Computing Reaction Rates Using Macro-states

Badi' Abdul-Wahid[1] Ronan Costaouec[2]
Eric Darve[2] Michelle Feng[1] Jesús Izaguirre[1]

[1]Notre-Dame University

[2]Stanford University

Tuesday July 3rd 2012 — 14:30–15:00

Computing reaction rates

- Consider a stochastic system such as a protein or bio-molecule. We assume a Brownian dynamics model for simplicity although this is not required:

$$dx(t) = \left(\nabla \mathbf{D} - \beta\, \mathbf{D}(x)\, \nabla U(x)\right) dt + \mathbf{R}(x)\, dW(t)$$
$$\mathbf{R}(x)\, \mathbf{R}(x)^T = 2\, \mathbf{D}(x)$$

- Reactant state: A, product state: B. These are open simply connected regions.
- Calculate: $\mathbf{rate}_{A \to B}$ **and** $\mathbf{rate}_{B \to A}$**.**
- A direct simulation by counting trajectories going from A to B and back can be extremely slow or intractable.

Definition of rates

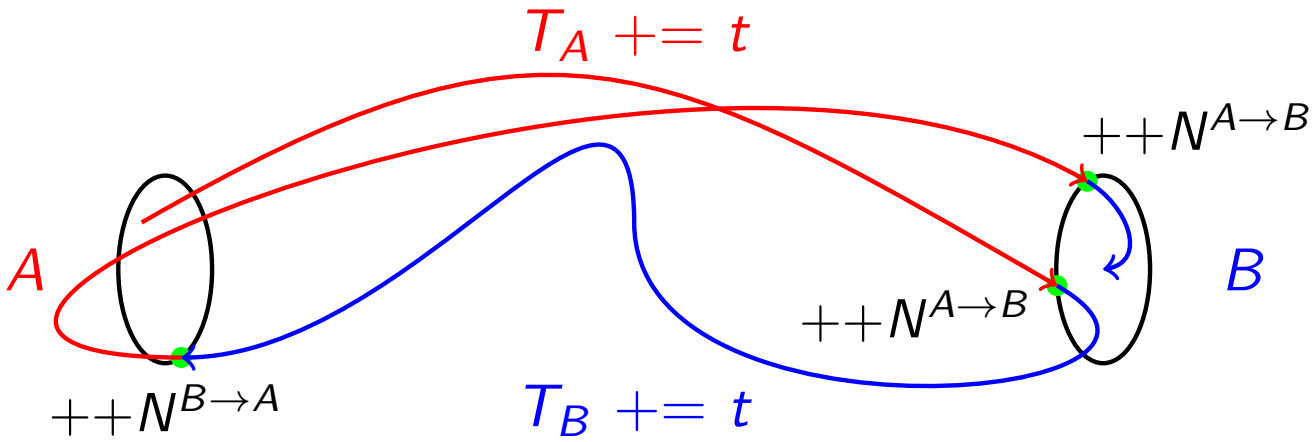
$$\frac{\partial \rho(x,t)}{\partial t} = -\frac{\partial}{\partial x}\left(\left[\nabla \mathbf{D} - \beta\, \mathbf{D}(x)\, \nabla U(x)\right] \rho(x,t)\right) + \frac{\partial^2}{\partial x^2}\left(\mathbf{D}(x)\, \rho(x,t)\right)$$

3 key rates can be defined based on the eigenvalues of the forward Fokker-Planck equation:

1. λ_2: second eigenvalue of FP. Describes the rate of convergence to equilibrium.
2. $k_{AB} = \rho_B \lambda_2$: reaction rate from A to B
3. $k_{BA} = \rho_A \lambda_2$: reaction rate from B to A

Reactive trajectories

Consider a very long trajectory of length T. Make note when a trajectory first enters A or first enters B.



T_A (resp. T_B): time spent between first entering A and first entering B.

$N^{A\to B}(T)$: number of transitions from A to B during T.

Rate definition using reactive trajectories

$$\lambda_2 = \lim_{T\to\infty} \frac{N^{A\to B}(T)}{T} = \lim_{T\to\infty} \frac{N^{B\to A}(T)}{T}$$

$$k_{AB} = \rho_B\ \lambda_2 = \lim_{T\to\infty} \frac{N^{A\to B}(T)}{T_A}$$

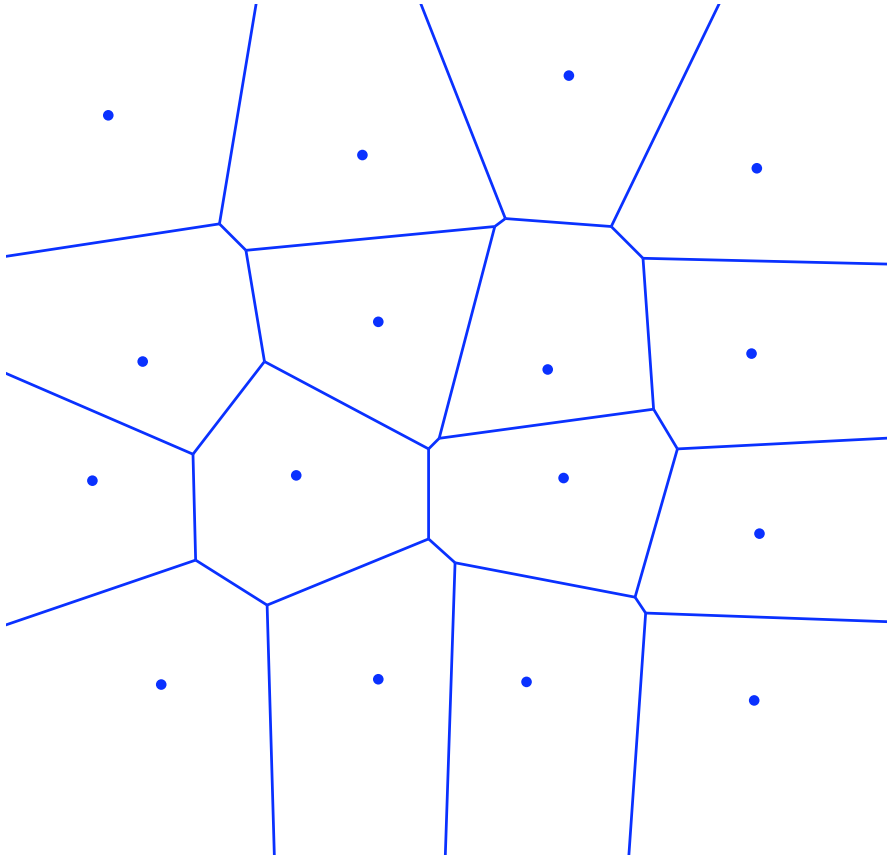
$$k_{BA} = \rho_A\ \lambda_2 = \lim_{T\to\infty} \frac{N^{B\to A}(T)}{T_B}$$

Direct approaches for computing rates

1. Solution of Fokker-Planck equation (e-value): PDE has too many dimensions.
2. Long trajectory and counting transitions (e.g., $N^{A\to B}(T)$): impractical since typically the reaction rates are small and the mean passage times very long.
3. Proposed solution: introduce macro-states = large cells forming a partition of space.
4. Advantages:
    1. Eigenvalue λ_2 can be computed because of the **small number of macro-states**.
    2. **Local sampling** is sufficient to reconstruct all kinetic information.
    3. **Exact rates** can be reconstructed under mild assumptions.

Definition of coarse states

Example construction based on Voronoi cells.



Four methods

1. Markov state models [MSM]

   Singhal et al., *J. Chem. Phys.*, 2004.; **Swope** et al., *Theory. J. Phys. Chem. B*, 2004; **Swope** et al., *J. Phys. Chem. B*, 2004; **Chodera et al.**, *Multiscale Model. Simul.*, 2006; **Chodera** et al., *J. Chem. Phys.*, 2007.

2. Tilted dynamics [TD]

   Vanden-Eijnden et al., *J. Chem. Phys.*, 2009.

3. Non-equilibrium umbrella sampling [NEUS]

   Warmflash et al., *J. Chem. Phys.*, 2007; **Dickson** et al. *J. Chem. Phys.*, 2009.

4. Accelerated weighted ensemble [AWE]

   Bhatt et al., *J. Chem. Phys.*, 2010; **Darve** et al., "Computing reaction rates in bio-molecular systems using discrete macro-states," in *Innovations in Biomolecular Modeling and Simulations*, ed: T. Schlick, 2012; **Abdul-Wahid** et al., "An Accelerated Weighted Ensemble for Protein Folding using Heterogeneous Distributed Computing," SC12, submitted.

All methods enhance sampling but different approximations are made:

- MSM: Markov approximation; reduced computational cost; accuracy difficult to control.
- TD: unbiased; increased computational cost; convergence is difficult to control.
- NEUS: unbiased; guaranteed convergence; complex data-structure.
- AWE: unbiased; guaranteed convergence; simple data-structure.

MSM: eigenvalues and rates

Define:

$$P_{ij}(\tau) = \mathbf{P}(\text{walker in cell } i \text{ at time } t_0 = 0 \text{ is in cell } j \text{ at } t_1 = \tau)$$

- First e-vector with e-value 1 is the equilibrium density, which is related to the free energy of states.
- Second e-vector with e-value μ_2 close to 1 provides the rate:

$$\text{rate} = \frac{-\ln(\mu_2)}{\tau}$$

- The entries P_{ij} can be accurately computed, independently of the rate (e.g., slow rate).
- However the accuracy of μ_2 is sensitive to τ: the memory or non-Markovity effect.

Memory

- At small lag time τ, we have a significant bias: the rate is over-estimated. Markov assumption is not satisfied: $P(k\tau) \neq P(\tau)^k$.
- This can be difficult to detect and fix, as one is required to run simulations with different τs and monitor convergence.
- Memory effect disappears under two independent assumptions:
  - Long lag-time τ
  - Choice of optimal cells based on committor function

Markov state model error

- Two results were derived. Eigenvector expansion of the Fokker-Planck equation:

$$\rho(x, t|x_0, 0) = \sum_k \psi_k(x_0)\, \rho_k(x)\, e^{-\lambda_k t}, \quad \mu_2 \approx e^{-\lambda_2 \tau}$$

- **Assumption 1: long lag time τ:**

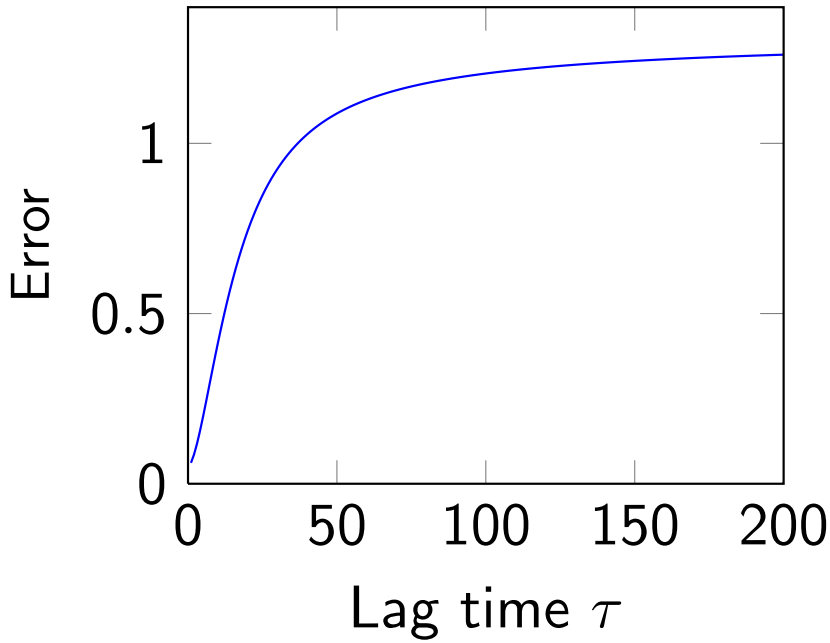
$$\text{Error in } \mu_2 \; = O(e^{-\lambda_3 \tau})$$

- **Or** it is possible to get accurate results even with a short lag-time. **Assumption 2: the cells are defined by:**

$$C_i = \{x \,|\, i\varepsilon \leq \psi_2(x) < (i+1)\varepsilon\}, \quad i \text{ integer}$$

Dependence of statistical error on τ

- The statistical error when computing μ_2 depends on τ.
- Accounting for the statistical errors in the entries $P_{ij}(\tau)$, we can calculate the variance of μ_2.

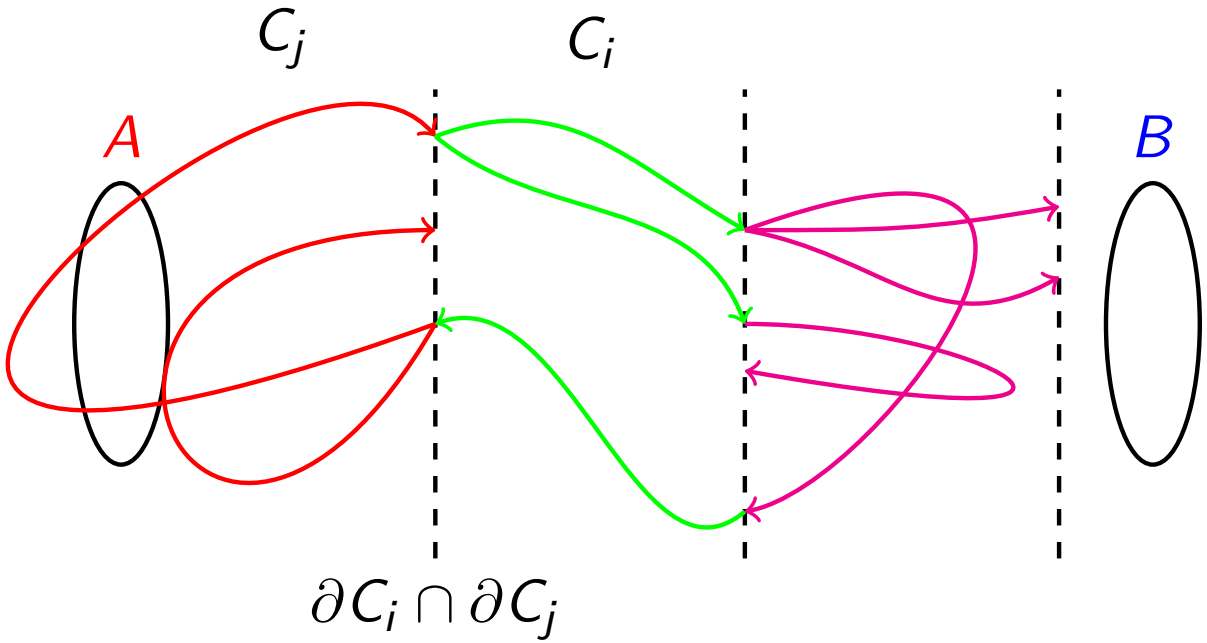


Long lag time increase of error

- At "long" lag times statistical errors increase rapidly.
- Plateau corresponds to $\tau \gg \lambda_3^{-1}$ = "direct" rate calculation = sampling trajectories going from one basin to the other. Benefits of MSM are lost.
- We need to find a trade-off between two difficulties. 1) At short lag times, we have a systematic error; 2) At long lag times, statistical errors are large.
- Accuracy of rate depends strongly on choice of cells.

Tilted dynamics

- Remove Markov approximation. Unbiased estimate. Reactive trajectories viewpoint.
- This requires (as for NEUS and AWE) a "global" convergence of statistics.
- B is a cemetery state: absorbing boundary condition.
- This means that the calculation is out of equilibrium with a "flux" of particles from A to B.



Tilted dynamics algorithm

Iterate until convergence:

1. Local sampling in each state; when a trajectory attempts to exit a state, record event and randomly choose a point on the boundary that is "re-entering." This is done using a database of interface points.
2. Based on statistics of exit points, calculate the weight of each macro-state and the non-equilibrium flux between states. This data is required for step 1 to compute the probability of choosing a state interface for re-entry.

$$P_{\partial C_i \cap \partial C_j} = \frac{\rho_j \ \nu_{ji}}{\sum_{k \neq i} \rho_k \ \nu_{ki}}, \qquad \nu_{ki}\text{: probability to exit } k \text{ through } i$$

Convergence

Convergence is difficult to monitor. We need to converge simultaneously:

- ν_{ji}: obtained through local sampling; statistical errors. Required input: $P_{\partial C_i \cap \partial C_j}$.
- $P_{\partial C_i \cap \partial C_j}$: requires global convergence of fluxes. Required input: ν_{ji}.

As the global convergence of flux estimates converge ($P_{\partial C_i \cap \partial C_j}$), statistical error in ν_{ji} needs to be reduced accordingly.

Non-equilibrium umbrella sampling

- Scheme is similar to TD.
- Non-equilibrium simulation: B is a cemetery state.
- Additional ingredient: when a trajectory exits a state ($j \to i$), a fraction of its weight is transferred to state i. This ensures a convergence of the state weights.
- Convergence is in principle easier to monitor.
- Global convergence still required: convergence of state weights affects the local sampling, which in turn determines the steady-state state weights.

Accelerated weighted ensemble

- As in TD and NEUS, we run many walkers. This time, walkers are allowed to leave macro-states.
- In order to maintain the population of walkers in a macro-state, a resampling algorithm is used: after a time τ, walkers are split if a state is depleted or merged if a state is over-crowded.

Re-sampling algorithm

- Goal: produce statistically unbiased results.
- Allow in-(de-)creasing the number of walkers in each macro-state to maintain a target number.
- Achieved by assigning a statistical weight to each walker.
- The sum of the weights of walkers in a state is the probability of the state.
- As a result of the specific algorithm we use, all walkers in a given macro-state have identical probabilistic weight after resampling algorithm.

```
# list0: initial list
# list1: final list of walkers after resampling
x = list0.pop()

while True: # while loop exits using a break.
    Wx = weights[x]
    if (Wx >= tw or len(list0) == 0):
        # walker splitting
        r = max(1, int(floor( Wx / tw )))
        r = min(r,ntargetwalkers-nwalkerlist1)
        nwalkerlist1 += r
        for item in repeat(x,r):
        # insert r copies of walkers in list1
            list1.append(item)
            newweights.append(tw)
        if nwalkerlist1 < ntargetwalkers and
            Wx - r*tw > 0.0:
            list0.append(x)
            weights[x] = Wx - r*tw
        ...
```

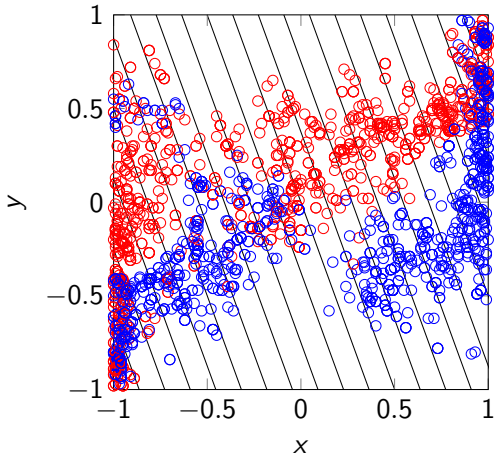
```
while True: # while loop exits using a break.
    Wx = weights[x]
    if (Wx >= tw or len(list0) == 0):

        ...

        if len(list0)>0:
            x = list0.pop()
        else:
            break
    else:
        y = list0.pop()
        Wy = weights[y]
        # walker merging
        Wxy = Wx + Wy
        p = random.random()
        # randomly select a walker
        if p < Wy / Wxy:
            x = y
        weights[x] = Wxy
```

AWE with colored walkers

- Method to calculate both the forward and backward rates.
- Red walkers: walkers who last entered A.
- Blue walkers: walkers who last entered B.
- Rate: fraction of walkers changing color (red to blue for forward and blue to red for backward) per unit time.
- Method is always unbiased. (Converges to the committor function.)
- Initially accelerated by updating the macro-state weights using the transition matrix P (reduces initial bias due to initial conditions).

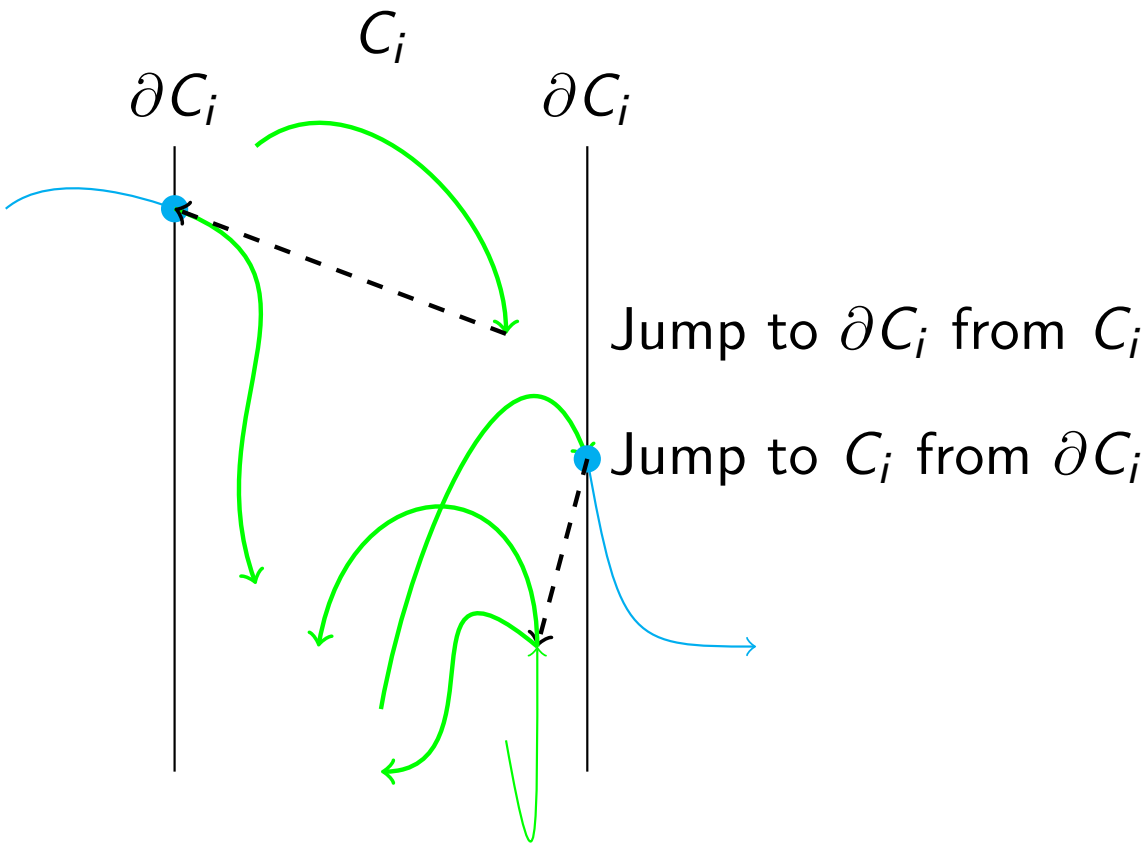


Relation to previous methods: AWE/MSM

- As in MSM, the rate can be obtained from the eigenvalue of the transition matrix P.
- However because a non-equilibrium calculation is carried out (with multi-coloring), the calculation is exact and not dependent on the Markov approximation.
- Note: if a single iteration is carried out (no-resampling), the two methods are identical.
- Distribution of walkers in states accounts for the non-equilibrium flux of walkers from A to B.

Resampling with $\tau = \Delta t$

Special case of the general algorithm with resampling every step.



AWE/TD-NEUS

- Take $\tau = \Delta t =$ one time step. Then two methods become comparable.
- TD always chooses a new walker on the boundary. Whereas for AWE:
  1. If walker leaves, a point inside C_i is randomly duplicated
  2. Whenever a walker enters, a walker is randomly deleted
- TD: random jumps from ∂C_i to ∂C_i.
- AWE: random jumps from ∂C_i to C_i (exit $==$ (1)) and from C_i to ∂C_i (entry $==$ (2)).
- **AWE does not require maintaining a database of re-entry points as in TD/NEUS.**

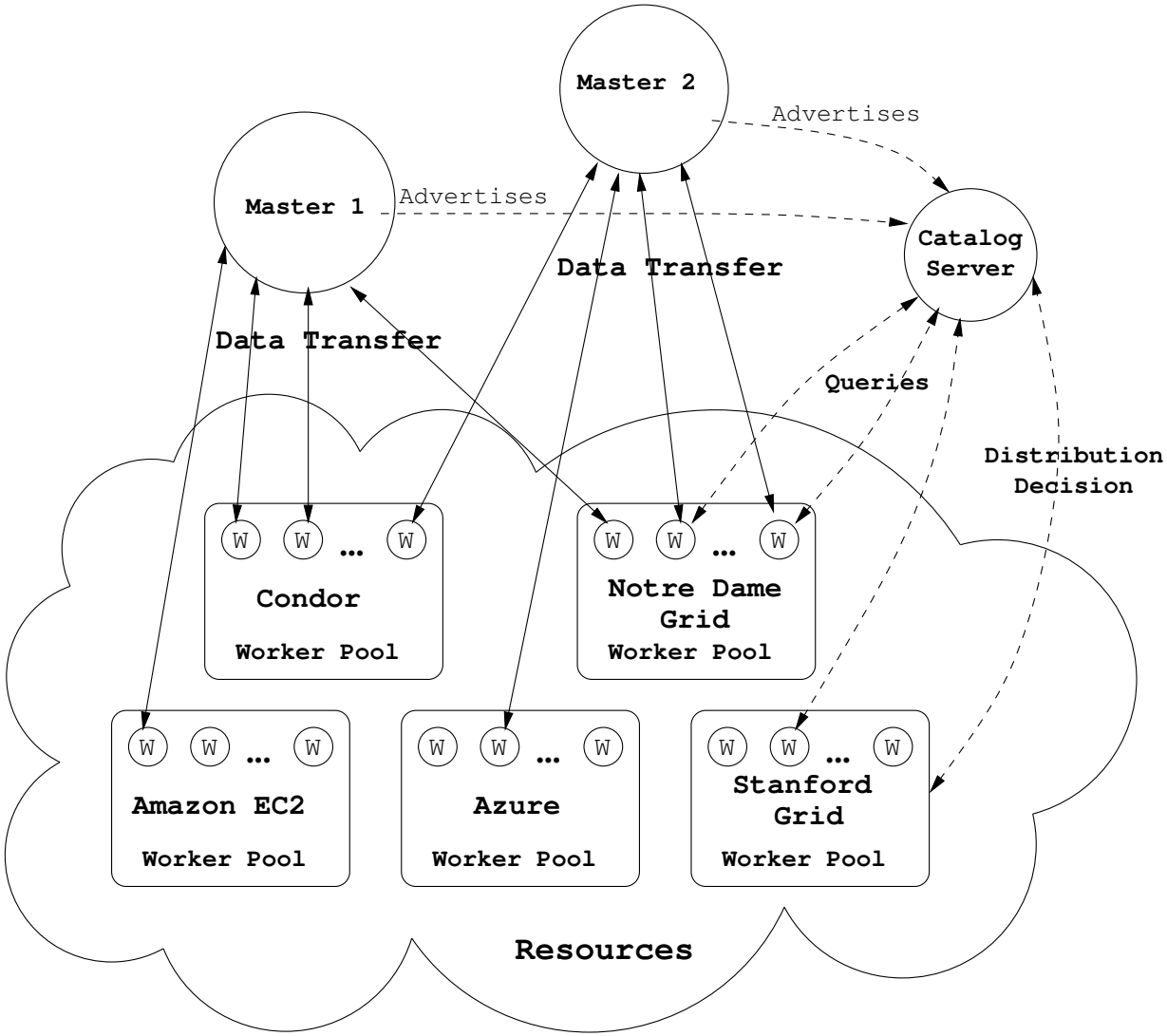
AWE with arbitrary resample time τ

- A choice $\tau \gg \Delta t$ is preferable because it leads to additional scalability on parallel machines.
- Iterate the following steps:
  1. Resample.
  2. Integrate forward each walker by a time τ.
- Summary:
  - Unbiased
  - Simple convergence monitoring
  - Scalable
  - Simple implementation. Key routine is the weight resampling.

Parallel implementation

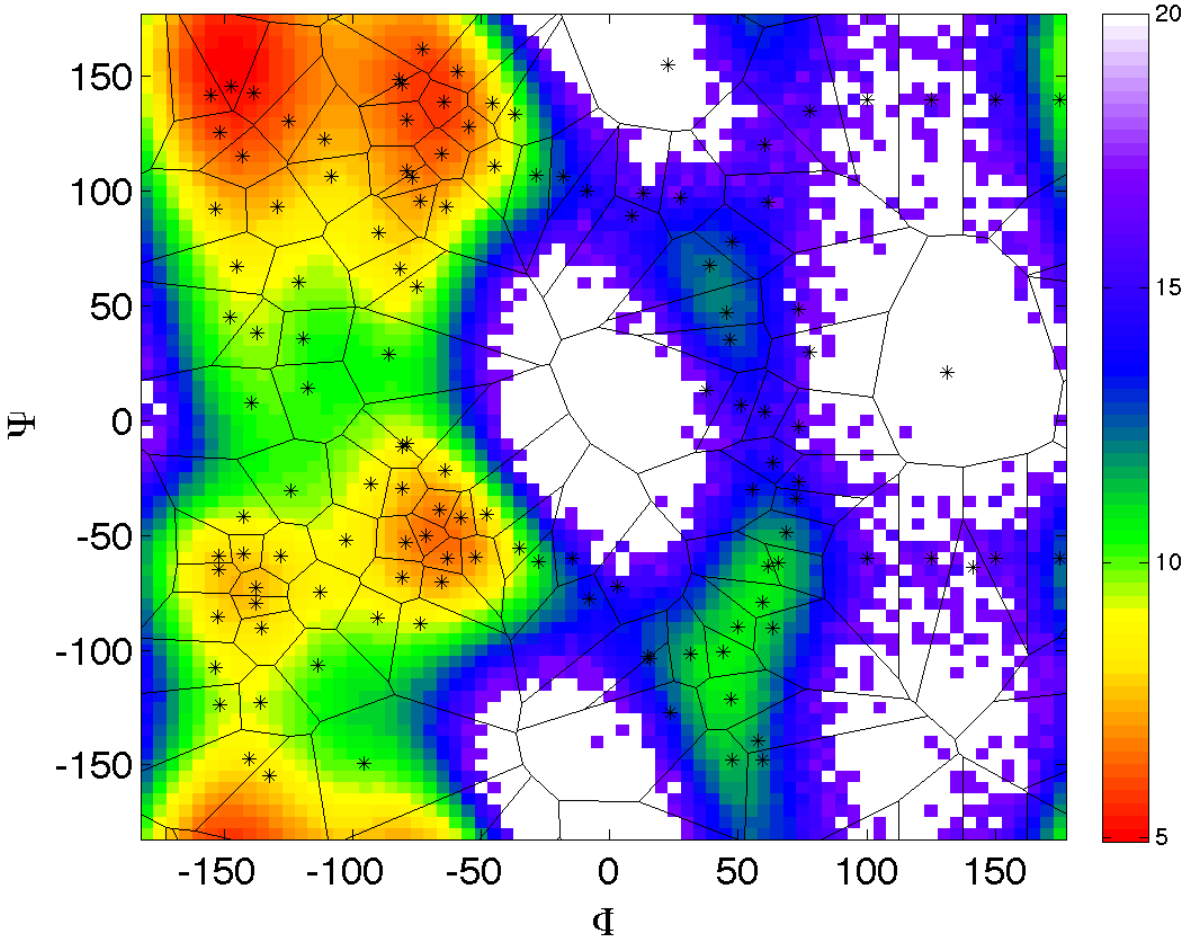
- The method was implemented using Work Queue.
- WQ is a framework to run parallel codes on all possible parallel platforms: cloud (Amazon), grid (Condor), multi-core cluster, GPU cluster.
- Requires a molecular dynamics code. The code runs unmodified. **BYOMDC.**
- Calculation is driven by a python script. **BYOA.**
- Interaction with MD code is only through input and output files.

Work Queue


Master 2
Advertises
Master 1
Advertises
Catalog Server
Data Transfer
Data Transfer
Queries
Distribution Decision
W W ... W
Condor
Worker Pool
W W ... W
Notre Dame Grid
Worker Pool
W W ... W
Amazon EC2
Worker Pool
W W ... W
Azure
Worker Pool
W W ... W
Stanford Grid
Worker Pool
Resources

Numerical results

Alanine dipeptide; Amber 96, Generalized Born, $T = 300$



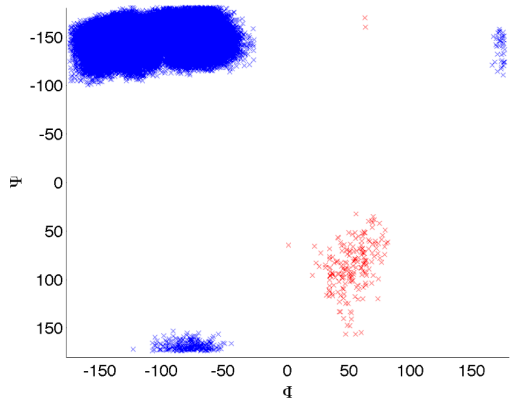
Eigenvectors for $A \to B$ and $B \to A$

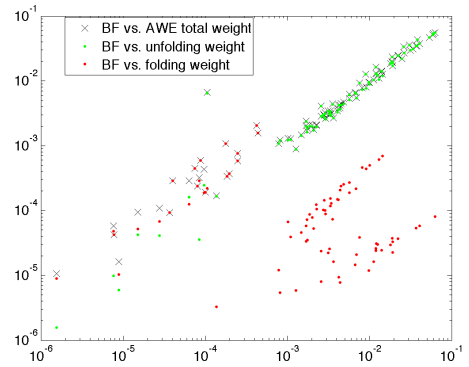
Top left: definition of A and B

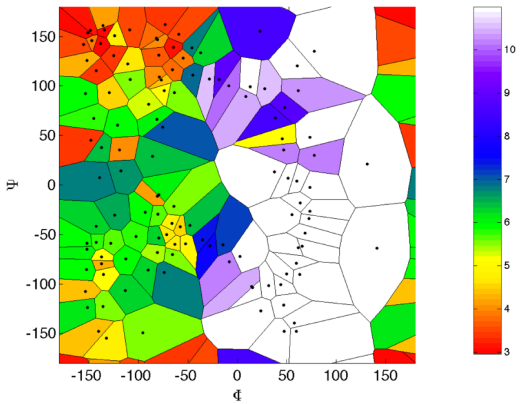
Top right: state weights; x-axis: brute force; y-axis: AWE, forward/backward evec

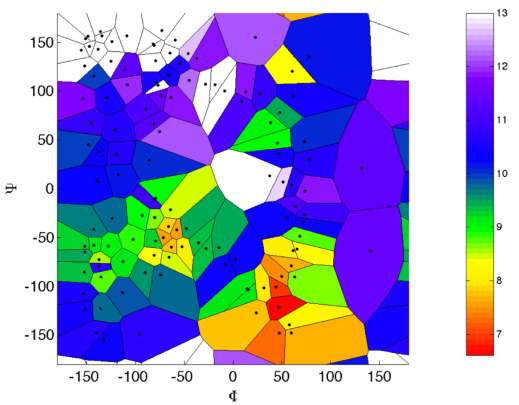
Bottom left: forward evec

Bottom right: backward evec

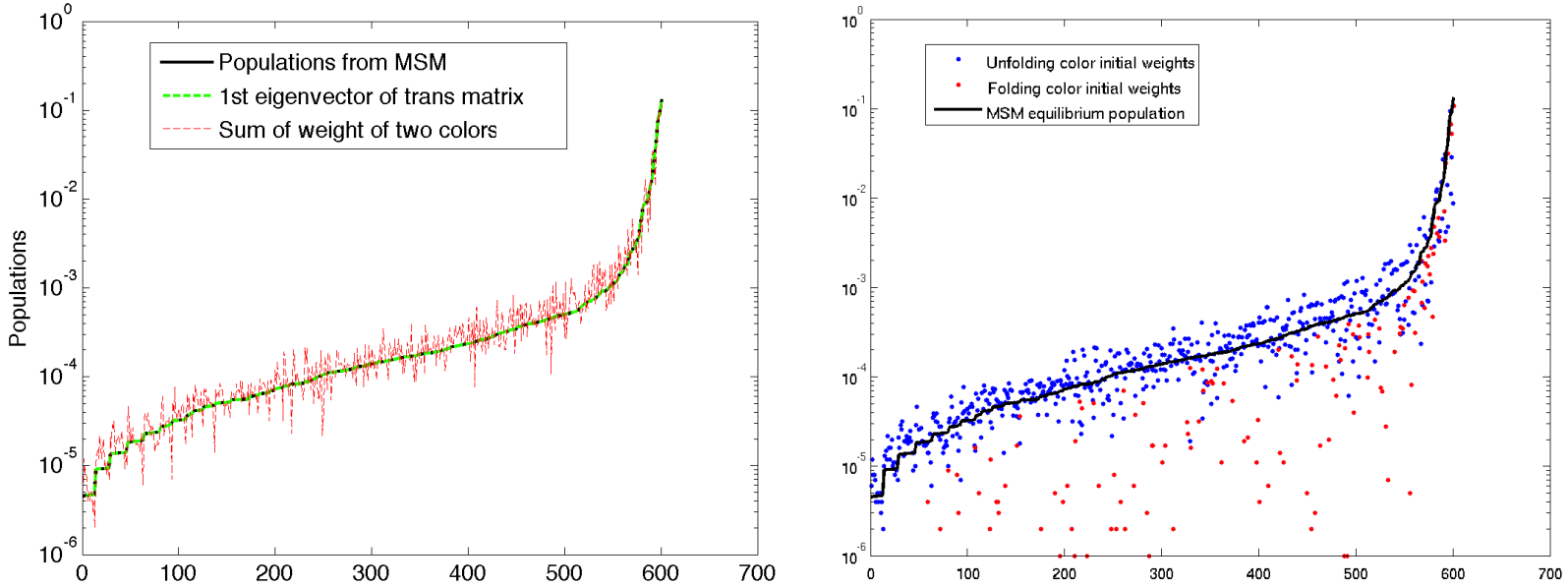








Macro-state weight convergence

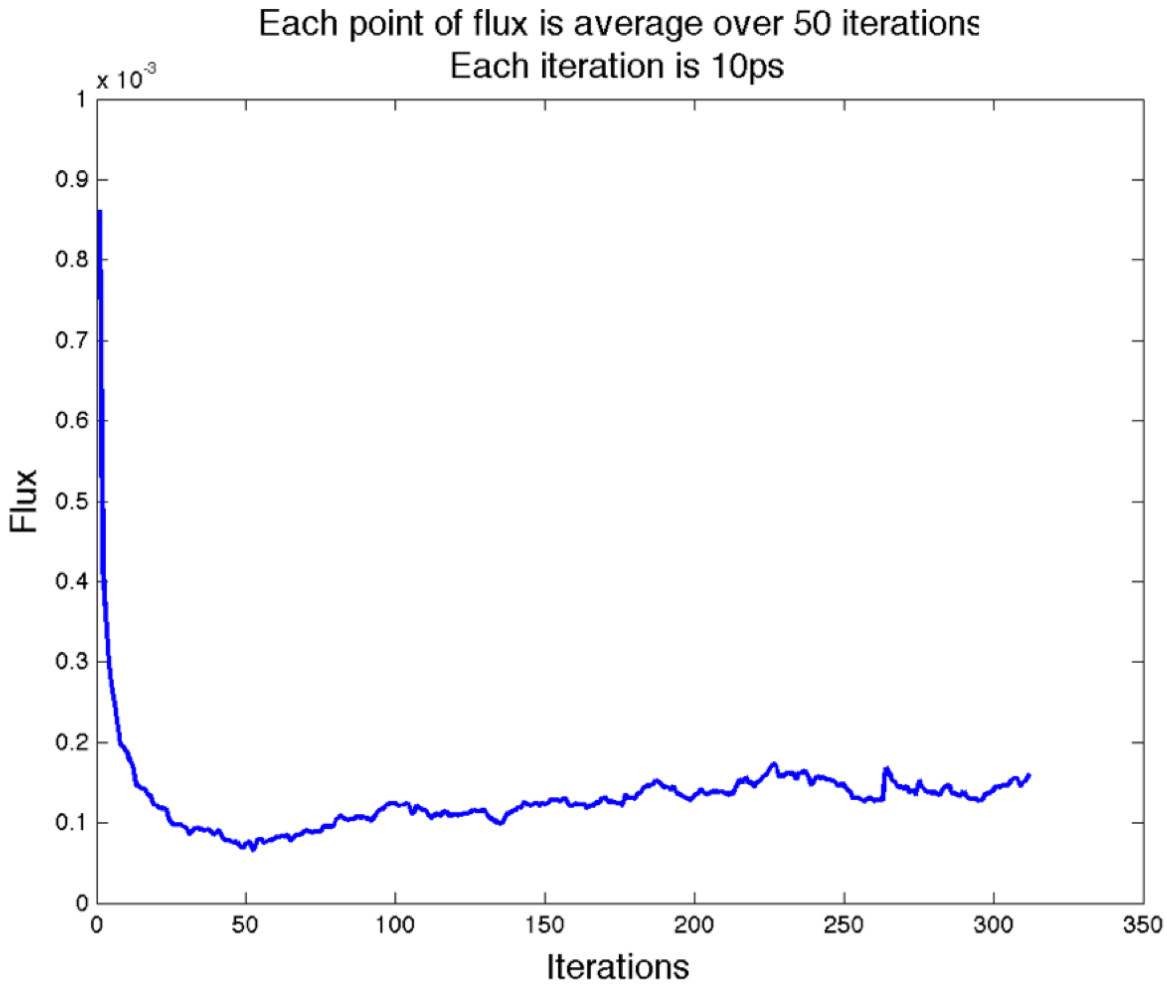


Left: state weights vs index for the MSM simulation, the 1st e-vector from MSM, and the state weights in AWE.

Right: state weights vs index, and forward and backward e-vectors for AWE.

Convergence of the flux

Mean = 0.013/ns, std = 0.006, brute force flux = 0.013/ns



Conclusion

Discussion of 4 methods based on macro-states to calculate slow reaction rates in high-dimensional stochastic systems.

Method	Markov Aspt	Global Cvg	Iterative Cvg	Cplx data structure
MSM	**Yes**	**No**	**No**	**No**
TD	**No**	**Yes**	**Yes**	**Yes**
NEUS	**No**	**Yes**	**No**	**Yes**
AWE	**No**	**Yes**	**No**	**No**

MSM: Markov state model; TD: tilted dynamics; NEUS: non-equilibrium umbrella sampling; AWE: accelerated weighted ensemble; Aspt = assumption; Cvg = convergence; Cplx = complex; Iterative Cvg: requires the simultaneous convergence of flux data and statistical errors.