ACS OMEGA

http://pubs.acs.org/journal/acsodf

Article

Wepy: A Flexible Software Framework for Simulating Rare Events with Weighted Ensemble Resampling

Samuel D. Lotz and Alex Dickson*

Cite This: *ACS Omega* 2020, 5, 31608–31623

Read Online

ACCESS | Metrics & More | Article Recommendations | Supporting Information

ABSTRACT: Here, we introduce the open-source software framework wepy (https://github.com/ADicksonLab/wepy) which is a toolkit for running and analyzing weighted ensemble (WE) simulations. The wepy toolkit is in pure Python and as such is highly portable and extensible, making it an excellent platform to develop and use new WE resampling algorithms such as WExplore, REVO, and others while leveraging the entire Python ecosystem. In addition, wepy simplifies WE-specific analyses by defining out-of-core tree-like data structures using the cross-platform HDF5 file format. In this paper, we discuss the motivations and challenges for simulating rare events in biomolecular systems. As has previously been shown, high-dimensional WE resampling algorithms such as WExplore and REVO have been successful at these tasks, especially for rare events that are difficult to describe by one or two collective variables. We explain in detail how wepy facilitates implementation of these algorithms, as well as aids in analyzing the unique structure of WE simulation results. To explain how wepy and WE work in general, we describe the mathematical formalism of WE, an overview of the architecture of wepy, and provide code examples of how to construct, run, and analyze simulation results for a protein−ligand system (T4 Lysozyme in an implicit solvent). This paper is written with a variety of readers in mind, including (1) those curious about how to leverage WE rare-event simulations for their domain, (2) current WE users who want to begin using new high-dimensional resamplers such as WExplore and REVO, and (3) expert users who would like to prototype or implement their own algorithms that can be easily adopted by others.

1. INTRODUCTION

Biomolecular simulation is a valuable tool to gain insight into the atomic-level mechanisms of biological systems. Molecular processes such as ligand (un)binding, protein (un)folding, protein−protein association/dissociation, conformational changes, transport, and enzymatic reactions underlie the functioning of all living organisms. Because the length scales of these phenomena are too small to be observed experimentally at atomic resolution, computer simulations have long attempted to serve as a proxy. This application of molecular dynamics (MD) has been limited by two factors: the accuracy of force fields and available computational power. Despite huge breakthroughs in both force fields[1,2] and hardware,[3−5] it is still difficult to perform simulations of sufficient length to capture the biomolecular processes of interest. For instance, the waiting time for ligand-unbinding events of pharmacologically relevant ligands can extend to tens of minutes or hours,[6] while MD trajectories are typically limited to microsecond timescales. Thus, despite our knowledge (from experiment) that these transitions occur readily at macroscopic timescales, the occurrence of these events during a microsecond molecular simulation can be seen as a "rare event".

A series of algorithms have been developed called "enhanced sampling" methods, which attempt to gain information about long-timescale processes using only short-timescale simulations. Many of these methods use perturbations to the system by either applying biasing forces or increasing the system temperature. This includes enhanced sampling methods such as replica exchange,[7] metadynamics,[8−11] temperature-accelerated MD,[12,13] and umbrella sampling.[14] These methods invoke an assumption that the molecular systems are in equilibrium, usually in the form of a canonical probability density function. Additionally, although there are some approaches to approximate rates based on biased simulations,[8] this approach makes it very difficult to recover full atomistic detail of transition-state ensembles that govern the forward and backward rate constants for a given transition.

There is thus a need for a method for accelerating the simulation of rare-event processes that generate trajectories using the unbiased Hamiltonian of interest. One such class of methods is called "path sampling", which are unique in that they apply a sampling process over a collection of trajectories rather than single conformations.[15] Importantly, they do not require modifications to the underlying dynamics model and provide

Received: August 12, 2020
Accepted: October 15, 2020
Published: December 2, 2020

contiguous trajectories of transition paths. This makes them suitable for studying all varieties of path-dependent observables [*e.g.,* (un)binding rates], as well as providing detailed atomistic models of transition states. There are a variety of path-sampling methods including transition path sampling (TPS),[16−18] forward flux sampling (FFS),[19] multilevel splitting,[20] and weighted ensemble (WE).[15,21] Of these, WE has advantages in which it does not in general require a Markovian assumption or *a priori* knowledge of full trajectories connecting two states to start[15,22] and does not require the definition of a progress coordinate, which will be discussed further below. At the heart of WE is a **resampling** process that uses a set of cloning and merging operations among a set of parallel simulation replicas. More formal definitions of resampling and WE are discussed in Section 3.1 as well as in refs.[15,22,23] WE is a conceptually portable method that can be applied to any field of study including molecular biophysics,[23−33] systems biology,[34−41] telecommunications,[42] and aerospace and engineering.[43]

Another benefit of WE is the flexibility of its framework, allowing for resampling methods that avoid the predefinition of progress coordinates and collective variables (CVs).[20,44−46] Approaches involving CVs are typically limited to low-dimensional representations (often ≤3), while WE-based resamplers such as WExplore[45] and REVO[46] perform well in high-dimensional spaces. High-dimensional adaptive resampling algorithms have been especially successful in obtaining preliminary rare-event simulations for systems with waiting times well beyond what is typical.[33,41,47−49] Notably, WExplore simulations produced unbinding trajectories of a drug ligand (TPPU) from its target (soluble epoxide hydrolase) that has an experimentally determined mean first passage time of 11 min, using less than 1 μs of simulation and a speedup of 10^9-fold.[48a] WExplore and REVO have shown to be particularly useful for discovering multiple pathways. Both algorithms were able to discover multiple ligand dissociation pathways for the trypsin−benzamidine system, which requires substantial rearrangement of the loops comprising the trypsin ligand-binding pocket.[46,47] In contrast, single and low-dimensional projections such as reaction coordinates often constrict the search space to particular paths, which precludes the discovery of alternative paths (and transition states) between macrostates. Finding multiple pathways can be particularly useful for applications such as kinetics-based drug design when we want to understand the structure of the ligand-binding transition state not only for a particular ligand but also for closely related ligands. Finally, adaptive algorithms such as WExplore and REVO (as well as the history-augmented Markov state model WE method[50]) require less upfront parametrization such as the definition of bin boundaries.

Despite the advantages of high-dimensional adaptive WE algorithms such as WExplore and REVO, their adoption has been hindered for a number of reasons. First, the implementation of these resampling algorithms is complex and difficult to implement correctly. Second, independent implementations of WE algorithms lack interoperability of produced data and so are difficult to compare. Third, the barrier to entry for other researchers to write an implementation of the resampling algorithm as well as progress metrics for their system of interest is prohibitive.

Here, we introduce the open-source wepy software framework for running WE simulations that attempts to address these issues. We first describe a software and data architecture that both reflect a simple mathematical formalism (described in 3.1) and also decompose into multiple modular components. The software architecture allows for reuse of vetted resampling algorithm implementations written by researchers with domain-specific progress metrics written by users. The data architecture solves interoperability through the introduction of a general purpose decision record design (described in the Appendix).

Wepy is implemented in the Python 3 programming language and thus allows users to natively leverage a massive ecosystem for scientific computing. Other benefits of a pure-Python implementation are that it (1) increases portability between platforms, (2) has a uniform interface that can be used as a library and embedded into other software easily, and (3) only requires knowledge of a single popular programming language (Python), which if necessary has facilities for writing extremely high performance code (*e.g.,* numba,[51] dask[52]). Currently, wepy is tightly integrated with the OpenMM[53] MD engine and provides excellent support for running GPU MD simulations. The architecture of wepy, however, is agnostic to the underlying dynamics engine as well as to any particular parallel computing strategy or framework. The wepy project also introduces a high-performance single-file storage format and schema for cloning-merging-type simulations implemented in HDF5.[54b] Use of HDF5 also provides "out-of-core" data structures which allow access to simulation data that do not fit entirely into computer main memory. On top of this, an extensive interface [application programmer interface (API)] is provided to make querying, analysis, conversion to other formats of complex path trajectories easy. Part of this interface is to support the intuitive representation of WE trajectories which have been cloned and merged as trees (referred to as "tree-like" data structures).

Although there are two other WE frameworks that have been developed, these have different strengths and design goals. The AWE-WQ[23] system provides an implementation of the accelerated weighted ensemble (AWE) along with a "Work Queue"-distributed computing framework. However, AWE-WQ is less flexible than wepy in that it solely implements AWE simulations and is opinionated about the distributed computing framework. The WESTPA[55] software suite is a popular implementation of many binning-based WE resamplers and provides excellent support for running simulations on large clusters of CPU cores and GPUs. It is implemented in Python and typically run using a combination of UNIX-like shell scripts and YAML configuration files. It shares many of the same benefits of wepy discussed above including modularity and the use of HDF5 files for simulation data. WESTPA however did not satisfy our requirements because the architecture is oriented around fixed-topology explicit binning approaches making implementation of binless algorithms such as REVO[46] difficult. Additionally, we also favor the configuration-as-code approach where simulation components are constructed in Python code.

In this paper, we first briefly describe WE, resampling, and rate calculations along with a sketch of some of the adaptive resamplers that motivated the construction of wepy. We then introduce a mathematical formalism that provides an overview of the design and architecture of the system followed by a description of the major software components in wepy including how to initialize, run, and analyze simulations. Finally, we present an example ligand-unbinding scenario, using a Lysozyme model system, along with concrete code examples and explanations.

https://dx.doi.org/10.1021/acsomega.0c03892
ACS Omega 2020, 5, 31608−31623

2. ALGORITHMS

2.1. Weighted Ensemble Algorithm. The WE algorithm is a general strategy for simulating rare or long-timescale events in stochastic systems.[21] Its fundamental features are as follows. A set of trajectories are propagated forward in time in a parallel fashion, each one assigned a statistical weight (p). Periodically, the trajectories are "resampled" to rebalance the computational effort toward lower-probability regions. This is done by cloning trajectories in sparsely populated, lower-probability regions, and merging together trajectories in overpopulated, higher-probability regions. To maintain proper statistics, the weight of a cloned trajectory is divided among its progeny, and the weights of merged trajectories are summed and given to the resulting trajectory. Here, we refer to this process as "resampling".

It was shown by Zhang *et al.*[44] that this is a statistically exact process in that ensemble averages were obtained with WE converge to those of the underlying distribution (*e.g.*, the canonical distribution) in the limit of large sampling, regardless of the particular resampling function. One of the first strategies for resampling is a simple binning of trajectories along a progress coordinate. Trajectories are then cloned in under-represented bins and merged in over-represented bins until the count in each bin is equal to a target value. However, because of the combinatorial explosion for tessellating high-dimensional search spaces with uniform bins, algorithms such as WExplore and REVO were developed to effectively leverage the WE algorithm for processes that take place in inherently high-dimensional spaces.

2.2. WExplore Resampler. The main problem with defining bins in a high-dimensional space is that the number of bins needed to cover the space scales exponentially with the dimensionality of a space. This number of bins is proportional to the overall computational effort of the simulation, as a target number of trajectories are to be run in each bin. The WExplore algorithm was developed by Dickson and Brooks to circumvent this difficulty.[45] The key is to construct a set of hierarchical regions: a small set of large regions that tile the entire space, which are each subdivided by a set of smaller regions, which are in turn subdivided by even smaller regions. In the WExplore resampler, these regions are Voronoi polyhedra that are defined by points (called "images") in a high-dimensional space. In order to assign a trajectory to a region, we must measure the distance from that trajectory state to each image. The trajectory is then assigned to the region with the closest image. More information on this algorithm can be found in the original paper.[45]

In a typical WExplore simulation, we start with only a single image and define new images as they are visited by the set of trajectories. A new image is defined when a trajectory in the ensemble reaches a new region of space, or more precisely, when the distance to the closest image is greater than a critical value. The list of critical values ($d_{\min} = (d_1, d_2, ..., d_n)$) thus control the sizes of the regions at each level of the hierarchy and are parameters of the WExplore resampler. As the set of images grows over the course of the simulation, the WExplore resampling function (denoted $\mathcal{R}$ below) can change with each resampling step.

2.3. REVO Resampler. Although WExplore presents a viable means of indexing tessellations of high-dimensional spaces, there are still some behaviors related to the construction of regions that are nonoptimal—most notably, the discontinuous behavior related to reaching new levels of the hierarchy for the first time. For this reason, a new resampling algorithm was recently proposed which avoids the construction of sampling regions altogether. In the REVO algorithm (resampling of ensembles by variation optimization), an objective quantity called the "trajectory variation" (denoted, V) is used to guide the merging and cloning process.[33,46] We calculate V before and after each proposed merging and cloning operation and execute only the operations that cause V to increase.

The variation is given by

$$V = \sum_i \sum_j \left(\frac{d(X_i, X_j)}{d_0} \right)^{\alpha} \phi(X_i)\phi(X_j) \tag{1}$$

where $d(X_i, X_j)$ represents the distance between trajectories i and j and $\phi(X)$ is a function that describes the importance of individual trajectories. The exponent α allows us to balance the relative strength of the distances and the importance functions, and the d_0 value does not affect resampling but serves to keep the variation function unitless. On the whole, the variation function increases as the ensemble of trajectories gets further apart.

The importance functions can be defined to take the probabilistic weight of the trajectories into account

$$\phi(X_i) = \log p_i + C \tag{2}$$

where C is a constant. This has the effect of prioritizing not only a broad ensemble of trajectories but one where the highest weighted trajectories are distributed as far as possible from each other. This strategy can minimize the error in the calculation of observables that depend on the weights of rare trajectories, such as transition rates.

2.4. Calculating Transition Rates Using Boundary Conditions. The wepy software supports the construction of nonequilibrium ensembles to calculate transition rates. In this technique,[56−58] a set of history-dependent ensembles are defined using a set of "basins". For instance, if we are studying ligand-binding transition pathways, then the basins will be the "bound" and the "unbound" states. The **unbinding ensemble** is then defined as the set of trajectories that have most recently visited the bound state. When a trajectory from the unbinding ensemble enters the unbound state, it transitions to the **binding ensemble**. In a typical wepy simulation, we can initialize trajectories in a given basin and use boundary conditions that terminate trajectories that enter the opposite basin. The weights of these trajectories are used to calculate a transition flux (probability per unit time), which is used to calculate a rate constant (*e.g.*, k_{on} or k_{off}). These trajectories are then "warped" back to the initial state, retaining the same statistical weight.

3. DESIGN AND ARCHITECTURE

3.1. Formalism. Let us first define ensemble resampling simulations in general. First, we define an ensemble to be a finite multiset of walkers, w, of size N

$$\boldsymbol{W} = \{w_i\}, \qquad \text{for } i = 0, 1, ..., N$$

where a walker is a set of two elements: a probabilistic weight, p, and a state, X

$$w_i = (p_i, X_i)$$

An ensemble of walkers defines a probability distribution, $P(X)$, and must be normalized such that

$$\sum_{0<i<N} p_i = 1$$

https://dx.doi.org/10.1021/acsomega.0c03892
ACS Omega 2020, 5, 31608−31623

There are at least two steps in an ensemble resampling simulation: propagating dynamics and resampling. We can also introduce a third step for the so-called boundary conditions that allow for running nonequilibrium simulations and calculation of rate constants. An ensemble resampling simulation process, $\mathcal{A}$ (for "apparatus"), is made up of three components: a runner function $\mathcal{D}$, a boundary condition function $\mathcal{B}$, and a resampling function $\mathcal{R}$.

$$\mathcal{A} = (\mathcal{D}, \mathcal{B}, \mathcal{R})$$

The dynamics can be any stochastic dynamical process such as MD, Monte Carlo simulations, and so forth. Formally, a runner function has the form

$$\mathcal{D}[\tau](X) \rightarrow X'$$

where $\mathcal{D}[\tau]$ is a stochastic function such that multiple evaluations of an input state X might not yield identical X's and τ is the number of steps of propagation. Although $\mathcal{D}$ acts independently on each walker state X, it is convenient to write it as

$$\mathcal{D}[\tau](\boldsymbol{W}) \rightarrow \boldsymbol{W}'$$

The resampling function, $\mathcal{R}$, maps an ensemble, $\boldsymbol{W}$, to another ensemble $\boldsymbol{W}'$ along with an updated resampling function $\mathcal{R}'$

$$\mathcal{R}(\boldsymbol{W}) \rightarrow (\mathcal{R}', \boldsymbol{W}')$$

The walkers in this new ensemble must satisfy these constraints

$$\begin{aligned} \boldsymbol{W}' &= \{(X_i, p_i'): i = 0, 1, \ldots N'\} \\ &\text{where} \quad N' \text{ is a postive integer} \\ & X_i \subseteq \{\mathbf{State}(w_j) \text{ for } w_j \text{ in } \boldsymbol{W}\} \\ & \sum_{0<i<N'} p_i' = 1 \end{aligned} \tag{3}$$

In words, $\boldsymbol{W}'$ consists of walkers with states chosen from the states of the $\boldsymbol{W}$ ensemble. Note that these constraints are necessary but not sufficient to ensure that the sampled ensembles are consistent with the cloning and merging process in the WE algorithm.

The updated resampler $\mathcal{R}'$ allows for algorithms that take into account history dependence. Thus, a resampling function

$$\mathcal{R}(\boldsymbol{W}) \rightarrow (\mathcal{R}, \boldsymbol{W}')$$

is said to be stateless and does not take into account any walker history.

Although more commonly stateless, boundary conditions and runners also have equivalent history-dependent definitions

$$\mathcal{B}(\boldsymbol{W}) \rightarrow (\mathcal{B}', \boldsymbol{W}'')$$

$$\mathcal{D}[\tau](\boldsymbol{W}) \rightarrow (\mathcal{D}', \boldsymbol{W}'')$$

These functions typically utilize an application strategy that follows

$$\mathcal{R}_0(\boldsymbol{W}_0) \rightarrow (\mathcal{R}_1, \boldsymbol{W}_1)$$

$$\mathcal{R}_1(\boldsymbol{W}_1) \rightarrow (\mathcal{R}_2, \boldsymbol{W}_2)$$

$$\vdots$$

$$\mathcal{R}_{n-1}(\boldsymbol{W}_{n-1}) \rightarrow (\mathcal{R}_n, \boldsymbol{W}_n)$$

To simplify this, we write it as

$$\mathcal{R}^{[n]}(\boldsymbol{W}_0) \rightarrow \boldsymbol{W}_n$$

Similarly, a single **cycle** of simulation is the application of the three components of the apparatus $\mathcal{A} = (\mathcal{D}[\tau], \mathcal{B}, \mathcal{R})$ to an initial ensemble, $\boldsymbol{W}_0$

$$\mathcal{D}[\tau](\boldsymbol{W}_0) \rightarrow \boldsymbol{W}'$$

$$\mathcal{B}(\boldsymbol{W}') \rightarrow \boldsymbol{W}''$$

$$\mathcal{R}(\boldsymbol{W}'') \rightarrow \boldsymbol{W}_1$$

This interleaved application of apparatus components can be simplified to

$$\mathcal{A}(\boldsymbol{W}_0) \rightarrow (\mathcal{A}', \boldsymbol{W}_1)$$

where $\mathcal{A}' = (\mathcal{D}', \mathcal{B}', \mathcal{R}')$ which can easily adopt the notation mentioned above for n cycles applied to the initial walkers

$$\mathcal{A}^{[n]}(\boldsymbol{W}_0) \rightarrow \boldsymbol{W}_n$$

A final useful construct is the **snapshot** which is the complete state of a simulation

$$\mathcal{K} = (\mathcal{A}, \boldsymbol{W})$$

3.2. Software Components. In this section, we describe concretely the software components that make up the wepy framework and how they are integrated. We begin by describing the **simulation manager**, which implements the apparatus described above and handles the reporting of output to the user. We then discuss some unique features of WE algorithms that can lead to challenges during data analysis and tools provided by wepy that make this analysis easier.

3.2.1. Building and Running Simulations. To run simulations, we need two things: an ensemble of initial walkers and a simulation manager. A simulation manager is an object that contains all of the apparatus necessary to move the walkers forward.

The flowchart in Figure 1 describes the functioning of a simulation manager acting on a set of initial walkers. First, the input walkers have their states propagated by the **runner**, which can be any sort of dynamics engine. The initial release of wepy supports OpenMM as an MD engine, as well as experimental support for other engines such as NAMD. Because the runner propagation is typically extremely compute-intensive, we also provide another **work mapper** component, which can be customized for different computing environments. Wepy includes a serial implementation that can be used for testing on a single core or device, as well as two additional work mappers that use the python built-in multiprocessing module that are suitable for simulations using hardware on a single node. Because the work mapper has been factored out of the implementation of the runner itself, these mappers will work for all different MD engines with little modification.

After propagation by the runner, each walker is tested by the boundary condition function to see if it has met the criteria. If any walker satisfies the boundary condition, the function is free to modify the state of those walkers; this is called a **warping** event. In addition to the new walkers, a set of warping records are produced that describe how the warping event(s) occurred. These records are then made available for generating the final data structure and any other report that requests them. In wepy, two kinds of warping events are recognized: **continuous** and **discontinuous**. A discontinuous warping event is one where dynamical variables of the walker are modified, for example,

https://dx.doi.org/10.1021/acsomega.0c03892
ACS Omega 2020, 5, 31608−31623

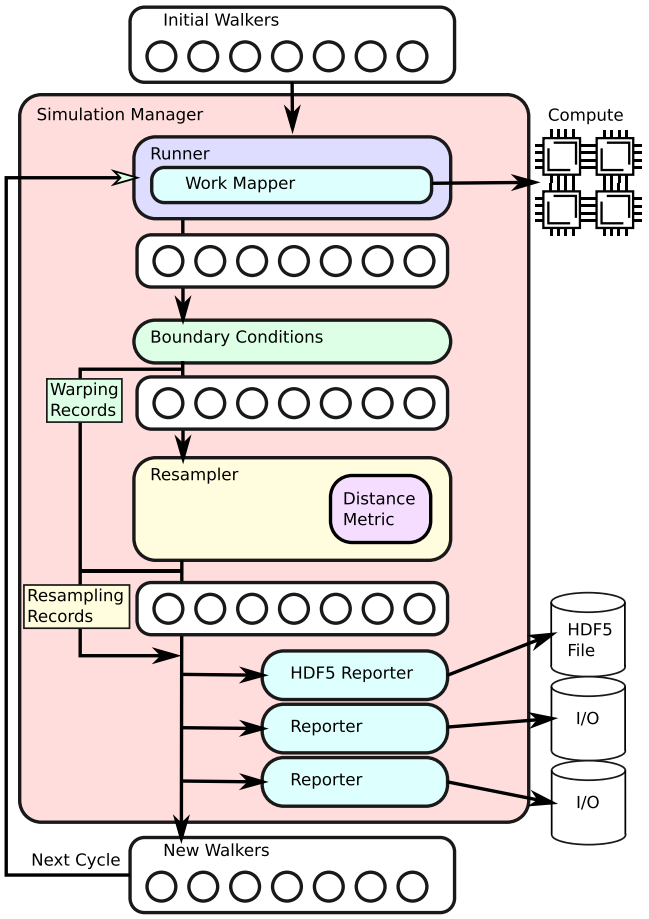

Figure 1. Diagram showing the components and flow of data for the main simulation loop of the simulation manager. Dynamics are performed in the runner component and parallelized onto the compute nodes *via* the work mapper. The warping and resampling records are shown in the data flow, which are serialized and saved using an HFD5 reporter. Other reporters can be defined to record simulation data, which is indicated by the I/O elements.

restarting walkers at the initial state after reaching a target state. A continuous warping event is one in which none of those dynamical variables are modified. This could be an auxiliary attribute such as a "color" after a walker passes through some boundary[59] or none at all in which case only a record will be produced.

After the boundary conditions are applied, the resampler resamples the walkers. Again, a collection of records is produced for the resampling; however, unlike the warping records, a record for each walker is produced every cycle. These resampling records contain critical information about the lineages of walker states that is necessary for reconstructing continuous trajectories. It is useful to use a diagram to depict the histories of each walker as they pass through these stages in a single cycle, an example of which we have shown in Figure 2. Finally, at the end of a cycle, the walkers and records are passed off to an arbitrary number of **reporters** which can record whatever output is necessary or desired.

Although reporters can be customized for a particular simulation, there are a number of useful reporters included with wepy. The WepyHDF5 reporter generates HDF5 output files, which are a major component of wepy and will be discussed on their own later in relation to analysis. The rest of the reporters are designed to give real-time insights to potentially long-

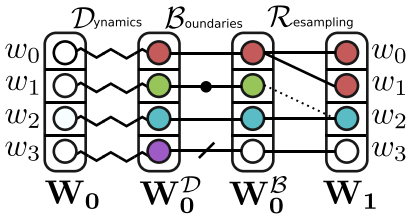

Figure 2. Walker history schematic. The vertically stacked boxes are the ensembles of walkers with labels below. The index of the "box" is the walker index in that ensemble. The color of the circle inside the box represents the state of the walker, where we treat white as the initial state of the simulation. The jagged lines indicate propagation of the walker state through the dynamics of the runner. The lines for the boundary condition step indicate whether a warping event occurred, where the dot indicates that a continuous warp occurred and the slash indicates a discontinuous warp, in this case, returning the walker state to the initial conditions. The lines in the resampling phase show cloning and merging, where a solid line indicates that a child inherits the state of its parent and a dashed line indicates a transfer of weight to another walker. In this example, w_0 has cloned itself to produce two child walkers while w_1 has been "squashed" and its state forgotten. It is "merged" and its weight was added to the resultant w_2.

running simulations as well as to give an accessible summary of the simulation results. The dashboard is an executive summary of the simulation provided in a simple plaintext file that is formatted in emacs org-mode that makes it easy to read a potentially large output file in a hierarchical folding manner. The walker ensemble reporter is for visualizing the current state of the walkers that simply outputs a reference topology file and a DCD trajectory file that can be used for visualizing in any of the main 3D molecular visualization programs, while these snapshots of simulations show only transient information that they are useful to indicate rough sketches of progress or for debugging purposes. Finally, the resampling tree output will generate a GEXF formatted XML file that shows the "family tree" of all walkers in the simulation. This is useful for helping understand both when and where resampling and warping events are taking place. These reporters are provided only for convenience as all of this information can be generated from WepyHDF5 files.

3.2.2. Data Format and Analysis. The WE algorithm is built on branching trajectories: simulations that have a single starting point may have multiple ending points. We call these kinds of tree-like branching trajectories **nonlinear** and a diagram of the difference between them and more traditional linear trajectories can be seen in Figure 3. In wepy, we call a locally linear selection of frames from the entire nonlinear dataset a **trace**. These are shown in Figure 3 as the solid colored lines drawn next to the frames they encompass. Nonlinear trajectories introduce additional complexity and typically require modifications to common time-dependent analyses.

First, visualizing a single linear trajectory requires a selection step and the input of information. For instance, in Figure 3, the single red and blue traces for the linear trajectories are trivial, whereas there is some redundancy in the nonlinear trajectories from the tree. In a nonlinear tree, we can choose a frame for which we are interested in its history and recover its **lineage** as a trajectory. These linear trajectories can then be exported to a common format (*e.g.,* CHARMM DCD) and visualized.

Second, state network models such as Markov State models (MSMs) are a common method for representing and understanding large amounts of simulation data. This is a perfect

https://dx.doi.org/10.1021/acsomega.0c03892
ACS Omega 2020, 5, 31608−31623

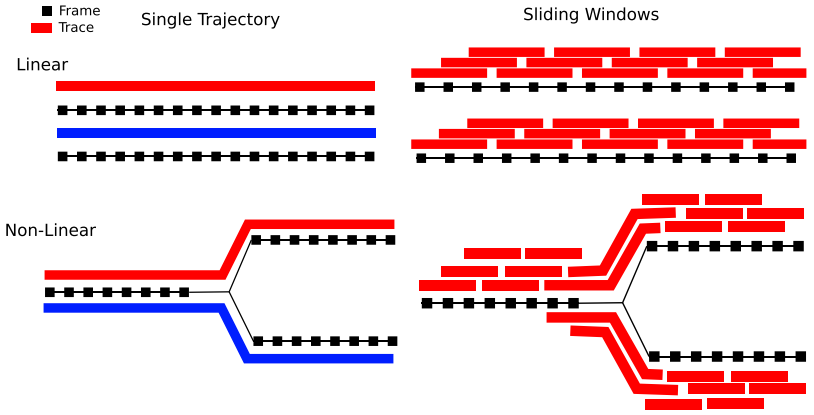

Figure 3. Nonlinear trajectory comparison. The black squares indicate single frames which are connected in time by the thin black lines. The larger lines indicate a locally linear trajectory, here called a "trace".

match for WE simulations, as the trajectory segments are typically generated using unbiased dynamics. However, current tools for constructing MSMs do not support construction of the transition matrices from nonlinear trajectory trees. Fundamentally, the problem involves avoiding the double-counting frames when counting the transitions for lag times greater than our cycle length (τ). A comparison for the solution for the nonlinear case is compared to the linear case in Figure 3, which is implemented in wepy.

Third, when calculating free energies of macrostates, the weights of trajectory observations must be taken into account. This simply amounts to doing weighted sums rather than counts for macrostate bins and requires careful association of trajectory frames to the instantaneous values of trajectory weights.

Finally, observations in different trajectories cannot be assumed to be statistically independent. Take for example, a set of observed warping events take place where a ligand molecule reaches the threshold of unbinding for a receptor. Each single instance contributes to the overall rate of unbinding proportional to its weight *via* the Hill relation.[15] However, the calculation of macroscopic observable uncertainties (such as the probability of an unbinding event) is complicated by the duplication of single observations *via* cloning. The degree to which two observations can be seen as "independent" depends on the time point of their last common ancestor.

To deal with these fundamental differences in the abstract structure of data generated by a WE simulation, wepy implements a new mechanism for storing trajectory data. To accomplish this, we have designed and implemented a storage layer using the common HDF5 format, which is suited to storing large amounts of heterogeneous data. In this implementation, all data results from a simulation are contained in a single monolithic binary file. Although the data can be accessed with any tool that supports HDF5, in wepy, we provide an extensive API for creating, accessing, querying, processing, and transforming the data at a useful semantic level.

In the WepyHDF5 format, we bundle together the three critical and interdependent data pieces into a single file: walker data (including weights and states), resampling data, and boundary condition warping data. This combination is what allows us to generate views of linear trajectories from nonlinear data. The resampling data inform the parent-child relationships between walkers and the warping data alerts to the presence of discontinuities in dynamics of these walker lineages. These primitives solve most of the major problems listed above for dealing with nonlinear trajectories with tools provided in wepy.

4. RESULTS

In this section, we first discuss code examples on how to set up, run (Section 4.1), and analyze (Section 4.2) a simulation for the test system used in this paper: Lysozyme ligand unbinding in an implicit solvent. We then describe all the parameters and details about the simulations that were run, the results of which are briefly discussed. This small experiment is shown to give an example of the kinds of results and analysis that are typical for wepy simulations.

4.1. Code for Running Simulations. Here, we give a brief sketch of how these components were constructed and put together into a simulation manager in a Python script. The complete code is given in the Supporting Information. The system and parameter choices will be discussed in Section 4.3.

We first need to set up our wepy runner for OpenMM, which requires an OpenMM system and integrator objects. Using OpenMM-Systems helper library (installed with wepy), we can easily create a ready-to-go MD system.

```
from openmm_systems.testsystems \
    import LysozymeImplicit
test_sys = LysozymeImplicit()
```

The integrator can be constructed using the OpenMM constructor

```
from simtk.openmm \
    import LangevinIntegrator
from simtk.unit import \
        kelvin, picosecond

integrator = LangevinIntegrator(
                300.0*kelvin,
                1/picosecond,
                0.002*picosecond)
```

where the three arguments are the temperature, friction coefficient, and dynamics time step, respectively. We can then create a runner object that contains everything needed to propagate the system, where the reference platform specifies a cross-platform reference implementation on a CPU. Note that our production simulations for this work were run with the CUDA platform.

https://dx.doi.org/10.1021/acsomega.0c03892
ACS Omega 2020, 5, 31608−31623

```
from wepy.runners.openmm \
    import OpenMMRunner

runner = OpenMMRunner(
             test_sys.system,
             test_sys.topology,
             integrator,
             platform='Reference')
```

Second, we need to create the initial ensemble of walker objects from which to start our simulations. We use a wepy helper function gen_walker_state to generate state objects directly from OpenMM systems.

```
from wepy.runners.openmm \
    import gen_walker_state
init_state = gen_walker_state(
               test_sys.positions,
               test_sys.system,
               integrator)
```

init_state is a WalkerState object which can be put inside a walker. Because we are using a Langevin integrator which has a stochastic component (required by all WE simulations), we can copy the same structure for all starting replicas. Each worker manually has a weight assigned to them; in this case, it is a uniform distribution.

```
from wepy.walker import Walker
from copy import copy

n_walkers = 48
init_weight = 1.0 / n_walkers

init_walkers = []
for i in range(n_walkers):
    walker = Walker(
        copy(init_state),
        init_weight)
    init_walkers.append(walker)
```

Third, we construct a resampler to use. Both the WExplore and REVO resamplers require a metric, which is a way of measuring the distance between two walkers. Details regarding the distance metrics are discussed in more detail in Appendix 6.2. For receptor ligand-based systems, there are distance metrics already included in wepy:

```
import wepy.resampling.distances \
    as wepy_dists
from wepy_dists.receptor \
    import UnbindingDistance

distance = \
    UnbindingDistance(
        ligand_idxs=lig_idxs,
        binding_site_idxs=bs_idxs,
        ref_state=init_state)
```

where lig_idxs and bs_idxs are the atomic indices of the ligand and the binding site in the system. A user can also easily make their own distance metrics; a recipe for doing this is shown below

```
import wepy.resampling.distances \
    as wepy_dists

from wepy_dists import Distance \
    as Dist

from geomm.rmsd import calc_rmsd
from geomm.superimpose \
    import superimpose

class MyUnbindingDistance(Dist):

    def __init__(self,
        ligand_idxs,
        binding_site_idxs):

        self.lig_idxs = \
            ligand_idxs
        self.bs_idxs = \
            binding_site_idxs

    def image_distance(self,
                       state_a,
                       state_b):

        sup_b = superimpose(
            state_a['positions'],
            state_b['positions'],
            idxs=self.bs_idxs)

        lig_rmsd = calc_rmsd(
            state_a['positions'],
            sup_b,
            idxs=self.lig_idxs)

        return lig_rmsd
```

The dependencies here are the distance base class and some helper functions for performing geometric operations on arrays from our custom library geomm. The distance object is created in a similar way, except that we do not need the reference state (which is needed in UnbindingDistance for performance reasons):

```
my_distance = MyUnbindingDistance(
                        lig_idxs,
                        bs_idxs)
```

Fourth, we construct the resampler that we are going to use. Here, we create an instance of the WExploreResampler using the distance and init_state objects that we created earlier as well as some additional algorithm parameters (as discussed in Section 2.2)

```
import wepy.resampling.resamplers \
    as wepy_resamplers
from wepy_resamplers.wexplore \
    import WExploreResampler

NUM_REG = (10, 10, 10, 10)
REG_SIZE = (1, 0.5, 0.35, 0.25)
resampler = WExploreResampler(
          init_state=init_state,
          distance=distance,
          max_n_regions=NUM_REG,
          max_region_sizes=REG_SIZE,
          pmin=1e-12,
          pmax=0.5)
```

Fifth, to set up a nonequilibrium ligand-unbinding simulation, we will construct boundary conditions that capture walkers as they cross into the unbound state. Wepy also comes with some

https://dx.doi.org/10.1021/acsomega.0c03892
ACS Omega 2020, 5, 31608−31623

built-in modules under receptor-based boundary conditions, which we can import and parametrize as follows

```
import wepy.boundary_conditions \
    as wepy_bc
from wepy_bc.receptor \
    import UnbindingBC

CUTOFF = 1.0 # nanometers

bc = UnbindingBC(
       cutoff_distance=CUTOFF
       initial_state=init_state,
       topology=json_top,
       ligand_idxs=lig_idxs,
       receptor_idxs=prot_idxs)
```

where json_top is an internal JSON-based topology format in wepy, more information on this is given in the supplemental example script. Again, this is easy to customize for your application; we show a simplified example of a custom boundary condition below. The warp_walkers function computes the minimum distance between the ligand and the receptor atoms for each walker and if it exceeds a threshold, we "warp" it by replacing that walker state with the initial bound state, while keeping its weight constant.

```
import wepy.boundary_conditions \
    as wepy_bc
from wepy_bc.boundary \
    import BoundaryConditions \
        as BC

class MyUnbindingBC(BC):

  def __init__(self,
               initial_state,
               ligand_idxs,
               receptor_idxs,
               cutoff_distance=1.0):

    self.initial_state = \
           initial_state
    # etc
    ...

  def warp_walkers(self,
           walkers,
           cycle):

    new_walkers = []
    for walker in walkers:

      dists = \
        compute_distances(
          walker,
          self.ligand_idxs,
          self.receptor_idxs)

      if (min(dists) >=
          self.cutoff_distance):

        warped_walker = Walker(
            self.initial_state,
            walker.weight)

        new_walkers.append(
            warped_walker)
      else:
        new_walker.append(walker)

    return new_walkers, [], [], []
```

Finally, we assemble these components into a simulation manager

```
from wepy.sim_manager \
    import Manager

sim_manager = Manager(
    init_walkers,
    runner=runner,
    resampler=resampler,
    boundary_conditions=bc)
```

Once the simulation manager is constructed, all we need to do now is to tell it to run. A simulation for 10 cycles, each having 10,000 steps per walker per cycle runs as follows

```
final_walkers, final_apparatus = \
  sim_manager.run_simulation(
      10,
      10000)
```

This returns the walkers at the end of the simulation along with the final state of the runner, resampler, and boundary conditions which are contained in the final_apparatus.

Another important component is the use of reporters. In addition to the walkers and apparatus returned by the simulation manager functions, wepy supports a plugin system to output data as the simulation is progressing. In the following example, we will show you how to use two different reporters: one for creating the HDF5 files and another that produces a plaintext dashboard file for every cycle to show the progress of long running simulations in a high-level overview. If you have the resampler and boundary conditions already constructed, it is very simple to make the HDF5 reporter. This will work out of the box but should likely be customized as the default is to save all data, including the velocities, at each step.

```
from wepy.reporter.hdf5 \
    import WepyHDF5Reporter

h5_reporter = WepyHDF5Reporter(
    file_path='myresult.wepy.h5',
    topology=json_top,
    resampler=resampler,
    boundary_conditions=bc)
```

The dashboard is composed of different sections for the different components. There is a generic one that displays the number of cycle runs, walker weights, and total simulation time, and there are component-specific sections for information on resampling, boundary conditions, and so forth.

```
from wepy.reporter.dashboard \
    import \
        DashboardReporter, \
        BCDashboardSection
```


https://dx.doi.org/10.1021/acsomega.0c03892
ACS Omega 2020, 5, 31608−31623

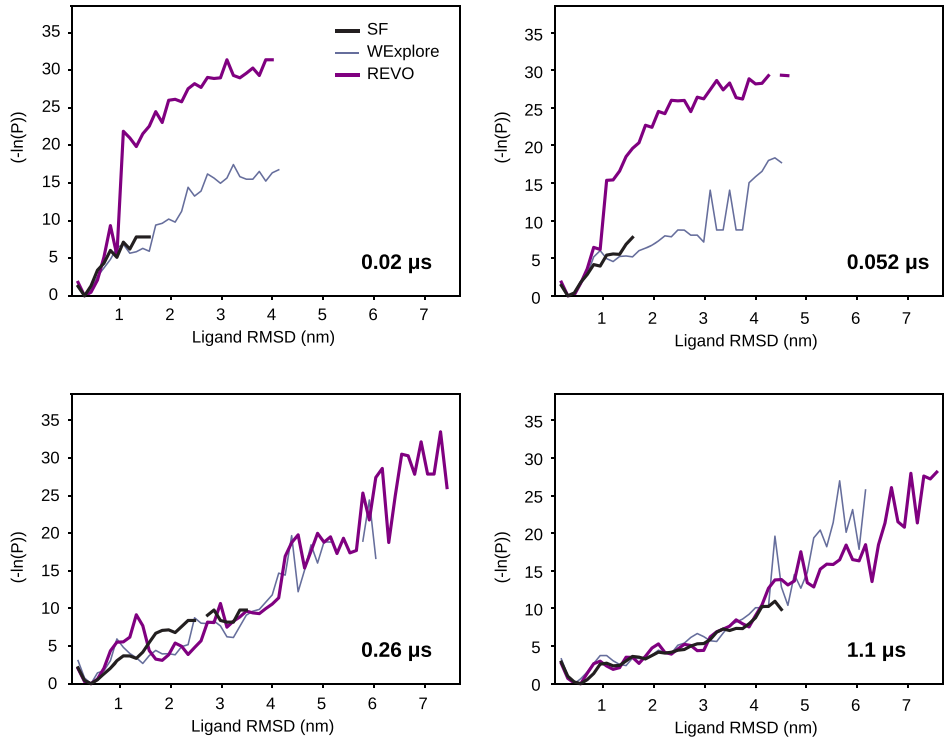

Figure 4. Series of $-\log(p)$ distributions for ensemble simulation groups at different time points. The time shown in the bottom right of each panel is the total amount of sampling across all replicas in the ensemble.

```
from wepy.reporter.openmm import \
    OpenMMRunnerDashboardSection

import wepy.reporter.wexplore \
    as wexplore_reporter

from wexplore_reporter \
    import WExploreDashboardSection

import wepy.reporter.receptor \
    as receptor_reporter

from receptor_reporter.dashboard \
    import \
    UnbindingBCDashboardSection

runner_dash = \
    OpenMMRunnerDashboardSection(
        runner)

wexplore_dash = \
    WExploreDashboardSection(
        resampler)

bc_dash = \
    UnbindingBCDashboardSection(
        bc)

dash_reporter = DashboardReporter(
    file_path='myresult.dash.org',
    step_time=0.002*picosecond,
    runner_dash=runner_dash,
    resampler_dash=wexplore_dash,
    bc_dash=bc_dash,
)
```

These reporters are attached to the simulation manager at creation time

```
sim_manager = Manager(
    init_walkers,
    runner=runner,
    resampler=resampler,
    boundary_conditions=bc,
    reporters=[h5_reporter,
               dash_reporter])
```

As mentioned in Section 3.2.2, the HDF5 reporter is of utmost importance and is a purpose-built fully featured storage format implemented in HDF5. Saving your data in the HDF5 format will let you use an extensive API designed specifically for manipulating WE data (the WepyHDF5 class in the wepy.hdf5 module) as well as allowing lower-level manipulations *via* libraries such as h5py. Furthermore, a fairly comprehensive analysis toolkit is made available in the wepy.analysis module. This toolkit makes it easy to transform and structure data to interoperate with other analysis toolkits such as scipy,[60] dask,[52] mdtraj,[61] and gephi.[62] In the next section, we show some examples of how to leverage these tools to perform common analyses and visualizations.

4.2. Code for Analysis and Visualizations. *4.2.1. Probability Distributions.* One of the most common ways to visualize simulation data is to project structural data to a small number of dimensions and then plot this as a probability distribution. In practice, free-energy profiles are computed by binning the projection domain and then computing the weighted histogram over those bins by summing the weights of the samples in those bins. The bin values are then normalized to get a probability distribution, which is then transformed by $-\ln(p)$ to get a free energy-like value. Typically, the term free energy refers to probability distributions over equilibrium ensembles. However, for convenience, here, we refer to probability distributions transformed as described as "free energy" or more specifically "nonequilibrium free energy" even if the underlying ensemble is not necessarily an equilibrium one. In a single linear MD

https://dx.doi.org/10.1021/acsomega.0c03892
ACS Omega 2020, 5, 31608−31623

trajectory, the frames are equally weighted, but in an ensemble of walkers in WE, the weights of each frame can be different and vary over time. As such, the trajectory coordinate data are associated with the weights in the wepy HDF5 file.

Here, we show how to create 1D free-energy profiles for each experimental group projected onto the ligand RMSD relative to the bound pose (see Figure 4). The process is to first compute the ligand RMSD for all of the simulation frames as an "observable" and then to compute the free-energy profiles which can be plotted with a tool such as matplotlib.[63]

To compute the ligand RMSD observable, we open the file with the WepyHDF5 API in Python

```
from wepy.hdf5 import WepyHDF5

wepy_h5 = WepyHDF5(
    'my_data_file.wepy.h5',
    mode='r+')

wepy_h5.open()
```

Then, we define a function for computing RMSD that will be mapped over all of the data

```
import numpy as np
from geomm.rmsd import calc_rmsd

def calc_rmsd_observable(fields):

  rmsds = []
  for frame in fields['positions']:

      rmsd = calc_rmsd(
          reference_positions,
          frame,
          idxs=ligand_idxs)

  return np.array(rmsds)
```

We then apply the function to the data

```
observable = \
    wepy_h5.compute_observable(
        calc_rmsd_observable,
        ['positions'],
        (),
        save_to_hdf5='lig-rmsd')
```

where the second argument asks to retrieve the relevant data from each frame, which here is just the positions. This also saves the observable into the database as the field "lig-rmsd". By default, the calc_rmsd_observable function is applied in serial. In wepy, we also provide a distributed version which can connect to a dask cluster server for distributed parallelism on large computing clusters (see documentation).

Now, we can create the free-energy profile for this observable. One intermediate step, although, is to use another representation called the "contig tree", which makes combining multiple contiguous simulations (such as when a simulation is restarted) much easier to analyze. Construction of a contig tree requires (1) a dataset, (2) a "decision" (used internally by the resampler see Appendix 6.1 for more details), and optionally (3) the boundary condition class that was used for the simulation if any.

```
from wepy.analysis.contig_tree \
    import ContigTree

decision = \
    WExploreResampler.DECISION

bcc = UnbindingBC

contigtree = ContigTree(
  wepy_h5,
  decision_class=decision,
  boundary_condition_class=bcc)
```

With the contig tree constructed, we can then feed it to the profiler, bin the domain, and calculate the free energies for each.

```
from wepy.analysis.profiles \
    import ContigTreeProfiler

profiler = ContigTreeProfiler(
               contigtree)

bin_edges = profiler.bin_edges(
              'auto',
              'lig-rmsd')

fe_profile = profiler.fe_profile(
                 0,
                 'lig-rmsd',
                 bins=bin_edges)
```

A line plot of fe against bin is shown for each simulation type in Figure 4 and discussed in Section 4.4.1.

4.2.2. Visualizing Ligand-Unbinding Events. To make rate estimates for a process or to analyze the transition path ensemble, we need an efficient way to examine the pathways from one basin to another. There are tools in wepy to help analyze this kind of boundary condition data, which we call the "warping" records. The data for all the warping events can be found in the HDF5 and are accessed through either the WepyHDF5 object or the ContigTree. Here, we can make a pandas data frame table for run 0, which can easily be exported to any number of formats.

```
warp_df = \
  wepy_h5.warping_records_dataframe(
    [0])
```

Each row of this table contains the index, weight, and point in time that the event took place.

To get trajectories from these warping events to the starting structure, we use the following functions on a Contig object which is a single simulation from a ContigTree

```
contig = contigtree.span_contig(0)

with contig:
  warp_points = \
    contig.warp_contig_trace()

  lineage_trace = list(
    contig.lineages(warp_points)
  )[0]

  contig_h5 = contig.wepy_h5

  trace_fields = \
    contig_h5.get_trace_fields(
                  lineage_trace,
                  ['positions',
                   'box_vectors'])

  traj = \
    contig_h5.traj_fields_to_mdtraj(
      trace_fields)

traj.save(filepath)
```


https://dx.doi.org/10.1021/acsomega.0c03892
ACS Omega 2020, 5, 31608−31623

On the first line, we first generate a linearized Contig from the ContigTree which is necessary for traversal. Then, we open a context for the contig to open the HDF5 file where we can then access the simulation data. We then query for a "trace" of all frames in which a warping event occurred as the warp_trace. From each of these events, we then produce the individual histories ("lineages") of that walker to the initial state. For this example, we only choose to look at one of these, which is set as lineage_trace. We now want to actually retrieve the simulation data to be able to visualize and perform analysis on it. To do this, we use the lineage trace and choose which attributes we want. For this example, we get only the positions and box vectors because our purpose is to generate visualizations. Now that we have the trace_fields for this trajectory, we can easily convert to an mdtraj object which can then be used to save the trajectory to a file format visualization software can read.

4.3. Simulating Lysozyme Ligand Unbinding. To showcase the use of wepy for performing simulations of rare events in real systems, we simulate the small-molecule p-xylene unbinding from the T4 lysozyme L99A mutant protein, which we refer to henceforth as "lysozyme". Lysozyme is a common model system for ligand-unbinding studies both experimentally and computationally.[64] Lysozyme-unbinding pathways have been simulated through a variety of enhanced and brute-force methods.[65−68] Here, we simulate lysozyme interacting with the p-xylene ligand in an implicit solvent where the event of interest, ligand unbinding, is not trivial to observe but is still tractable with straightforward MD simulations.

The system was prepared in an OBC GBSA implicit solvent, using Amber ff96 for the protein force field and a GAFF and AM1-BCC parametrization of the ligand. A Langevin integrator was used with a 2 fs time step, a temperature of 300 K, and a friction coefficient of 1 ps^{-1}. Here, we determine a target p-xylene residence time in an implicit solvent using a total of 3.385 μs of straightforward MD simulation, where p-xylene is "warped" back to the binding site upon unbinding a total number of 11 times. This resulted in an unbinding rate of 3.25 μs^{-1}, which we use as a target rate for comparison with both WE simulations and trajectory ensembles using wepy without resampling. The test groups that were simulated are as follows: ensemble of 48 walkers with no resampling, that is, straightforward (SF group); ensemble of 48 walkers with the REVO resampler (REVO group); ensemble of 48 walkers with the WExplore resampler (WExplore group).

For each group, four independent simulations were run, each for a total sampling time (summation across all replicas of ensemble) of 1 μs. All simulations used the same boundary condition criterion, which is that the minimum of all ligand−protein interatomic distances is greater than 1.0 nm. Boundary conditions are checked at the end of every cycle, which was τ = 20 ps for every group. The WE simulations (REVO and WExplore) employed the following parameters: a minimum walker weight of 10^{-12} and a maximum walker weight of 5.0×10^{-1}. Both REVO and WExplore used the UnbindingDistance, where the distance between two walker structures is computed as the distance between their ligands after alignment of their binding sites to the initial starting structure. This is included in wepy and has been used with success in other ligand release simulations.[28,33,47−49]

For the WExplore simulation, the same parameters from previous publications[28,47,48] were used; four region hierarchy levels with cutoffs d = 10, 5, 3.5, and 2.5 Å with a maximum of 10 subregions per parent region. For REVO,[46] the following parameters were used: a characteristic distance of 1 Å, a merge distance of 2.5 nm[c], and a distance exponent of 4, and we use the weight-based importance function, as described in eq 2.

4.4. Analysis of Simulation Results. *4.4.1. Probability Distributions.* Figure 4 shows a series of plots of free energies of each simulation group projected onto the RMSD of the ligand to the starting pose as a function of aggregate simulation time. Again, the "free energies" here are more accurately the negative logarithm of the nonequilibrium probability distributions because we are warping unbound walkers back to the starting position. Accordingly, we see a local free-energy minimum at RMSD close to 0 Å but no corresponding minimum in the unbound state.

REVO and WExplore reach much larger ligand RMSD values than the straightforward (SF) simulations. Both WExplore and REVO observed close to 5 nm RMSD values at 0.052 μs (Figure 4B), whereas SF has only reached ligand RMSD values of around 1.7 nm. The probability of large RMSD states tends to be underestimated early on, as can be seen by comparison between panel A and B and the final estimates in panel D.

By the end of the simulations (Figure 4D), all the profiles are very similar between the different groups until around 4 nm. The curves for WExplore and REVO beyond that are noisy because of the difficulty in reaching very large distances (*e.g.*, 7 nm) without reaching the unbinding boundary condition. In general, note that early time predictions of WE free-energy profiles can differ significantly from equilibrium free-energy profiles. These should instead be viewed as direct estimates of a conditional time-dependent probability distribution: $P(x, t|x_0, t_0)$, or the probability of being at a point x and time t, given that we began at point x_0 at time t_0. Although we expect these to converge to equilibrium probabilities only for $t \rightarrow \infty$, we can often learn valuable information from the tails of these distributions.

We note that a peculiar artifact in this system is that the p-xylene ligand sticks strongly to the surface of lysozyme, and breaking out of the binding pocket was much easier than leaving the surface of the protein. This likely explains high-variance ligand RMSD values above 4 nm and performance could be improved by incorporating the latter process (leaving the surface) more directly into the distance metric that governs the resampling process.

4.4.2. Unbinding Events and Trajectories. A primary interest in the study of rare events in nonequilibrium systems is to understand the kinetics of transition paths. Although we can use a variety of simulation methods to estimate the rate constants associated with events, it can be much more difficult to obtain accurate data on the actual mechanistic determinants of these rates. Primarily, this is the structural details of transition states, but all structures along a path can potentially be useful for design purposes. Transition paths obtained with path sampling methods, such as WE, are especially meaningful because the Hamiltonian is unperturbed throughout the sampling process. Here, we show how wepy can easily obtain and analyze continuous, unbiased trajectories of ligand unbinding.

Table 1 shows that both REVO and WExplore outperform the SF methods in terms of the absolute number of unbinding events that are observed. More important, although, is that the initial times of the first observations for REVO and WExplore are much faster than the SF ensemble simulations. All replicates for REVO and WExplore simulations have exit points within the first 100 ns, while the first among SF simulations takes about 3 times as long (at 300 ns) and is highly variable among replicates. The slowest SF simulation does not observe an unbinding event until

https://dx.doi.org/10.1021/acsomega.0c03892
ACS Omega 2020, 5, 31608−31623

Table 1. Unbinding Events and Final Rate Estimates of Simulations

group	num warps
SF (numerical target)	11
SF (ensemble)	39
WExplore	486
REVO	4337

around 700 ns. This highlights the utility of high-dimensional resamplers such as WExplore and REVO for generating observations of rare events with a more modest investment of computing power. In Figure 5, we show structures along the first

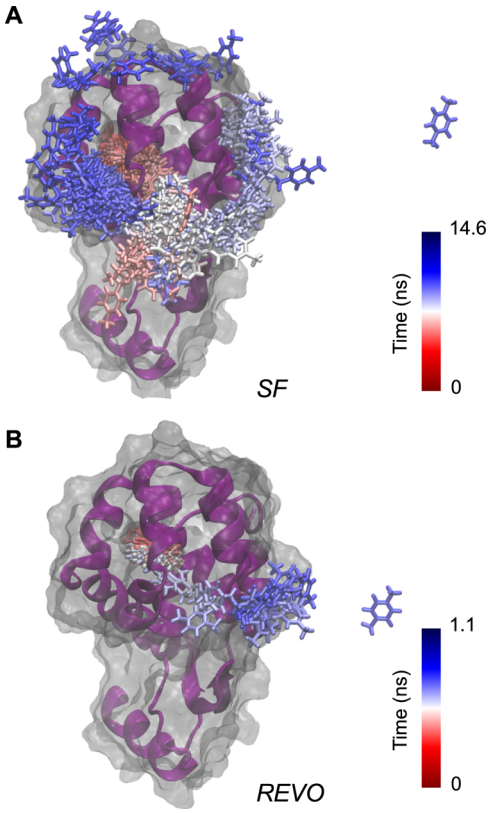

Figure 5. Ligand-unbinding trajectories. These 3D renderings of lysozyme protein (from the trajectory seed structure) showing the positions of the p-xylene ligand from the bound positions (red) to unbound positions (blue). A cartoon representation of lysozyme is in purple surrounded by a gray surface representation. Positions of the ligand are shown for the first observed unbinding trajectory from each of the two simulation groups shown at the end of each WE cycle (τ, *i.e.,* every 20 ps). (A) Unbinding trajectory from the SF group (no resampling). (B) Trajectory from the REVO group.

unbinding paths generated by the SF and REVO simulations. From this, we can see that the REVO simulation (panel B) takes a much shorter, more directed path to unbinding compared to the SF one (panel A). The ligand in the SF simulation can be seen to move around to multiple other locations on the surface of the protein before ultimately unbinding.

5. DISCUSSION

5.1. Successes. Wepy is a useful and flexible implementation of advanced WE simulations with a growing number of applications.[33,46,49,69] In our experience, wepy has been particularly useful in three major ways. First, it has made the prototyping of new methods very easy even for researchers with little experience in programming or the Python language. The initial goal of wepy was to simplify and modularize the original implementation of the WExplore algorithm.[45] Following this, the REVO algorithm was fully prototyped, tested, and eventually used as a resampler for a variety of problems.[33,46,69] This process was made much simpler and faster by the sharing of the same infrastructure that was already developed. In addition, the binless nature of the REVO algorithm was actually conceived of partly as a response to the abstractions formalized by wepy.

This highlights the second point: not only does wepy provide useful software to perform simulations but it also provides a common language with which researchers could communicate with each other about their simulations. We have found the use of diagrams such as Figures 1 and 2 to be valuable not only for reasoning about our programs but also in explaining WE and nonequilibrium simulations in general.

Finally, wepy has made the process of analyzing WE simulations dramatically simpler. The biggest contribution here is the support for out-of-core data structures (*via* the HDF5 format) and expressive in-memory representations that reflect the tree-like trajectory structures (*i.e.,* WepyHDF5 and ContigTree). Using this data structure, wepy also provides facilities for easy analysis and visualization of "resampling trees".

As mentioned in the Introduction (Section 1), a big asset to the WE method is that the microscopic trajectories are generated using the unbiased Hamiltonian. In wepy, we ensure that these data can be fully leveraged however the user sees fit. One major use case for these data is the construction of MSMs with long lag times, which in turn can be used to predict steady-state probabilities or identify transition states. Although not mentioned here, wepy provides facilities for constructing macrostate models [such as conformation state networks (CSNs) and MSMs].

Others are encouraged to use, share components for, or even contribute to wepy which is open source with an MIT license. The source code is currently hosted on github (https://github.com/ADicksonLab/wepy) and documentation is currently available at https://adicksonlab.github.io/wepy. At the time of publication, the 1.00 release of wepy has been made and has been archived and given a persistent identifier from Zenodo.org (DOI: 10.5281/zenodo.3973431).

5.2. Opportunities for Future Development. Although wepy provides many essential and novel features, there are areas for potential improvements that could make it even more widely useful. Currently, it has not yet been integrated with major MD engines such as GROMACS,[70] CHARMM,[71] Amber,[72] and Desmond,[73] and there is currently only experimental support for NAMD[74] and ASE.[75] We note that the architecture of some of these engines makes it difficult to interface without going through a UNIX-like system environment (a difficulty for all software using these tools). We note that OpenMM allows for the use of force fields native to each engine (*e.g.,* CHARMM, AMBER) to be used within it, so the issue is more about choice of implementation rather than the content of the simulations.

Second, the priority for wepy developers has been on implementing and prototyping new adaptive and high-dimensional resampling algorithms rather than implementing standard static binning methods or accelerated WE.[23,59] Fortunately, many of these methods are not complex to implement as resampler objects in wepy and we hope that these will either be included in future releases or as standalone libraries. We note

https://dx.doi.org/10.1021/acsomega.0c03892
ACS Omega 2020, 5, 31608−31623

that a goal between WESTPA[55] and wepy developers is to enable a modular program design, where resampler objects could be used interchangeably between the two WE implementations.

Third, for protein and other macromolecular simulations, wepy resorted to using an *ad hoc* serialization format in JSON for molecular topologies (the schema of which was borrowed from an internal representation in the mdtraj HDF5 implementation). In principle, wepy is agnostic to topology formats but in practice, this is an extremely important component in building simulations and performing analyses. Despite there being a large number of software packages implementing in-memory representations of molecular topologies, there are no formats suitable for serialization and communication between software. We encourage users to consider the merits of the JSON topology used in wepy but by no means recommend it as a general purpose standard nor do we wish it to become a *de facto* standard.

Finally, although the use of an HDF5 based file format has proven to be a good choice for many reasons, it is currently not supported natively by any molecular visualization software. Thus, trajectories must be converted and duplicated into separate files with a supported format (a simple task using the integration with the mdtraj library). In our work flows, we treat these files as temporary intermediates; however, this can bloat the necessary disk space needed and cause some time delays when (re)generating them. We note, although, that visualization tools could benefit immensely by adopting a random-access format such as HDF5 which would allow for visualizing single trajectories which do not fit in memory, a problem we frequently encounter when attempting to view very long trajectories.

APPENDIX: CREATING AND DEVELOPING RESAMPLERS

As mentioned in Section 3.1, resamplers can do whatever they want as long as they satisfy the constraints given in eq 3. There is thus great flexibility in wepy for advanced users who wish to design new resampling algorithms. However, in practice, not much is developed in a vacuum and much of the functionality between resamplers can be shared. Here, we describe two core abstractions that are provided by wepy to aid in design and construction of resamplers: decision classes and distance metrics. These two components are the foundations for the two high-dimensional resamplers provided in wepy, WExplore, and REVO, in addition to other in-progress research resamplers.

6.1. Decision Classes

The first useful abstraction we identify is the **decision class** (denoted D). We would like to report on the resampling process such that there is no loss of information. Although this is not completely necessary and resamplers have the privilege of not divulging how they resampled a given ensemble of walkers, it is rather useful to know *post hoc*. Indeed, if we do not have information on how an ensemble of walkers was derived from a former one, then we have no way of connecting them together for visualization or analysis.

One way to represent the resampling process that satisfies these requirements is to model a resampling process as a set of discrete actions applied to each walker. We then require that the resampler generates an **action record** for every walker in a single applicative step

$$\mathcal{R}(\boldsymbol{W}) \rightarrow (\mathcal{R}', \boldsymbol{W}', A)$$

where $\mathcal{R}$ is the resampler, $\boldsymbol{W}$ is the set of walkers, and A is a list of the actions a_i that were applied to each walker i, where n is the number of walkers in $\mathbf{W}$

$$A = (a_0, a_1, \ldots a_{n-1})$$

In order to support more general situations, we allow for this "net" action record to be described as a set of microactions that when applied successively, k times, yield the net action record: $\overline{A} = (A_0, A_1, \ldots A_{k-1})$. This structure was chosen because it supports a multipass iterative approach without losing any information. For single-pass approaches, we can use a set of microactions with size one, that is, $\overline{A} = (A_0)$. However, the net action record, $\overline{A}$, is all that is needed when analyzing ancestries and for this discussion we can ignore the microactions.

This representation is quite general as it leaves the structure and content of the individual action records a_i unspecified. The decision class adds structure to these records by regarding each action record as a decision that was made regarding the fate of a walker. Minimally, this simply means defining a set of decision types (or symbols) and choosing one for each walker.

As an example, for the canonical WE cloning and merging use case, we choose the decision symbols: "CLONE", "MERGE", "SQUASH", and "NOTHING". Here, "CLONE" indicates to make a copy of the walker, "MERGE" indicates that the state of this walker will be kept and weight from a selection of the "SQUASH" walkers will be donated to it, and "NOTHING" indicates that no action will be taken for this walker. The definition of these symbols is implemented as a Python Enum that assigns an integer value to each symbol (useful for efficient storage). To encode the meanings of these symbols in a resampler, a decision class uses two methods: action and parents.

The action method can be seen by expanding the previous application of the resampling function into a two-step process

$$\mathcal{R}(\boldsymbol{W}) \rightarrow (\mathcal{R}', A)$$

$$\text{action}_D(\boldsymbol{W}, A) \rightarrow \boldsymbol{W}'$$

The clone-merge decision class in wepy covers most use cases and allows for multiple clones from a single walker. In addition to the decision symbol, it also requires an additional list of indices indicating the other walkers they target. For "CLONE", the target indices are the locations (and thus implicitly reveal the number of clones) that the newly minted walkers will occupy. For "SQUASH", the target indicates which "MERGE" walker it will donate its weight to. Target indices for "MERGE" and "NOTHING" designate the index of the surviving walker. It is up to the resampler to ensure the integrity and consistency of these records.

As an example, the action record A from the resampling step of Figure 2 would take the form

$$A = ((\text{CLONE}, (0, 1)), (\text{SQUASH}, 2), (\text{MERGE}, 2), (\text{NOTHING}, 3))$$

This decision class is the only one supported in wepy currently, although others are currently under investigation.

The parent function is a function which allows for the recovery of the lineage of walkers

$$\text{parents}_D(A) \rightarrow (p_0, p_1, \ldots, p_{n'-1})$$

where p_i is the index of the walker in $\boldsymbol{W}$ that is the parent of the walker i in $\boldsymbol{W}'$ and n' is the number of walkers in $\boldsymbol{W}'$. For the example mentioned above, the parents would be (0, 0, 2, 3).

https://dx.doi.org/10.1021/acsomega.0c03892
ACS Omega 2020, 5, 31608−31623

Resamplers can then be equipped with decision classes both in order to aid in generating the records of resampling and identifying the decision protocol that it adheres to. These both reduce the amount of code a resampler must implement but reifies an interface such that multiple components can identify a resampler as having certain properties. Reification of the decision class drastically improves the serializability of WE simulation data and thus the interoperability. This can be seen in the straightforward representations of these records into the HDF5 storage format as a simple table of values.

6.2. Distance Metrics

The goal of any resampler in a WE simulation is to refocus computational effort toward walkers that are deemed to be more valuable than others for accomplishing a particular research objective. In both WExplore and REVO, a central object is the distance metric, which is used to compare walkers to each other (in the case of REVO) or to compare walkers to a set of "images" that are constructed over the course of the simulation (in the case of WExplore). Using a distance allows us to focus on walkers that are different from the others, which can increase our odds of breaking out of deep free-energy minima. Note that distance metrics are more general than projection onto collective variables, which is the basis of many other enhanced sampling methods (including other applications of the WE method). For more detailed arguments for the use of distance metrics beyond those made here, we refer to refs.[76,78]

Distance metrics are defined as independent objects in wepy that can be used to build resamplers, boundary conditions, or for analysis. These can be defined very generally and need not be Euclidean. One example is the characterization of molecular conformations by a "string" of booleans indicating the presence or absence of secondary structure features or intermolecular interactions. Although this does not form a vector space, it is still able to be compared using metrics based on Jaccard distances such as the Tanimoto distance which is frequently used in chemoinformatics for comparing molecular structure features. In fact, there is a great diversity of nonvector space distance metrics that could be used to express the goal of a simulation:

- Root-mean square metrics such as molecular RMSD,[79]
- Mahalanobis distance for characterizing protein surfaces,[80]
- Hausdorff distances for characterizing shapes from continuous topologies, and
- network/graph structure Wiener index for characterizing network/graph structures again used in chemoinformatics.[81]

Distance metrics have a simple and uniform abstraction that is mathematically well-studied. Generally, a distance metric can be defined as a function, d, that maps two states, X_i and X_j, in a metric space, J, to a single positive real number[77,82]

$$d(X_i, X_j) = d_{ij};\ \ d\colon \mathbb{R}^J, \mathbb{R}^J \rightarrow \mathbb{R}$$

An example of a simple wepy distance metric is given in Figure 6. The necessary details of this are that the class inherits from the distance superclass and that it implements a method called image. With this implementation, we are free to run simulations with either WExplore or REVO because they both support this interface.

In practice, distance metrics should be defined to reflect the goals of a particular simulation. This meshes well with the wepy implementation as one can use any external library available in the vast Python ecosystem. We highlight, for instance, that the scipy.spatial library provides over 20 different general purpose distance metrics at the time of writing.[60] Furthermore, more molecular focused analysis tools such as MDTraj,[61] MDAnalysis,[83] ProDy,[84] and our own geomm can be leveraged for constructing more complex distance metrics.

```
import wepy.resampling.distances \
    as wepy_dists

from wepy_dists import Distance \
    as Dist

import numpy as np

class XYEuclideanDistance(Dist):

  def image_distance(self,
          image_a,
          image_b):

    xx = (image_a[0] - image_b[0])**2
    yy = (image_a[1] - image_b[1])**2

    return np.sqrt(xx + yy)
```

Figure 6. Code for a wepy distance metric class that implements a 2-dimensional Euclidean distance.

■ ASSOCIATED CONTENT

Supporting Information

The Supporting Information is available free of charge at https://pubs.acs.org/doi/10.1021/acsomega.0c03892.

Example folder with included data and scripts to generate and analyze the data presented above; plaintext README file with instructions for wepy installation and for the running the scripts and examples; and for more up-to-date information and examples, consult the wepy documentation at https://github.com/ADicksonLab/wepy (ZIP).

■ AUTHOR INFORMATION

Corresponding Author

Alex Dickson − *Department of Biochemistry & Molecular Biology and Department of Computational Mathematics, Science and Engineering, Michigan State University, East Lansing 48824, Michigan, United States;* orcid.org/0000-0002-9640-1380; Email: alexrd@msu.edu

Author

Samuel D. Lotz − *Department of Biochemistry & Molecular Biology, Michigan State University, East Lansing 48824, Michigan, United States;* orcid.org/0000-0001-6159-615X

Complete contact information is available at: https://pubs.acs.org/10.1021/acsomega.0c03892

Notes

The authors declare no competing financial interest.

■ ACKNOWLEDGMENTS

ARD acknowledges support from the National Science Foundation grant DMS 1761320 and the National Institutes of Health grant R01GM130794.

■ ADDITIONAL NOTES

[a]The computationally determined mean first passage time however was 42 s.

[b]A thorough description of this format can be found in the documentation for the wepy project https://github.com/ADicksonLab/wepy.

[c]The large value chosen here allowed overly dissimilar walkers in the ensemble to be merged. We expect that a smaller value for this quantity could improve REVO performance, for example, 2.5 Å.

REFERENCES

(1) Levitt, M.; Warshel, A. Computer simulation of protein folding. *Nature* **1975**, *253*, 694.

(2) McCammon, J. A.; Gelin, B. R.; Karplus, M. Dynamics of folded proteins. *Nature* **1977**, *267*, 585.

(3) Stone, J. E.; Hardy, D. J.; Ufimtsev, I. S.; Schulten, K. GPU-accelerated molecular modeling coming of age. *J. Mol. Graph. Model.* **2010**, *29*, 116−125.

(4) Shaw, D. E.; Deneroff, M. M.; Dror, R. O.; Kuskin, J. S.; Larson, R. H.; Salmon, J. K.; Young, C.; Batson, B.; Bowers, K. J.; Chao, J. C.; et al. Anton, a Special-purpose Machine for Molecular Dynamics Simulation. *Commun. ACM* **2008**, *51*, 91−97.

(5) Moore, G. E. Cramming more components onto integrated circuits. *Electronics* **1965**, *38*, 114.

(6) Lu, H.; Tonge, P. J. Drug-target residence time: critical information for lead optimization. *Curr. Opin. Chem. Biol.* **2010**, *14*, 467−474.

(7) Hansmann, U. H. E. Parallel tempering algorithm for conformational studies of biological molecules. *Chem. Phys. Lett.* **1997**, *281*, 140−150.

(8) Tiwary, P.; Parrinello, M. From Metadynamics to Dynamics. *Phys. Rev. Lett.* **2013**, *111*, 230602.

(9) Barducci, A.; Bonomi, M.; Parrinello, M. Metadynamics. *Wiley Interdiscip. Rev.: Comput. Mol. Sci.* **2011**, *1*, 826−843.

(10) Casasnovas, R.; Limongelli, V.; Tiwary, P.; Carloni, P.; Parrinello, M. Unbinding kinetics of a p38 MAP kinase type II inhibitor from metadynamics simulations. *J. Am. Chem. Soc.* **2017**, *139*, 4780−4788.

(11) Limongelli, V.; Bonomi, M.; Parrinello, M. Funnel metadynamics as accurate binding free-energy method. *Proc. Natl. Acad. Sci. U.S.A.* **2013**, *110*, 6358−6363.

(12) Maragliano, L.; Vanden-Eijnden, E. A temperature accelerated method for sampling free energy and determining reaction pathways in rare events simulations. *Chem. Phys. Lett.* **2006**, *426*, 168−175.

(13) Abrams, C. F.; Vanden-Eijnden, E. Large-scale conformational sampling of proteins using temperature-accelerated molecular dynamics. *Proc. Natl. Acad. Sci. U.S.A.* **2010**, *107*, 4961−4966.

(14) Torrie, G. M.; Valleau, J. P. Nonphysical sampling distributions in Monte Carlo free-energy estimation: Umbrella sampling. *J. Comput. Phys.* **1977**, *23*, 187−199.

(15) Zuckerman, D. M.; Chong, L. T. Weighted Ensemble Simulation: Review of Methodology, Applications, and Software. *Annu. Rev. Biophys.* **2017**, *46*, 43−57.

(16) Pratt, L. R. A statistical method for identifying transition states in high dimensional problems. *J. Chem. Phys.* **1986**, *85*, 5045−5048.

(17) Bolhuis, P. G.; Dellago, C.; Chandler, D. Sampling ensembles of deterministic transition pathways. *Faraday Discuss.* **1998**, *110*, 421−436.

(18) van Erp, T. S.; Moroni, D.; Bolhuis, P. G. A novel path sampling method for the calculation of rate constants. *J. Chem. Phys.* **2003**, *118*, 7762−7774.

(19) Allen, R. J.; Warren, P. B.; ten Wolde, P. R. Sampling Rare Switching Events in Biochemical Networks. *Phys. Rev. Lett.* **2005**, *94*, 018104.

(20) Cérou, F.; Guyader, A. Adaptive multilevel splitting for rare event analysis. *Stoch. Anal. Appl.* **2007**, *25*, 417−443.

(21) Huber, G. A.; Kim, S. Weighted-ensemble Brownian dynamics simulations for protein association reactions. *Biophys. J.* **1996**, *70*, 97−110.

(22) Aristoff, D. Analysis and Optimization of Weighted Ensemble Sampling. *ESAIM: Math. Modell. Numer. Anal.* **2018**, *52*, 1219.

(23) Abdul-Wahid, B.; Feng, H.; Rajan, D.; Costaouec, R.; Darve, E.; Thain, D.; Izaguirre, J. A. AWE-WQ: Fast-Forwarding Molecular Dynamics Using the Accelerated Weighted Ensemble. *J. Chem. Inf. Model.* **2014**, *54*, 3033−3043.

(24) Adelman, J. L.; Grabe, M. Simulating rare events using a weighted ensemble-based string method. *J. Chem. Phys.* **2013**, *138*, 044105.

(25) Rojnuckarin, A.; Livesay, D. R.; Subramaniam, S. Bimolecular Reaction Simulation Using Weighted Ensemble Brownian Dynamics and the University of Houston Brownian Dynamics Program. *Biophys. J.* **2000**, *79*, 686−693.

(26) Zhang, B. W.; Jasnow, D.; Zuckerman, D. M. Efficient and verified simulation of a path ensemble for conformational change in a united-residue model of calmodulin. *Proc. Natl. Acad. Sci. U.S.A.* **2007**, *104*, 18043−18048.

(27) Saglam, A. S.; Chong, L. T. Highly Efficient Computation of the Basal kon using Direct Simulation of Protein-Protein Association with Flexible Molecular Models. *J. Phys. Chem. B* **2016**, *120*, 117−122.

(28) Dickson, A.; Lotz, S. D. Ligand Release Pathways Obtained with WExplore: Residence Times and Mechanisms. *J. Phys. Chem. B* **2016**, *120*, 5377−5385.

(29) Dickson, A.; Mustoe, A. M.; Salmon, L.; Brooks, C. L., III Efficient in silico exploration of RNA interhelical conformations using Euler angles and WExplore. *Nucleic Acids Res.* **2014**, *42*, 12126.

(30) Adelman, J. L.; Grabe, M. Simulating Current-Voltage Relationships for a Narrow Ion Channel Using the Weighted Ensemble Method. *J. Chem. Theory Comput.* **2015**, *11*, 1907−1918.

(31) Zwier, M. C.; Pratt, A. J.; Adelman, J. L.; Kaus, J. W.; Zuckerman, D. M.; Chong, L. T. 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. *J. Phys. Chem. Lett.* **2016**, *7*, 3440−3445.

(32) Feng, H.; Costaouec, R.; Darve, E.; Izaguirre, J. A. A comparison of weighted ensemble and Markov state model methodologies. *J. Chem. Phys.* **2015**, *142*, 214113.

(33) Dixon, T.; Lotz, S. D.; Dickson, A. Predicting ligand binding affinity using on- and off-rates for the SAMPL6 SAMPLing challenge. *J. Comput.-Aided Mol. Des.* **2018**, *32*, 1001−1012.

(34) Allen, R. J.; Frenkel, D.; ten Wolde, P. R. Simulating rare events in equilibrium or nonequilibrium stochastic systems. *J. Chem. Phys.* **2006**, *124*, 024102.

(35) Morelli, M. J.; ten Wolde, P. R.; Allen, R. J. DNA looping provides stability and robustness to the bacteriophage switch. *Proc. Natl. Acad. Sci. U.S.A.* **2009**, *106*, 8101−8106.

(36) Donovan, R. M.; Tapia, J.-J.; Sullivan, D. P.; Faeder, J. R.; Murphy, R. F.; Dittrich, M.; Zuckerman, D. M. Unbiased Rare Event Sampling in Spatial Stochastic Systems Biology Models Using a Weighted Ensemble of Trajectories. *PLoS Comput. Biol.* **2016**, *12*, No. e1004611.

(37) Donovan, R. M.; Sedgewick, A. J.; Faeder, J. R.; Zuckerman, D. M. Efficient stochastic simulation of chemical kinetics networks using a weighted ensemble of trajectories. *J. Chem. Phys.* **2013**, *139*, 115105.

(38) Tse, M. J.; Chu, B. K.; Roy, M.; Read, E. L. DNA-binding kinetics determines the mechanism of noise-induced switching in gene networks. *Biophys. J.* **2015**, *109*, 1746−1757.

(39) Daigle, B. J., Jr; Roh, M. K.; Gillespie, D. T.; Petzold, L. R. Automated estimation of rare event probabilities in biochemical systems. *J. Chem. Phys.* **2011**, *134*, 01B628.

(40) Roh, M. K.; Daigle, B. J., Jr; Gillespie, D. T.; Petzold, L. R. State-dependent doubly weighted stochastic simulation algorithm for automatic characterization of stochastic biochemical rare events. *J. Chem. Phys.* **2011**, *135*, 234108.

(41) Tse, M. J.; Chu, B. K.; Gallivan, C. P.; Read, E. L. Rare-event sampling of epigenetic landscapes and phenotype transitions. *PLoS Comput. Biol.* **2018**, *14*, No. e1006336.

(42) Villén-Altamirano, M.; Villén-Altamirano, J. RESTART: a straightforward method for fast simulation of rare events. *Proceedings of Winter Simulation Conference*, 1994; pp 282−289.

https://dx.doi.org/10.1021/acsomega.0c03892
ACS Omega 2020, 5, 31608−31623

(43) Morio, J.; Balesdent, M. *Estimation of Rare Event Probabilities in Complex Aerospace and Other Systems: A Practical Approach*; Woodhead Publishing, 2015.
(44) Zhang, B. W.; Jasnow, D.; Zuckerman, D. M. The "weighted ensemble" path sampling method is statistically exact for a broad class of stochastic processes and binning procedures. *J. Chem. Phys.* **2010**, *132*, 054107.
(45) Dickson, A.; Brooks, C. L. WExplore: hierarchical exploration of high-dimensional spaces using the weighted ensemble algorithm. *J. Phys. Chem. B* **2014**, *118*, 3532−3542.
(46) Donyapour, N.; Roussey, N. M.; Dickson, A. REVO: Resampling of ensembles by variation optimization. *J. Chem. Phys.* **2019**, *150*, 244112.
(47) Dickson, A.; Lotz, S. D. Multiple ligand unbinding pathways and ligand-induced destabilization revealed by WExplore. *Biophys. J.* **2017**, *112*, 620−629.
(48) Lotz, S. D.; Dickson, A. Unbiased Molecular Dynamics of 11 min Timescale Drug Unbinding Reveals Transition State Stabilizing Interactions. *J. Am. Chem. Soc.* **2018**, *140*, 618−628.
(49) Dickson, A. Mapping the Ligand Binding Landscape. *Biophys. J.* **2018**, *115*, 1707−1719.
(50) Copperman, J.; Zuckerman, D. Accelerated estimation of long-timescale kinetics by combining weighted ensemble simulation with Markov model "microstates" using non-Markovian theory. 2019, arxiv:1903.04673. http://arxiv.org/abs/1903.04673.
(51) Contributors, N. Numba. 2020, https://github.com/numba/numba (accessed March 17, 2020).
(52) Contributors, D. dask. 2020, https://github.com/dask/dask (accessed March 17, 2020).
(53) Eastman, P.; Friedrichs, M. S.; Chodera, J. D.; Radmer, R. J.; Bruns, C. M.; Ku, J. P.; Beauchamp, K. A.; Lane, T. J.; Wang, L.-P.; Shukla, D.; et al. OpenMM 4: A Reusable, Extensible, Hardware Independent Library for High Performance Molecular Simulation. *J. Chem. Theory Comput.* **2013**, *9*, 461−469.
(54) Folk, M.; Cheng, A.; Yates, K. HDF5: A file format and I/O library for high performance computing applications. *Proceedings of Supercomputing*, 1999; pp 5−33.
(55) Zwier, M. C.; Adelman, J. L.; Kaus, J. W.; Pratt, A. J.; Wong, K. F.; Rego, N. B.; Suárez, E.; Lettieri, S.; Wang, D. W.; Grabe, M.; et al. WESTPA: An Interoperable, Highly Scalable Software Package for Weighted Ensemble Simulation and Analysis. *J. Chem. Theory Comput.* **2015**, *11*, 800−809.
(56) Dickson, A.; Warmflash, A.; Dinner, A. R. Separating forward and backward pathways in nonequilibrium umbrella sampling. *J. Chem. Phys.* **2009**, *131*, 154104.
(57) Vanden-Eijnden, E.; Venturoli, M. Exact rate calculations by trajectory parallelization and tilting. *J. Chem. Phys.* **2009**, *131*, 044120.
(58) Suárez, E.; Lettieri, S.; Zwier, M. C.; Stringer, C. A.; Subramanian, S. R.; Chong, L. T.; Zuckerman, D. M. Simultaneous Computation of Dynamical and Equilibrium Information Using a Weighted Ensemble of Trajectories. *J. Chem. Theory Comput.* **2014**, *10*, 2658−2667.
(59) Costaouec, R.; Feng, H.; Izaguirre, J.; Darve, E. Analysis of the accelerated weighted ensemble methodology. *Conference Publications* **2013**, 171−181.
(60) Virtanen, P.; Gommers, R.; Oliphant, T. E.; Haberland, M.; Reddy, T.; Cournapeau, D.; Burovski, E.; Peterson, P.; Weckesser, W.; Bright, J.; et al. SciPy 1.0: Fundamental Algorithms for Scientic Computing in Python. *Nature Methods* **2020**, *17*, 261−272.
(61) McGibbon, R. T.; Beauchamp, K. A.; Harrigan, M. P.; Klein, C.; Swails, J. M.; Hernández, C. X.; Schwantes, C. R.; Wang, L.-P.; Lane, T. J.; Pande, V. S. MDTraj: A Modern Open Library for the Analysis of Molecular Dynamics Trajectories. *Biophys. J.* **2015**, *109*, 1528−1532.
(62) Bastian, M.; Heymann, S.; Jacomy, M. Gephi: an open source software for exploring and manipulating networks. *International AAAI Conference on Weblogs and Social Media*, 2009; Vol. *8*, pp 361−362.
(63) Hunter, J. D. Matplotlib: A 2D graphics environment. *Comput. Sci. Eng.* **2007**, *9*, 90−95.
(64) Baase, W. A.; Liu, L.; Tronrud, D. E.; Matthews, B. W. Lessons from the lysozyme of phage T4. *Protein Sci.* **2010**, *19*, 631−641.
(65) Wang, Y.; Papaleo, E.; Lindorff-Larsen, K. Mapping transiently formed and sparsely populated conformations on a complex energy landscape. *eLife* **2016**, *5*, No. e17505.
(66) Nunes-Alves, A.; Zuckerman, D. M.; Arantes, G. M. Escape of a Small Molecule from Inside T4 Lysozyme by Multiple Pathways. *Biophys. J.* **2018**, *114*, 1058−1066.
(67) Schiffer, J. M.; Feher, V. A.; Malmstrom, R. D.; Sida, R.; Amaro, R. E. Capturing invisible motions in the transition from ground to rare excited states of T4 lysozyme L99A. *Biophys. J.* **2016**, *111*, 1631−1640.
(68) Miao, Y.; Feher, V. A.; McCammon, J. A. Gaussian accelerated molecular dynamics: Unconstrained enhanced sampling and free energy calculation. *J. Chem. Theory Comput.* **2015**, *11*, 3584−3595.
(69) Dixon, T.; Uyar, A.; Ferguson-Miller, S.; Dickson, A. Membrane-mediated ligand unbinding of the PK-11195 ligand from the translocator protein (TSPO). *bioRXiv* **2020**, 1.
(70) Van Der Spoel, D.; Lindahl, E.; Hess, B.; Groenhof, G.; Mark, A. E.; Berendsen, H. J. C. GROMACS: fast, flexible, and free. *J. Comput. Chem.* **2005**, *26*, 1701−1718.
(71) Brooks, B. R.; Brooks, C. L.; Mackerell, A. D.; Nilsson, L.; Petrella, R. J.; Roux, B.; Won, Y.; Archontis, G.; Bartels, C.; Boresch, S.; et al. CHARMM: The biomolecular simulation program. *J. Comput. Chem.* **2009**, *30*, 1545−1614.
(72) Pearlman, D. A.; Case, D. A.; Caldwell, J. W.; Ross, W. S.; Cheatham, T. E., III; DeBolt, S.; Ferguson, D.; Seibel, G.; Kollman, P. AMBER, a package of computer programs for applying molecular mechanics, normal mode analysis, molecular dynamics and free energy calculations to simulate the structural and energetic properties of molecules. *Comput. Phys. Commun.* **1995**, *91*, 1−41.
(73) Bowers, K. J.; Chow, D. E.; Xu, H.; Dror, R. O.; Eastwood, M. P.; Gregersen, B. A.; Klepeis, J. L.; Kolossvary, I.; Moraes, M. A.; Sacerdoti, F. D. Scalable algorithms for molecular dynamics simulations on commodity clusters. *SC'06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, 2006; p 43.
(74) Phillips, J. C.; Braun, R.; Wang, W.; Gumbart, J.; Tajkhorshid, E.; Villa, E.; Chipot, C.; Skeel, R. D.; Kalé, L.; Schulten, K. Scalable molecular dynamics with NAMD. *J. Comput. Chem.* **2005**, *26*, 1781−1802.
(75) Larsen, A. H.; Mortensen, J. J.; Blomqvist, J.; Castelli, I. E.; Christensen, R.; Dułak, M.; Friis, J.; Groves, M. N.; Hammer, B.; Hargus, C. The atomic simulation environment − a Python library for working with atoms. *J. Phys.: Condens. Matter* **2017**, *29*, 273002.
(76) Samet, H. *Foundations of Multidimensional and Metric Data Structures*; Morgan Kaufmann, 2006.
(77) Chávez, E.; Navarro, G.; Baeza-Yates, R.; Marroquín, J. L. Searching in metric spaces. *ACM Comput. Surv.* **2001**, *33*, 273−321.
(78) Skala, M. Measuring the Difficulty of Distance-Based Indexing. *String Processing and Information Retrieval*; Springer: Berlin, Heidelberg, 2005; pp 103−114.
(79) Theobald, D. L. Rapid calculation of RMSDs using a quaternion-based characteristic polynomial. *Acta Crystallogr., Sect. A: Found. Adv.* **2005**, *61*, 478−480.
(80) Ofran, Y.; Rost, B. Analysing Six Types of Protein-Protein Interfaces. *J. Mol. Biol.* **2003**, *325*, 377−387.
(81) Gao, W.; Farahani, M. R.; Imran, M.; Kanna, M. R. R. Distance-based topological polynomials and indices of friendship graphs. *SpringerPlus* **2016**, *5*, 1563.
(82) Mao, R.; Miranker, W. L.; Miranker, D. P. Pivot selection: Dimension reduction for distance-based indexing. *J. Discrete Algorithm.* **2012**, *13*, 32−46. , Best Papers from the 3rd International Conference on Similarity Search and Applications (SISAP 2010)
(83) Michaud-Agrawal, N.; Denning, E. J.; Woolf, T. B.; Beckstein, O. MDAnalysis: A toolkit for the analysis of molecular dynamics simulations. *J. Comput. Chem.* **2011**, *32*, 2319−2327.
(84) Bakan, A.; Meireles, L. M.; Bahar, I. ProDy: Protein Dynamics Inferred from Theory and Experiments. *Bioinformatics* **2011**, *27*, 1575−1577.

https://dx.doi.org/10.1021/acsomega.0c03892
ACS Omega 2020, 5, 31608−31623