HTMD: High-Throughput Molecular Dynamics for Molecular Discovery

S. Doerr,[†] M. J. Harvey,[‡] Frank Noé,[¶] and G. De Fabritiis*[,§]

[†]Computational Biophysics Laboratory (GRIB-IMIM), Universitat Pompeu Fabra, Barcelona Biomedical Research Park (PRBB), C/Doctor Aiguader 88, 08003 Barcelona, Spain

[‡]Acellera, Barcelona Biomedical Research Park (PRBB), C/Doctor Aiguader 88, 08003 Barcelona, Spain

[¶]Department of Mathematics, Computer Science and Bioinformatics, Free University of Berlin, Berlin, Germany

[§]Institució Catalana de Recerca i Estudis Avançats (ICREA), Passeig Lluis Companys 23, Barcelona 08010, Spain

Supporting Information

ABSTRACT: Recent advances in molecular simulations have allowed scientists to investigate slower biological processes than ever before. Together with these advances came an explosion of data that has transformed a traditionally computing-bound into a data-bound problem. Here, we present HTMD, a programmable, extensible platform written in Python that aims to solve the data generation and analysis problem as well as increase reproducibility by providing a complete workspace for simulation-based discovery. So far, HTMD includes system building for CHARMM and AMBER force fields, projection methods, clustering, molecular simulation production, adaptive sampling, an Amazon cloud interface, Markov state models, and visualization. As a result, a single, short HTMD script can lead from a PDB structure to useful quantities such as relaxation time scales, equilibrium populations, metastable conformations, and kinetic rates. In this paper, we focus on the adaptive sampling and Markov state modeling features.

1. INTRODUCTION

Protein folding, protein−ligand binding, and protein−protein interactions are some of the most studied phenomena in biophysics. If understood, these phenomena can lead to the development of novel drugs as well as an improved understanding of cellular function.[1−4] There are several methodologies for investigating these problems including experimental, such as NMR, fluorescence, and X-ray, but we believe that simulation-based discovery is going to reach a prime spot in the long term due to the exponential growth of information technologies.[5]

Molecular dynamics simulations (MD) can provide an atomic level resolution of biological processes at very high temporal resolution,[1,2,6−8] but it comes with its own set of limitations with the most pronounced being the accuracy of the force fields and time sampling limitations. However, we believe that there are further important problems: the data analysis and reproducibility of experiments. In the past few years, specialized hardware,[9] high-throughput methods,[10−12] and advanced sampling techniques[13−15] have been able to significantly improve molecular dynamics, allowing them to reach aggregate simulation times of multiple milliseconds. Force fields have also dramatically improved.[16−20] This increase of simulation accuracy and data has led to the necessity of a more standardized methodology for preparing, executing, and handling thousands of individual trajectories.

Investigating biological processes using MD usually requires the processing of large amounts of data and files using various tools and adapting to peculiarities of many different software packages developed over several decades. With all of these fragile sets of tools, it is hard to follow the steps of a workflow that leads from the original PDB to the results, even for the scientist who wrote the workflow. Second, it is hard to extend the functionality of the tools because of such diversity of languages and the absence of a common programming environment in which to introduce new extensions. In this paper, we illustrate our vision of a unified platform: a programmable workspace for simulation-based molecular discovery. We named it high-throughput molecular dynamics (HTMD)[21] to indicate the fact that it allows for the handling of thousands of simulations and multiple systems in a controlled manner.

HTMD (https://www.htmd.org) extends the Python programming language with functions and classes to handle molecular systems at different levels while abstracting implementation details and best-practice knowledge. Python is a scripting language that enjoys widespread usage in the scientific community and thus provides an ideal platform on which to develop and distribute HTMD. HTMD's functionalities span from molecular structure manipulation to visualization,

Received: January 15, 2016
Published: March 7, 2016

preparing and executing molecular simulations on different computing resources, and data analysis, including the use of Markov state models (MSMs) to identify slow events, kinetic rates, affinities, and pathways.

The need for better tools to deal with large amounts of simulation data has led to the recent development of toolkits covering various facets of the molecular simulation pipeline. Multiple C++,[22,23] Python libraries,[24−26] and web-based environments[27] exist that support reading and writing of PDB files and trajectories as well as various simulation projections (RMSD, contacts, etc.). On a higher level, software packages like Ensembler[28] can combine various tools and libraries like Rosetta[29] and MSMBuilder[30] to allow the building preparation and simulation of whole protein superfamilies, including support for the simulation of proteins of unknown structure through homology modeling, protonation, solvation, and model refinement. The Copernicus[31] platform provides tools for running and managing simulations in distributed environments. These software solutions cover various lengths and phases of the molecular simulation pipeline. To our knowledge, however, HTMD is the first platform to integrate all of the functionalities required for molecular discovery.

2. METHODS

2.1. Integrated Platform for Molecular Modeling.

Listing 1: Manipulating molecular structures.

```
1  # Download the PDB structure of Barnase
2  mol = Molecule('2F4Y')
3  # Set the chain ID of Barnase to 'Y'
4  mol.set('chain', 'Y', sel='protein')
5  # Remove the carbon alpha of residue 15
6  mol.remove('residue 15 and name CA')
7  # Append Barstar to Barnase
8  mol.append(Molecule('2HXX'))
9  # Visualize in VMD or NGL
10 mol.view()
11 # Solvate in water box
12 solvate(mol, minmax=[[-50, -50, -50],
13                      [50 ,  50,  50]])
```

HTMD provides the user with an integrated platform for in silico molecular simulation discovery. Its functionalities range from molecular structure manipulation to system building, docking, MD simulations, simulation management, clustering, Markov models, and adaptive sampling. HTMD can be used from any python interpreter in the form of self-executing scripts as well as interactively using ipython or jupyter notebooks. The jupyter notebooks allow a user to combine code, documentation, and figures in one document, thus integrating a whole experiment, setup, and report in a single file, which can help increase the reproducibility of experiments. Additionally, notebooks provide the possibility for full remote execution of HTMD via a server (e.g., Amazon EC2) and a browser.

Structure Manipulation. Molecular structure information is usually encoded in the form of atomic coordinates in PDB files, which have to be manipulated to prepare simulations. The format of PDB files does not present itself for easy manual manipulation and operations on atoms can become exceedingly complicated. HTMD provides a class for storing and manipulating structural information. It can read, write, and combine PDB files (e.g., Listing 1, lines 2,8) as well as simulation trajectories. Modifications to atom information can be performed (e.g., Listing 1, line 4) using the same powerful atom selection language as the VMD software.[32] Furthermore, HTMD allows the user to do residue mutations and add or delete atoms (e.g., Listing 1, line 6) and provides functions for typical coordinate manipulation like translations and rotations. The visualization of the structures is directly built into HTMD using VMD to inspect structures and modifications (e.g., Listing 1, line 10), although this requires the separate installation of VMD. Furthermore, a webGL embedded viewer[33] is also available from jupyter notebooks to allow for remote execution.

System Building. For system preparation, HTMD tries to encode best practices as well as decouple the system preparation from the respective force fields and simulation software, making the code reusable and allowing the user to change force fields on the fly. HTMD extends the structure manipulation functionality previously described by providing solvation, ionization, and capping as well as simple, interchangeable building of systems for CHARMM and AMBER.

Molecular Simulation. Furthermore, simulation deployment and management are integrated into HTMD. HTMD abstracts these processes by providing a common interface for managing simulations on various computational resources and software, such as local CPU/GPU clusters or remote simulations on Amazon Cloud,[34] allowing the user to quickly switch between a local test run and a production run sent to a remote cluster. Configurations for common procedures such as system equilibration and production runs are encoded in a set of protocols that can be used directly or modified to the user's needs. HTMD provides a common software-independent interface for managing simulations with different simulation software. Currently, protocols and simulation interfaces are provided for ACEMD; however, the set of supported applications is directly extensible by users, and more software will be supported in subsequent versions of HTMD. Likewise, HTMD currently supports the XTC trajectory file format used by ACEMD and GROMACS with more file formats to be supported.

Adaptive Sampling. Adaptive sampling is a key component of HTMD. Traditional MD sampling protocols are relatively inefficient at exploring the conformational space as computation is spent exploring oversampled regions of the conformational space due to metastable minima. More intelligent "adaptive" sampling protocols can better explore the conformational space in an iterative stepwise manner.[15] Adaptive sampling is intended as the normal mode of sampling in HTMD and can therefore be performed in a very simple and automated manner.

Projecting and Clustering. HTMD provides various classes, as shown in Table 1, supporting the calculation of interatom distances, RMSD, dihedral angles, and more. Further projection classes can be written by implementing new classes sharing a given interface. Using HTMD, MD simulations can easily be clustered from a set of millions of protein conformations to a well-defined set of clusters and cluster centers, representing the diversity of conformations found in these simulations as well as defining the boundaries for each conformational cluster.

Markov State Models. Markov state models[35,36] in HTMD are used for simulation analysis as well as adaptive sampling. For simulation analysis, they provide a powerful method for detecting metastable states and calculating kinetics and free energies by integrating any number of simulations into a single statistical model.[11,37−39] For adaptive sampling, Markov state models allow for more advanced sampling methodologies based on metastable states and kinetics. MSM estimation in HTMD is performed using some functionality (TICA,[40,41] PCCA,[42] transition matrix estimation, etc.) from PyEMMA[26] and Scikit-learn[43] for clustering.

Table 1. Projection Classes

metric	description
`MetricDistance`	distances or contacts between two sets of atoms
`MetricSelfDistance`	distances or contacts between all atoms in a single set
`MetricRmsd`	RMSD of a set of atoms from a given reference structure
`MetricSecondaryStructure`	secondary structure of the protein
`MetricDihedral`	dihedral angles (phi/psi) of the protein
`MetricDeviation`	deviations of atoms from reference positions
`MetricShell`	atom densities in concentric shells centered around another set of atoms[44]
`MetricCoodrinate`	coordinates of a set of atoms
`TICA`	time-based independent component analysis

We plan to collaborate with the PyEMMA developers on low-level tools while retaining the application-centric vision of HTMD. In particular, from HTMD it is possible to access lower-level PyEMMA functionalities so as to provide a simple, user-friendly interface for the construction of Markov state models but also directly use PyEMMA when extra MSM functionalities are desired.

2.2. Adaptive Sampling in HTMD. Adaptive sampling is an intelligent sampling protocol designed to sample the conformational space more efficiently than traditional high-throughput MD protocols. By incrementally obtaining knowledge from past simulations, it is able to identify conformational subspaces that are under-sampled and start new simulations from those regions to improve their sampling.

The adaptive protocol used in HTMD is outlined in Figure 1. The protocol begins by running an initial set of `nmax` simulations in what we call the first epoch. It then keeps periodically polling for completed simulations and checking if enough simulations have been completed (i.e., if the number of currently running simulations `nrun` is lower than a threshold `nmin`). If enough (i.e., `nmax`−`nmin`) simulations have been completed, the next epoch will start. Simulations of previous epochs will keep running and will be used once completed. The two `nmax` and `nmin` thresholds are used to make efficient use of asynchronous and asymmetrical computational resources, such as clusters or volunteer computing grids, where some simulations can be delayed for longer periods of time. In the new epoch, an analysis is performed from all previously completed simulations, and the new respawning conformations are selected from the analysis. Then, new simulations are sent out to once again reach `nmax` concurrently running simulations, and the process repeats until `nepochs` number of epochs have completed. All of these steps are completely automated in HTMD and transparent to the user.

In HTMD, the default analysis method used for selecting new respawning conformations utilizes Markov state models. Markov state models prove especially useful in an adaptive sampling scenario as they are able to join short, independent simulations into a single statistical model. The default strategy is different from the adaptive sampling scheme used in ref 15, yet it reproduces the same performance on a more sound mathematical base. By default, the MSM adaptive analysis implemented in HTMD selects the respawning conformations inversely proportional to the amount of sampling of the macrostates of the Markov model in what we call the $\frac{1}{M_c}$ method.

The motivation for this choice is the following. Consider that we have found $n-1$ states and the possibility that there is a

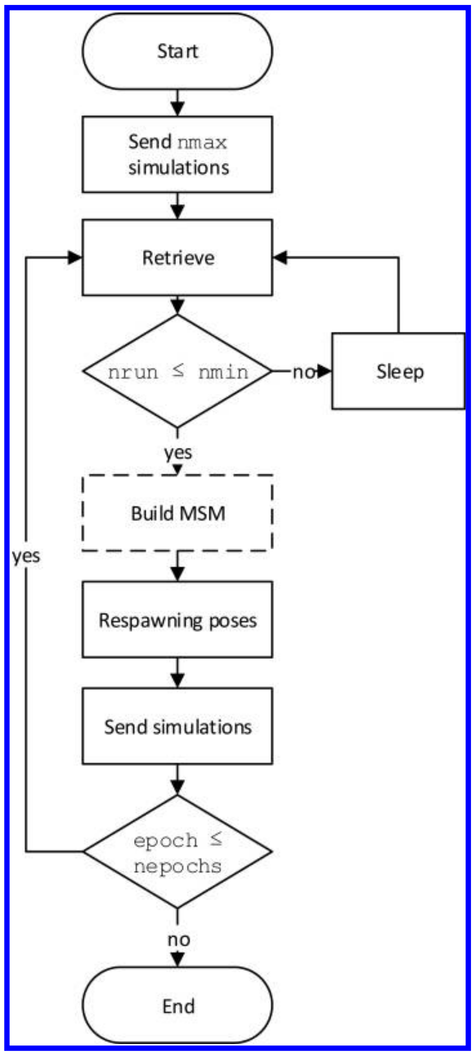

Figure 1. Adaptive protocol flowchart. After an initial set of simulations are sent, completed simulations are polled. Once enough simulations have been completed, an analysis (MSM) is performed to calculate the starting poses of the simulations of the next epoch. Finally, the simulations of the new epoch are sent and the loop repeats. The dotted line is used to indicate that the MSM analysis can readily be replaced by other adaptive sampling algorithms while keeping the backbone of the algorithm fixed.

representative nth state that has not yet been discovered. For a given state i, the outgoing transition probabilities $p_{i,1}$ to $p_{i,n}$ are distributed according to the Dirichlet distribution

$$\mathbb{P}(p_{i,1}, ..., p_{i,n-1}, p_{i,n}) = \frac{1}{B(c_{i,1}+1, ..., c_{i,n-1}+1, c_{i,n}+1)} \prod_j T_{ij}^{c_{ij}+1} \quad (1)$$

where B is the Beta distribution and $\mathbf{C} = [c_{ij}]$ are posterior transition counts. The probability of finding a hypothetical nth state from state i is given by

$$\mathbb{E}(p_{i,n}) = \int_{p_i} \mathbb{P}(p_{i,1}, ..., p_{i,n-1}, p_{i,n}) p_{i,n} \, \mathrm{d}p_i \quad (2)$$

Using a uniform prior, this evaluates to

$$\mathbb{E}[p_{i,n}] = \frac{1}{z_i + n} \approx \frac{1}{z_i} \quad (3)$$

where z_i is the number of observed counts in microstate i. On the basis of this simple idea, we sample macrostates proportional to z_i^{-1}. The reason to use macrostates rather than microstates is simple but important as it allows for minimizing the statistical error present in microstates due to poor clustering. For further details on the adaptive equations, see Supporting Information section I. As demonstrated here and in ref 45, count-based adaptive sampling allows for the discovery of new states with orders of magnitude less simulation time than conventional sampling.

The optimal criteria for the selection of the respawning conformations, however, is still an open question in adaptive sampling. Different criteria can be defined for adaptive sampling, such as reducing the sampling error in a Markov model,[46] faster sampling of binding events,[15] sampling along the slowest pathways, and many more.[15,31,45−49] Therefore, the adaptive protocol in HTMD is written with modularity in mind, and the analysis method can be easily replaced by any other analysis that the user wants to perform on the simulations (not limited to Markov models), allowing for the development of further adaptive protocols and easy experimentation.

3. RESULTS

To demonstrate the capabilities of HTMD, we provide three examples: first, a protein folding analysis on a large data set of folding and unfolding simulations of the villin headpiece; second, the sampling of the same system using the adaptive sampling functionalities of HTMD compared with the large data set, and third, the adaptive sampling of a free ligand binding example for Thrombin and a small ligand. Short HTMD code listings are provided for all examples.

3.1. Protein Folding Analysis.

Listing 2: Markov models

```
1  # Collect and link simulation files
2  sims = simlist(glob('data/*/'),
     ↪  glob('input/*/structure.pdb'))
3  # Calculate protein contact maps
4  met = Metric(sims)
5  met.projection( MetricSelfDistance(
     ↪  'protein and name CA',
     ↪  metric='contacts') )
6  data = met.project()
7  # Project on slowest 10 TICA dimensions
8  tica = TICA(data, 20)
9  dataTica = tica.project(10)
10 # Cluster data with kmeans
11 dataTica.cluster(
     ↪  MiniBatchKMeans(n_clusters=1000))
12 # Link a model with the data
13 model = Model(dataTica)
14 # Calculate the Markov model
15 model.markovModel(300, 4)
16 # Macrostates equilibrium distribution
17 model.eqDistribution()
18 # Visualize the macrostates in VMD
19 model.viewStates()
20 # Calculate the kinetics
21 kin = Kinetics(model, temperature=360)
22 kin.getRates()
```

Here, we demonstrate the HTMD Markov state model analysis of the protein villin using the same setup as Piana et al.[50] Villin is a tissue-specific protein that binds to actin. In this setup, we use the double norleucine K65Nle/K70Nle mutant of the 35 amino acid C-terminal headpiece of villin. The data set consists of 1614 simulations of 120 ns each with an aggregate simulation time of 193.6 μs simulated using ACEMD[51] on the GPUGRID.net distributed computing infrastructure.[10]

First, all simulation folders that will be used in the analysis are added to a list using the `simlist` function (Listing 2, line 2). Individual trajectories are assumed to be located in separate folders, and whose names are used as identifiers for the trajectories. Multiple trajectory files stored in a single folder will be assumed to be continuous pieces of the same trajectory and will be appended internally. The `simlist` function also links trajectory folders to their corresponding structure files, thus providing all of the atom information needed by HTMD to calculate the various projections.

Clustering methods can be used to define states for a Markov model given simulation trajectories. However, as the molecular system coordinates are very high-dimensional, using them as feature vectors complicates clustering.[52] Additionally, because in the model we are interested in conformational clusters, feature vectors of coordinates are highly unsuitable as they are not a rotation and translation invariant representation, which is typically desired when comparing different protein conformations or protein−ligand interactions. Therefore, it is preferred to project the simulations on a lower dimensional representation before clustering using the projection classes described in Table 1.

In this example, protein contact maps were calculated from the simulation coordinates using the `MetricSelfDistance` class (Listing 2, line 5). In detail, contact maps were constructed by calculating the distances between all carbon-alpha atoms of the protein and setting them to 1 if the distance is smaller than 8 Å and 0 otherwise. Different projection methods could be used in this case to represent protein conformations, such as backbone dihedral angles or protein secondary structure; however, protein contact maps tend to represent the various conformations in a protein well, giving good results. The contact maps were further projected using time-lagged independent component analysis (`TICA`) onto the 10 slowest varying subspaces (Listing 2, lines 8 and 9) to aid the clustering method in placing the clusters on the transition regions between metastable states, which improves the model quality.[35]

The TICA coordinates were then clustered using the `MiniBatchKMeans` class of Scikit-learn[43] into 1000 clusters (Listing 2, line 11). The selection of the clustering method and number of clusters is left up to the user as different clustering methods can perform better on different simulation sets. In some cases, it might be necessary for the user to try out various configurations to obtain the best results. A higher number of clusters produces smaller clusters, which increase the accuracy and spatial resolution of the Markov model. Small geometric clusters tend to also produce more kinetically homogeneous clusters by eliminating energetic barriers within states. However, a very large amount of small geometric clusters can lead to poor statistics causing a less precise model.[36] Therefore, a compromise has to be made when choosing the number of clusters.

Discrete-time Markov models, as in our case, model a system as a jump process between a set of states with a discrete jump time called the lag time τ. The most widely used heuristic for choosing a lag time for the Markov model is by inspecting the convergence of the implied time-scales of the model over increasing lag times. Given the convergence of the implied time

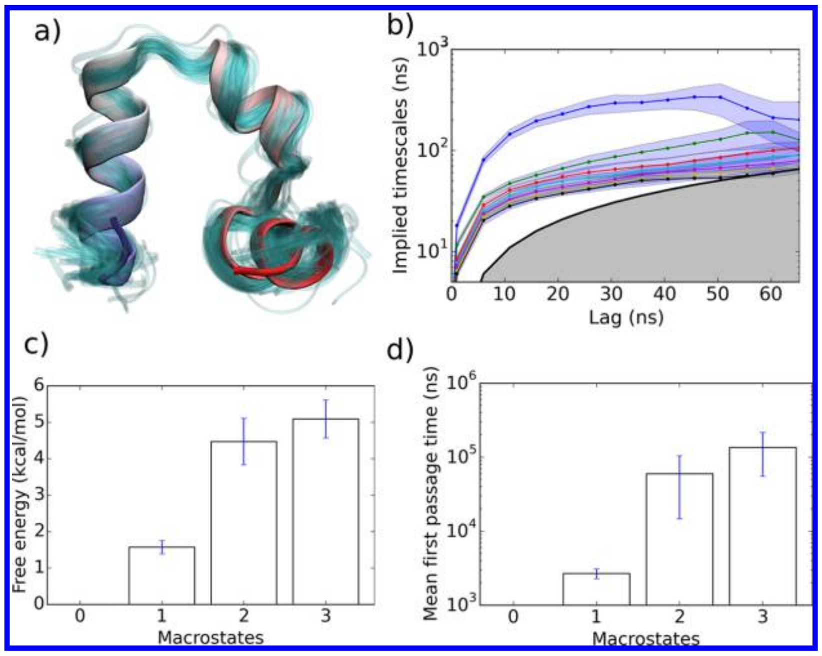

Figure 2. Results of the MSM analysis of the simulations of villin showing (a) the MSM detecting the folded state of villin as macrostate 1 (teal stripes) overlaid on the crystal folded structure in the colored cartoon (for the rest of the macrostates, see Figure S2), (b) an implied time scales plot showing the convergence of the slowest process of the MSM for lag times between 30 and 40ns, (c) the standard free energy of each macrostate, and (d) the mean first passage time from the unfolded macrostate to each macrostate. Macrostate 1 corresponds to the protein folding time. Macrostate 0 corresponds to the unfolded state. Error bars were calculated by bootstrapping the simulations and recalculating the MSM.

scales in Figure 2b, a lag time of 30 ns = 300 frames was chosen, and the clusters were lumped into four metastable macrostates using PCCA+[42] (Listing 2, line 15).

From the Markov model transition probability matrix, it is possible to obtain the equilibrium populations of each macrostate as well as the rates, mean first passage times, and standard free energies of the macrostates. These observables are of great importance to biologists and chemists as they can be measured experimentally such that MSMs provide a predictive and informative methodology.

The folding free energy ΔG is calculated as $\Delta G = -k_B T \log\left(\frac{p_F}{p_U}\right)$ where k_B is the Boltzmann constant, T is the temperature in Kelvin, p_F is the equilibrium probability of the protein being folded, and p_U is the equilibrium probability of the unfolded state. The folding time is calculated by the mean first passage time from the unfolded to folded state. The free energy and folding time of the protein were calculated (Listing 2, line 22) as 1.5(±0.2) kcal/mol and 2.6(±0.4) μs, respectively. These compare reasonably well with the values obtained by Piana et al.[50] (−0.6 kcal/mol and 3.2 μs).

3.2. Adaptive Sampling Protein Folding.

Listing 3: Adaptive sampling protein folding

```
1  md = AdaptiveRun()
2  md.project = 'Villin'
3  md.nmin=10
4  md.nmax=20
5  md.nepochs = 60
6  md.app = AcemdLocal()
7  md.metricsel1 = 'protein and name CA'
8  md.metrictype = 'contacts'
9  md.ticadim = 10
10 md.method = '1/Mc'
11 md.run()
```

Below, we describe the adaptive sampling methodology applied on sampling the protein folding process of villin. The simulation setup is the same one used for the analysis above with the exception of the trajectory lengths, which are made shorter (50 ns) to better suit the adaptive protocol.

We start off with a set of one or more simulation input folders provided by the user or constructed using the HTMD build system. Code Listing 3 then shows the adaptive configuration script for HTMD. Lines 1−10 define the parameters for the adaptive run and line 11 starts the execution of the adaptive protocol. The adaptive protocol starts by reading the initial "generator" simulation input files from a default path, which can be modified, and spawns the first set of `nmax` simulations (specified in line 4). Lines 3 and 4 define the maximum and minimum number of simulations that should be running at any moment in the adaptive run. Once the number of running simulations `nrun` falls under `nmin` (e.g., `nmax` − `nmin` simulations have completed since the last epoch), a Markov state model will be built using all available simulations. Inside the `AdaptiveRun.run()` method, a projection class calculates the contact map (Listing 3, line 8) between all carbon alpha atoms of the protein (Listing 3, line 7) for all trajectories. These contact maps are further projected onto the 10 slowest dimensions using `TICA` (Listing 3, line 9). The `TICA` projected data is then clustered using `MiniBatchKMeans`; an MSM is built from the projected simulations, and the respawning conformations are calculated using the $\frac{1}{M_c}$ method (Listing 3, line 10). The new simulations are then run using ACEMD[51] on local GPUs as defined in Listing 3, line 6.

Figure 3 demonstrates the progress of the folding and unfolding time estimates over 60 epochs of adaptive sampling (36.6 μs), comparing it to the progress using nonadaptive sampling of equivalent aggregate simulation time and longer trajectories. The adaptive sampling method shows a much faster

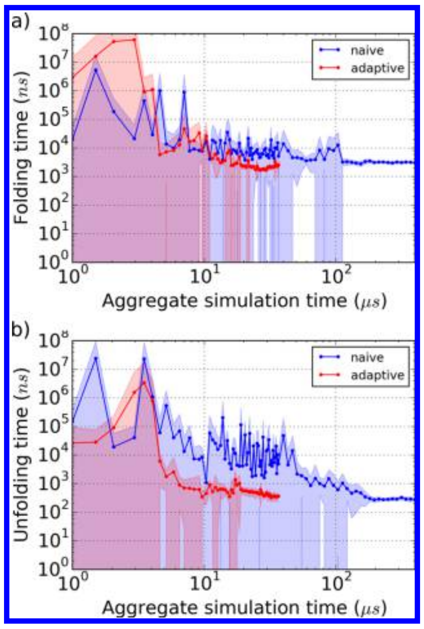

Figure 3. Plots show progression of the estimation of (a) folding and (b) unfolding times for villin over 60 adaptive epochs (red) compared to equal amounts of simulation data produced by naive sampling (blue).

convergence in folding and unfolding time compared to naive sampling as expected from previous studies.[15,45]

3.3. Adaptive Sampling Ligand Binding.

Listing 4: Adaptive sampling ligand binding

```
md = AdaptiveRun()
md.project = 'Thrombin'
md.nmin=5
md.nmax=10
md.nepochs = 60
md.app = AcemdLocal()
md.metricsel1 = 'protein and name CA'
md.metricsel2 = 'resname MOL and noh'
md.metrictype = 'contacts'
md.ticadim = 3
md.method = '1/Mc'
md.run()
```

In the following example, we demonstrate the adaptive sampling methodology applied on sampling the binding process of the ligand piperidine-1-carboxamidine to the protein thrombin taken from PDB ID 3D49. The procedure followed here is similar to the one used for villin. The main differences are the following: Fewer simulations are performed per epoch due to the faster nature of ligand binding, meaning that we do not need to sample each epoch as thoroughly as in the case of villin. A different metric is used, in this case, the contacts between the ligand heavy atoms and the carbon-alpha atoms of the protein, to allow the Markov model to detect protein−ligand interactions. Fewer TICA dimensions are also used, as we know from previous analyses of similar systems (benzamidine-trypsin) that 3 TICA dimensions are enough to resolve the binding of the ligand.

Figure 4 demonstrates the progress of the slowest implied time scale as well as the progress of the standard free energy of binding over 60 epochs of adaptive sampling. Our results of −5.97(±0.50) kcal/mol of binding free energy in epoch 60 compare favorably to the results published by Ruhmann et al.[53]

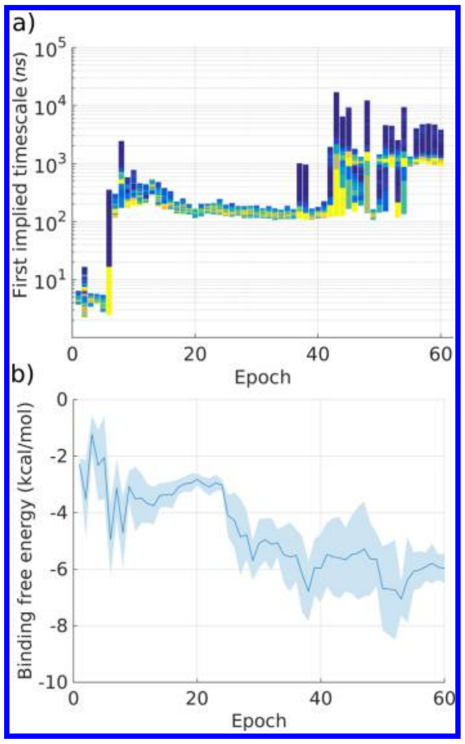

Figure 4. (a) The distributions of the estimated implied time scales (in ns) of the slowest process detected in the simulations calculated over various epochs. Blue to orange coloring shows increasing amounts of bootstrapped models estimating the given time scale value. (b) The estimated binding free energy of piperidine-1-carboxamidine to thrombin calculated over various epochs. The light blue outline is the standard deviation.

(−5.41(±0.22) kcal/mol) using isothermal titration calorimetry for the binding of the ligand. The implied time scale plot demonstrates how the Markov model detects slower processes over increasing epochs. Each epoch is visualized as a vertical histogram showing the distribution of the values of the top time scale calculated for that epoch using bootstrapping. Two large jumps can be seen at epochs 7 and 43 attributed to the discovery of a metastable prebound and bound state with corresponding time scales on the order of 10^2 and 10^3 ns. Between epochs 43 and 54, the distributions appear bimodal because of the lack of statistics and bootstrapping of the data; the various calculated Markov models are not all able to consistently identify the newly discovered slowest process of 10^3 ns. After epoch 54, we have acquired enough statistics to converge to the correct time scale of 10^3 ns, which in turn leads to a more converged free energy estimate in the free energy plot.

4. CONCLUSIONS

With HTMD, we intend to integrate the whole workflow of molecular simulation-based discovery in a single environment while abstracting unnecessary technical details. This reduces preventable errors in the workflow, increases reproducibility of molecular dynamics experiments, allows for the manipulation of large amounts of simulation data, and opens the way for biologists and medicinal chemists to utilize simulations while focusing on the real biological problems they are trying to solve. In this paper, we described the capabilities of HTMD on analyzing MD simulations using Markov models and demonstrated its application on improving conformational space exploration through adaptive sampling. Additionally, it is our goal to allow for easy development of new protocols and methods by the community, which can be included in future releases of HTMD by providing access to the code repository. The HTMD software, documentation, tutorials, and examples are available at https://www.htmd.org, where it can be downloaded for free by academic users.

ASSOCIATED CONTENT

Supporting Information

The Supporting Information is available free of charge on the ACS Publications website at DOI: 10.1021/acs.jctc.6b00049.

Additional equations explaining the adaptive sampling method, figure showing how the number of clusters is chosen in the adaptive sampling, and figure of all macrostates of the Markov model of villin (PDF)

AUTHOR INFORMATION

Corresponding Author

*E-mail: gianni.defabritiis@upf.edu.

Notes

The authors declare the following competing financial interest(s): G.D.F. and M.J.H. are shareholders of Acellera Ltd., which provides the non-academic version of HTMD. S.D. is partially funded by Acellera Ltd.

ACKNOWLEDGMENTS

We thank Acellera Ltd. for funding. G.D.F. aknowledges support from MINECO (BIO2014-53095-P) and FEDER. We thank the volunteers of GPUGRID for donating their computing time for the simulations.

REFERENCES

(1) Dror, R. O.; Green, H. F.; Valant, C.; Borhani, D. W.; Valcourt, J. R.; Pan, A. C.; Arlow, D. H.; Canals, M.; Lane, J. R.; Rahmani, R.; Baell, J. B.; Sexton, P. M.; Christopoulos, A.; Shaw, D. E. *Nature* **2013**, *503*, 295−299.

(2) Nygaard, R.; Zou, Y.; Dror, R. O.; Mildorf, T. J.; Arlow, D. H.; Manglik, A.; Pan, A. C.; Liu, C. W.; Fung, J. J.; Bokoch, M. P.; Thian, F. S.; Kobilka, T. S.; Shaw, D. E.; Mueller, L.; Prosser, R. S.; Kobilka, B. K. *Cell* **2013**, *152*, 532−542.

(3) Nolan, T. L.; Geffert, L. M.; Kolber, B. J.; Madura, J. D.; Surratt, C. K. *ACS Chem. Neurosci.* **2014**, *5*, 784−792.

(4) Shukla, D.; Meng, Y.; Roux, B.; Pande, V. S. *Nat. Commun.* **2014**, *5*, 3397.

(5) Kurzweil, R. *The singularity is near: when humans transcend biology*; Viking: New York, 2005.

(6) Lindorff-Larsen, K.; Piana, S.; Dror, R. O.; Shaw, D. E. *Science* **2011**, *334*, 517−520.

(7) Jensen, M.; Jogini, V.; Borhani, D. W.; Leffler, A. E.; Dror, R. O.; Shaw, D. E. *Science* **2012**, *336*, 229−233.

(8) Arkhipov, A.; Shan, Y.; Das, R.; Endres, N. F.; Eastwood, M. P.; Wemmer, D. E.; Kuriyan, J.; Shaw, D. E. *Cell* **2013**, *152*, 557−569.

(9) Shaw, D. E.; Grossman, J. P.; Bank, J. A.; Batson, B.; Butts, J. A.; Chao, J. C.; Deneroff, M. M.; Dror, R. O.; Even, A.; Fenton, C. H.; Forte, A.; Gagliardo, J.; Gill, G.; Greskamp, B.; Ho, C. R.; Ierardi, D. J.; Iserovich, L.; Kuskin, J. S.; Larson, R. H.; Layman, T.; Lee, L.-S.; Lerer, A. K.; Li, C.; Killebrew, D.; Mackenzie, K. M.; Mok, S. Y.-H.; Moraes, M. A.; Mueller, R.; Nociolo, L. J.; Peticolas, J. L.; Quan, T.; Ramot, D.; Salmon, J. K.; Scarpazza, D. P.; Ben Schafer, U.; Siddique, N.; Snyder, C. W.; Spengler, J.; Tang, P. T. P.; Theobald, M.; Toma, H.; Towles, B.; Vitale, B.; Wang, S. C.; Young, C. Anton 2: Raising the Bar for Performance and Programmability in a Special purpose Molecular Dynamics Supercomputer. *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. Piscataway, NJ, USA, 2014; pp 41−53.

(10) Buch, I.; Harvey, M. J.; Giorgino, T.; Anderson, D. P.; De Fabritiis, G. *J. Chem. Inf. Model.* **2010**, *50*, 397−403.

(11) Buch, I.; Giorgino, T.; De Fabritiis, G. *Proc. Natl. Acad. Sci. U. S. A.* **2011**, *108*, 10184−10189.

(12) Kohlhoff, K. J.; Shukla, D.; Lawrenz, M.; Bowman, G. R.; Konerding, D. E.; Belov, D.; Altman, R. B.; Pande, V. S. *Nat. Chem.* **2014**, *6*, 15−21.

(13) Bowman, G. R.; Ensign, D. L.; Pande, V. S. *J. Chem. Theory Comput.* **2010**, *6*, 787−794.

(14) Bacci, M.; Vitalis, A.; Caflisch, A. *Biochim. Biophys. Acta, Gen. Subj.* **2015**, *1850*, 889−902.

(15) Doerr, S.; De Fabritiis, G. *J. Chem. Theory Comput.* **2014**, *10*, 2064−2069.

(16) Hornak, V.; Abel, R.; Okur, A.; Strockbine, B.; Roitberg, A.; Simmerling, C. *Proteins: Struct., Funct., Genet.* **2006**, *65*, 712−725.

(17) Best, R. B.; Buchete, N.-V.; Hummer, G. *Biophys. J.* **2008**, *95*, L07−L09.

(18) Lindorff-Larsen, K.; Piana, S.; Palmo, K.; Maragakis, P.; Klepeis, J. L.; Dror, R. O.; Shaw, D. E. *Proteins: Struct., Funct., Genet.* **2010**, *78*, 1950−1958.

(19) Piana, S.; Lindorff-Larsen, K.; Shaw, D. E. *Biophys. J.* **2011**, *100*, L47−49.

(20) Lindorff-Larsen, K.; Maragakis, P.; Piana, S.; Eastwood, M. P.; Dror, R. O.; Shaw, D. E. *PLoS One* **2012**, *7*, e32131.

(21) Harvey, M. J.; De Fabritiis, G. *Drug Discovery Today* **2012**, *17*, 1059−1062.

(22) Yesylevskyy, S. O. *J. Comput. Chem.* **2012**, *33*, 1632−1636.

(23) Romo, T. D.; Leioatts, N.; Grossfield, A. *J. Comput. Chem.* **2014**, *35*, 2305−2318.

(24) Michaud-Agrawal, N.; Denning, E. J.; Woolf, T. B.; Beckstein, O. *J. Comput. Chem.* **2011**, *32*, 2319−2327.

(25) McGibbon, R.; Beauchamp, K.; Harrigan, M.; Klein, C.; Swails, J.; Hernández, C.; Schwantes, C.; Wang, L.-P.; Lane, T.; Pande, V. *Biophys. J.* **2015**, *109*, 1528−1532.

(26) Scherer, M. K.; Trendelkamp-Schroer, B.; Paul, F.; Pérez-Hernández, G.; Hoffmann, M.; Plattner, N.; Wehmeyer, C.; Prinz, J.-H.; Noé, F. *J. Chem. Theory Comput.* **2015**, *11*, 5525−5542.

(27) Jeong, J. C.; Jo, S.; Wu, E. L.; Qi, Y.; Monje-Galvan, V.; Yeom, M. S.; Gorenstein, L.; Chen, F.; Klauda, J. B.; Im, W. *J. Comput. Chem.* **2014**, *35*, 957−963.

(28) Parton, D. L.; Grinaway, P. B.; Hanson, S. M.; Beauchamp, K. A.; Chodera, J. D. *bioRxiv* **2015**, 018036.

(29) Rohl, C. A.; Strauss, C. E. M.; Misura, K. M. S.; Baker, D. *Methods Enzymol.* **2004**, *383*, 66−93.

(30) Beauchamp, K. A.; Bowman, G. R.; Lane, T. J.; Maibaum, L.; Haque, I. S.; Pande, V. S. *J. Chem. Theory Comput.* **2011**, *7*, 3412−3419.

(31) Pronk, S.; Larsson, P.; Pouya, I.; Bowman, G.; Haque, I.; Beauchamp, K.; Hess, B.; Pande, V.; Kasson, P.; Lindahl, E. *High Performance Computing, Networking, Storage and Analysis* **2011**, 1−10.

(32) Humphrey, W.; Dalke, A.; Schulten, K. *J. Mol. Graphics* **1996**, *14*, 33−38.

(33) Rose, A. S.; Hildebrand, P. W. *Nucleic Acids Res.* **2015**, *43*, W576−W579.

(34) Harvey, M. J.; De Fabritiis, G. *J. Chem. Inf. Model.* **2015**, *55*, 909−914.

(35) Prinz, J.-H.; Wu, H.; Sarich, M.; Keller, B.; Senne, M.; Held, M.; Chodera, J. D.; Schütte, C.; Noé, F. *J. Chem. Phys.* **2011**, *134*, 174105−174105−23.

(36) Bowman, G. R.; Beauchamp, K. A.; Boxer, G.; Pande, V. S. *J. Chem. Phys.* **2009**, *131*, 124101.

(37) Shukla, D.; Hernández, C. X.; Weber, J. K.; Pande, V. S. *Acc. Chem. Res.* **2015**, *48*, 414−422.

(38) Lane, T. J.; Bowman, G. R.; Beauchamp, K.; Voelz, V. A.; Pande, V. S. *J. Am. Chem. Soc.* **2011**, *133*, 18413−18419.

(39) Plattner, N.; Noé, F. *Nat. Commun.* **2015**, *6*, 7653.

(40) Pérez-Hernández, G.; Paul, F.; Giorgino, T.; De Fabritiis, G.; Noé, F. *J. Chem. Phys.* **2013**, *139*, 015102.

(41) Schwantes, C. R.; Pande, V. S. *J. Chem. Theory Comput.* **2013**, *9*, 2000−2009.

(42) Deuflhard, P.; Weber, M. *Linear Algebra Appl.* **2005**, *398*, 161−184.
(43) Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; Duchesnay, E. *J. Mach. Learn. Res.* **2011**, *12*, 2825−2830.
(44) Harrigan, M. P.; Shukla, D.; Pande, V. S. *J. Chem. Theory Comput.* **2015**, *11*, 1094−1101.
(45) Zimmerman, M. I.; Bowman, G. R. *J. Chem. Theory Comput.* **2015**, *11*, 5747−5757.
(46) Singhal, N.; Pande, V. S. *J. Chem. Phys.* **2005**, *123*, 204909.
(47) Hinrichs, N. S.; Pande, V. S. *J. Chem. Phys.* **2007**, *126*, 244101.
(48) Weber, J. K.; Pande, V. S. *J. Chem. Theory Comput.* **2011**, *7*, 3405−3411.
(49) Preto, J.; Clementi, C. *Phys. Chem. Chem. Phys.* **2014**, *16*, 19181−19191.
(50) Piana, S.; Lindorff-Larsen, K.; Shaw, D. E. *Proc. Natl. Acad. Sci. U. S. A.* **2012**, *109*, 17845−17850.
(51) Harvey, M. J.; Giupponi, G.; De Fabritiis, G. *J. Chem. Theory Comput.* **2009**, *5*, 1632−1639.
(52) Kriegel, H.-P.; Kröger, P.; Zimek, A. *ACM Trans. Knowl. Discovery Data* **2009**, *3*, 1−58.
(53) Rühmann, E.; Betz, M.; Fricke, M.; Heine, A.; Schäfer, M.; Klebe, G. *Biochim. Biophys. Acta, Gen. Subj.* **2015**, *1850*, 647−656.