Greedy Algorithms for Fast Discovery of Macrostates from Molecular Dynamics Simulations

Haoyun Feng
Department of Computer Science and Engineering, Notre Dame, US
Email: hfeng@nd.edu

Geoffrey Siwo
Institute for Quantitative Biomedical Sciences, Geisel School of Medicine, Dartmouth College, US
Email: siwomolbio@gmail.com

Jesus A. Izaguirre, Douglas Thain, and Badi Abdul-Wahid
Department of Computer Science and Engineering, Notre Dame, US
Email: {izaguirr, dthain, cabdulwa}@nd.edu

***Abstract*—With development of distributed computing systems, it is possible to significantly accelerate long-term molecular dynamics simulations by using ensemble algorithms, such as Markov State Models (MSM) and Weighted Ensemble (WE). Decomposing the conformational space of molecule into macrostates is an important step of both methods. To ensure efficiency and accuracy of ensemble methods, it is necessary that the macro states are defined according to certain kinetic properties. Monte Carlo simulated annealing (MCSA) has been widely applied to define macro states with optimal metastability of the dynamical system. This article proposes two greedy algorithms, G1 and G2, based on different definitions of local search space to improve efficiency and scalability of MCSA on distributed computing system. Numerical experiments are conducted on two biological systems, alanine dipeptide and WW domain. The numerical experiments demonstrate that G1 is the most efficient of the three on a single core machine and distributed computing system. Sequential version of G2 is the slowest but it gains the most speed up on distributed computing systems.**

***Index Terms*—molecular dynamics, metastable states, unsupervised clustering, greedy algorithm.**

I. Introduction

Molecular Dynamics simulation [1] is a numerical tool for simulating movements of molecules. It generates a sequence of coordinates representing Brownian motion of molecules, which is called a trajectory. Biological studies such as drug design usually require longterm (mili-seconds) simulations that may cost several years of CPU clock time. However, with development of distributed computing systems, it is possible to generate a large amount of short trajectories in parallel. Recovering long-term dynamics from a large number of short trajectories has drawn the attention of many researchers and a number of techniques have been developed for such purpose. The two most popular techniques are Markov State Models (MSM) [2]-[6] and Weighted Ensembles (WE) [2], [7]-[11].

Partitioning the continuous conformational space into macrostates plays a key role in both techniques. MSM calculates implied time scales of the dynamics from the transition matrix, within which each entry represents transition probability between two macrostates. The quality of macrostate definitions affects accuracy of MSM. On the other hand, WE is a distributed algorithm that provides efficient sampling on conformational space and reliable estimation of reaction rate between free energy wells. Macrostates are defined so that WE samples the conformational space evenly by maintaining a constant number of samples in every macrostate. Efficiency of WE depends on the underlying partition.

A number of studies have been done on the automatic discovery of macrostates. Chodera proposed to find clusters for MSM using Perron cluster cluster analysis (PCCA) and Monte Carlo simulated annealing (MCSA) [12]. PCCA is efficient in time complexity but has statistical errors. MCSA takes PCCA result as initial solution and continues to reduce the statistical errors. The advantage of MCSA is that it guarantees for finding the optimal clusters, but the time complexity is high and unpredictable, especially the sequential version demonstrated in [12]. Chang and Yao proposed to reduce statistical errors of PCCA by clustering densely and sparsely sampled conformational spaces separately, so that the sparsely sampled space won't affect accuracy on the overall clustering [13], [14]. However, a disadvantage of PCCA is that it only finds free energy wells. It is not suitable for any other clustering criteria. This article proposes two greedy algorithms that generate clusters optimizing any predefined target functions. The greedy

Manscript received October 12, 2014; revised January 5, 2015.

doi: 10.12720/lnit.2.4.302-309

algorithms are more efficient than MCSA and more scalable on distributed computing systems.

All three algorithms, G1, G2 and MCSA, are applied to study two biological systems, alanine dipeptide and WW domain. On single core machine, G1 is the most efficient of the three. On distributed computing system, as the number of steps executed in parallel increases, the total number of steps required for convergence also increases. MCSA has the most increase in the number of steps, which leads to the worst scalability. G2 has almost no increase in the number of steps, so achieves the best scalability. G1 gains less speed up than G2 on distributed computing system, but since the sequential version of G1 is much more efficient than G2, G1 is still the most efficient on distributed computing systems.

The paper is organized as the following. Section II introduces metastability, the target function used throughout this article, describes the existing method MCSA and the two greedy algorithms G1 and G2 developed by authors. Section III shows the experimental results of applying these algorithms on two biological systems, alanine dipeptide and WW domain. Theoretical performance is examined on the simpler protein alanine dipeptide while actual performance is examined on the medium sized protein WW domain.

II. Methodology

All clustering algorithms are applied on a set of samples of molecule coordinates. There are two stages of clustering. In the first stage, coordinates of conformations are clustered into Voronoi bins. A commonly used algorithm for this purpose is the K-centroid algorithm. Once microstates are defined, one could calculate transition probability from one microstate to another. Clustering algorithms discussed in this article work for the second stage, which clusters microstates into macrostates to satisfy certain dynamical properties. Here we consider the metastability as the target property. Hence, a macrostate, also considered as metastable state, consists of many microstates with large transitional probability between each other, while different metastable states have small transitional probability between them. In this section, the formula for computing metastability is discussed in Section A. MCSA, two greedy algorithms, G1 and G2, and a design of implementation on distributed system are introduced in Sections B, C, D and E respectively.

A. Metastability

Metastability quantitatively measures the quality of metastable states. Given a partition of underlying conformational space into metastable states $\{S_i\}_{i=1}^{L}$, $P(S_i, S_j, \tau)$ the transitional probability between S_i and S_j in time lag τ, metastability Q is defined as the sum of transition probabilities within each metastable-state.

$$Q = \sum_{i=1}^{L} P(S_i, S_j, \tau) \tag{1}$$

More specifically, an MD trajectory can be considered as a list of states that propagates over time. $MS(t)$ - the metastable state at time t in trajectory, $P(S_i, S_j, \tau)$ can be calculated using raw trajectories with length T according to the following formula.

$$P(S_i, S_j, \tau) = \frac{\sum_{t=0}^{T-\tau} \delta(MS(t) = S_i, MS(t+\tau) = S_j)}{\sum_{t=0}^{T-\tau} \delta(MS(t) = S_i)} \tag{2}$$

where $\delta(\bullet)$ equals to one if the statement $(\bullet)$ is true, zero otherwise. The transitional probability from S_i to S_j in time lag τ is equal to the number of transitions made from S_i to S_j in τ divided by the total number of transitions made from S_i to any other states in τ.

B. Monte Carlo Simulated Annealing

Monte Carlo Simulated Annealing (MCSA) is a stochastic algorithm for searching global optima. It requires an initial partition, which could be generated randomly, or by other fast algorithms with poor accuracy, e.g. PCCA. In each iteration of MCSA, one microstate m_i and one metastable state S_j are picked randomly. Moving m_i from its original metastable state to S_j will cause change of metastability Q to $Q + \Delta Q$, m_i will be moved to S_j with probability $\min\left(1, e^{\beta \Delta Q}\right)$, where β represents inverse of temperature in MC procedure. In [1], β is set to number of MCSA steps that have been executed, which ensures that the probability of accepting movement decreases with increasing MCSA steps.

Each iteration of MCSA takes $O(N/L)$ on average and $O(N)$ in the worst case, where N denotes the number of microstates and L denotes the number of macrostates. The stochastic nature of MCSA guarantees that the search process won't get stuck in a local minimum and will reach global optima at the end. However, the number of iterations required for convergence is highly unpredictable, which makes the algorithm inefficient. On the other hand, in sequential version, every MCSA step makes stochastic move towards maximum metastability from previous MCSA step. Parallelizing m steps means that the $(i+m)$th step will rely on assignment in i^{th} step. Without realizing changes made by i, $i+1$, ... , $i+m-1$ steps, the $(i+m)^{th}$ move will contain significant noise.

C. Greedy Algorithm (G1)

Unlike the stochastic algorithm, greedy algorithms are deterministic. At one iteration, a set of local movements is defined and the one that increases the metastability the most will be selected. Many greedy algorithms have been discussed for the optimization problem [15]. Here, two greedy algorithms are proposed for the specific problem of finding macrostates.

Similar to MCSA, greedy algorithms need initial solution to start as well. In one iteration of this approach, a microstate is selected by order, it examines the increase in metastability by moving this microstate from its original macrostate to all other macrostates, choose the macrostates with maximum increase of metastability ΔQ and make the move. Formally, we have

$$S^{new}(m_{i0}) = \arg\max_{j=1}^{L}(\Delta Q(m_{i0} -> S_j)) \quad (3)$$

where $S^{new}(m_{i0})$ denotes the updated metastable state for microstate m_{i0}, $\Delta Q(m_{i0} -> S_j)$ denotes the ΔQ obtained by moving m_{i0} from its original metastable state to S_j. The pseudo code of this algorithm is as follows:

```
Input: Part = {S(m_i)}_{i=1,2,..., N}
       //S(m_i) denotes the index of macrostate assigned
       //to microstate i initially
Output: Optimal macrostate assignment {S(m_i)}_{i=1,2,..., N}

// meta(.) calculates metastability of current partition
Q0 = meta(Part)
Qdiff = threshold+1

//Check if metastability has converged
WHILE Qdiff > threshold
    FOR i = 1,2, …, N
        Qdiff_i = 0
        Part0 = Part
        FOR j = 1, 2, …, L
            Part1 = Part
            Part1(S(m_i)) = j
            IF meta(Part1)-meta(Part) > Qdiff_i
                Qdiff_i = meta(Part1) – meta(Part)
                Part0 = Part1
            END IF
        END FOR
        Part = Part0
    END FOR
    Qdiff = meta(Part) – Q0
    Q0 = meta(Part)
END WHILE
```

In a single iteration of G1, microstate m_i is moved to S_j to attain maximum gain on metastability ΔQ. This requires computing ΔQ L times for assigning m_i to $S_1, \ldots, S_L$ respectively. The computational time for assigning m_i to S_j is *O(# of microstates in S_j)*. In all L macrostates, there are N microstates in total. Therefore, the computational time for one iteration is *O(N)*. With reasonable initialization, this algorithm converges in *O(N)* iterations, so overall time complexity is approximately $O(N^2)$.

The greedy algorithm is more scalable on distributed computing system than MCSA, because it takes a longer time to make one update of $S(m_i)$ that increases metastability more significantly than a single update in MCSA. Having more updates required by MCSA for achieving global optima, it is hard to determine which updates are more dependent to each other and should be assigned on single computing node. In summary, the greedy algorithm G1 takes less effort for convergence and is more scalable on distributed computing system.

D. Greedy Algorithm (G2)

In one iteration of G1, the local search space is defined as moving a specific microstate m_{i0} to all other macrostates. It looks for the macrostate S_j that gains maximum $\Delta Q(m_{i0} -> S_j)$. On the other hand, G2 defines a larger local search space, that is all possible matches between microstates in a specific macrostate S_{j0} to all other macrostates. More specifically, it tries to assign any $m_i \in S_{j0}$ to all other macrostates S_j to find maximum $\Delta Q(m_i -> S_j), \forall m_i \in S_{j0}$. Formally, we have

$$S^{new}(j0) = \arg\max_{m_i \in S_{j0}, j=1,2,\ldots,L}(\Delta Q(m_i -> S_j)) \quad (4)$$

The pseudo code is presented in the following.

```
Input: Part = {S(m_i)}_{i=1,2,..., N}
       //S(m_i) denotes the index of macrostate assigned
       //to micro state i initially
Output: Optimal macrostate assignment {S(m_i)}_{i=1,2,..., N}

// meta(.) calculates metastability of current partition
Q0 = meta(Part)
Qdiff = threshold+1

//Check if metastability has converged
WHILE Qdiff > threshold
    FOR i = 1,2, …, L
        Qdiff_i = 0
        Part0 = Part
        FOR all m_k that have S(m_k)=i
            FOR j = 1, 2, …, L
                Part1 = Part
                Part1(S(m_k)) = j
                IF meta(Part1)-meta(Part) > Qdiff_i
                    Qdiff_i = meta(Part1) – meta(Part)
                    Part0 = Part1
                END IF
            END FOR
        END FOR
        Part = Part0
    END FOR
    Qdiff = meta(Part) – Q0
    Q0 = meta(Part)
END WHILE
```

In each macrostate, there are N/L microstates on average. Each microstate takes *O(N)* time to search the macrostate S_j with maximum $\Delta Q(m_a -> S_j)$. Therefore, the time complexity for one iteration is $O(N*N/L)=O(N^2/L)$ on average Since G2 defines a larger local search space than G1, it takes longer time to make single move that leads to greater ΔQ. It is easier to define independent computations in G2 than in G1 or MCSA. Therefore, G2 is supposed to have the best scalability on distributed computing system. However, when the initial partition is poor, G2 will consume a longer time to make enough updates than G1 and MCSA, which decreases it efficiency on single core machines.

E. Design of Distributed Algorithms

WorkQueue [16], [17] is a master/worker framework for distributing computations to many computing nodes. The framework consists of a master process and multiple

worker processes. In general, the computation should be split into many independent tasks. The workers processes execute tasks in parallel and once a worker is finished, it will call home to the master process, which is in charge of summarizing results from all workers and generating tasks for next iteration of execution.

Implementations of MCSA, G1 and G2 under WorkQueue framework follow the same procedure. All three algorithms make a series of updates on initial partition to achieve maximum metastability. The central idea is that we want to group a certain number of updates as a task and execute many tasks in parallel on multiple workers. Once a worker finishes, it will send the master node a list of updates that should be made on current partition. When all workers are finished, the master will compute metastability of the new partition that includes updates computed by all workers and compare it with the metastability of previous iteration partition. According to the difference of metastability, the master determines if the algorithm has converged. If the algorithm has not converged, the master will update the input data files for executing another iteration of tasks on workers.

More specifically, taking G1 as an example, given an initial partition, assume there are X worker nodes, the first worker will update microstates m_1, m_2, …, $m_{N/X}$ to new macrostate for maximum gain of metastablility, the second worker will update micro states $m_{N/X+1}$, …, $m_{2N/X}$ and so on. This is different from sequential version of G1, because every worker makes updates on the initial partition without knowledge of other workers' updates. This may cause a number of non-optimal updates, thereby increasing the total number of updates required for convergence. As discussed in the previous sections, MCSA updates are stochastic, so a large number of non-optimal updates will be generated by parallelization. On the other hand, G2 makes the most deterministic update selected from large local search space. Therefore, distributing computations should have the least effect on convergence of G2.

III. Experiments

Alanine dipeptide is a simple biological system consisting of 22 atoms, two dihedral angles ψ and ϕ. Conformations can be illustrated on the 2D $\psi - \phi$ space.

This system is simple enough to display free energy landscape on the two dimensional conformations space, which is good for a theoretical analysis. On the other hand, it is also interesting to study because it has several free energy wells. In this section, theoretical performance of MCSA, G1 and G2 will be investigated on this system.

To convert theoretical running time of the three algorithms into same unit, we have the following relation: *1 G2 step = N/L G1 step = N MCSA steps*. In this experiment, the efficiency of an algorithm is represented by number of MCSA steps required for convergence.

Fig. 1 left illustrates the structure of alanine dipeptide and Fig. 1 right shows a reference partition of alanine dipeptide into six metastable states, which is defined manually by Chodera *et al.* [12]. This reference partition will be used for validating quality of partitions made by MCSA and greedy algorithms. In the preparation stage, 100 MD trajectories are generated by GROMACS 4.5, using force field Amber96 and implicit solvent. The 2D conformational space is split into 30 by 30 grids, constructing 144 microstates. A transition matrix is computed on the initial microstates partition, using a time lag of 200ps. MCSA, G1 and G2 are all initialized by a random partition with 10 steps of MCSA.

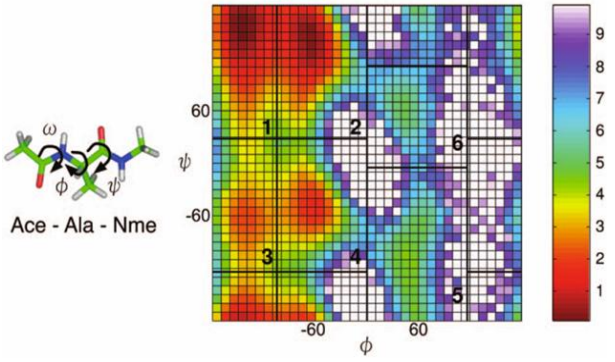


Figure 1. Alanine dipeptide structure and energy plots. Left: Structure of alanine dipeptide. Right: Ramachandran plot for showing free energy landscape of alanine dipeptide on $\psi - \phi$ space. Black lines and number mark boundaries of six metastable states.

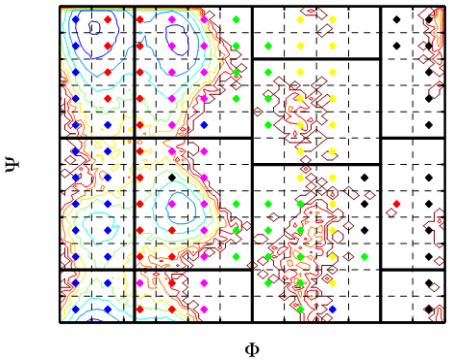


Figure 2. Random partition of metastable states for initializing MCSA, G1 and G2.

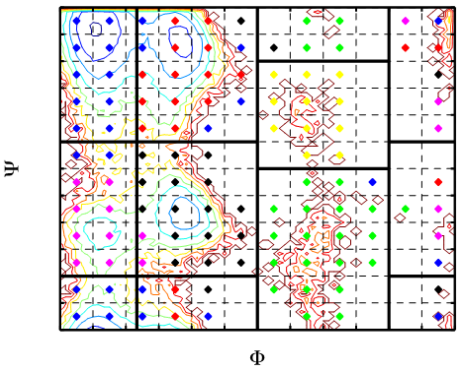


Figure 3. Partition generated by MCSA. Black lines mark boundaries of metastable states made manually. Colored dots mark metastable states discovered by MCSA. One specific color marks a single metastable state.

Fig. 2 shows the initial partition; Fig. 3-Fig. 5 shows partitions after convergence of MCSA, G1 and G2. Different macrostates are marked by distincted colored

dots at the center of microstates. It is observed that all three methods generate reasonably good partitions compared to the reference partition in Fig. 1. To examine the result more qualitatively, Fig. 6 shows the convergence of metastability over theoretical running time. G1 takes the shortest time to converge, which is equivalent to 1,000 MCSA steps. MCSA takes 2,000 steps to converge. The sequential version of G2 takes longest time 8,000 steps to converge.

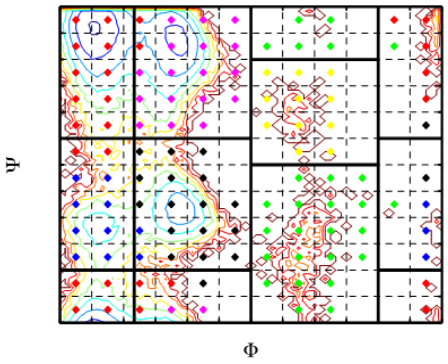


Figure 4. Partition generated by G1

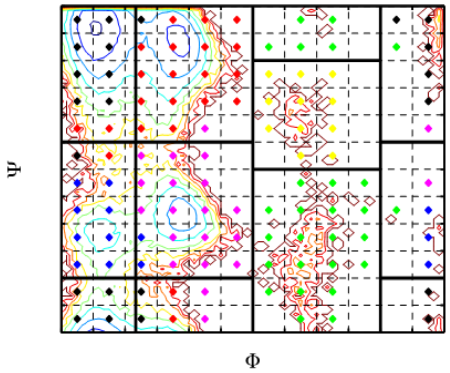


Figure 5. Partition generated by G2

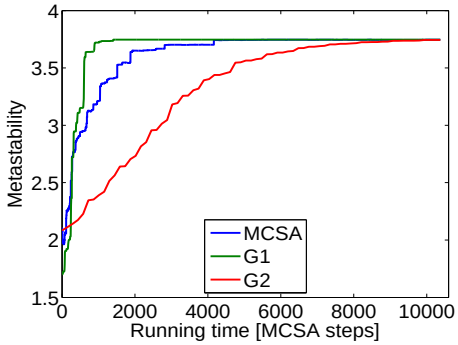


Figure 6. Convergence of three algorithms on metastability over theoretical running time.

Given 144 computing nodes, theoretical performances of distributed MCSA, G1 and G2 methods are shown in Fig. 7. Instead of the total running time of all machines, the x-axis shows the running time on a single machine, which is the real time required on distributed computing systems. The overhead of handling computing nodes is not considered in the theoretical study. The real running time will be discussed in the experiment on the more complex system WW domain. The alanine dipeptide experiment only studies the impact of degrees of distribution on the speed of convergence. From Fig. 7, we can conclude that the distributed G1 is still the most efficient of the three, distributed G2 is the second efficient, while distributed MCSA is the slowest algorithm.

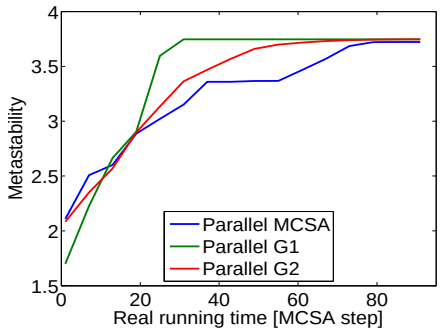


Figure 7. Convergence of distributed MCSA, G1 and G2 algorithms over theoretical running time, given 144 computing nodes available.

Fig. 8, Fig. 9, Fig. 10 compares convergence of sequential and distributed versions of MCSA, G1 and G2 respectively. The x-axis shows the total running time on all computing nodes in unit of MCSA steps. As expected, the distributed G2 requires almost no increase in the total number of updates for convergence compared to sequential G2, which means distributing this algorithm on *X* computing nodes gets *X* times speed up. Distributed G1 increases the total running time required for convergence from 1000 to 4000. MCSA has the worst scalability. It increases the running time from 2000 to 11000.

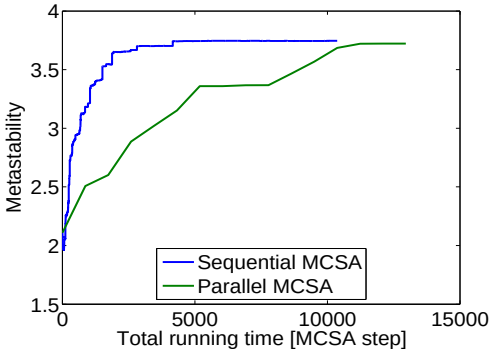


Figure 8. Comparison of the convergence of sequential and distributed MCSA

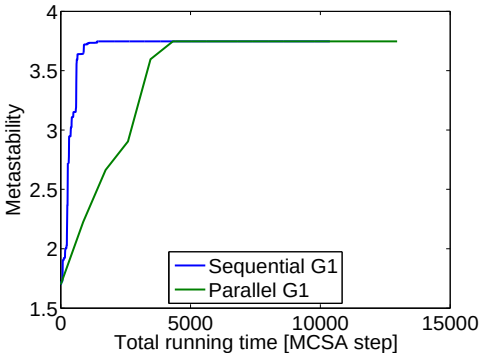


Figure 9. Comparison of the convergence of sequential and distributed G1

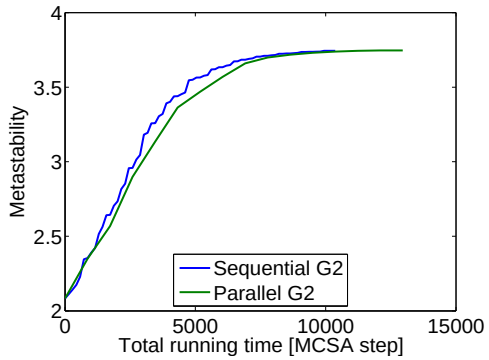


Figure 10. Comparison of the convergence of sequential and distributed G2

WW domain is a medium sized system consisting of 545 atoms, 33 residues. In this experiment, the real running time on a distributed computing system is examined for comparing performance of clustering algorithms. 200 micro-seconds trajectory is generated by GROMACS4.5 using force field Amber96 and explicit solvent. 10,000 microstates are defined by the k-centroid algorithm. MCSA, G1 and G2 are applied to assign the 10,000 microstates into 100 macrostates.

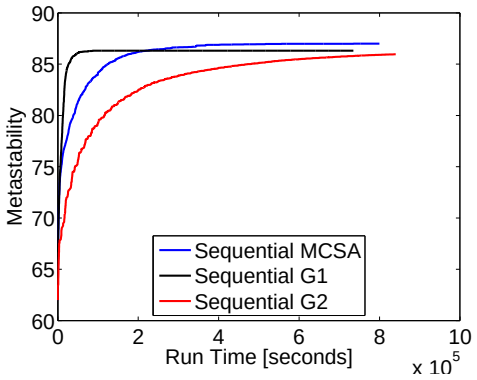


Figure 11. Convergence of metastability of sequential MCSA, G1 and G2 algorithms on WW domain over running time in seconds.

Fig. 11 shows the time it took (in the unit of seconds) for convergence using all algorithms. Sequential G1 took the shortest time, 447 minutes to converge at a metastability of 85. MCSA is 5 times slower than G1, which takes 2,082 minutes for reaching a metastability of 85. G2 is the slowest taking 7,943 minutes for convergence.

TABLE I. REAL RUNNING TIME FOR CLUSTERING WW DOMAIN

	Time (minutes) for reaching metastability >= 8	
Algorithm	Sequential	Distributed
MCSA	2,082	887
G1	447	24
G2	7,943	214

Now, to implement the three algorithms on a distributed computing system, 50 worker nodes are used, each node handles multiple steps of updates at one iteration. Table I lists running time (in minutes) taken for convergence. G1 is the most efficient on single core machine and on the distributed computing system. Using 50 computing nodes, G1 attains 19 times speed up over its sequential version. The sequential G2 algorithm is the slowest of the three, while distributed G2 attains 37 times speed up and distributed MCSA only attains 2 times speed up, making distributed G2 more efficient than distributed MCSA.

Given constant number of computing nodes, the number of steps that are executed per node affects the efficiency. If small numbers of steps are executed per node, making a certain number of updates will consume heavy workload on handling communications between the master and the workers. On the other hand, if a large numbers of steps are executed per node, speed of convergence will be slowed down significantly. Fig. 12 shows the convergence of distributed G1 using 50 computing nodes, with 1, 10 and 100 steps of updates operated at once, respectively. In the case of WW domain, operating 10-100 steps per node leads to reasonably good performance in 17-24 minutes for convergence.

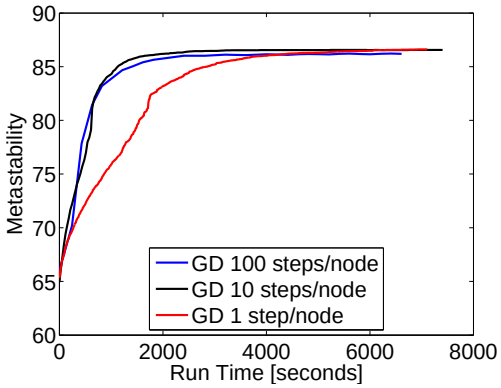


Figure 12. The convergence of distributed G1 using 50 computing nodes at 1, 10 and 100 steps. With varying number of steps operated on one node in one iteration, the plot shows computing time taken for convergence of the distributed G1 algorithm.

The numerical results show that the distributed greedy algorithms are more efficient for automatic discovery of metastable states compared to Monte Carlo simulated annealing. G1 has a smaller local search space than G2. Due to the right balance of computational time per update and quality of each update, G1 is the most efficient in both sequential and distributed versions. G2 searches on a large local search space for better quality updates. Because of the high computational time for a single G2 update, sequential G2 is the slowest of the three. However, the better quality of G2 update leads to close to perfect scalability. Therefore, on distributed computing system, G1 is still the most efficient, G2 is the second efficient and MCSA is the slowest. In practice, these algorithms are applied to assist Markov State Model (MSM) analysis and Weighted Ensemble (WE) sampling. Efficiency is especially important for WE because it runs the clustering algorithm periodically. On the other hand, since both MSM and WE require distributed computing system, it is reasonable to use the same computational resources for efficient clustering to avoid long idle time

of computers. Under such circumstance, greedy algorithms are better choice than MCSA.

ACKNOWLEDGEMENT

The authors thank Dr. E. Darve and Dr. R. Costaouec at Stanford University for useful discussions on Weighted Ensemble methodology. Thank James Sweet for generating MD trajectories for WW domain. Funding has been provided by NSF CCF-1018570, NIH 1R01 GM101935- 01, NIH 7R01 AI039071.

REFERENCES

[1] T. Hansson, C. Oostenbrink, and W. Gunsteren. "Molecular dyanmics simulations," *Curr. Opin. Struc. Biol*, vol. 12, no. 2, pp. 190-196, 2002.

[2] B. Abdul-Wahid, H. Feng, D. Rajan, R. Costaouec, E. Darve, D. Thain, and J. Izaguirre, "AWE-WQ: Fast-forwarding molecular dynamics using the accelerated weighted ensemble," *J. Chem. Info. Model*, 2014.

[3] G. Jayachandran, V. Vishal, and V. S. Pande, "Using massively parallel dynamics simulation and Markovian models to study protein folding: examining the dynamics of the villin headpiece," *J. Chem. Phys.*, vol. 124, no. 16, pp. 164902, 2006.

[4] A. L. Beberg, D. L. Ensign, G. Jayachandran, S. Khaliq, and V. S. Pande, "Folding@home: Lessons from eight years of volunteer distributed computing," *Lect. Notes. Comput. Sc.*, May 2009.

[5] A. Dickson. "Quantifying hub-like behavior in protein folding networks," *J. Chem. Theory. Comput.*, vol. 8, no. 9, pp. 3044-3052, 2011.

[6] V. S. Pande, K. Beauchamp, and G. R. Bowman, "Everything you wanted to know about Markov State Models but were afraid to ask," *Methods*, vol. 52, no. 1, pp. 99-2015, 2010.

[7] E. Darve and R. Ryu, "Computing reaction rates in bio-molecular systems using discrete macrostates," *Innovations in Biomolecular Modeling and Simulations*, vol. 1, pp. 138-206, 2012.

[8] R. Costaouec, H. Feng, J. Izaguirre, and E. Darve, "Analysis of the accelerated weighted ensemble methodology," *Lect. Notes. Math.*, pp. 171-181, 2013.

[9] B. W. Zhang, D. Jasnow, and D. M. Zuckerman, "The weighted ensemble path sampling method is statistically exact for a broad class of stochastic processes and bining procedure," *J. Chem. Phys.*, vol. 132, no. 5, pp. 054107, 2010.

[10] D. Bhatt, B. W. Zhang, and D. M. Zuckerman, "Steady-state simulations using weighted ensemble path sampling," *J. Chem. Phys.*, vol. 133, no. 1, pp. 014110, 2010.

[11] G. A. Huber and S. Kim, "Weighted-ensemble Brownian dynamics simulations for protein association reaction," *Biophys. J.*, vol. 70, no.1, pp. 97-110, 1996.

[12] J. D. Chodera, N. Singhal, V. S. Pande, K. A. Dill, and W. C. Swope, "Automatic discovery of metastable states for the construction of Markov State Models of macromolecular conformational dynamics," *J. Chem. Phys.*, vol. 126, no. 15, 2007.

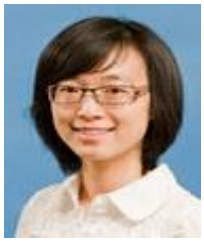
[13] H. Chang, S. Bacallado, V. S. Pande, and G. E. Carlsson, "Persistent topology and metastable state in conformational dynamics," *PloS one*, vol. 8, no. 4, pp. 58699, 2013.

[14] Y. Yao, R. Z. Cui, G. R. Bowman, D. Silva, J. Sun, and X. Huang, "Hierachical nystrom methods from construction Markov state models for conformational dynamics," *J. Chem. Phys.*, vol. 138, no. 17, pp. 174106, 2013.

[15] S. Fortunato, "Community detection in graphs," *Phys. Rep.*, vol. 486, no. 3, pp. 75-174, 2010.

[16] P. Bui, D. Rajan, B. Abdul-Wahid, J. Izaguirre, and D. Thains, "Work Queue+Python: A framework for scalable scientific ensemble applications," in *Proc. Workshop on Python for High Performance and Scientific Computing*, 2011.

[17] M. Albrecht, P. Donnelly, P. Bui, and D. Thain, "Makeflow: A portable abstraction for data intensive computing on clusters, clouds, and grid," in *Proc. 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies*., 2012, pp. 1.



Haoyun Feng is a Ph.D. candidate and Research Assistant in computational biology, Department of Computer Science and Engineering at University of Notre Dame, IN, USA. She received M.S. degree in machine learning from Department of Computer Science, Columbia University and B.S. degree in Mathematics from Beijing University of Aeronautics and Astronautics.

Having background in mathematics and computer science, she focuses on algorithm development and validation for distributed molecular dynamics (MD) simulations during her Ph.D. study. She joined development of Python library AWE-WQ, which implements a distributed weighted ensemble (WE) algorithm for accelerating MD simulations and coauthored papers on development and analysis of WE algorithm.

Geoffrey Siwo is a Research Associate at the Geisel School of Medicine, Dartmouth College where he has initiated the United Genomes Project- an education and open innovation platform for genomic medicine challenges, especially in African populations. His research interests are in developing scalable solutions for enabling rapid discovery in medicine. He is active in open science competitions and led the top teams in developing solutions for estimating human age from DNA and predicting the activity of genes from DNA sequences in the IBM DREAM challenges. He is a co-founder of Helix Nanotechnologies- a company developing a device for recording information inside cells on DNA- and Fit2Cure - a game that crowdsources drug discovery. He holds a B.Sc degree in Biomedical Science from Egerton University, Kenya and a PhD degree in Biology from the University of Notre Dame, IN, USA. He is the recipient of several awards including a TED Global Fellow (2014), Young Investigator Award from Sage Bionetworks (2012), an IBM PhD Scholarship (2011) and an Eck Institute of Global Health Fellowship (2010) at the University of Notre Dame.

Jesus A. Izaguirre is an Associate Professor in Department of Computer Science and Engineering at University of Notre Dame, IN, USA. He received Ph.D. and M.S. degrees from Department of Computer Science at University of Illinois at Urbana-Champaign, IL, USA and B.S. degree from Electronic Systems Engineering, Instituto Tecnologico y de Estudios Superiores de Monterrey, Mexico. His research focuses on development of scalable algorithms and software for computational chemistry and computational biology. This is driven by collaborative applications, particularly in developmental and cellular molecular biology (e.g., chicken limb development, immune system molecular recognition, and anti breast-cancer drug design). His research is funded by NSF.

Douglas Thain is an Associate Professor in the Department of Computer Science and Engineering at the University of Notre Dame. He received the B.S. in Physics from the University of Minnesota - Twin Cities and the M.S. and Ph.D. in Computer Sciences from the University of Wisconsin - Madison, where he contributed to the Condor distributed computing system. At Notre Dame, he works closely with researchers in multiple fields of science and engineering to attack scientific problems using large scale computing. His research team creates and publishes open source software that is used around the world to harness large scale computing systems such as clusters, clouds, and grids.

Badi' Abdul-Wahid is a Ph.D. cadidate in Department of Computer Science and Engineering at University of Notre Dame. He received his M.S. degree in Computer Science from University of Notre Dame, IN, USA and B.S. degree in Computer Science and Chemistry from Central Washington University, USA. His research focuses on development of scalable distributed algorithms for accelerating molecular dynamics simulations. He released Python library AWE-WQ and published papers on journals and conferences on this topic.