Basic Monte Carlo (chapter 3)

Algorithm

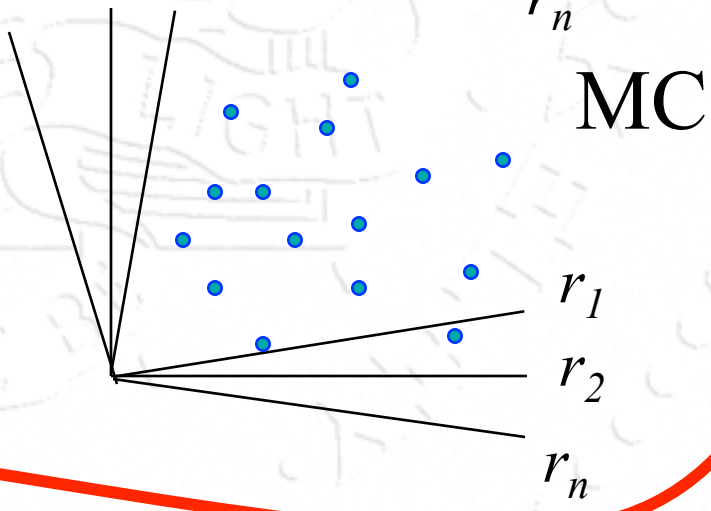
Detailed Balance

Other points

Molecular Simulations

- Molecular dynamics: solve equations of motion

- Monte Carlo: importance sampling



Statistical Thermodynamics

Partition function

$$Q_{NVT} = \frac{1}{\Lambda^{3N} N!} \int \mathrm{d}\mathrm{r}^N \exp\left[-\beta U\left(\mathrm{r}^N\right)\right]$$

Ensemble average

$$\left\langle A \right\rangle_{NVT} = \frac{1}{Q_{NVT}} \frac{1}{\Lambda^{3N} N!} \int \mathrm{d}\mathrm{r}^N A\left(\mathrm{r}^N\right) \exp\left[-\beta U\left(\mathrm{r}^N\right)\right]$$

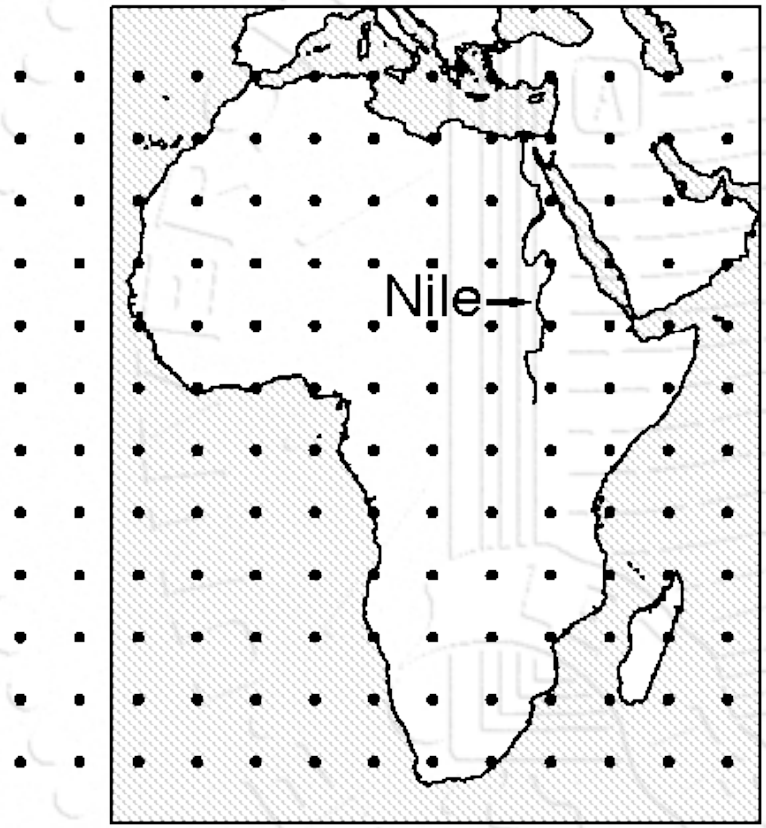
Probability to find a particular configuration

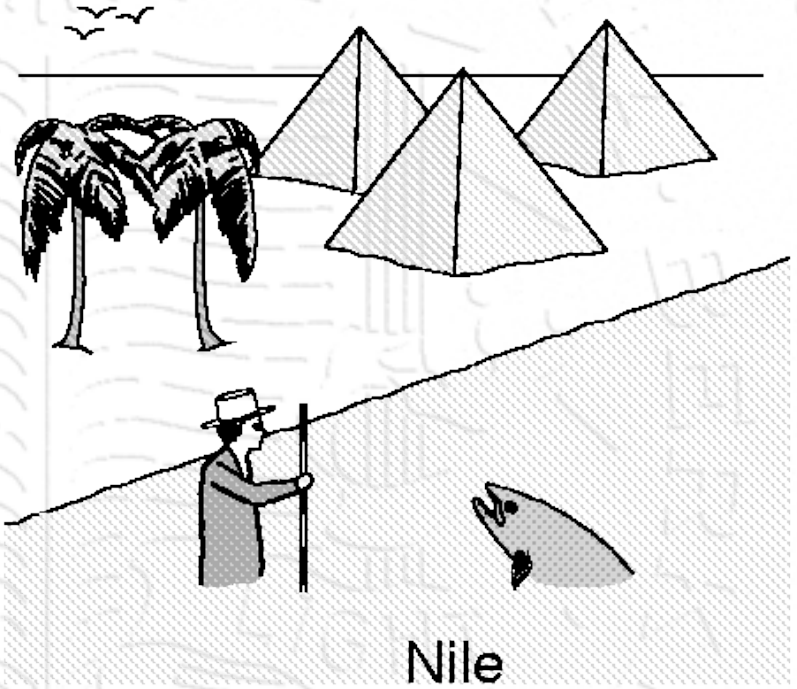
$$\mathrm{N}\left(\mathrm{r}^N\right) = \frac{1}{Q_{NVT}} \frac{1}{\Lambda^{3N} N!} \int \mathrm{d}\mathrm{r}'^N \delta\left(\mathrm{r}'^N - \mathrm{r}^N\right) \exp\left[-\beta U\left(\mathrm{r}'^N\right)\right] \propto \exp\left[-\beta U\left(\mathrm{r}^N\right)\right]$$

Free energy

$$\beta F = -\ln\left(Q_{NVT}\right)$$

Monte Carlo simulation





Ensemble average

$$\langle A \rangle_{NVT} = \frac{1}{Q_{NVT}} \frac{1}{\Lambda^{3N} N!} \int \mathrm{d}\mathbf{r}^N A(\mathbf{r}^N) \exp\left[-\beta U(\mathbf{r}^N)\right]$$

$$= \int \mathrm{d}\mathbf{r}^N A(\mathbf{r}^N) P(\mathbf{r}^N) = \frac{\int \mathrm{d}\mathbf{r}^N A(\mathbf{r}^N) P(\mathbf{r}^N)}{\int \mathrm{d}\mathbf{r}^N P(\mathbf{r}^N)}$$

$$P(\mathbf{r}^N) = \frac{\exp\left[-\beta U(\mathbf{r}^N)\right]}{Q_{NVT} \Lambda^{3N} N!}$$

$$= \frac{\int \mathrm{d}\mathbf{r}^N A(\mathbf{r}^N) C \exp\left[-\beta U(\mathbf{r}^N)\right]}{\int \mathrm{d}\mathbf{r}^N C \exp\left[-\beta U(\mathbf{r}^N)\right]} = \frac{\int \mathrm{d}\mathbf{r}^N A(\mathbf{r}^N) \exp\left[-\beta U(\mathbf{r}^N)\right]}{\int \mathrm{d}\mathbf{r}^N \exp\left[-\beta U(\mathbf{r}^N)\right]}$$

Generate configuration using MC:

$$\left\{\mathbf{r}_1^N, \mathbf{r}_2^N, \mathbf{r}_3^N, \mathbf{r}_4^N \cdots, \mathbf{r}_M^N\right\}$$

$$\bar{A} = \frac{1}{M} \sum_{i=1}^{M} A(\mathbf{r}_i^N) = \frac{\int \mathrm{d}\mathbf{r}^N A(\mathbf{r}^N) P^{MC}(\mathbf{r}^N)}{\int \mathrm{d}\mathbf{r}^N P^{MC}(\mathbf{r}^N)}$$

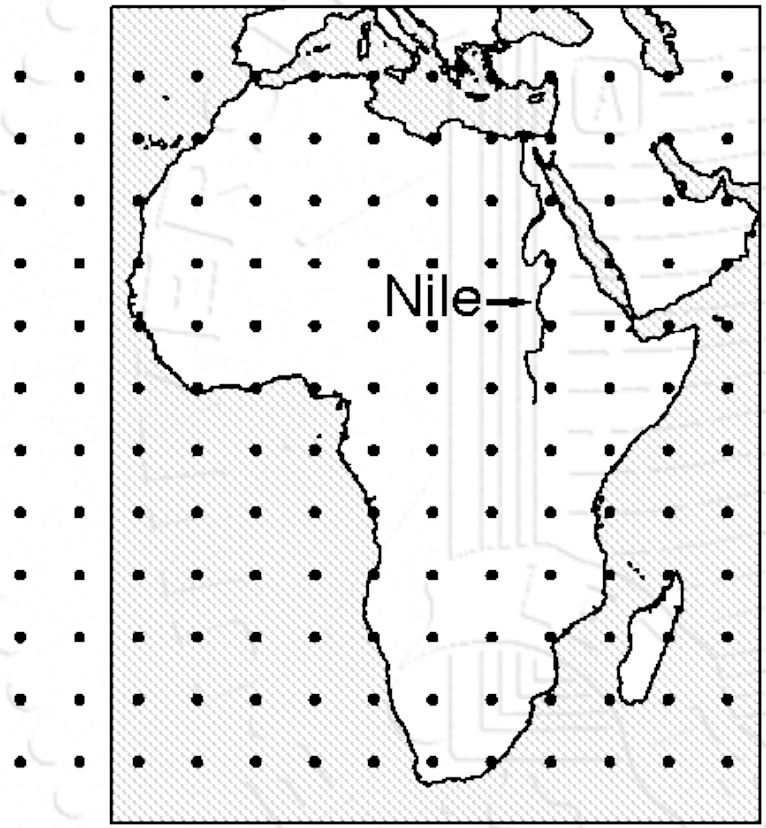
$$= \frac{\int \mathrm{d}\mathbf{r}^N A(\mathbf{r}^N) C^{MC} \exp\left[-\beta U(\mathbf{r}^N)\right]}{\int \mathrm{d}\mathbf{r}^N C^{MC} \exp\left[-\beta U(\mathbf{r}^N)\right]}$$

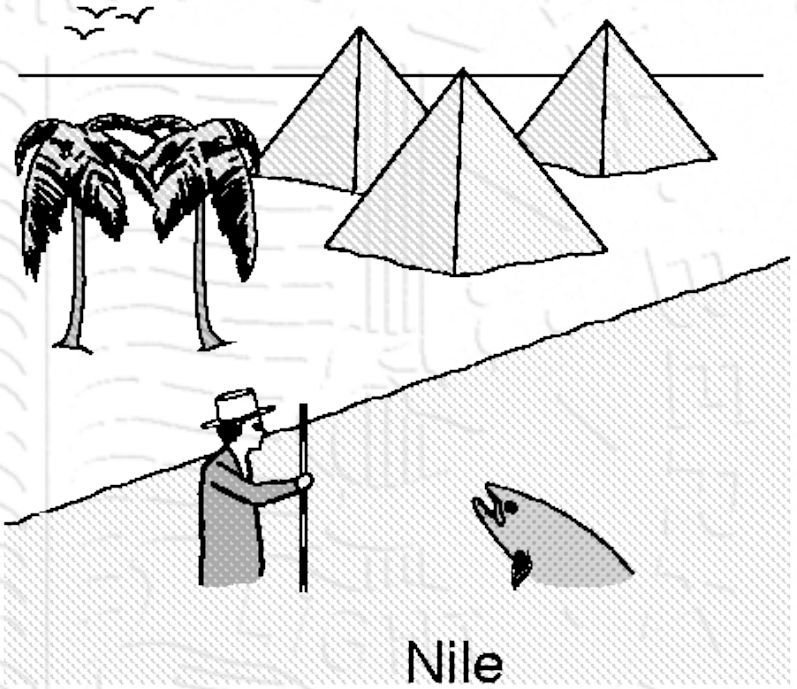
with

$$P^{MC}(\mathbf{r}^N) = C^{MC} \exp\left[-\beta U(\mathbf{r}^N)\right]$$

$$= \frac{\int \mathrm{d}\mathbf{r}^N A(\mathbf{r}^N) \exp\left[-\beta U(\mathbf{r}^N)\right]}{\int \mathrm{d}\mathbf{r}^N \exp\left[-\beta U(\mathbf{r}^N)\right]}$$

Monte Carlo simulation





Algorithm 1 (Basic Metropolis Algorithm)

```
   PROGRAM mc                                  basic Metropolis algorithm

   do icycl=1,ncycl                            perform ncycl MC cycles
     call mcmove                               displace a particle
     if (mod(icycl,nsamp).eq.0)
  +    call sample                            sample averages
   enddo
   end
```

Comments to this algorithm:

1. *Subroutine* `mcmove` *attempts to displace a randomly selected particle (see Algorithm 2).*
2. *Subroutine* `sample` *samples quantities every* `nsampth` *cycle.*

Algorithm 2 (Attempt to Displace a Particle)

```
   SUBROUTINE mcmove                    attempts to displace a particle

   o=int(ranf()*npart)+1                select a particle at random
   call ener(x(o),eno)                  energy old configuration
   xn=x(o)+(ranf()-0.5)*delx            give particle random displacement
   call ener(xn,enn)                    energy new configuration
   if (ranf().lt.exp(-beta              acceptance rule (3.2.1)
  +    *(enn-eno)) x(o)=xn              accepted: replace x(o) by xn
   return
   end
```

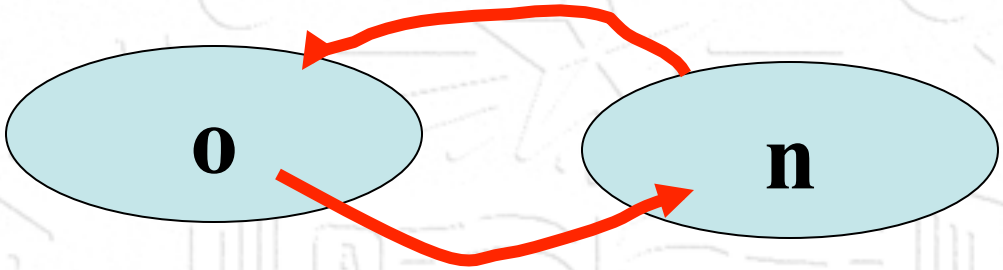
Comments to this algorithm:

1. *Subroutine* `ener` *calculates the energy of a particle at the given position.*
2. *Note that, if a configuration is rejected, the old configuration is retained.*
3. *The* `ranf()` *is a random number uniform in* $[0, 1]$.

Questions

- How can we prove that this scheme generates the desired distribution of configurations?
- Why make a random selection of the particle to be displaced?
- Why do we need to take the old configuration again?
- How large should we take: `delx`?

Detailed balance



$$K(o \to n) = K(n \to o)$$

$$K(o \to n) = N(o) \times \alpha(o \to n) \times \mathrm{acc}(o \to n)$$

$$K(n \to o) = N(n) \times \alpha(n \to o) \times \mathrm{acc}(n \to o)$$

$$\frac{\mathrm{acc}(o \to n)}{\mathrm{acc}(n \to o)} = \frac{N(n) \times \alpha(n \to o)}{N(o) \times \alpha(o \to n)} = \frac{N(n)}{N(o)}$$

NVT-ensemble

$$N(n) \propto \exp\left[-\beta U\left(n\right)\right]$$

$$\frac{\mathrm{acc}(o \to n)}{\mathrm{acc}(n \to o)} = \frac{N(n)}{N(o)}$$

$$\frac{\mathrm{acc}(o \to n)}{\mathrm{acc}(n \to o)} = \exp\left[-\beta\left[U\left(n\right) - U\left(o\right)\right]\right]$$

Algorithm 2 (Attempt to Displace a Particle)

```
      SUBROUTINE mcmove                    attempts to displace a particle

      o=int(ranf()*npart)+1                select a particle at random
      call ener(x(o),eno)                  energy old configuration
      xn=x(o)+(ranf()-0.5)*delx            give particle random displacement
      call ener(xn,enn)                    energy new configuration
      if (ranf().lt.exp(-beta              acceptance rule (3.2.1)
     +    *(enn-eno)) x(o)=xn               accepted: replace x(o) by xn
      return
      end
```

Comments to this algorithm:

1. *Subroutine* `ener` *calculates the energy of a particle at the given position.*
2. *Note that, if a configuration is rejected, the old configuration is retained.*
3. *The* `ranf()` *is a random number uniform in* $[0, 1]$.

Questions

- How can we prove that this scheme generates the desired distribution of configurations?
- Why make a random selection of the particle to be displaced?
- Why do we need to take the old configuration again?
- How large should we take: `delx`?

Algorithm 2 (Attempt to Displace a Particle)

```
   SUBROUTINE mcmove                    attempts to displace a particle

   o=int(ranf()*npart)+1                select a particle at random
   call ener(x(o),eno)                  energy old configuration
   xn=x(o)+(ranf()-0.5)*delx            give particle random displacement
   call ener(xn,enn)                    energy new configuration
   if (ranf().lt.exp(-beta              acceptance rule (3.2.1)
  +    *(enn-eno)) x(o)=xn              accepted: replace x(o) by xn
   return
   end
```

Comments to this algorithm:

1. *Subroutine* `ener` *calculates the energy of a particle at the given position.*
2. *Note that, if a configuration is rejected, the old configuration is retained.*
3. *The* `ranf()` *is a random number uniform in* $[0, 1]$.

Questions

- How can we prove that this scheme generates the desired distribution of configurations?
- Why make a random selection of the particle to be displaced?
- Why do we need to take the old configuration again?
- How large should we take: `delx`?

Mathematical

Transition probability:

$$\pi(o \to n) = \alpha(o \to n) \times \mathrm{acc}(o \to n)$$

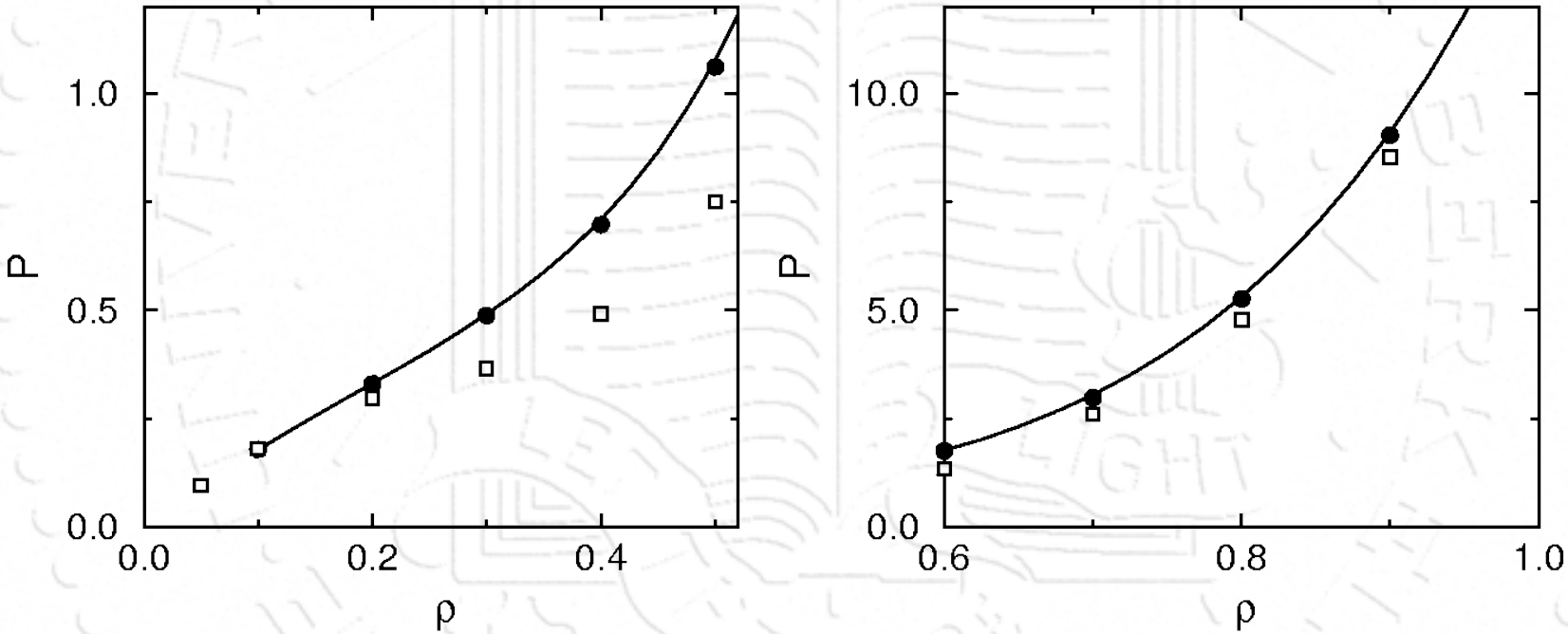
$$\sum_n \pi(o \to n) = 1$$

Probability to accept the old configuration:

$$\pi(o \to o) = 1 - \sum_{n \neq o} \pi(o \to n)$$

≠0

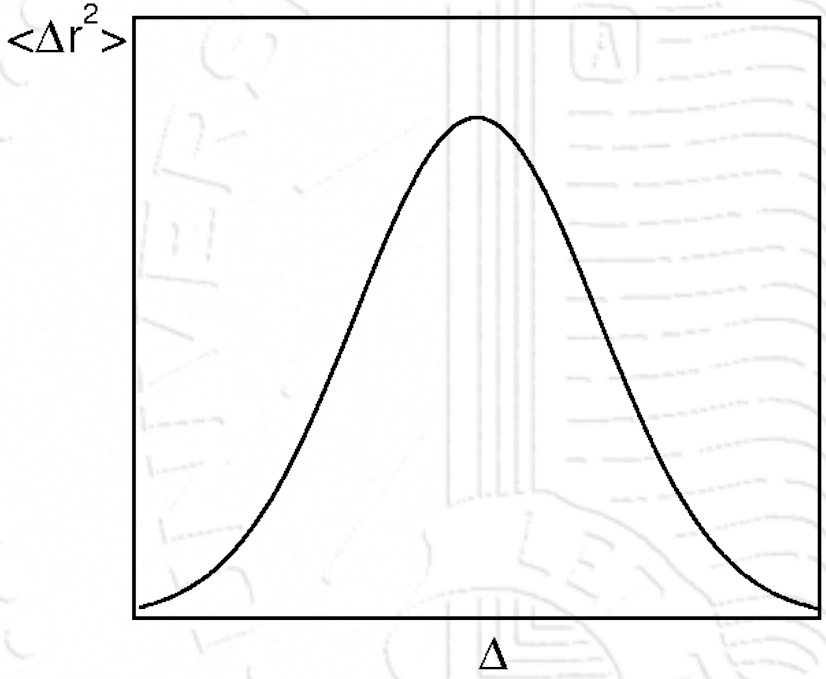
Keeping old configuration?

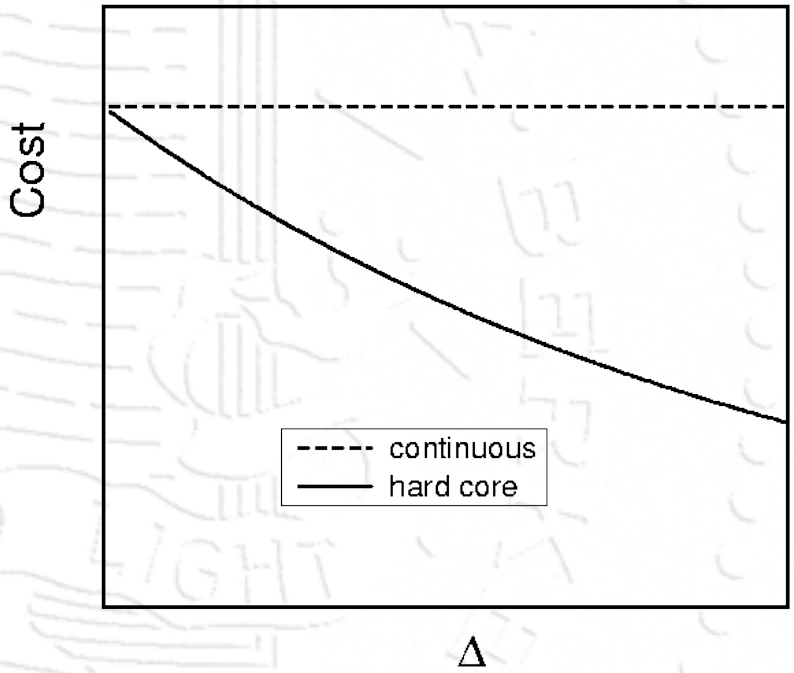


Questions

- How can we prove that this scheme generates the desired distribution of configurations?
- Why make a random selection of the particle to be displaced?
- Why do we need to take the old configuration again?
- How large should we take: `delx`?

Not too small, not too big!





Non-Boltzmann sampling

$$\langle A \rangle_{NVT_1} = \frac{1}{Q_{NVT_1}} \frac{1}{\Lambda^{3N} N!} \int \mathrm{d}\mathbf{r}^N A(\mathbf{r}^N) \exp\left[-\beta_1 U(\mathbf{r}^N)\right]$$

$$= \frac{\int \mathrm{d}\mathbf{r}^N A(\mathbf{r}^N) \exp\left[-\beta_1 U(\mathbf{r}^N)\right] \exp\left[\beta_2 U(\mathbf{r}^N) - \beta_2 U(\mathbf{r}^N)\right]}{\int \mathrm{d}\mathbf{r}^N \exp\left[-\beta_1 U(\mathbf{r}^N)\right] \exp\left[\beta_2 U(\mathbf{r}^N) - \beta_2 U(\mathbf{r}^N)\right]}$$

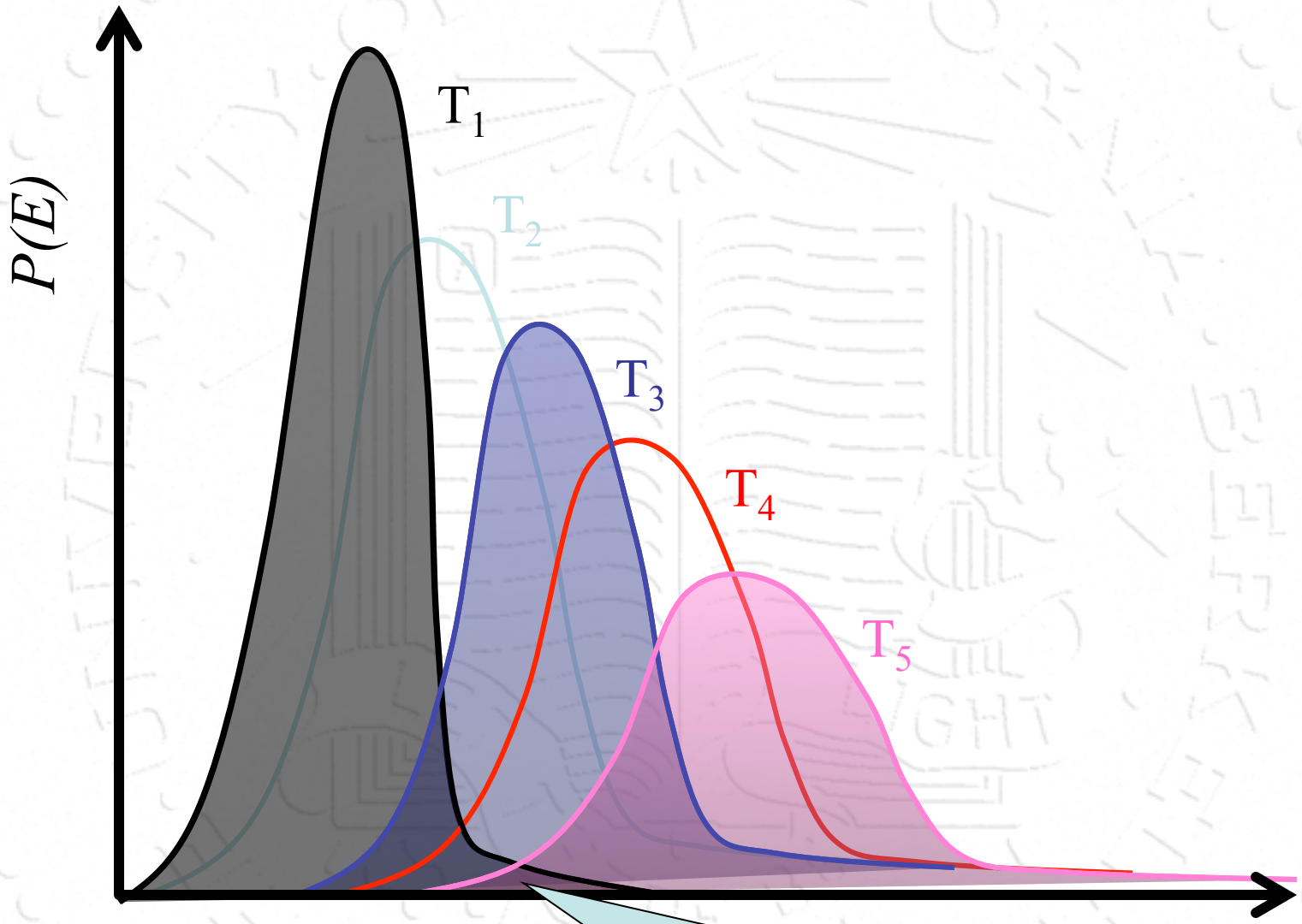
$$= \frac{\left\langle A \exp\left[(\beta_2 - \beta_1) U\right]\right\rangle_{NVT_2}}{\left\langle \exp\left[(\beta_2 - \beta_1) U\right]\right\rangle_{NVT_2}}$$

T_1 is arbitrary!

Why are we not using this?

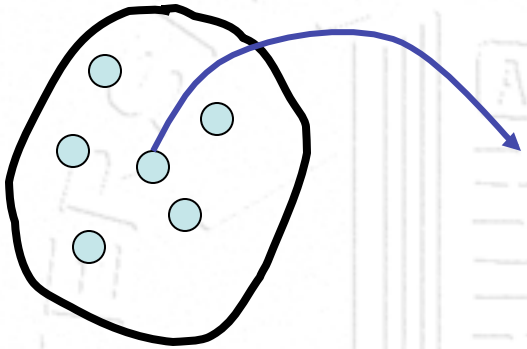
We only need a *single* simulation!

We perform a simulation at $T=T_2$ and we determine A at $T=T_1$



Overlap becomes very small

How to do *parallel* Monte Carlo



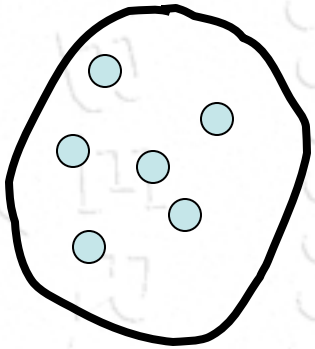
- Is it possible to do Monte Carlo in parallel
  - Monte Carlo is sequential!
  - We first have to know the fait of the current move before we can continue!

Parallel Monte Carlo

Algorithm (*WRONG*):

1. Generate *k* trial configurations in parallel
2. Select out of these the one with the lowest energy

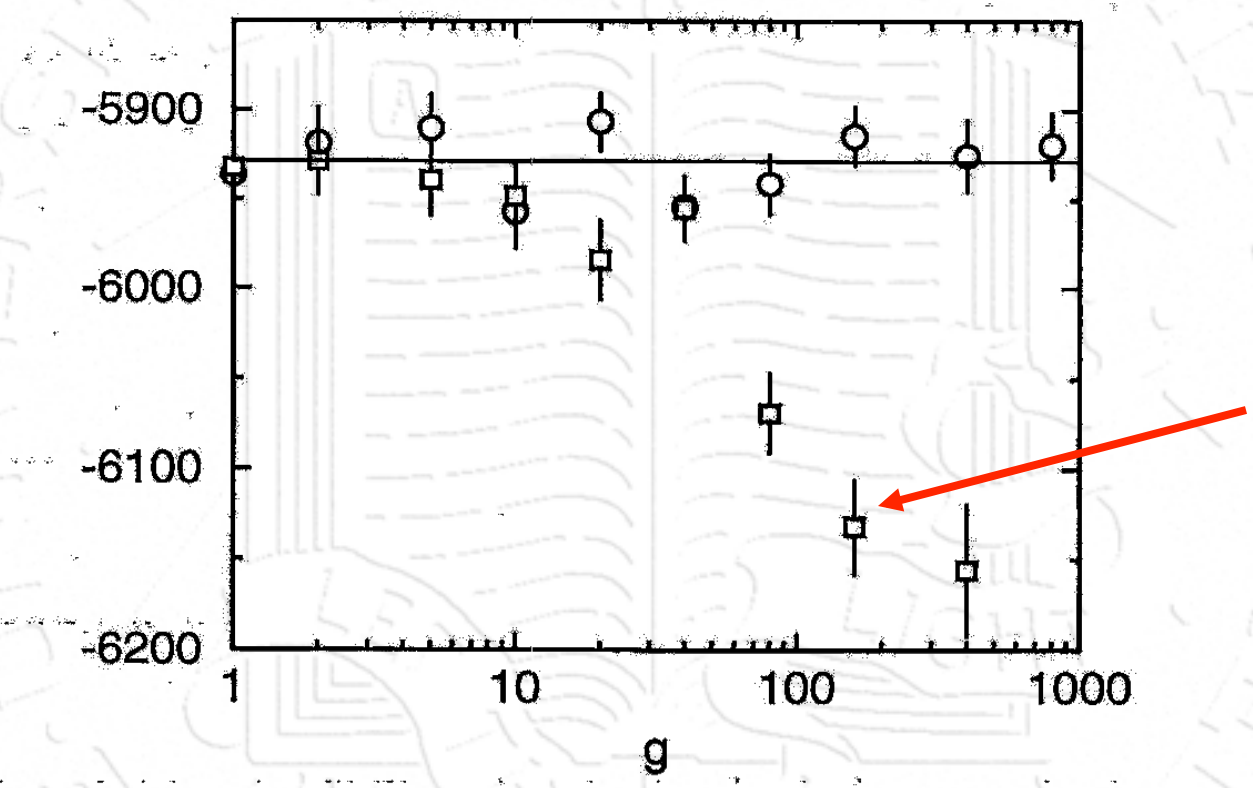
$$P(n) = \frac{\exp\left[-\beta\left(U_n\right)\right]}{\sum_{j=1}^{g} \exp\left[-\beta\left(U_j\right)\right]}$$



3. Accept and reject using normal Monte Carlo rule:

$$\mathrm{acc}\left(o \to n\right) = \exp\left[-\beta\left(U_n - U_o\right)\right]$$

Conventional acceptance rule



Conventional acceptance rules leads to a ***bias***

W

Detailed balance!

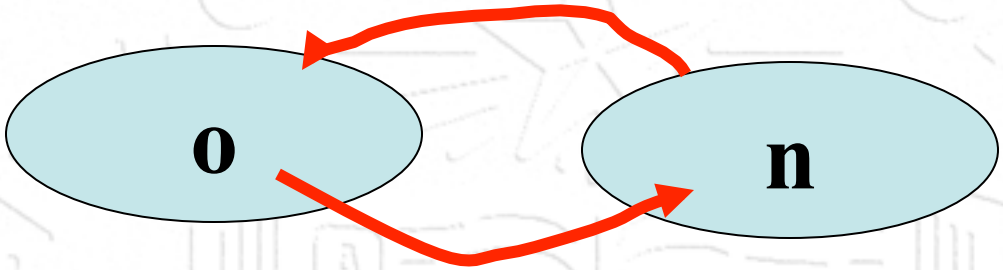
$$K(o \rightarrow n) = K(n \rightarrow o)$$

$$K(o \rightarrow n) = N(o) \times \alpha(o \rightarrow n) \times \mathrm{acc}(o \rightarrow n)$$

$$K(n \rightarrow o) = N(n) \times \alpha(n \rightarrow o) \times \mathrm{acc}(n \rightarrow o)$$

$$\frac{\mathrm{acc}(o \rightarrow n)}{\mathrm{acc}(n \rightarrow o)} = \frac{N(n) \times \alpha(n \rightarrow o)}{N(o) \times \alpha(o \rightarrow n)} = \frac{N(n)}{N(o)}$$

Detailed balance



$$K(o \to n) = K(n \to o)$$

$$K(o \to n) = N(o) \times \alpha(o \to n) \times \mathrm{acc}(o \to n)$$

$$K(n \to o) = N(n) \times \alpha(n \to o) \times \mathrm{acc}(n \to o)$$

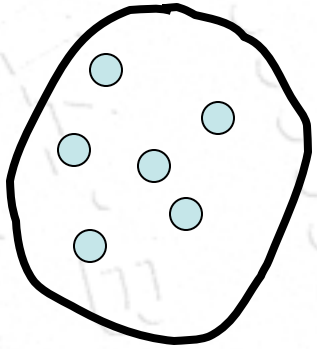
$$\frac{\mathrm{acc}(o \to n)}{\mathrm{acc}(n \to o)} = \frac{N(n) \times \alpha(n \to o)}{N(o) \times \alpha(o \to n)} = \frac{N(n)}{N(o)}$$

?

$$K(o \to n) = N(o) \times \alpha(o \to n) \times \mathrm{acc}(o \to n)$$

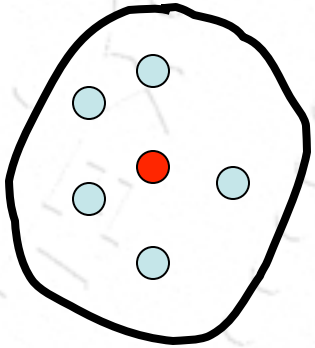
$$\alpha(o \to n) = \frac{\exp\left[-\beta\left(U_n\right)\right]}{\sum_{j=1}^{g} \exp\left[-\beta\left(U_j\right)\right]}$$

$$\alpha(o \to n) = \frac{\exp\left[-\beta\left(U_n\right)\right]}{W\left(n\right)}$$



$$\alpha(n \to o) = \frac{\exp\left[-\beta\left(U_o\right)\right]}{\sum_{j=1}^{g} \exp\left[-\beta\left(U_j\right)\right]}$$

$$\alpha(n \to o) = \frac{\exp\left[-\beta\left(U_o\right)\right]}{W\left(o\right)}$$

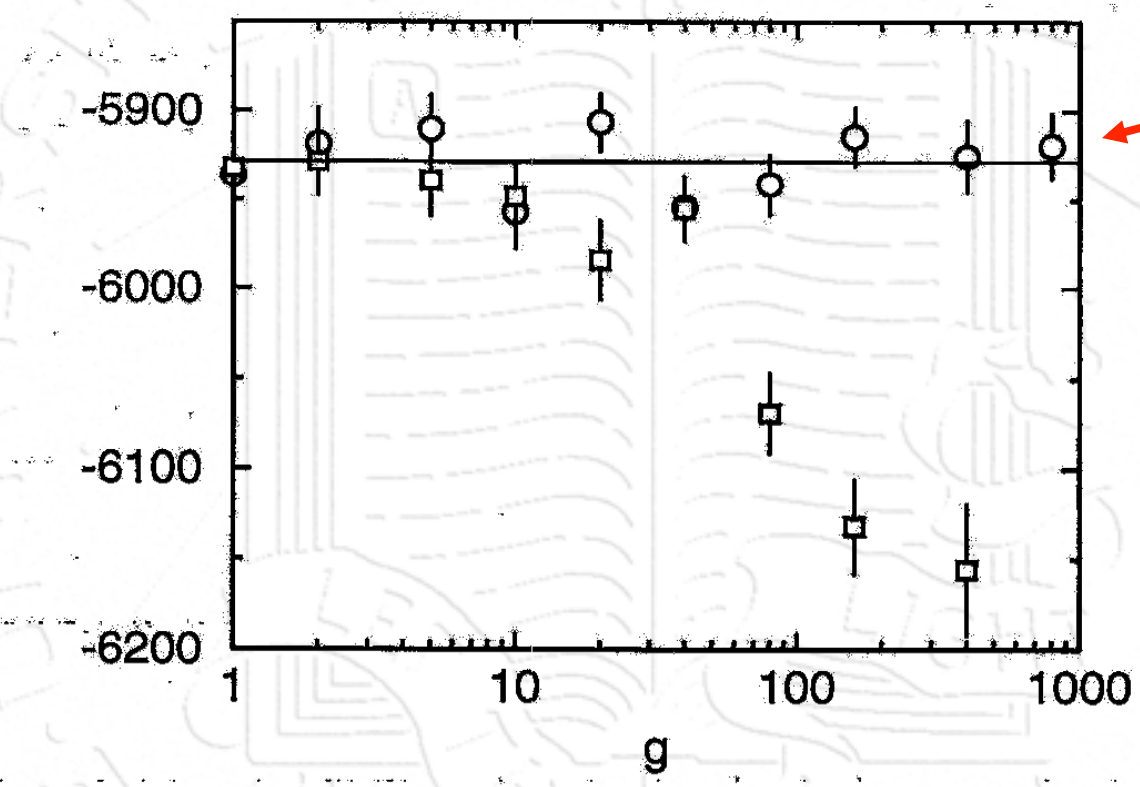


$$\frac{\mathrm{acc}(o \to n)}{\mathrm{acc}(n \to o)} = \frac{N(n) \times \alpha(n \to o)}{N(o) \times \alpha(o \to n)} = \frac{N(n)}{N(o)}$$

$$\alpha(o \to n) = \frac{\exp\left[-\beta\left(U_n\right)\right]}{W\left(n\right)} \quad \alpha(n \to o) = \frac{\exp\left[-\beta\left(U_o\right)\right]}{W\left(o\right)}$$

$$\frac{\mathrm{acc}(o \to n)}{\mathrm{acc}(n \to o)} = \frac{N(n) \times \dfrac{\exp\left[-\beta\left(U_o\right)\right]}{W\left(o\right)}}{N(o) \times \dfrac{\exp\left[-\beta\left(U_n\right)\right]}{W\left(n\right)}} = \frac{W(n)}{W(o)}$$

Modified acceptance rule



Modified acceptance rule remove the ***bias*** exactly!