in **Advanced Tutorial 7.6**. To enable the use of the HDF5 framework in your own simulation, you may use the current tutorials directory as an example. The location of the trajectory h5 files will need to be specified in the `west.cfg` file, and the appropriate restart and topology files will need to be copied to the locations specified in the `get_pcoord.sh` and `runseg.sh` files. You will also need to make sure that the file extensions for any trajectory files are readable by `mdtraj.load()`, (e.g., Amber restart files must end in .ncrst) which simply requires renaming. To save disk space, trajectory files outputted by the dynamics engine can be deleted after every iteration in the `post_iter.sh` file, which is located in the `westpa_scripts/` directory.

7.5.5 Monitoring and Analyzing the WE Simulation

Combining Multiple WE Simulations for Analysis. To combine multiple WE simulations into a single aggregate simulation file for analysis, we can use the `w_multi_west` tool, which creates a single `multi.h5` file that contains the data from all of the `west.h5` files of each WE simulation. Each WE iteration in the `multi.h5` file contains all of the trajectory segments from the corresponding iterations of the individual WE simulations, all normalized to the total weight for that iteration. For backwards compatibility, a version of `w_multi_west` for use with previously run WESTPA 1.0 simulations (v2020.XX) has been available since version 2020.04.

To apply the multitool to a combination of `west.h5` files, place the `west.h5` file for each simulation in a numbered directory starting with `01/`. If all of the simulations used a custom grouping function (such as in `group.py`), you must also include that file in the top-level analysis directory.
The files will be organized as follows:

```
01/
  west.h5
02/
  west.h5
group.py [if used in simulation]
```

Next, in this directory, run the following to merge the `west.h5` files:

```
$ w_multi_west -m . -n 2
```

The `-m` flag specifies the path to your directories and the `-n` flag specifies the number of WE simulations to combine for analysis. To combine auxiliary datasets, one can add either an `--aux=NAME_OF_DATASET` flag for a specific dataset or an `--auxall` flag for all auxiliary datasets; note that the inclusion of auxiliary datasets will substantially extend the time needed to combine the simulation data. The above `w_multi_west --auxall` command will generate a list of WE simulation datasets to combine based on the datasets listed in `01/west.h5` and generate a `multi.h5` file with the combined simulation datasets. The `--ibstates` flag will merge the initial and basis states if the `bstates` dataset is identical across all the simulations. You may want to rename this file to `west.h5` in order to apply the `w_pdist` tool to the combined simulation dataset. Note that the `w_multi_west` tool will only merge up to the *N-1* WE iteration, ignoring the last WE iteration. The resulting `multi.h5` file will not link to the individual iteration HDF5 files generated using the HDF5 framework.

Post-Simulation Analysis. As mentioned above, WESTPA 2.0 enables efficient post-simulation analysis of trajectory data by storing trajectory data in highly compressed HDF5 files. The `w_crawl` tool can then be used to "crawl" through the trajectory data in single HDF5 files per WE iteration rather than millions of trajectory files. Results from the analysis are written to a dataset in a new HDF5 file. Before crawling through an entire simulation dataset, we recommend that users first test their analysis scheme in the `wcrawl_functions.py` file to ensure that the scheme works as expected. For this tutorial, we will only be using a single CPU core for these `w_crawl` calculations but also include a sample script as an example of how to use `w_crawl` on multiple CPU cores in parallel. The `wcrawl_functions.py` file contains the main analysis code. This script first identifies the final frame of a segment's parent trajectory file from the previous WE iteration and makes sure this is eventually combined with the trajectory segment from the current WE iteration. The inclusion of the parent structure at the beginning of the current iteration trajectory is necessary for using the crawled dataset with WESTPA's kinetics analysis tools. In this example, trajectory coordinates of only Na^+ and Cl^- are extracted using the MDTraj analysis suite and multiplied by 10 to convert from nm to Å. Resulting per-iteration coordinate values are then saved to an array, which is subsequently saved to a `coord.h5` file. The `coord.h5` file is formatted similarly to a `west.h5` file, where the new per-iteration values are stored under `iterations/iter_{n_iter:08d}/coord`. To ease analysis, a `copy_h5_dataset.py` script is provided to copy `coord.h5`'s contents into a west.h5 file as an auxiliary dataset. Note that if you store a WESTPA simulation's trajectory HDF5 files in a separate directory from what is used in this tutorial, you need to specify the directory where the `iter_XXXXXX.h5` files are located in the `wcrawl_functions.py` file.

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

The `run_w_crawl.sh` shell script runs the `w_crawl` tool at the command line and provides options for running the tool in serial or parallel modes. In this tutorial example, we will run the `w_crawl` tool in the serial mode using the `--serial` flag, analyzing one WE iteration at a time on a single CPU core. While the serial mode is sufficient for "crawling" relatively small datasets, the parallel mode using the `--parallel` flag is desirable for datasets with over 100 trajectory segments per WE iteration and/or hundreds of WE iterations. In the parallel mode, each CPU core of a single compute node analyzes a different WE iteration at the same time. To run the `w_crawl` tool across multiple nodes, one can use the ZMQ work manager. Once satisfied with the `wcrawl_functions.py` and `run_w_crawl.sh` files, run the `w_crawl` tool locally:

```
$ ./run_w_crawl.sh
```

or on a multi-node cluster using the Slurm workload manager:

```
$ sbatch run_w_crawl.sh
```

To monitor the progress of the analysis, we examine the `w_crawl.log` file, which contains analysis results for each WE iteration and each trajectory segment. Finally, to copy the `coord.h5` file to the `west.h5` file, run the `copy_h5_dataset.py` script.

7.5.6 Conclusion

After completing this tutorial, users will gain an understanding of how to configure the upgraded resampler module for using a binless scheme, initiate a WE simulation from multiple structures using the new HDF5 trajectory storage framework, and apply the `w_multi_west` and `w_crawl` post-simulation analysis tools.

7.6 Advanced Tutorial: Simulations of Membrane Permeation by 1-Butanol

7.6.1 Introduction

The ability of a drug-like molecule to cross (or permeate) a lipid bilayer has been of great interest to drug discovery [52], but is challenging to simulate due to the long timescales involved. In this tutorial, we will use WESTPA 2.0 to simulate pathways for membrane permeation by a small molecule (1-butanol) and calculate the permeability coefficient. Our WE protocol employs and explains two new features in WESTPA 2.0 [2]: (i) the Minimal Adaptive Binning (MAB) scheme [19], and (ii) the HDF5 framework for efficient restarting, storage, and analysis of a WE simulation.

Learning Objectives. This tutorial demonstrates how steady state WE simulations can be used to generate pathways and permeability coefficients for membrane permeation by a small molecule. Specific learning objectives include:

1. How to set up a double membrane bilayer system for permeability studies;
2. How to use the highly scalable HDF5 framework for more efficient restarting, storage, and analysis of simulations;
3. How to apply the minimal adaptive binning (MAB) scheme.

7.6.2 Prerequisites

In addition to completing the Basic and Intermediate WESTPA Tutorials [38], a prerequisite to this advanced tutorial is completion of the above **Advanced Tutorial 7.6**. Also required is a working knowledge of the CHARMM-GUI membrane builder, PACKMOL, OpenEye Scientific's OEChem and Omega toolkits (for system preparation only), MDTraj analysis suite, and the OpenMM 7.6 dynamics engine (for running WE).

Computational Requirements. The membrane permeability tutorial simulation runs best using, at minimum, a dual-GPU workstation. For this tutorial, simulations were tested with a compute node containing both a NVIDIA Titan X (Pascal) GPU and a NVIDIA GTX 1080 GPU, as well as a 16-core Intel Xeon X5550 CPU running at 2.67 GHz with a total of 100 GB of system memory. In the case a user does not have a GPU and only CPUs, switch between OpenMM's GPU and CPU platforms by changing the platform name in line 22 of `memb_prod.py` to `CPU` instead of `CUDA`. The complete tutorial simulation run length (37 iterations) required ~4 days of continuous wall clock time on both GPUs, as well as ~30 GB of hard disk space with the HDF5 framework and MAB options turned on.

This tutorial uses the OpenMM 7.6 dynamics engine [33] and MDTraj 1.9.5 analysis suite (https://www.mdtraj.org/1.9.5/index.html) for progress coordinate calculations. Force fields used in this tutorial can be installed via openmmforcefields (https://github.com/openmm/openmmforcefields). System setup and equilibration were performed separately using OpenMM. In order to run the companion Jupyter notebook, nglview, matplotlib are required for visualization purposes. Other dependencies, including NumPy and MDTraj, are installed with WESTPA 2.0 itself.

Quick Start for this Tutorial. Users can run the following command within the `tutorial7.6-membrane/` directory to install all the software dependencies to an existing conda environment:

```
$ conda env update --name <your WESTPA conda env>
                   --file environment.yml
```


https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

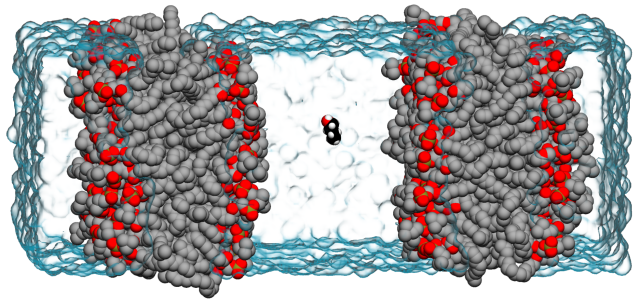

Figure 14. The equilibrated double-membrane bilayer system used for initiating WE simulations of membrane permeation by 1-butanol in this tutorial. Both the POPC membrane bilayers (gray; hydrogens removed for clarity) and 1-butanol (black) are represented in van der Waals representation, while layers of explicit water molecules are shown as a transparent blue surface.

7.6.3 Preparing the simulation

The following preparation steps have already been completed and are presented to instruct the reader on how to prepare similar systems for WE simulation.

Our system consists of a 1-palmitoyl-2-oleoyl-sn-glycero-3-phosphatidylcholine (POPC) membrane bilayer. The 1-butanol-double POPC membrane bilayer system was prepared by piecing together several smaller molecular systems in the following way. First, a single POPC membrane bilayer was generated using CHARMM-GUI's membrane builder with 50 lipids per leaflet and zero salt concentration. This membrane was then equilibrated using a single GPU with the OpenMM dynamics engine using the standard CHARMM-GUI procedure. The membrane plus the outer aqueous layer to the membrane, once combined, (see **Figure 14**) was equilibrated for an additional 500 ns. A 2D representation of 1-butanol was generated from an input SMILES string (CCCCO) using OEChem, and converted to a 3D structure using the Omega TK toolkit. The 3D structure of 1-butanol was then solvated with a 2 nm slab of water molecules at a density of 1 gm/cm^3 using PACKMOL along with the OEChem TK and Omega TK toolkits from OpenEye. Finally, the full double-membrane bilayer system was assembled by placing the butanol-embedded slab of water at the origin, with a single-membrane system at each z-edge of the water slab. The resulting system was then subjected to energy minimization and equilibrated before the initiated WE simulations of butanol permeating the membrane bilayer were initialized.

The System. In this tutorial, we will run a WE simulation of 1-butanol crossing one membrane of a double POPC membrane bilayer system. To run the WESTPA 2.0 simulation, the AMBER LIPID17 force field is applied to all POPC lipids, explicit water molecules are represented by the TIP3P model, while the parameters for 1-butanol were taken from the GAFF 2.11 force field. All force field parameters were applied using the openmmforcefields Python package.

Progress Coordinate. The progress coordinate (z) is defined as the (signed) distance from the center of mass (COM) of the butanol molecule to that of the closest membrane. The width of a single leaflet of the membrane is roughly 2 nm, so a z < -2 nm, between -2 nm and 2 nm, or >2 nm indicates that the butanol molecule has not yet crossed, is currently crossing, or has crossed the membrane, respectively. A target state of z $\geq$3.5 nm is used to recycle trajectory walkers. The actual computation is performed by measuring the signed distances between the COMs of butanol and each of the two membranes, z_1 and z_2, using the MDTraj analysis suite and then taking the larger value of the two, z=max(z_1, z_2).

Preparing the Simulation Environment. Once we have constructed and equilibrated the 1-butanol membrane system, we will prepare the WESTPA system environment. First, we will analyze the equilibrated double membrane bilayer system to define an initial progress coordinate. The progress coordinate, equilibrated coordinate file (e.g. XML file), and `bstates.txt` file describing the initial basis states are placed in the `bstates/` directory. Second, we will edit the `west.cfg` file with options for using the MAB scheme and HDF5 framework. To initialize the WESTPA 2.0 environment, we will run `./init.sh`. This command will source the WESTPA 2.0 environment, construct the `seg_logs/`, `traj_segs/`, and `istates/` directories, and will run the `w_init` command with the correct settings for the target state (z = 3.5 nm) and the basis states constructed above.

Adaptive Binning using the MAB Scheme. By default, this tutorial uses a manual, fixed binning scheme, but can be modified to use the Minimal Adaptive Binning (MAB) module, which adaptively positions bins along the progress coordinate. To enable this adaptive binning scheme, uncomment the MAB-related lines in the `west.cfg` file, which specify the `MABBinMapper` as the primary bin mapper type, and comment the lines related to the inner `RectilinearBinMapper`. Next, define `n_bins` (the number of MAB bins placed per progress coordinate dimension) in the same section of the `west.cfg` (e.g., if a two-dimensional progress coordinate is being used, `[20, 20]` indicates 20 bins in each dimension). It is important to note that if the recycling of trajectories at a target state is desired within the MAB framework, recursive bins must be specified by adding a `MABBinMapper` inside of a `RecursiveBinMapper` outer bin and defining the target state in terms of the recursive outer bins. An example of a MAB recursive binning scheme is shown below:

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

```
west:
  system:
    system_options:
      bins:
        type: RecursiveBinMapper
        base:
          type: RectilinearBinMapper
          boundaries:
            - [-inf, -44, 34, inf]
          mappers:
          - type: MABBinMapper
            nbins: [20] # Number of bins
            direction: [0] # Split both directions
            skip: [0] # Bin along this dimension
            at: [0]  # Location of MAB mapper
                          relative to base mapper
```

In the above example, the `at` option in the last line specifies which outer bin to place the MAB scheme inside of range `[-44, 34]`. For a two-dimensional progress coordinate, this option will require a list with two values, one for each dimension. The `nbins` option specifies the number of MAB linear bins that will be used inside the bin, plus two more bins for extrema and bottleneck trajectories, respectively. Optional `direction`, `skip` and `mab_log` parameters can also be specified for the MAB scheme. The `direction` parameter (0, -1, or 1) can be used to specify the direction along the progress coordinate for splitting of trajectories, where 0 indicates both directions, -1 indicates the direction of decreasing values along the progress coordinate, and 1 indicates the direction of increasing values along the progress coordinate. The `skip` parameter (1 or 0) designates whether a particular dimension along the progress coordinate will be binned during the simulation, but will be used to define the target state (1 indicates that the dimension will be skipped for binning and 0 indicates that the dimension will not be skipped for binning). The `mab_log` parameter, when enabled with `true`, will print MAB-related statistics such as the progress coordinate values of extrema walkers to the `west.log` file. Multiple MAB schemes can be added to a recursive binning setup, but only one MAB scheme may be used per each outer bin.

If users choose to combine the application of the MAB scheme with the weighted ensemble steady-state (WESS) plugin [22], which reweights trajectories towards a non-equilibrium steady state, they must provide fixed bins for the reweighting procedure. The positions of these fixed bins can be specified in the WESS plugin section of the `west.cfg` file:

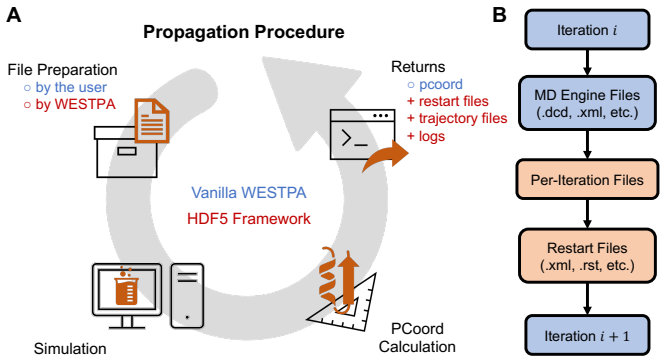

Figure 15. Diagrams showing the differences in the propagation procedure between a vanilla WESTPA run and a run with the HDF5 framework. A) Procedures for propagating one WE iteration using the original WESTPA 1.0 framework (blue text) and WESTPA 2.0 HDF5 framework (red text). Using the WESTPA 2.0 HDF5 framework, the WESTPA software prepares the input files while the user is responsible for returning the progress coordinate (pcoord), restart, trajectory, and optional log files. B) Workflow for using the WESTPA 2.0 HDF5 framework. Blue: Steps and files from the original WESTPA 1.0 procedure. Red: Files generated or prepared by the WESTPA 2.0 HDF5 framework.

```
plugins:
  - plugin: westpa.westext.wess.WESSDriver
    enabled: true
    do_reweighting: true
    window_size: 0.75
    bins:
      type: RectilinearBinMapper
      boundaries:
        - ['-inf', 0.5, 1.0, 1.5, 2.0, 2.5, 'inf']
```

HDF5 Framework. The setup for a WESTPA simulation with the HDF5 framework is similar to a vanilla one with the addition of the following procedures, which are a more detailed list of the same steps discussed briefly in **Advanced Tutorial** 7.5 above:

1. An `iteration` entry was provided in `west.cfg` under `west.data.data_refs` to specify where and how the per-iteration HDF5 files should be saved and named.
2. All the necessary files needed for propagating the next segment, such as state/restart and topology files, are passed on to WESTPA through the environment variable, `$WEST_RESTART_RETURN`, after initialization and propagation of each iteration (**Figure 15A**). This information is typically placed in the `get_pcoord.sh` and `runseg.sh` files.
3. All the trajectory files, and topology files if the topology is not stored as part of the trajectory file, are provided to WESTPA through the environment variable,

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

`$WEST_TRAJECTORY_RETURN`, after the propagation of each iteration. This, again, is typically placed in the `get_pcoord.sh` and `runseg.sh` files. The coordinates of the basis states can be provided through the environment variable during initialization to be stored as the "trajectories" of the zero-th iteration. Note that the procedures described in step 2 and 3 are similar to how the progress coordinates are returned through `$WEST_PCOORD_RETURN` in the vanilla WESTPA simulation. The trajectory and restart files will be saved as part of the per-iteration HDF5 files. In turn, these files do not need to be located and copied over to the directory for propagating the next segment, and they will be automatically extracted and put into the segment folder by WESTPA instead (Figure 15**B**).

These additional procedures simplify the data management on the user's end for two reasons. First, all the trajectories are stored in a standard way which enables fast and easy access to these trajectories with their associated WE-related data using the newly added analysis module (see Section 7.5.4). Second, the restart/state files are much easier to locate as when they are generated than when they are needed in the next iteration, so letting the users pass trajectory and restart files to WESTPA for it to manage frees users from tracking those files themselves which would require the critical knowledge of how two WESTPA-assigned environment variables work (namely, `$WEST_CURRENT_SEG_INITPOINT_TYPE` and `$WEST_PARENT_DATA_REF`).

For this membrane permeation example, the per-iteration HDF5 files are saved in a folder named `traj_segs/` and named following the pattern `iter_XXXXXX.h5`. The basis states are returned to WESTPA in `get_pcoord.sh` as both the "trajectories" of the zero-th iteration and the restart files for propagating the first iteration and recycled walkers. The dynamics is propagated using OpenMM for 100 ps for each iteration, and the output trajectory files and state XML files are returned to WESTPA in `runseg.sh`. These files are deleted once they are returned in order to save disk space. See the sample project setup files for detail.

7.6.4 Running the WE Simulation

Similar to other examples, the simulation can be run using `./run.sh` from the top-level permeability tutorial directory.

7.6.5 Analyzing the WE Simulation

The analysis of the membrane permeation simulation can be found in the accompanying Jupyter notebook titled Membrane Permeability Tutorial (Analysis). In this tutorial notebook, we demonstrate how to extract a complete, continuous pathway of a membrane-crossing event and calculate the incoming flux to the target state from the WE membrane permeation simulation. Note that this tutorial assumes that you already have a completed simulation using the HDF5 framework with at least one crossing event (~40 iterations).

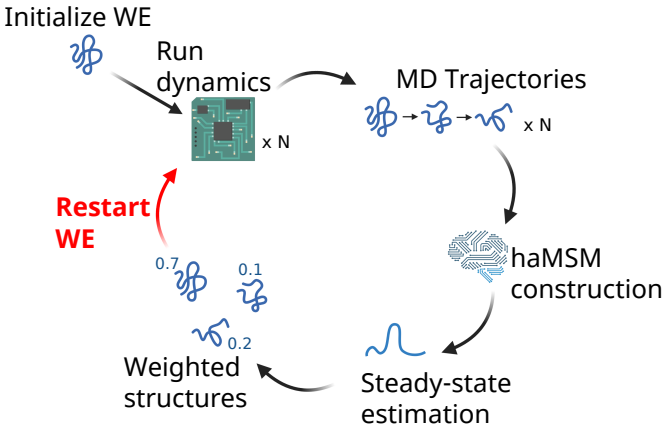

Figure 16. Schematic of haMSM restarting procedure. Trajectories from one or more WE runs are used together to build an haMSM. An estimate of steady-state is obtained from the haMSM, and is used to assign weights to all sampled structures. New WE runs are initiated from these steady-state weighted structures, and the procedure repeats. Figure reprinted with permission from [2]. Further permission related to the source material should be requested from ACS.

7.6.6 Conclusion

In this tutorial, we have illustrated the relative ease in which one may use the WESTPA 2.0 software package to perform advanced WE path sampling simulations of membrane permeation for a small molecule (butanol). Using a single workstation with two GPUs, our WE simulation can generate membrane permeation events within a few days of wall-clock time. WE simulations, when using the WESTPA 2.0 HDF5 framework and MAB binning scheme, are relatively cost effective, both in terms of total computing time and disk storage.

7.7 Advanced Tutorial: Analysis and restarting with haMSMs: NTL9 Protein Folding

7.7.1 Introduction

Although the WE strategy provides an efficient framework for unbiased rare-event sampling, slow relaxation to steady state and impractically large variance in rate constant estimates may still be limiting factors for complex systems. History-augmented Markov state models (haMSMs) have been demonstrated to provide estimates of steady state from transient, relaxation-phase WE data, which can be used to start new WE simulations [18]. As shown in **Figure 16**, the haMSM plugin for the WESTPA 2.0 software package automatically constructs an haMSM from one or

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

more independent WE simulations to estimate steady-state observables and then can automatically initiate new simulations from those estimates and iteratively repeat this procedure when those simulations complete. The underlying haMSM analysis library, `msm_we`, can also be used to perform stand-alone haMSM analysis of existing WESTPA data.

Learning Objectives. This tutorial demonstrates the use of an haMSM restarting workflow in WE simulations of the ms-timescale folding process of the NTL9 protein. Specific objectives are:

1. How to apply the haMSM plugin for periodic restarting of simulations;
2. How to use the `msm_we` package to build an haMSM from WE data;
3. How to estimate the distribution of first passage times from the haMSM, using `msm_we`.

7.7.2 Prerequisites

The Basic and Intermediate WESTPA Tutorials [38] should be completed before running this tutorial.

Computational Requirements. This tutorial can be completed on a computer with a single NVIDIA GTX 1080 GPU and a 2.4GHz Intel Xeon E5-2620 in 90 min. The WE simulation will generate ~5 GB of data, though on a typical cluster filesystem, overhead associated with data redundancy may increase this to ~15 GB. A version of the Amber software package compatible with Amber 16 restart and topology files must be installed to propagate the dynamics and calculate the WE progress coordinate.

The `msm_we` Python package must also be installed, which can be done by first cloning the repository and then installing it into your existing conda environment (with WESTPA already installed) by running:

```
$ git clone https://github.com/westpa/msm_we
$ cd msm_we
$ conda env update --name <your WESTPA conda env>
        --file environment.yml
```

7.7.3 Plugin functionality

Once we initiate a WESTPA run with the haMSM plugin enabled, the plugin will execute a series of independent WE simulations (runs) from the same starting configuration for the number of WE iterations specified in `west.cfg`. For this tutorial, the runs will not use the HDF5 trajectory-saving framework.

If none of the WE runs have reached the target state, the haMSM plugin will sequentially extend each run for a number of iterations specified in `west.cfg`. This extension procedure will be repeated until at least one run has reached the target

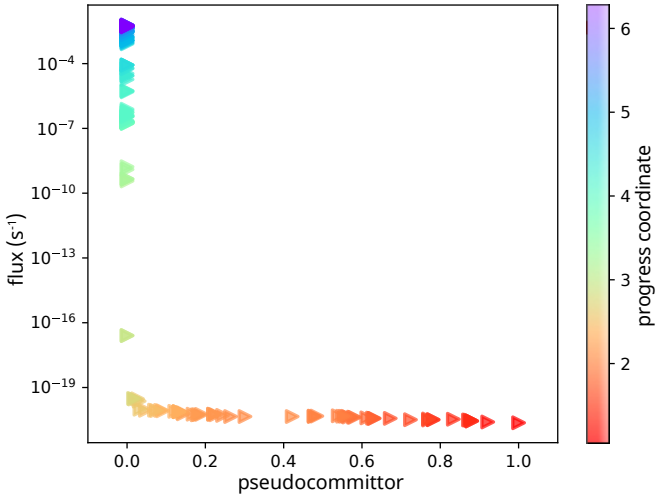

Figure 17. Sample flux profile generated by haMSM restarting plugin after a restart. The x-axis is labeled "pseudocommittor", since these are committor values generated from a unidirectional path ensemble. This plot should flatten in successive restarts until steady state is reached. The color scale indicates the progress coordinate associated with each pseudocommittor—notably, most of the dynamic range in committor-space is restricted to a small range of progress coordinate values. This image can be found in restart0/plots/psuedocomm-flux_plot_pcoordcolor.pdf after restart is performed.

state. For consistency, all of the other runs in the set will be extended to match the length of this run. As a result of this extension procedure, runs used for the first restart may be longer than runs in subsequent restarts.

After completing the extension procedure, the plugin will construct an haMSM from these runs, and estimate the steady-state distribution and flux into the target state. All structures sampled by the set of runs are used to build this haMSM, and are then weighted according to a steady state. Note that this haMSM uses only the first and last frame of each WE iteration, which effectively sets the lag-time equal to the WE resampling time. A number of plots are automatically generated from the model. The flux profile, shown in **Figure 17**, provides an important metric of convergence, and should be examined carefully. A flatter flux profile indicates more converged weighted ensemble.

A new set of runs from the resulting weighted structured are initialized. These runs are correlated but independent from this point onward. As a technical note, when initializing the new WE simulations, these structures are used as "start states". Within the WESTPA 2.0 framework, start states are a third category of state, in addition to basis states and target states. Like basis states, start states are used for seeding trajectory walkers when initializing a simulation with `w_init`; however, *unlike* basis states, start states are not used after

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

this point and walkers reaching the target state will **not** be recycled into start states but rather only to the basis states. The new WE runs are executed for the number of WE iterations specified in `west.cfg`, in series. At this point, the model is saved, a new restart is prepared, and the process repeats from that point onward.

Please see [2, 18] for more theoretical background on the models used by this plugin.

7.7.4 Preparing the system

The system. For our simulation of the NTL9 protein folding process, we use a stochastic Langevin thermostat with low-friction (collision frequency γ = 5 ps^{-1}) and a generalized Born implicit solvent model. The system consists of ~600 atoms. We will use the haMSM restart plugin to automatically perform three independent WESTPA runs serially before constructing an haMSM. A single restart will be performed from the haMSM steady state estimate, and then WE simulation will be continued for another 106 WE iterations for each of the three runs.

To reduce runtime for this tutorial, we provide a partially completed set of three independent WE simulations. In this set of simulations, the first two runs have completed, and the third is nearing completion. None of these simulations have yet reached the target state.

After continuing this set of WE simulations, the third simulation will finish and reach its maximum number of WE iterations. With very high probability, none will have reached the target folded state, and pre-restart extensions will be triggered. The initialized runs were chosen such that it is unlikely that the third run will reach the target state before the next restart. However, it is possible that in the remaining few WE iterations of the third simulation, this simulation will reach the target state, in which case we will skip the extension procedure.

After a single round of extensions, the target state should be reached in run 2, though not necessarily runs 1 or 3. We will construct the haMSM from the three extended runs, and restart a new set of three runs from the steady-state estimates.

Structure of Plugin-Specific Files. A list of important files used and generated in `$WEST_SIM_ROOT` by the haMSM plugin is shown in **Table 3**.

The following files contain more adjustable parameters or are more tightly integrated in the workflow and therefore warrant a more in-depth explanation.

`west.cfg`: The haMSM restarting plugin requires a number of parameters to be set in the appropriate section of `west.cfg`. Details regarding these parameters are listed at https://westpa.readthedocs.io/en/latest/documentation/ext/westpa.westext.hamsm_restarting.html#west-cfg.

Table 3. haMSM plugin organization and file explanations

`restart.dat`	Tracks current restart/run
`restart_initialization.json`	User provided on start, modified by plugin during run
`west.h5`	Generated by WESTPA for the currently active run
`restart0/`	Stores data from the first restart. 0-indexed.
`/JtargetSS.txt`	haMSM target steady-state flux estimate
`/pSS.txt`	haMSM steady-state distribution estimate
`/hamsm.obj`	Pickled msm_we.ModelWE object
`/startstates.txt`	Used for next restart, holds all weighted structures
`/basisstates.txt`	The initial set of basis states supplied to `w_init`
`/targetstates.txt`	The initial set of target states supplied to `w_init`
`/struct/`	Complete set of structure files for all structures in `startstates/`
`/run*/`	Backed up `traj_segs/`, `seg_logs/`, and `west.h5` from each run in this marathon. 1-indexed.
`/plots/*.pdf`	Various auto-generated plots
`restart*/`	Other restarts

`restart_initialization.json`: When initializing each run, the plugin needs to know what configuration it should be launched with. After the first restart, this is automatically generated. However, *before* the first restart (i.e. in producing the initial set of runs in Workflow Step 1), there is no way for the plugin to determine how the first run was initialized. So, the parameters initially passed to `w_init` must be manually entered into `restart_initialization.json`.

`westpa_scripts/restart_overrides.py`: When building the haMSM, some dimensionality reduction is typically necessary as it's generally neither practical nor useful to analyze the model on the full set of coordinates. This dimensionality reduction is highly system-specific, so no general procedure is distributed with the plugin. Instead, the user is required to define a function which takes in an array of full-atomic coordinates of shape `(n_segments, n_atoms, 3)`, perform the desired dimensionality reduction, and then return the reduced coordinates in an array of shape `(n_segments, n_features)`. This functions are then loaded by the haMSM analysis code at run-time, and used throughout. More details are available at https://westpa.readthedocs.io/en/latest/documentation/ext/westpa.westext.hamsm_restarting.html#featurization-overrides.

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

Preparing the WE Simulation Environment. To prepare the system for using the haMSM restarting plugin, first clone the tutorial repository. In `env.sh`, change `$TEMPDIR_ROOT` to point to a directory on your filesystem where temporary files will be created (on a cluster, this should ideally be some node-local scratch/temp space that supports I/O, but on a local workstation can be a new folder such as `$WEST_SIM_ROOT/temp`). This temp space will be used for temporary files created during progress coordinate calculation. In the same file, change `AMBER_EXEC` and `CPPTRAJ` to point to your AMBER and CPPTRAJ executables. In `west.cfg`, change `ray_tempdir` to point to the same directory as `TEMPDIR_ROOT`. Then examine haMSM plugin-specific configuration files above to familiarize yourself with them, though for this tutorial, no further changes are required. To download and extract the prepared files for the in-progress simulation this tutorial uses, run the following command from within the main simulation directory.

```
$ bash pull_sample_data.sh
```

7.7.5 Running the WE simulation

Now, we are ready to (re)start the WE simulation. The haMSM plugin will automatically perform the restarting and analysis. Typically, we would initialize the system using `w_init` as we do for all WE simulations using the WESTPA 2.0 software package. However, to reduce the runtime for this tutorial, we have provided a pre-prepared system, and running `w_init` is not required. Once we have configured the haMSM plugin through `west.cfg`, we can restart the WE simulation and run the simulation for a few iterations, by simply executing the following command.

```
$ ./run.sh
```

If the target state has not been reached after running for the specified number of iterations, additional rounds of restarting/extending the WE simulation will automatically be launched. Once the target state has been reached, the plugin will build an haMSM, update statistical weights for each sampled structure, and restart a new set of WE trajectories initialized from those structures with updated weights. This will all be done automatically.

Start States. After performing a restart, we will find under the `restart0/` directory a `startstates.txt` reference file that lists all the structures used for the restart (start states) and their associated weights for initializing new WE simulations from the first round of restarting. As noted, these are distinct from the basis states. The `startstates.txt` text file is formatted with three columns, which define the names (e.g., "b21s0"), associated probabilities, and name of the directory containing the structure files of the start states (relative to the path defined under `west.data_refs.basis_states` in `west.cfg`). Structure files corresponding to these start states are in `restart0/structs/` and are named according to the structure's haMSM bin and the structure's index within that bin. Start states can be added to the pool of potential structures for WE initialization by adding the `--sstates-from` or `--sstates` option to the `w_init` command. Similar to the `--bstates` options for basis states in the above **Intermediate Tutorial 7.3**, the `--sstates-from` option is used to indicate a text file with a list of start states and the `--sstates` option is used to append additional start states through the command line.

7.7.6 Analyzing the WE simulation

After the plugin finishes running, you will find the associated `west.h5` for each run and the associated haMSM pickled `hamsm.obj` objects for each marathon in the `restart*/` directories. Although the plugin will automatically build the haMSMs and perform some of the analysis based on the configuration files, haMSM analysis can also be manually performed post-simulation on WESTPA data with the `msm_we` library (as used internally by the plugin).

For this tutorial, you can use either the data generated by the steps above, or the pre-prepared `west.h5` files containing data generated from a similar simulation configuration. This analysis largely follows the `msm_we` usage instructions provided in the `msm_we` documentation (https://msm-we.readthedocs.io/en/latest/usage.html).

For detailed instructions on how to analyze your simulation results, please refer to the Jupyter notebook distributed along with this tutorial.

7.7.7 Conclusion

Complex systems may exhibit relaxation slow enough to prevent direct measurement of rate constants using probability flux in WE. This tutorial therefore presents the haMSM plugin for leveraging relaxation-phase WE simulations by automatically (i) building a haMSM; (ii) generating an estimate of the steady-state probability distribution and the corresponding steady flux; and (iii) if desired, restarting a new WE simulation or set of simulations from the estimated steady state. Each haMSM yields an estimate for the MFPT and FPT distribution using `msm_we`.

7.8 Advanced Tutorial: Creating Custom Analysis Routines and Calculating Rate Constants

7.8.1 Introduction

In this tutorial, we will go over how to create custom analysis routines using the `westpa.analysis` Python API and how

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

to calculate rate constants using the Rate from Event Durations (RED) analysis scheme, which enables rate-constant estimates from transient, pre- steady-state data and therefore shares the same motivation as the haMSM analysis scheme [2, 18] used in the above **Advanced Tutorial 7.7**. For the creation of custom analysis routines, we will focus on the membrane permeability simulations completed in **Advanced Tutorial 7.6**. For the calculation of rate constants, we will focus on previously published protein-protein binding simulations involving the barnase/barstar system [6].

Learning objectives. Specific learning objectives for this tutorial include:

1. How to access simulation data in a `west.h5` file using the high-level `Run` interface of the `westpa.analysis` Python API and how to retrieve trajectory data using the `BasicMDTrajectory` and `HDF5MDTrajectory` readers;
2. How to access steady-state populations and fluxes from the `assign.h5` and `direct.h5` data files, convert fluxes to rate constants, and plot the rate constants using an appropriate averaging scheme;
3. How to apply the RED analysis scheme to estimate rate constants from shorter trajectories.

7.8.2 Prerequisites

In addition to completing the **Basic and Intermediate Tutorials 7.1-7.4** [38], a prerequisite to this tutorial is completion of the above **Advanced Tutorials 7.5 and 7.6**.

Computational requirements. Users should have access to at least 1 CPU core for running the analysis tools. For larger datasets, one may want to parallelize some of the tools (especially `w_direct`). The size of a dataset is mainly determined by the number of iterations. For a dataset of greater than 1000 iterations, it may be best to use at least 4 CPU cores at a time.

7.8.3 Creating custom analysis routines

For this part of the tutorial, we will create custom analysis routines using the `westpa.analysis` API for the membrane permeability simulations completed in **Advanced Tutorial 7.6**.

The main abstraction of the `westpa.analysis` API is the `Run` class, which provides a read-only view of the data in the main WESTPA output file (`west.h5`). We will start by opening the `west.h5` file from the permeability run. We assume that the current working directory is the simulation root directory you are interested in analyzing, though the resulting `west.h5` file from **Advanced Tutorial 7.6** is linked to the main directory of this tutorial for convenience. Open a Python interpreter and run the following commands:

```
>>> from westpa.analysis import Run
>>> run = Run.open('west.h5')
>>> run
<WESTPA Run with 500 iterations at 0x7fcaf8f0d5b0>
```

We now have convenient access to a wealth of information about the permeability simulation, including all trajectory segments at each WE iteration and any data associated with those segments, including values of the progress coordinate and other auxiliary data. Iterating over a run yields a sequence of `Iteration` objects, each of which is a collection of `Walker` objects. For example, the following loop iterates over all trajectory walkers in a run, but does nothing with each trajectory walker:

```
>>> for iteration in run:
...     for walker in iteration:
...         pass
```

We can access a walker by providing its (1-based) iteration number and (0-based) segment ID:

```
>>> walker = run.iteration(10).walker(4)
>>> walker
Walker(4, Iteration(10, <WESTPA Run
  with 500 iterations at 0x7fcaf8f0d5b0>))
```

To access the progress coordinates of a certain trajectory walker, we use the `pcoords` attribute:

```
>>> pcoords = walker.pcoords
```

Other properties available through this Python API include the weight, parent and children of a trajectory walker. We can access auxiliary data by looking up the dataset of interest in the `auxiliary_data` dictionary attribute (note that the following auxiliary dataset is not actually present, and the command is provided as an example):

```
>>> auxdata = walker.auxiliary_data['test_data']
```

We can also view a list of all recycled (successful) trajectory walkers and choose one walker to trace its pathway through the membrane:

```
>>> walkers = run.recycled_walkers
>>> walker = max(walkers,
...     key=lambda walker: walker.weight)
```

The history of a trajectory walker can be traced by using the `trace()` method, which returns a `Trace` object:

```
>>> trace = walker.trace()
```


https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

Using the WE iteration and IDs of the trajectory segments obtained from this trace, we can plot the data of our traced trajectory to see how that property is changing. Remember that the `test_data` auxiliary dataset does not actually exist, but can be replaced with an auxiliary dataset of your choice.

```
>>> xs = [walker.iteration.number
...       for walker in trace]
>>> ys = [walker.auxiliary_data['test_data']
...       for walker in trace]
>>> import matplotlib.pyplot as plt
>>> plt.plot(xs, ys)
```

One goal of the `westpa.analysis` API is to simplify the retrieval of trajectory coordinates. Two built-in readers are provided for retrieving MD trajectory coordinates: (1) `BasicMDTrajectory`, which handles trajectory files outputted by the dynamics engine; or (2) `HDF5MDTrajectory`, which handles trajectories stored using the new HDF5 framework, as is done in the above **Tutorials 7.5 and 7.6**. For users requiring greater flexibility, custom trajectory readers can be implemented using the more general `Trajectory` class. Here we provide a brief overview of both the `BasicMDTrajectory` and the `HDF5MDTrajectory` readers. The following is included only as an example, since the trajectory files required are not provided. MD trajectories stored in the traditional manner may be retrieved using the `BasicMDTrajectory` reader with its default settings:

```
>>> from westpa.analysis import BasicMDTrajectory
>>> trajectory = BasicMDTrajectory()
```

Here, `trajectory` is a callable object that takes either a `walker()` or a `trace()` object as input and returns an `mdtraj.Trajectory()` object (https://mdtraj.org/1.9.5/api/generated/mdtraj.Trajectory.html). To retrieve the trajectory of the trace obtained above, then save the coordinates to a DCD file (e.g., for visualization using the VMD program), we can run the following:

```
>>> traj = trajectory(trace)
>>> traj.save('trace-coords.dcd')
```

Note that in the code above, we have relied on the fact that the `traj_segs/` directory of interest is contained in the current working directory. In the general case, the name of the simulation root directory may be provided to the trajectory reader via the optional `sim_root` parameter. Minor variations of the "basic" trajectory storage protocol (e.g., use of different file formats) can be handled by changing the parameters of the `BasicMDTrajectory` reader:

```
>>> trajectory = BasicMDTrajectory(
  traj_ext='.nc', state_ext='.ncrst', top=None)
```

However, suppose that instead of storing the coordinate and topology data for trajectory segments in separate files (`seg.dcd` and `bstate.pdb`), we store them together in an HDF5 trajectory file (such as `iter_XXXXXX.h5`) using the new HDF5 restarting framework available in WESTPA 2.0. This change can be accommodated by using the `HDF5MDTrajectory` reader:

```
>>> trajectory = HDF5MDTrajectory()
```

The examples above highlight the flexibility and convenience provided by the `westpa.analysis` package and provide the building blocks available to a user wanting to explore the `west.h5` file and create custom analysis routines using data in the `west.h5` file.

7.8.4 Calculating rate constants using the original WE scheme

For the two remaining sections of this tutorial, we will focus on applying the RED analysis scheme [20] to calculate the association rate constant from previously published protein-protein binding simulations involving the barnase/barstar system [6]. The RED scheme involves three steps:

1. Calculate State Populations and the Flux into the Target State. The target state can be defined using either the WE progress coordinate or auxiliary coordinates. The analysis bins and state definitions are placed in the analysis section of the `west.cfg` file.

```
west:
  analysis:
    kinetics:
         evolution: cumulative
       analysis_schemes:
         OVERALL:
           enabled: True
           bins:
             - type: RectilinearBinMapper
               boundaries:
                 - [0, 3.5, 'inf']
                 - [0, 3, 15, 'inf']
           states:
             - label: unbound
               coords:
                 - [0.5, 50.0]
                 - [50.0, 50.0]
             - label: bound
               coords:
                 - [0.5, 0.5]
```


https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

To calculate the flux into a state defined by the progress coordinate, we can use the `w_ipa` program. Flux calculation with `w_ipa` involves two steps: (1) assigning trajectory segments to states using the `w_assign` command-line tool, and (2) calculating the probability flux between each pair of defined states using the `w_direct` command-line tool. Given the large size of the barnase-barstar simulation HDF5 file, we have provided the resulting `assign.h5` and `direct.h5` files for the remainder of the tutorial. To use `w_ipa` for flux analysis, one would run the following at the command line:

```
$ w_ipa -ao
```

This command will analyze the `pcoord` data from `west.h5` using the scheme for bins and states defined in `west.cfg` and not drop the user into an IPython environment (to make use of that functionality, remove the `-ao` options from the above command). The resulting `assign.h5` and `direct.h5` files, the latter containing the fluxes, will be outputted to a newly created directory that is named for the relevant scheme (in this case that will be in `./ANALYSIS/OVERALL/`. The `evolution:cumulative` option (which is the default option) ensures that all evolution datasets are calculated with a rolling average, a requirement for using the RED scheme (see below).

To calculate the flux into a state defined by auxiliary coordinates, we still use the scheme for bins and states defined in the `west.cfg` file. However, instead of using the `w_ipa` program, we use the command-line tools, `w_assign` and `w_direct`. Before using these tools, we need to copy `module.py` to your current directory first (by default, `module.py` is located in the `./scripts/` directory). Then, we can assign trajectory segments to specified states using the `w_assign` tool:

```
$ w_assign --config-from-file --scheme OVERALL
  --construct-dataset module.load_auxdata --serial
```

The `--config-from-file` option tells the program to read analysis parameters from the `west.cfg` file's analysis section and the `--scheme` option specifies the relevant scheme for bins and states. The `--construct-dataset` option provides a function to `w_assign` for loading in the auxiliary data which is located in the file `module.py`. The `--serial` option tells `w_assign` to run the assignment in serial mode. Running `w_assign` will generate an `ANALYSIS/` directory and place an `assign.h5` file in a scheme-specific folder there. Next, we apply the `w_direct` tool to the `assign.h5` file.

```
$ w_direct all -a ./ANALYSIS/OVERALL/assign.h5
```

This command will generate a `direct.h5` file in the same directory where we ran the `w_direct` command. Move this file to the `analysis/` folder that was generated by `w_assign` and proceed with the analysis. Note that the above is not part of the following tutorial and is only included to provide an example to users of how to perform an analysis on auxiliary data.

2. Correct the Fluxes using the RED Scheme. To correct the calculated fluxes using the RED scheme, we apply the `w_red` command-line tool, which will read in the `rate_evolution` and durations datasets in your `direct.h5` file and calculates a correction factor for the flux value at each iteration. To use this tool, add the following to the analysis section of your `west.cfg` file:

```
red:
  scheme: OVERALL
  istate_label: unbound
  fstate_label: bound
  nstiter: 21
  nstrep: 1
```

For the scheme option, specify the desired scheme for bins and states to use from your `west.cfg` analysis section and for the `istate_label` and `fstate_label` options, specify the initial and target states, respectively. The `nstiter` parameter is the number of frames per WE iteration that were saved during the WESTPA simulation and `nstrep` is the frequency of outputting progress of the RED calculation. After setting all parameters, run the `w_red` tool from the command line:

```
$ w_red
```

A new dataset containing the corrected fluxes, named `red_flux_evolution`, will be added to your `direct.h5` file. If that dataset already exists, the `w_red` tool will ask if you want to overwrite the existing dataset.

The new `red_flux_evolution` dataset is created by adjusting the `rate_evolution` dataset by the correction factor determined by the RED scheme. The `rate_evolution` dataset, in turn, is composed of the relevant `conditional_flux_evolution` dataset (from `direct.h5`) normalized by the steady state populations from `labeled_populations` (from `assign.h5`). Therefore, when using the RED scheme (or the original `rate_evolution` dataset), no explicit normalization of the fluxes by the steady state populations is necessary.

3. Convert the Flux to a Rate Constant. The fluxes that we just calculated are already rate constants in units of inverse τ (the resampling interval used for your WE simulation).

For a unimolecular process, the final units of the rate constant should be inverse time (e.g., s^{-1}) and can be obtained by converting from units of per τ for the extracted flux array to the desired time unit. Note that the τ value needs to be provided by the user and is not currently recorded in the `west.h5` file.

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

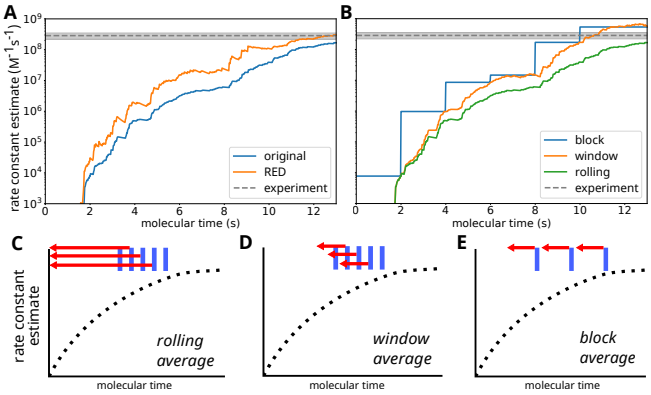

Figure 18. A) A comparison of the RED and original schemes for estimating the rate constant for protein-protein association involving the barnase/barstar system from previously published WE simulations [6]. In this case, the simulation has not converged to a steady state, as indicated by different rate-constant estimates using the original and RED schemes. B) A comparison of different averaging schemes using the same dataset. While the RED scheme is, in principle, only compatible with rolling averages, the use of block and window averaging can still be informative for monitoring convergence of the simulation. C-E) Schematic illustrations of rolling, window, and block averaging schemes, indicating points where averages are calculated (blue) and the extent of data used for the averages (red).

For a bimolecular process, which is the case for our `OVERALL` scheme, the rate constant should be in units of inverse time and inverse concentration (e.g., $M^{-1}s^{-1}$) and can be obtained by first converting to the desired time unit, as done for unimolecular processes, and dividing the resulting flux values by the effective concentration of the solutes involved in the bimolecular process given the volume of the simulation box. In the case of our barnase-barstar system, the τ value was 20 ps and the effective concentration for the barstar ligand was 1.7 mM. We will therefore divide all of the `conditional_flux` values by 20×10^{-12} s and 0.0017 M to obtain per iteration rate constants in units of $M^{-1}s^{-1}$.

7.8.5 Monitoring Convergence of the Rate Constant

To monitor the convergence of the rate-constant estimate, we can plot the time-evolution of the rate constant using both the original and RED schemes and assess how close our original estimate is to the RED estimate. If the two schemes are converging to the same value, that can be one indication that the simulation has begun to converge.

To obtain the rate constant using the original scheme [3], simply convert the `rate_evolution` dataset to the appropriate units as discussed above. The time-evolution of this rate-constant estimate can then be compared with the RED-scheme estimate to assess convergence of the simulation to a steady state (**Figure 18A**).

A note on averaging schemes. There are three main types of averaging schemes that can be used to monitor convergence when plotting the rate constant evolution using the original scheme. It may be useful to plot different schemes depending on the behavior of your specific system. A few examples are shown here with instructions on how to generate the plots.

The first averaging scheme is the default which is a rolling average (**Figure 18B**), which can be achieved by specifying `evolution: cumulative` in the analysis section of `west.cfg` and setting `step_iter: 1`. This method of visualizing the rate constant evolution offers the smoothest curve and is recommended as the most stringent way of assessing convergence, as it incorporates information from the entire simulation. A rolling average is also implicitly incorporated into the RED scheme, which by design never excludes data from the start of a simulation. When analyzing rate constant estimates generated by the RED scheme, specify `evolution: cumulative` to ensure that only the implicit rolling average is performed.

The second averaging scheme is a window average, which can be achieved in `w_ipa` by specifying `evolution: cumulative` and `step_iter: 10`, or whatever your desired averaging window is. A recommended starting averaging window size is 10% of the length of your simulation, but the most robust would be the lag time of your simulation as determined from an autocorrelation of the flux plot. A windowed average is not as smooth as the rolling average but can give a better indication of convergence at different stages of your simulation relative to other stages.

The third and final averaging scheme is a block average. This will require setting `evolution: blocked` and `step_iter: 10`. The rationale behind choosing the block size here is the same as the window size discussed above. The block average will appear like a step function where each block an average of the preceding block is plotted. This method of plotting is the least smooth, but can be best for obtaining a final value of the rate constant that is not influenced by earlier, lower values.

7.8.6 Conclusion

Among the upgrades introduced in the WESTPA 2.0 software package are ones that enable the creation and execution of more streamlined analysis of simulations and more efficient estimation of rate constants. The `westpa.analysis` subpackage can be utilized to more carefully inspect WESTPA trajectory data and to create custom analysis routines. The RED analysis scheme for correcting rate constants based on the "ramp-up" time in the fluxes is implemented in the `w_red` command-line tool. The files contained in this tutorial for utilizing the RED scheme are intended to provide useful starting points for analyzing the kinetics of WESTPA simulations.

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

7.9 Advanced Tutorial: M-WEM Simulations of Alanine Dipeptide

7.9.1 Introduction

The Markovian Weighted Ensemble Milestoning (M-WEM) approach [11] is a modified version of the Weighted Ensemble Milestoning (WEM) [53, 54] approach. Both approaches are designed to use the WE strategy to enhance the efficiency of the milestoning method in calculating equilibrium and non-equilibrium properties (e.g., free energy landscape and rate constants, respectively).

In the milestoning method [12, 13] the reaction coordinate is stratified using multiple high-dimensional interfaces—or milestones. Short trajectories are propagated between the interfaces and using the principles of continuous time Markov chains, the properties of a long timescale process can be calculated. But the milestones need to be placed significantly far from each other to lose memory of the previous milestone. But converging the transition statistics between distant milestones can be expensive depending on the underlying free energy landscape and the complexity of the system. Because of the use of shorter trajectories that do not require trajectories to transit from the initial to final states, the WEM and M-WEM calculations are computationally less expensive than a WE simulation. On the downside, however, one cannot obtain continuous pathways due to the lack of continuous trajectories between the starting and the final state.

In the M-WEM approach, regions between the milestones are referred to as "cells". WE simulation is performed within the cell with half-harmonic walls present at the milestone interfaces to prevent trajectory escape [11, 55, 56]. In this tutorial, we will use M-WEM to calculate the mean first passage time (MFPT), free energy landscape and committor function for the conformational transition in the alanine dipeptide system.

Learning objectives. This tutorial covers the installation of and use of the Markovian Weighted Ensemble Milestoning (M-WEM) software in combination with WESTPA to compute the kinetics and the free energy landscape of an alanine dipeptide. Specific learning objectives include:

1. How to install the M-WEM software and perform an M-WEM simulation;
2. How to create milestones to define the M-WEM progress coordinate;
3. How to analyze an M-WEM simulation to compute the mean first passage time, committor, and free energy landscape.

7.9.2 Prerequisites

The users should have a basic understanding of running WE simulations using the Minimal Adaptive Binning (MAB) scheme, and should have completed **Advanced Tutorial 7.6** before commencing the M-WEM tutorial. Also, a basic idea of the Markovian Milestoning framework is necessary. For that purpose, the users should refer to the following manuscripts [11, 55, 56].

Computational requirements. In terms of software, this tutorial requires several Python modules (NumPy, SciPy, and Matplotlib) in addition to the WESTPA 2.0 software and NAMD 2.14 simulation package.

Note: M-WEM is implemented using the Colvars module in NAMD. Please check out the NAMD tutorial (http://www.ks.uiuc.edu/Training/Tutorials/namd-index.html) and colvars tutorial (https://colvars.github.io/colvars-refman-namd/colvars-refman-namd.html).

In terms of computer hardware, this tutorial will require approximately 4 GB of disk space. Running the simulation 100 WE iterations for each milestone takes ~85 min on an Intel Core i5-8250U CPU @ 1.60GHz processor with 4 CPU cores. For 8 milestones that amounts to about 11-12 hr if the calculation is performed serially with 4 CPU cores used at a time. But if a computer cluster is available, each milestone should be run in parallel which will significantly reduce the wall clock time. The analysis for each milestone takes ~5 min for each milestone with the same computing hardware but with 1 CPU core. For 8 milestone cells that would be ~40 min, but similar to the simulation, the analysis for each milestone can be done in parallel.

7.9.3 Installation of the M-WEM software package

The `M-WEM` software package can be downloaded from https://github.com/dhimanray/MWEM. To install the package go to the main directory that contains the `setup.py` file and run the following command:

```
$ python -m pip install .
```

The M-WEM software should be installed in the same conda environment in which WESTPA 2.0 is installed.

7.9.4 Setting up a M-WEM environment

Overview. For performing the M-WEM simulation, we need to first create the milestone anchors along the transition pathway. This is a typical prerequisite for milestoning simulation. It is typically done using steered MD simulation [57] which is a common technique in MD simulation. To avoid spending extra time and possible variability in the results, we have generated the milestone anchors and provided them in the anchors directory.

The system. We will be studying the conformational change of gas phase alanine dipeptide using the M-WEM scheme. The details of this example are provided in [11].

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

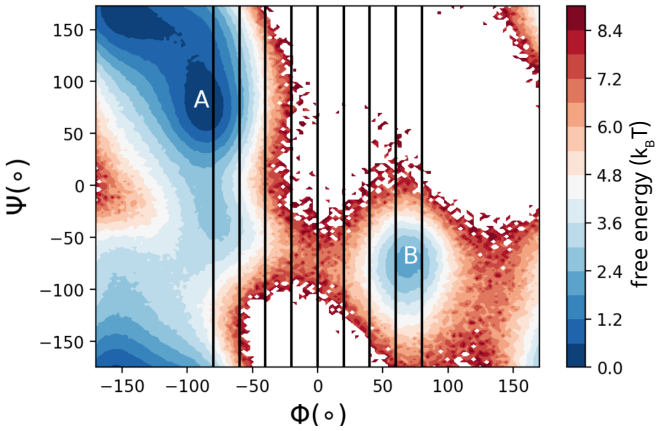

Figure 19. Free energy landscape of alanine dipeptide in the gas phase. Milestone positions are shown as vertical lines. Our aim is to simulate the MFPT of the transition from state A to state B.

The free energy landscape for the system is shown in **Figure 19**. To simulate the transition from state A to state B, 9 milestones are placed at ϕ = -80°, -60°, -40°, -20°, 0°, 20°, 40°, 60°, 80°. This created 8 cells bound by the milestones. The anchors are chosen in a way that they are located approximately in the middle of each cell.

Preparing the simulation environment. The `tutorial7.9-mwem/` directory contains all simulation and analysis files and will be referred to as the simulation home directory for the rest of the tutorial. The milestone anchors for all cells generated from the steered MD simulation are provided as pdb files in the `anchors/` directory.

The `build.py` script is a python script which will set up all the milestones for the simulation. Each milestone cell will be simulated in a different directory, numbered from 0 to 7.

The `template/` directory is a generic template for M-WEM simulation for any one cell. It contains all necessary files except for the pdb files which are specific to each cell (which are in the `anchors/` directory). Also, the `colvars.in` files have replaceable strings which are used by the `build.py` script to create cell specific files. For example, there are terms like "CENTER", "HIGH", and "LOW". These are places where the position of the center and the two milestones are written by the `build.py` code. Make sure to edit the `env.sh` file to include the path to your NAMD installation.

In the `common_files/` directory the topology (.psf), the parameter (.prm) and the NAMD configuration files are provided. The structure file (.pdb) in this directory and in the `equilibration/` directory are prepared by the `build.py` script, and are different for each milestone cell.

Once you have your M-WEM setup ready, prepare the cell-specific folders by running the following command:

```
$ python build.py
```

7.9.5 Running the M-WEM simulation

In the `equilibration/` directory of each cell, a constrained equilibration of the anchor will need to be performed to keep the anchor approximately in the middle of the cell. The `milestone_equilibration.colvars.traj` file contains the collective variable information, which, in this case, are the Phi and Psi torsion angles of the alanine dipeptide. The `milestone_equilibration.xsc`, `milestone_equilibration.coor` and `milestone_equilibration.vel` files are NAMD restart files that will be used to start WE trajectories from the endpoint of the equilibration simulation.

The running of the M-WEM simulation for each individual cell is done separately, for the convenience of parallelizing it in a computing cluster. The `run.sh` script performs the simulation via the command `./run.sh`. Unlike typical WESTPA setups, the initialization setup code is included in the `run.sh` script here. Although it does not make a significant difference for a small system like this, we found it is more convenient to submit multiple milestone jobs to a computer cluster by using a single `run.sh` script. Both the equilibration and running of each cell is performed by executing the following command from within each cell-specific folder:

```
$ ./run.sh
```

In the first part of the script, equilibration is performed. Then relevant files are copied to the `bstates/` directory, from which they are read by the `westpa_scripts/runseg.sh`. Then the WE simulation is initialized and propagated as usual by the `w_init` and `w_run` commands. In this example script, only one trajectory is propagated at a time. But this can be parallelized based on the computing resources available. Alternative Slurm scripts for running and restarting the simulation are also provided in the same directory.

The total number of iterations performed per milestone is 100. The user may choose to change this number according to their preference. The results reported in this work are from 100 iterations. The convergence is achieved after 40 iterations in our calculation. But it may slightly vary for independent calculations.

7.9.6 Analyzing the M-WEM simulations

After the M-WEM simulations are completed, it is important to properly analyze the results. Please refer to the M-WEM publication for the theoretical details of the analysis [11]. We perform the analysis in two steps:

Step 1. Move into the `analysis/` directory and execute the following command:

```
$ python analysis_build.py
```


https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

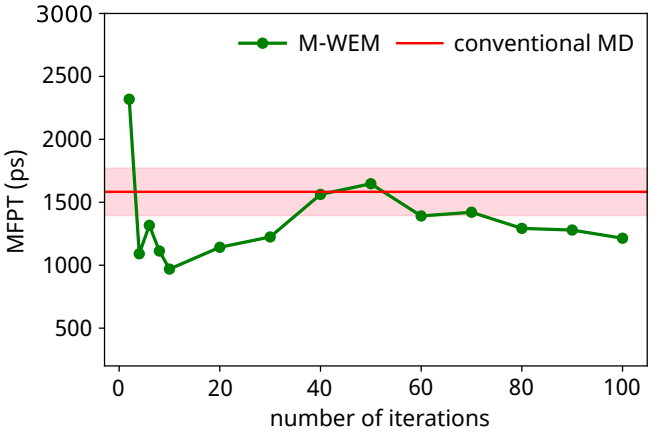

Figure 20. Convergence of the mean first passage time (MFPT) as a function of M-WEM iterations.

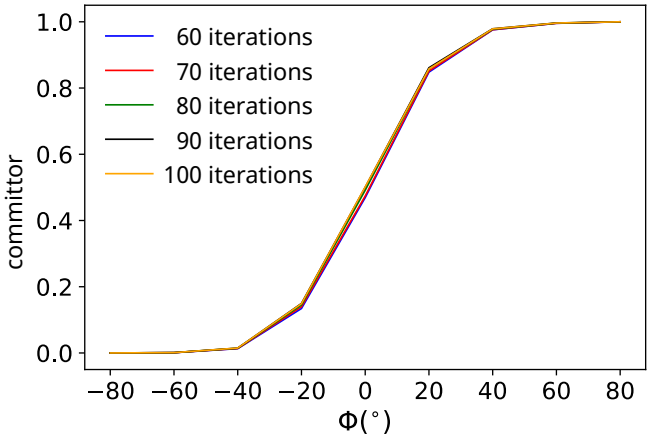

Figure 21. Committor function along the ϕ coordinate from M-WEM simulation.

The `analysis_build.py` script will produce directories `cell_0/` through `cell_7/` and copy the corresponding `west.h5` files (WESTPA output files) from the `propagation/` directory into each cell. It will also copy the `west.cfg` files (different from the `west.cfg` files for propagation), and the `analysis.py` files from `westpa_analysis_files/` directory to each cell. The `analysis.py` file also has strings like "LOW" and "HIGH", which will be replaced by floating point numbers corresponding to the left and right milestones.

Running the `analysis.py` script from within a specific cell directory will produce the `trajectories.pkl`, `crossings.pkl` and `weights.txt` files. The files generated by `analysis.py` contain information on the trajectory traces (history of the segments in the final iteration), the time and location (which milestone right or left) of the milestone crossings, and the weights of each traced trajectory respectively.

Perform analysis in all cells by running the following command from within the `analysis/` directory:

```
$ ./analyze_all_convergence.sh
```

This script will execute `python analysis.py` from within each cell-specific directory to produce the .pkl and .txt outputs for the final iteration. But, for the sake of checking convergence of our results, it will also produce similar files for some subsequent iterations. To do that, the script will copy the `analysis.py` to `analysis_x.py` (where x = iteration number) and replace the `w.niters` inside each `analysis.py` to the corresponding iteration number. Then, it will produce `trajectories_x.pkl`, `crossings_x.pkl` and the `weights_x.txt` files for each x. This step can take several minutes to a few hours depending on the computing hardware. If you have access to a computing cluster, you may choose to submit this as a job. Note that the `analyze_all_convergence.sh` script is customizable. For example, if you want to run all cells in parallel on a cluster you can create separate bash scripts for each cell. Also, `analysis_build.py` will produce the following directories for milestoning analysis in Step 2: `cell_probability/`, `N_i_j_files/`, `R_i_files/`, and `committor/`.

Step 2. After the analysis of the WESTPA output files are done, we will proceed to analyze our results using the Markovian milestoning framework in two Jupyter notebooks: `kinetics.ipynb` and `free-energy-landscape.ipynb`.

First, run the `kinetics.ipynb` notebook to obtain the mean first passage time and the committors. This will also produce the probability distribution file in the milestone space. Details can be found inside the notebook. The MFPT convergence plots and the committor functions should look like **Figures 20 and 21**.

Next, run the `free-energy-landscape.ipynb` to reconstruct the free energy landscape along Phi and Psi coordinates from the M-WEM data. It will first produce the unscaled probability distribution, rescale it and then compute the rescaled free energy landscape. The final free energy landscape should look like **Figure 22**.

Note that before executing any notebook, you will need to set the kernel to the environment in which you installed the M-WEM software. If the kernel is not available, activate the Jupyter notebook for that environment by executing:

```
$ python -m ipykernel install --user --name=westpa2
```

Replace `westpa2` with the environment in which you installed M-WEM.

7.9.7 Conclusion

This tutorial presents the Markovian Weighted Ensemble Milestoning (M-WEM) Python package for use with the WESTPA 2.0 software package to estimate equilibrium and

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

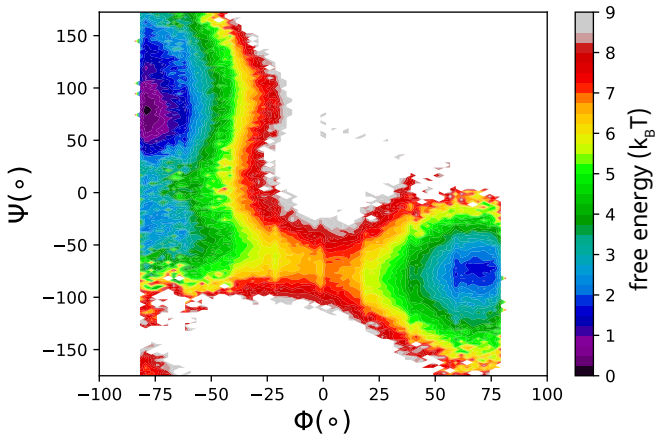

Figure 22. Free energy landscape reconstructed from an M-WEM simulation.

non-equilibrium observables for the alanine dipeptide. In the M-WEM approach, the WE strategy is applied to enhance the efficiency of the Markovian milestoning approach to accelerate the convergence between milestones. While it is not possible to use this approach to generate continuous pathways between the initial and final states of a rare-event process, the M-WEM approach can be highly efficient in the calculation of "end-point properties" such as the MFPT and free energy differences between the two states. Beyond the alanine dipeptide, the M-WEM approach has been applied to more complex processes such as receptor-ligand binding, yielding the k_{on}, k_{off}, and binding free energy for the trypsin benzamidine complex [11].

7.10 Advanced Tutorial: Systems Biology Simulations using the WESTPA/BNG Plugin

7.10.1 Introduction

This tutorial focuses on a scenario in systems biology in which the WE strategy can be useful: enhanced sampling of rare events in a non-spatial model. Here we focus on a BioNetGen language (BNGL) rule-based model for a biological signaling network that consists of a set of structured molecule types and a set of rules that define the interactions between the molecule types. While the average steady-state behavior of the model can be obtained using ordinary differential equations, the full kinetics of the model can only be obtained from stochastic simulations. However, adequate sampling of any rare events in the model can be a challenge for stochastic simulations. In this tutorial, we will use WESTPA to orchestrate BNGL simulations that are propagated by the BNG software package. As mentioned above, WESTPA is interoperable with any stochastic dynamics engine, including the BNG software.

Learning objectives. We will simulate a BNGL rule-based model of a two-gene switch motif that exhibits mutually exclusive activation and inhibition. Specific learning objectives include:

1. How to install the WESTPA/BNG plugin and set up a WESTPA/BNG simulation;
2. How to apply adaptive Voronoi binning, which can be used for both non-spatial and molecular systems;
3. How to run basic analyses tailored for high-dimensional WESTPA/BNG simulations.

7.10.2 Prerequisites

Users should have a working knowledge of BNGL models (http://bionetgen.org) and the WESTPA 2.0 software package. This tutorial will make use of the WEBNG Python package, which facilitates the integration of WESTPA with the BNG software and requires Python 3.8 or later versions. To install the WEBNG package:

```
$ git clone
      https://github.com/ASinanSaglam/webng.git
$ python -m pip install -e .
```

For common installation issues, see https://webng.readthedocs.io/en/latest/quickstart.html#installation. Alternatively the user can use a Docker container where the environment is already prepared correctly using the command:

```
$ docker pull
        ghcr.io/westpa/westpa2_tutorials:webng
```

Note that this requires a Docker installation, for more information see Docker documentation (https://docs.docker.com/get-docker). Once the docker image is downloaded, you can run the image with:

```
$ docker run -it --entrypoint /bin/bash
          ghcr.io/westpa/westpa2_tutorials:webng
```

Computational requirements. This tutorial requires ~500 MB disk space. The simulation takes at most 1 hour of wall-clock time using a single CPU core of a 3.2GHz Intel Core i7 processor. We recommend using the `WEBNG` package on a Unix system. While the package has not been tested on Windows systems, one can try using the Windows subsystem for Linux (WSL; https://docs.microsoft.com/en-us/windows/wsl/install).

7.10.3 Setting up the simulation

The model. Our BNGL model consists of two genes, gene A and gene B, that are transcribed to produce proteins A and B, respectively (**Figure 23**). Protein A binds to the gene A promoter site to activate protein A production and to the gene

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

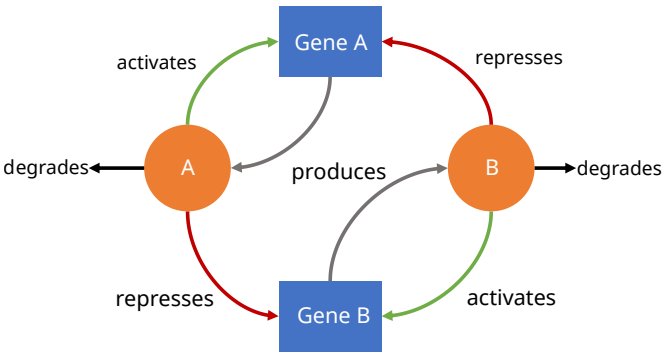

Figure 23. Two-gene network of exclusive mutual inhibition and self activation. Blue squares represent genes and orange circles represent proteins.

B promoter site to repress B production. Likewise, protein B activates gene B and represses gene A. The two most populated states therefore consist of either (1) high quantities of protein A and low quantities of protein B, or (2) high quantities of protein B and low quantities of protein A. Transitions between these two states are rare events.

Preparing the simulation environment. For this portion of the tutorial, you can use either your own BNGL model or the ExMISA model described above, which is the default WEBNG example. WEBNG uses a YAML configuration file to set up a WESTPA folder. The `WEBNG template` subcommand gives you a YAML config file with the same defaults which you can then edit and use to generate the WESTPA simulation folder.

If you are using the default example, the command to generate the template is the following:

```
$ webng template -o mysim.yaml
```

If you are using your own model file called `exmisa.bngl`, the command is:

```
$ webng template -i exmisa.bngl -o mysim.yaml
```

This command will generate the YAML file, `mysim.yaml`. For the full set of configuration options, see https://webng.readthedocs.io/en/latest/config.html. Path options are specified automatically using the libraries that are already installed. Propagator options will also be automatically populated according to the BNGL model. By default, this simulation setup will use an adaptive Voronoi binning scheme [16] due to the fact that rectilinear binning is not feasible for high-dimensional BNG models. The `center_freq` option sets the frequency of Voronoi bin addition, in units of WE iterations, `max_centers` is the maximum number of Voronoi bins that will be added, `traj_per_bin` is the number of trajectory walkers per Voronoi bin, and `max_iter` option sets the maximum number of WE iterations. All of these options can be modified after the simulation folder is set up (see https://github.com/westpa/westpa/wiki/User-Guide#Setting_Up_a_WESTPA_Simulation).

By default, the stochastic simulator is set to libroadrunner (http://libroadrunner.org). To use this simulator, we must first convert the BNGL model to a systems biology markup language (SBML) model. Next, we use the WEBNG software to compile the SBML model into a Python object, which allows for efficient simulation of the model. WEBNG also supports the use of the `BNG` simulation package. However, the use of this package will result in higher file I/O operations. Any other stochastic simulator will require the use of a custom WESTPA propagator.

7.10.4 Running the WE simulation

To run the simulation, we first need to generate a WESTPA folder using the YAML configuration file generated in the previous step:

```
$ webng setup --opts mysim.yaml
```

The above command will use the path option `sim_name` as the WESTPA folder, which is automatically set to the model name in the folder you ran the template command. Next, we initialize the simulation and run the model in a serial mode using a single CPU:

```
$ cd exmisa
$ ./init.sh
$ w_run --serial
```

To run the model in a parallel mode using multiple CPU cores, please refer to WESTPA documentation for options available with the `w_run` command-line tool. The resulting simulation can be found in the `exmisa/` folder directory.

7.10.5 Analyzing the WE simulation

To analyze the simulation, we will use the WEBNG package. To begin, we edit the YAML file under the folder that contains the configuration file `mysim.yaml`, setting `analyses.enable`, `analyses.average.enable`, and `analyses.evolution.enable` to `True`; and `analyses.average.first-iter` to the simulation half point (default: 50). To run the analysis, we use the following command:

```
$ webng analysis --opts mysim.yaml
```

The above command will generate an `analysis/` folder in the simulation folder, run the analyses, and generate associated figures.

By default, `average.png` provides an N×N matrix of plots of the average two-dimensional probability distributions of each observable (dimension) of the WE progress coordinate (**Figure 24**) and each of the other observables. The `evolution.png` file gives the time-evolution (number of WE

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

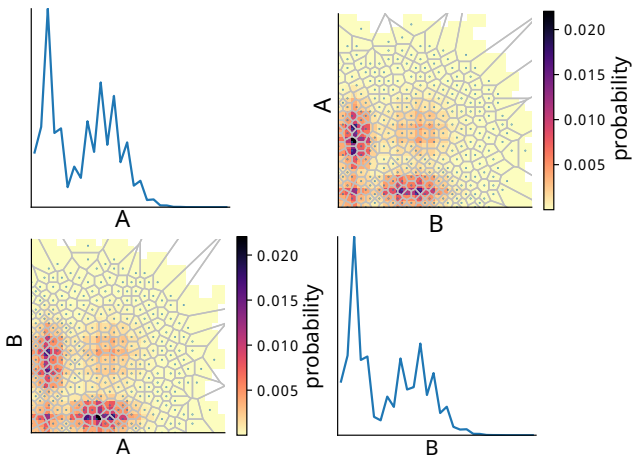

Figure 24. Probability distributions as a function of the two-dimensional progress coordinate (off-diagonal) and each dimension of the progress coordinate (diagonal). Axes A and B show the quantity of each protein in the two-gene model. Grey lines in each off-diagonal plot delineate adaptive Voronoi bins used during the WE simulation.

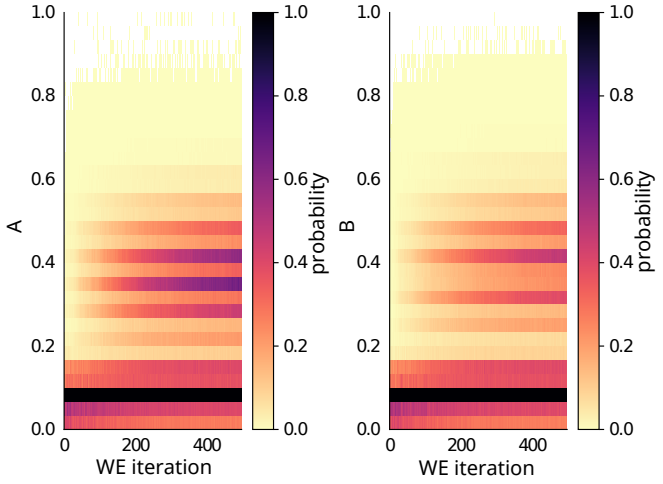

Figure 25. Probability distributions of each observable (A or B) of the two-dimensional progress coordinate as a function of WE iteration. The striped nature of the distributions is due to the fact that A and B are discrete as opposed to continuous observables.

iterations) of probability distributions for each observable (**Figure 25**) and can be used to assess the convergence of simulation, making modifications to the binning scheme if necessary.

The average two-dimensional probability distributions reveal a total of four states: a low A/low B state, the symmetric low A/high B state and high A/low B states, and a high A/high B state (**Figure 24**). The fourth state is the least probable while the first three states are all highly probable. Transitions from low A/high B to high A/low B states are difficult to sample and transitions from low A/high B to high A/high B states are even more difficult to sample. The WE algorithm allows the user to sample these states and transitions between the states. All analyses should be based on the portion of the simulation that is done evolving. If the simulation is still evolving, we recommend extending the simulation until the observables of interest are reasonably converged.

7.10.6 Conclusion

As demonstrated by this tutorial, the WEBNG Python package provides a framework for applying the WESTPA 2.0 software package to BNGL models with minimal user input and simplified installation. The adaptive Voronoi binning scheme enables efficient application of high-dimensional progress coordinates for both molecular and non-spatial systems. Voronoi bins can be effective for exploratory simulations, placing bins as far away as possible from previous bins to inform the creation of a custom binning scheme for sampling the rare-event process of interest. However, such bins may not be as effective for surmounting barriers (e.g., compared to the MAB scheme [19]), as demonstrated by the probability distribution as a function of the WE progress coordinate where many bins near the edges of the configurational space are occupied, but are not of interest. Future work with WEBNG will include more detailed analysis options such as automated clustering, generation of networks from bins and clusters, and the estimation of rate constants for transitions between the clusters.

8 Author Contributions

AT Bogetti, JMG Leung, JD Russo, S Zhang, JP Thompson, AS Saglam, and D Ray developed and wrote the Advanced tutorials; the primary author of each tutorial is designated as a co-first author of the overall manuscript. AT Bogetti, B Mostofian, AJ Pratt, AS Saglam, PO Harrison, JL Adelman, M Dudek, PA Torrillo, AJ DeGrave, and U Adhikari developed and wrote the Basic and Intermediate Tutorials 7.1-7.4 [38]. LT Chong, DM Zuckerman, DN LeBard, MC Zwier, JL Adelman, I Andricioaei, and JR Faeder provided guidance for tutorial development. AT Bogetti, JMG Leung, LT Chong, DM Zuckerman, and MC Zwier wrote the introductory sections leading up to the tutorials. RC Abraham helped generate trajectory data for **Advanced Tutorial 7.5**.

9 Other Contributions

We thank the many users of the WESTPA software who have provided important feedback for improving our tutorials and documentation over the years. We thank Michael Shirts for assistance in generating this article.

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

10 Potentially Conflicting Interests

The authors declare the following competing financial interest(s): L.T.C. is a current member of the Scientific Advisory Board of OpenEye Scientific and an Open Science Fellow with Psivant Sciences. S Zhang, JP Thompson, and DN LeBard are employees of OpenEye Scientific.

11 Funding Information

This work was supported by NIH R01 GM1151805 to LT Chong, DM Zuckerman, JR Faeder, and MC Zwier; NSF grants CHE-1807301 and MCB-2112871, and NIH R01 GM1115762 to LT Chong; a University of Pittsburgh Dietrich School of Arts and Sciences Graduate Fellowship to AT Bogetti; NIH T32-DK061296 to JL Adelman; a University of Pittsburgh Brackenridge Summer Undergraduate Fellowship to PA Torrillo; a University of Pittsburgh Andrew Mellon Predoctoral Fellowship to AT Bogetti; and a MolSSI Software Fellowship to JD Russo. Computational resources were provided by the University of Pittsburgh Center for Research Computing under NSF MRI 2117681.

12 Content and Links

All files needed for the tutorials can be found at https://github.com/westpa/tutorials.

Author Information

ORCID:

Anthony T. Bogetti: 0000-0003-0610-2879
Jeremy M. G. Leung: 0000-0001-7021-4619
John D. Russo: 0000-0002-2813-6554
She Zhang: 0000-0001-9265-4498
Jeff P. Thompson: 0000-0003-3134-9451
Ali S. Saglam: 0000-0002-6513-8401
Dhiman Ray: 0000-0002-7103-0886
Barmak Mostofian: 0000-0003-0568-9866
AJ Pratt: 0000-0002-5275-1020
Rhea C. Abraham: 0000-0003-3823-3822
Page O. Harrison: 0000-0003-2301-6016
Max Dudek: 0000-0001-8521-9543
Paul A. Torrillo: 0000-0002-4618-6061
Alex J. DeGrave: 0000-0001-9933-6273
Upendra Adhikari: 0000-0002-7568-1598
James R. Faeder: 0000-0001-8127-609X
Ioan Andricioaei: 0000-0002-6655-0721
Joshua L. Adelman: 0000-0001-9986-7533
Matthew C. Zwier: 0000-0002-0744-1146
David N. LeBard: 0000-0002-0979-800X
Daniel M. Zuckerman: 0000-0001-7662-2031
Lillian T. Chong: 0000-0002-0590-483X

References

[1] **Zwier MC**, Adelman JL, Kaus JW, Pratt AJ, Wong KF, Rego NB, Suárez E, Lettieri S, Wang DW, Grabe M, Zuckerman DM, Chong LT. WESTPA: An Interoperable, Highly Scalable Software Package for Weighted Ensemble Simulation and Analysis. Journal of Chemical Theory and Computation. 2015; 11(2):800–809. https://doi.org/10.1021/ct5010615.

[2] **Russo JD**, Zhang S, Leung JMG, Bogetti AT, Thompson JP, DeGrave AJ, Torrillo PA, Pratt AJ, Wong KF, Xia J, Copperman J, Adelman JL, Zwier MC, LeBard DN, Zuckerman DM, Chong LT. WESTPA 2.0: High-Performance Upgrades for Weighted Ensemble Simulations and Analysis of Longer-Timescale Applications. Journal of Chemical Theory and Computation. 2022; 18(2):638–649. https://doi.org/10.1021/acs.jctc.1c01154.

[3] **Huber GA**, Kim S. Weighted-ensemble Brownian dynamics simulations for protein association reactions. Biophysical Journal. 1996; 70(1):97–110. https://doi.org/10.1016/S0006-3495(96)79552-8.

[4] **Zuckerman DM**, Chong LT. Weighted Ensemble Simulation: Review of Methodology, Applications, and Software. Annual review of biophysics. 2017; 46:43–57. https://doi.org/10.1146/annurev-biophys-070816-033834.

[5] **Adhikari U**, Mostofian B, Copperman J, Subramanian SR, Petersen AA, Zuckerman DM. Computational Estimation of Microsecond to Second Atomistic Folding Times. Journal of the American Chemical Society. 2019; 141(16):6519–6526. https://doi.org/10.1021/jacs.8b10735.

[6] **Saglam AS**, Chong LT. Protein–protein binding pathways and calculations of rate constants using fully-continuous, explicit-solvent simulations. Chemical Science. 2019; 10(8):2360–2372. https://doi.org/10.1039/C8SC04811H.

[7] **Lotz SD**, Dickson A. Unbiased Molecular Dynamics of 11 min Timescale Drug Unbinding Reveals Transition State Stabilizing Interactions. Journal of the American Chemical Society. 2018; 140(2):618–628. https://doi.org/10.1021/jacs.7b08572.

[8] **Sztain T**, Ahn SH, Bogetti AT, Casalino L, Goldsmith JA, Seitz E, McCool RS, Kearns FL, Acosta-Reyes F, Maji S, Mashayekhi G, McCammon JA, Ourmazd A, Frank J, McLellan JS, Chong LT, Amaro RE. A glycan gate controls opening of the SARS-CoV-2 spike protein. Nature Chemistry. 2021; p. 1–6. https://doi.org/10.1038/s41557-021-00758-3.

[9] **Donovan RM**, Sedgewick AJ, Faeder JR, Zuckerman DM. Efficient stochastic simulation of chemical kinetics networks using a weighted ensemble of trajectories. The Journal of Chemical Physics. 2013; 139(11):115105. https://doi.org/10.1063/1.4821167.

[10] **Donovan RM**, Tapia JJ, Sullivan DP, Faeder JR, Murphy RF, Dittrich M, Zuckerman DM. Unbiased Rare Event Sampling in Spatial Stochastic Systems Biology Models Using a Weighted Ensemble of Trajectories. PLOS Computational Biology. 2016; 12(2):e1004611. https://doi.org/10.1371/journal.pcbi.1004611.

[11] **Ray D**, Stone SE, Andricioaei I. Markovian Weighted Ensemble Milestoning (M-WEM): Long-Time Kinetics from Short Trajectories. Journal of Chemical Theory and Computation. 2022; 18(1):79–95. https://doi.org/10.1021/acs.jctc.1c00803.

[12] **Faradjian AK**, Elber R. Computing time scales from reaction coordinates by milestoning. The Journal of Chemical Physics. 2004; 120(23):10880–10889. https://doi.org/10.1063/1.1738640.

[13] **West AMA**, Elber R, Shalloway D. Extending molecular dynamics time scales with milestoning: Example of complex kinetics in a solvated peptide. The Journal of Chemical Physics. 2007; 126(14):145104. https://doi.org/10.1063/1.2716389.

[14] **Harris LA**, Hogg JS, Tapia JJ, Sekar JAP, Gupta S, Korsunsky I, Arora A, Barua D, Sheehan RP, Faeder JR. BioNetGen 2.2: advances in rule-based modeling. Bioinformatics. 2016; 32(21):3366–3368. https://doi.org/10.1093/bioinformatics/btw469.

[15] **Tapia JJ**, Saglam AS, Czech J, Kuczewski R, Bartol TM, Sejnowski TJ, Faeder JR. MCell-R: A particle-resolution network-free spatial modeling framework. Methods in molecular biology (Clifton, NJ). 2019; 1945:203–229. https://doi.org/10.1007/978-1-4939-9102-0_9.

[16] **Zhang BW**, Jasnow D, Zuckerman DM. The "weighted ensemble" path sampling method is statistically exact for a broad class of stochastic processes and binning procedures. The Journal of Chemical Physics. 2010; 132(5):054107. https://doi.org/10.1063/1.3306345.

[17] **Donyapour N**, Roussey NM, Dickson A. REVO: Resampling of ensembles by variation optimization. The Journal of Chemical Physics. 2019; 150(24):244112. https://doi.org/10.1063/1.5100521.

[18] **Copperman J**, Zuckerman DM. Accelerated Estimation of Long-Timescale Kinetics from Weighted Ensemble Simulation via Non-Markovian "Microbin" Analysis. Journal of Chemical Theory and Computation. 2020; 16(11):6763–6775. https://doi.org/10.1021/acs.jctc.0c00273.

[19] **Torrillo PA**, Bogetti AT, Chong LT. A Minimal, Adaptive Binning Scheme for Weighted Ensemble Simulations. The Journal of Physical Chemistry A. 2021; 125(7):1642–1649. https://doi.org/10.1021/acs.jpca.0c10724.

[20] **DeGrave AJ**, Bogetti AT, Chong LT. The RED scheme: Rate-constant estimation from pre-steady state weighted ensemble simulations. The Journal of Chemical Physics. 2021; 154(11):114111. https://doi.org/10.1063/5.0041278.

[21] **Aristoff D**, Zuckerman DM. Optimizing Weighted Ensemble Sampling of Steady States. Multiscale Modeling & Simulation. 2020; 18(2):646–673. https://doi.org/10.1137/18M1212100.

[22] **Bhatt D**, Zhang BW, Zuckerman DM. Steady-state simulations using weighted ensemble path sampling. The Journal of Chemical Physics. 2010; 133(1):014110. https://doi.org/10.1063/1.3456985.

[23] **Zwier MC**, Kaus JW, Chong LT. Efficient Explicit-Solvent Molecular Dynamics Simulations of Molecular Association Kinetics: Methane/Methane, Na+/Cl-, Methane/Benzene, and K+/18-Crown-6 Ether. Journal of Chemical Theory and Computation. 2011; 7(4):1189–1197. https://doi.org/10.1021/ct100626x.

[24] **Zheng W**, Andrec M, Gallicchio E, Levy RM. Simulating replica exchange simulations of protein folding with a kinetic network model. Proceedings of the National Academy of Sciences. 2007; 104(39):15340–15345. https://doi.org/10.1073/pnas.0704418104.

[25] **Copperman J**, Aristoff D, Makarov DE, Simpson G, Zuckerman DM. Transient probability currents provide upper and lower bounds on non-equilibrium steady-state currents in the Smoluchowski picture. The Journal of Chemical Physics. 2019; 151(17):174108. https://doi.org/10.1063/1.5120511.

[26] **Zuckerman DM**, Discrete-state kinetics and Markov models;. http://physicallensonthecell.org/discrete-state-kinetics-and-markov-models.

[27] **Chong LT**, Saglam AS, Zuckerman DM. Path-sampling strategies for simulating rare events in biomolecular systems. Current Opinion in Structural Biology. 2017; 43:88–94. https://doi.org/10.1016/j.sbi.2016.11.019.

[28] **Mostofian B**, Zuckerman DM. Statistical Uncertainty Analysis for Small-Sample, High Log-Variance Data: Cautions for Bootstrapping and Bayesian Bootstrapping. Journal of Chemical Theory and Computation. 2019; 15(6):3499–3509. https://doi.org/10.1021/acs.jctc.9b00015.

[29] **Aristoff D**. Analysis and optimization of weighted ensemble sampling. ESAIM: Mathematical Modelling and Numerical Analysis. 2018; 52(4):1219–1238. https://doi.org/10.1051/m2an/2017046.

[30] **Zuckerman DM**, Woolf TB. Transition events in butane simulations: Similarities across models. The Journal of Chemical Physics. 2002; 116(6):2586–2591. https://doi.org/10.1063/1.1433501.

[31] **Zhang BW**, Jasnow D, Zuckerman DM. Efficient and verified simulation of a path ensemble for conformational change in a united-residue model of calmodulin. Proceedings of the National Academy of Sciences. 2007; 104(46):18043–18048. https://doi.org/10.1073/pnas.0706349104.

[32] **Case DA**, Aktulga HM, Belfon K, Ben-Shalom IY, Berryman JT, Brozell SR, Cerutti DS, III TEC, Cisneros GA, Cruzeiro VWD, Darden TA, Duke RE, Giambasu G, Gilson MK, Gohlke H, Goetz AW, Harris R, Izadi S, Izmailov SA, Kasavajhala K, et al., Amber 2022. University of California, San Francisco; 2022.

[33] **Eastman P**, Swails J, Chodera JD, McGibbon RT, Zhao Y, Beauchamp KA, Wang LP, Simmonett AC, Harrigan MP, Stern CD, Wiewiora RP, Brooks BR, Pande VS. OpenMM 7: Rapid development of high performance algorithms for molecular dynamics. PLOS Computational Biology. 2017; 13(7):e1005659. https://doi.org/10.1371/journal.pcbi.1005659.

[34] **Braun E**, Gilmer J, Mayes HB, Mobley DL, Monroe JI, Prasad S, Zuckerman DM. Best Practices for Foundations in Molecular Simulations [Article v1.0]. Living Journal of Computational Molecular Science. 2019; 1(1):5957–5957. https://doi.org/10.33011/livecoms.1.1.5957.

[35] **Zuckerman DM**, Russo JD. A gentle introduction to the nonequilibrium physics of trajectories: Theory, algorithms, and biomolecular applications. American Journal of Physics. 2021; 89(11):1048–1061. https://doi.org/10.1119/10.0005603.

[36] **Grossfield A**, Patrone PN, Roe DR, Schultz AJ, Siderius D, Zuckerman DM. Best Practices for Quantification of Uncertainty and Sampling Quality in Molecular Simulations [Article v1.0]. Living Journal of Computational Molecular Science. 2019; 1(1). https://doi.org/10.33011/livecoms.1.1.5067.

[37] **Zwier MC**, Pratt AJ, Adelman JL, Kaus JW, Zuckerman DM, Chong LT. Efficient Atomistic Simulation of Pathways and Calculation of Rate Constants for a Protein–Peptide Binding Process: Application to the MDM2 Protein and an Intrinsically Disordered p53 Peptide. The Journal of Physical Chemistry Letters. 2016; 7(17):3440–3445. https://doi.org/10.1021/acs.jpclett.6b01502.

[38] **Bogetti AT**, Mostofian B, Dickson A, Pratt AJ, Saglam AS, Harrison PO, Dudek M, Torrillo PA, DeGrave AJ, Adhikari U, Zweir MC, Zuckerman DM, Chong LT. A Suite of Tutorials for the WESTPA Rare-Events Sampling Software [Article v1.0]. Living Journal of Computational Molecular Science. 2019; 1(2):10607–10607. https://doi.org/10.33011/livecoms.1.2.10607.

[39] **Casalino L**, Dommer AC, Gaieb Z, Barros EP, Sztain T, Ahn SH, Trifan A, Brace A, Bogetti AT, Clyde A, Ma H, Lee H, Turilli M, Khalid S, Chong LT, Simmerling C, Hardy DJ, Maia JD, Phillips JC, Kurth T, et al. AI-driven multiscale simulations illuminate mechanisms of SARS-CoV-2 spike dynamics. The International Journal of High Performance Computing Applications. 2021; 35(5):432–451. https://doi.org/10.1177/10943420211006452.

[40] **Cerutti DS**, Duke R, Freddolino PL, Fan H, Lybrand TP. A Vulnerability in Popular Molecular Dynamics Packages Concerning Langevin and Andersen Dynamics. Journal of Chemical Theory and Computation. 2008; 4(10):1669–1680. https://doi.org/10.1021/ct8002173.

[41] **Pratt A**, Suarez E, Zuckerman D, Chong L. Extensive Evaluation of Weighted Ensemble Strategies for Calculating Rate Constants and Binding Affinities of Molecular Association/Dissociation Processes. bioRxiv. 2019; https://doi.org/10.1101/671172.

[42] **Joung IS**, Cheatham TE. Molecular Dynamics Simulations of the Dynamic and Energetic Properties of Alkali and Halide Ions Using Water-Model-Specific Ion Parameters. The Journal of Physical Chemistry B. 2009; 113(40):13279–13290. https://doi.org/10.1021/jp902584c.

[43] **Jorgensen WL**, Chandrasekhar J, Madura JD, Impey RW, Klein ML. Comparison of simple potential functions for simulating liquid water. The Journal of Chemical Physics. 1983; 79(2):926–935. https://doi.org/10.1063/1.445869.

[44] **Nguyen H**, Maier J, Huang H, Perrone V, Simmerling C. Folding Simulations for Proteins with Diverse Topologies Are Accessible in Days with a Physics-Based Force Field and Implicit Solvent. Journal of the American Chemical Society. 2014; 136(40):13959–13962. https://doi.org/10.1021/ja5032776.

[45] **Nguyen H**, Roe DR, Simmerling C. Improved Generalized Born Solvent Model Parameters for Protein Simulations. Journal of Chemical Theory and Computation. 2013; 9(4):2020–2034. https://doi.org/10.1021/ct3010485.

[46] **Kussie PH**, Gorina S, Marechal V, Elenbaas B, Moreau J, Levine AJ, Pavletich NP. Structure of the MDM2 Oncoprotein Bound to the p53 Tumor Suppressor Transactivation Domain. Science. 1996; 274(5289):948–953. https://doi.org/10.1126/science.274.5289.948.

[47] **Honda S**, Yamasaki K, Sawada Y, Morii H. 10 Residue Folded Peptide Designed by Segment Statistics. Structure. 2004; 12(8):1507–1518. https://doi.org/10.1016/j.str.2004.05.022.

[48] **Hill TL**. In: State Probabilities and Fluxes in Terms of the Rate Constants of the Diagram Springer New York; 1989. p. 39–88. https://doi.org/10.1007/978-1-4612-3558-3_2.

[49] **Michaud-Agrawal N**, Denning EJ, Woolf TB, Beckstein O. MDAnalysis: A toolkit for the analysis of molecular dynamics simulations. Journal of Computational Chemistry. 2011; 32(10):2319–2327. https://doi.org/10.1002/jcc.21787.

[50] **Gowers R**, Linke M, Barnoud J, Reddy T, Melo M, Seyler S, Domański J, Dotson D, Buchoux S, Kenney I, Beckstein O. MDAnalysis: A Python Package for the Rapid Analysis of Molecular Dynamics Simulations. In: *Python in Science Conference*; 2016. p. 98–105. https://doi.org/10.25080/Majora-629e541a-00e.

[51] **McGibbon RT**, Beauchamp KA, Harrigan MP, Klein C, Swails JM, Hernández CX, Schwantes CR, Wang LP, Lane TJ, Pande VS. MDTraj: A Modern Open Library for the Analysis of Molecular Dynamics Trajectories. Biophysical Journal. 2015; 109(8):1528 – 1532. https://doi.org/10.1016/j.bpj.2015.08.015.

[52] **Zhang S**, Thompson JP, Xia J, Bogetti AT, York F, Skillman AG, Chong LT, LeBard DN. Mechanistic Insights into Passive Membrane Permeability of Drug-like Molecules from a Weighted Ensemble of Trajectories. Journal of Chemical Information and Modeling. 2022; p. acs.jcim.1c01540. https://doi.org/10.1021/acs.jcim.1c01540.

[53] **Ray D**, Gokey T, Mobley DL, Andricioaei I. Kinetics and free energy of ligand dissociation using weighted ensemble milestoning. The Journal of Chemical Physics. 2020; 153(15):154117. https://doi.org/10.1063/5.0021953.

[54] **Ray D**, Andricioaei I. Weighted ensemble milestoning (WEM): A combined approach for rare event simulations. The Journal of Chemical Physics. 2020; 152(23):234114. https://doi.org/10.1063/5.0008028.

[55] **Vanden-Eijnden E**, Venturoli M. Markovian milestoning with Voronoi tessellations. The Journal of Chemical Physics. 2009; 130(19):194101. https://doi.org/10.1063/1.3129843.

[56] **Maragliano L**, Vanden-Eijnden E, Roux B. Free Energy and Kinetics of Conformational Transitions from Voronoi Tessellated Milestoning with Restraining Potentials. Journal of Chemical Theory and Computation. 2009; 5(10):2589–2594. https://doi.org/10.1021/ct900279z.

[57] **Izrailev S**, Stepaniants S, Isralewitz B, Kosztin D, Lu H, Molnar F, Wriggers W, Schulten K. Steered Molecular Dynamics. In: Griebel M, Keyes DE, Nieminen RM, Roose D, Schlick T, Deuflhard P, Hermans J, Leimkuhler B, Mark AE, Reich S, Skeel RD, editors. *Computational Molecular Dynamics: Challenges, Methods, Ideas*, vol. 4 Berlin, Heidelberg: Springer Berlin Heidelberg; 1999.p. 39–65. https://doi.org/10.1007/978-3-642-58360-5_2.

PROTOCOL FOR RUNNING WE SIMULATIONS

I Ready.

(1) **Organize files into directories.** The initial coordinate file(s) should go in `bstates` and the topology and MD input files should go in `common_files`. In addition, any scripts needed to calculate the progress coordinate should be placed in `common_files`. (**Section 7.1.3**)

(2) **Prepare the system environment.** Edit `env.sh` accordingly for your system environment, i.e. source any files needed to set up the dynamics engine and set variables equal to the full paths of programs (see **Section 7.1.3**).

(3) **Specify WE parameters.** In the `west.cfg` file, specify the number of WE iterations that will be carried out, progress coordinate, number of progress coordinate values that will be recorded per iteration, bin spacing, and number of trajectories per bin (see **Section 4**; **Table 2**). If a nested coordinate is desired, the bin spacing should be defined in `system.py` (See point 4 in **Section 4**). The τ value is specified in the main input file for dynamics propagation (in `common_files/`) through the total number of MD steps.

(4) **Set up calculations of auxiliary data.** Decide if you want to store any data that is auxiliary to the progress coordinate and add the corresponding datasets to `west.cfg`.

(5) **Specify whether to run equilibrium or steady-state WE.** Edit the `init.sh` file to include `$TSTATE_ARGS` if you plan to run steady-state WE. Also, edit `tstate.file` to include the target state progress coordinate value(s). (**Section 7.1.3**)

(6) **Calculate your initial progress coordinate value(s).** You can either set us this calculation manually, placing the contents in `pcoord.init` (see **Basic Tutorial 7.1**), or edit `get_pcoord.sh` to calculate it (see **Intermediate Tutorial 7.2**) before those values are passed to WESTPA. (**Section 7.1.3**)

II Set.

(1) **Initialize the WE simulation.** This is done by running `init.sh`.

(2) **Prepare to run the WE simulation.** Edit `runseg.sh` to run dynamics, calculate the progress coordinate and store any auxiliary data. (**Section 7.1.4**)

III Go!

(1) **Run the WE simulation.** To execute `w_run` on your cluster, run an appropriate submission script (e.g., using Slurm).

(2) **Monitor simulation progress.** Backup your `west.h5` file and use `w_pdist` and `plothist` to calculate and visualize probability distributions (**Section 7.1.6**). If your WE simulation is not making any bin-to-bin transitions stop the simulation and restart the simulation with a shorter τ, adjustments in the progress coordinate, and/or adjustments in bin spacings (**Tutorial 7.2.4**).

(3) **Assess simulation convergence.** Plot the average flux into the target state (or other observable of interest) as a function of WE iteration (**Section 7.1.7**).

IV Analyze.

(1) **Progress coordinate and auxiliary data.** Progress coordinate and auxiliary datasets are stored in the `west.h5` file and can be extracted using `hdfview` or the `h5py` Python package. The `w_ipa` tool can be used to calculate kinetic observables (e.g. rate constants), which are outputted to the `assign.h5` and `direct.h5 files`. (**Section 7.4.2**) To plot a dataset, use a plotting tool such as `matplotlib`.

(2) **Visualize a successful trajectory.** Start by running `w_succ` to obtain the WE iteration and segment of the final conformation from the successful trajectory (Section 7.1.6). Then, run `w_trace` to obtain the series of conformations in the trajectory by tracing backwards from the final to initial conformations of the trajectory. The resulting series of conformations can then be visualized using a software package such as VMD. (**Section 7.1.7**)

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655

CHECKLIST FOR TROUBLESHOOTING WE SIMULATIONS

Files for Dynamics Propagation

☐ Have you set up all of the files for propagating the dynamics using your dynamics engine (e.g. Amber, OpenMM)?

System Configuration (`west.cfg` file)

☐ Is `pcoord_len` set to the number of data points that corresponds to the frequency with which the dynamics engine outputs the progress coordinate? Note: Many MD engines (e.g. Gromacs) output the initial point (i.e. zero).
☐ Are the bins in the expected positions? You can easily view the positions of the bins using a Python interpreter.

Initializing the simulation (`init.sh` file)

☐ Is the directory structure for the trajectory output files consistent with specifications in the `west.cfg` file?
☐ Are the basis (`bstate`) states, and if applicable, target states (`tstate`), specified correctly?

Calculating the progress coordinate for initial states (`get_pcoord.sh` file)

☐ Ensure that the procedure to extract the progress coordinate works by calculating your progress coordinate manually for one (or more) basis state files.
☐ Examine structure(s) of the initial states using visualization software (e.g. VMD, PyMOL) to verify that the structure(s) match the progress coordinate in the H5 file

Segment implementation (`runseg.sh`)

☐ Ensure that the progress coordinate is being calculated correctly by manually running a single dynamics segment of length τ for a single trajectory walker. Check that your analysis pipeline works using the output from the single dynamics segment.
☐ Are you feeding the information (e.g., coordinates, velocities) that is required for continuing trajectories?
☐ Are you including the parent coordinate file at the beginning of your analysis?
☐ Are you imaging the trajectory before calculating the progress coordinate? Or equivalently using a function that can image your trajectory on-the-fly?
☐ Check the `seg_log` file to further ensure correct calculation of the progress coordinate.
☐ Ensure you are saving everything you might need to restart your simulation later on, including random seeds for stochastic dynamics as auxiliary data in the H5 file.

Storage

☐ WESTPA simulations can generate a lot of data! Make sure you have enough space before starting the simulation.
☐ Do you have a plan for backing up the simulation? Consider using tar to compress files from past iterations for easier backup. Keep file sizes =<300 GB.
☐ For all-atom explicit water simulations, it's a good idea to save a separate copy of your trajectories without water coordinates for more efficient analysis.

Simulation Progress (`west.h5` file)

☐ Check that the first WE iteration has been initialized by typing `h5ls west.h5/iterations` into the command line. You should see `iter_00000001` in the output.
☐ In addition, the progress coordinate should be initialized. Check this by using the command `h5ls -d west.h5/iterations/iter_00000001/pcoord`. If all is well, the output will show that the array is populated by zeros and the first point is the value calculated by `get_pcoord.sh`.

Analysis

☐ If you are running analysis on a shared computing resource, use the `--serial` flag with the analysis tool. Otherwise, many of the included tools default to parallel mode (e.g., `w_assign`), which will create as many Python threads as there are CPU cores available on your resource.

https://doi.org/10.33011/livecoms.5.1.1655
Living J. Comp. Mol. Sci. 2023, 5(1), 1655