

UFRJ — COPPE — PEE — CPE723 — Optimização Natural

Simulated Annealing — Aula 01

1953 — Metropolis, Rosenbluth, Rosenbluth e Teller — estimação de propriedades médias de substâncias consistindo de partículas individuais em interacç.

1983 — Kirkpatrick, Gelatt e Vecchi — estendem o algoritmo de Metropolis do modo a obter um procedimento "natural" (baseado em mecânica estatística) para aplicação em optimizaç. Extensão: loop de redução de temperatura T_k .

1984 — German e German — considere o "evento" = "configurações
(estados x) geradas pelo algoritmo são aquelas configurações
de energia mínima." Então, se a sequência de temperaturas
 T_k for limitada "por baixo" por $T_0/\log(k+1)$, à medida em
que k tende ao infinito, a probabilidade de "evento" tende a 1.
(Demonstração de convergência do SA para o ótimo global).

1987 — Szu e Hartley — Fast Simulated Annealing.

1998 — Rose — "Aproveitar o melhor dos dois mundos:
otimização determinística (sem vagar aleatoriamente pela
superfície de energia fazendo progresso incremental na média)
e resfriamento (evitando a atracção por mínimos locais
próximos)." Substituição do uso de simulações estocásticas
pelo uso do operador "valor esperado".

Algoritmo de Metropolis — Algumas Definições

N partículas — N é um número

grande. Por exemplo, da ordem de

6.02×10^{23} (constante de Avogadro).

Na nossa disciplina, vamos

considerar números muito menores.

" $N = 5$ "

$x(4)$

o

$x(5)$

o

$x(1)$

o

$x(3)$

o

$x(2)$

o

$$\text{Estado } \mathbf{x} = \begin{bmatrix} | & | & | & | & | \\ x(1) & x(2) & x(3) & x(4) & x(5) \\ | & | & | & | & | \end{bmatrix}, \quad \mathbf{x} \in \mathbb{R}^{2 \times 5}$$

$$\text{Estado } \mathbf{x} = \begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} & x_{15} \\ x_{21} & x_{22} & x_{23} & x_{24} & x_{25} \end{bmatrix}$$

↑
posicō da
partícula 1
no \mathbb{R}^2

↑
posicō da
partícula 5
no \mathbb{R}^2

No caso de $N=5$ partículas em $M=2$ dimensões, o número de componentes do estado x é 10, ou seja, o estado tem 10 dimensões.

Escrevemos $x \in \mathbb{R}^{10}$, em vez de $x \in \mathbb{R}^{2 \times 5}$.

Temperatura T — na aula de hoje, e também no artigo de Metropolis et al., assumimos T constante.

$J(x)$: função energia do sistema com N partículas (ou "função-custo" do respectivo sistema).

$$\text{Por exemplo: } J(x) = \underbrace{\frac{1}{N} \sum_n \|x(n)\|^2}_{\text{distância quadrática média das partículas à origem}} + \lambda \underbrace{\sum_m \sum_{n>m} \frac{1}{\|x(m) - x(n)\|^2}}_{\text{somatório das forças de repulsão entre as partículas multiplicador de Lagrange}}$$

Considere também $F(x)$: uma propriedade qualquer do sistema com N partículas.

Por exemplo: $F(x) = \frac{1}{N} \sum_n \|x(n)\|^2 \longrightarrow$ neste caso, $F(x)$ é de noua a distância quadrática média das partículas à origem.

Valor esperado da propriedade: $E[F(x)]$ (*)

$$E[F(x)] = \frac{\int f(x) e^{-\frac{E(x)}{kT}} dx}{\int e^{-\frac{E(x)}{kT}} dx}$$

distribuição (ou densidade)
"canônica", ou distribuição
de Boltzmann-Gibbs.

$(k = 1.38 \times 10^{-23} \text{ J/K, constante de Boltzmann})$

(*) Observação: "valor" esperado de um vetor aleatório X , dado que a sua função densidade de probabilidade é $f_X(x)$: $E[X] = \int x f_X(x) dx$

Valor esperado de uma função $g(x)$: $E[g(X)] = \int g(x) f_X(x) dx$

Toda p.d.f. satisfaz $\int f_X(x) dx = 1$. Caso usemos, em lugar de $f_X(x)$, pesos $w(x)$

que não satisfazem $\int w(x) dx = 1$, então é preciso normalizar:

$$E[g(X)] = \frac{\int g(x) w(x) dx}{\int w(x) dx}$$

A avaliação numérica da integral

$$E \subset F(x) = \frac{\int f(x) e^{\frac{-\beta(x)}{kT}} dx}{\int e^{\frac{-\beta(x)}{kT}} dx}$$

por métodos convencionais de integração é muito difícil. Usamos, em lugar

da integração numérica convencional, o método de Monte Carlo.

Vejamos, a seguir, dois exemplos...

$$\text{Exemplo 1: } \int_2^3 \frac{1}{x} dx = \left(\int_2^3 \frac{1}{x} f_x(x) dx = \right) \ln(x) \Big|_2^3 = \ln(3) - \ln(2) = 0.4055$$

$f_x(x) = 1$ (X com densidade uniforme no intervalo $x \in [2, 3]$)

MATLAB

```
x = rand(100000, 1);
```

```
x = x + 2;
```

```
sum(1./x)/100000
```

ans = 0.4054

(básico)
aprox.
maior

PYTHON

```
import numpy as np
```

```
N = 100000
```

```
x = np.random.uniform(0, 1, N) + 2
```

```
print(np.mean(1/x))
```

o comando print retorna 0.4056

$$\text{Exemplo 2: } \int_0^1 xe^{-x} dx = -xe^{-x} \Big|_0^1 + \int_0^1 e^{-x} dx = 1 - 2e^{-1} = 0.2644$$

$(u = x)$
 $du = e^{-x} dx$

$-e^{-1}$
 $1 - e^{-1}$

$N = 1000000$

`x = rand(1e6, 1);`

`mean(x.*exp(-x))`

`ans = 0.2644`

`x = random('exp', 1, 1e6, 1);`

`sum(x(x<1))/1e6`

`ans = 0.2639`

`x = np.random.uniform(0, 1, N)`

`print(np.mean(x * np.exp(-x)))`

0.2642

`x = np.random.exponential(1, N)`

`print(np.sum(x[x<1])/N)`

0.2643

Problema: com o uso do gerador de vetores aleatórios X com densidade uniforme temos, com alta probabilidade, $\omega(x) = \exp(-J(x)/(kT)) \approx 0$.



($\omega(x)$ é o peso com o qual x contribui para a integral $E[F(x)]$)

Na segunda parte do Exemplo 2, nós já geramos vetores aleatórios X com densidade "adequada" à integral, isto é, exponencial.

Solução para o problema: sortear os vetores x já com probabilidade proporcional a $\omega(x)$, e dar a todos $F(x)$ o mesmo peso na média que aproxima $E[F(x)]$.
(\hookrightarrow aparece no slide seguinte)

Pergunta: como fazer um gerador de vetores aleatórios X com probabilidade proporcional a $\exp(-\mathcal{J}(x)/kT)$?

Algoritmo de Metropolis:

x_0 qualquer; $n=0$; $N=1000000$; ϵ = número "pequeno" (ex.: 10^{-3})

$$\rightarrow \hat{x}_{n+1} = x_n + \epsilon R; \Delta \mathcal{J} = \mathcal{J}(\hat{x}_{n+1}) - \mathcal{J}(x_n);$$

$$q = \exp(-\Delta \mathcal{J}/(kT)); r = U(0,1); a = (0, \text{se } r > q; \text{ ou } 1, \text{ se } r < q)$$

$$x_{n+1} = \begin{cases} \hat{x}_{n+1}, & \text{se } \Delta \mathcal{J} < 0 \\ (1-a)x_n + a\hat{x}_{n+1}, & \text{se } \Delta \mathcal{J} \geq 0 \end{cases}$$

$n = n+1$; if $n == N$, end; else...

Do final do loop acima, fazemos então

(estimativa de $\in [F(x)]$)
obtida através de média
amostral)

$$\hat{F} = \frac{1}{M} \sum_{n=N-M+1}^N f(x_n)$$

Algoritmo de Metropolis escrito de forma mais compacta:

x qualquer; $n = 0$; $N = "1000000"$; $\epsilon = "10^{-3}"$; $M = "10\% \text{ de } N"$; $\hat{F} = 0$;

while $n < N$,

$$\hat{x} = x + \epsilon R;$$

(obs.: cada vez que esse comando é executado,

sorteia-se R de uma p.d.f. com média zero).

if $U(0,1) < \exp((\beta S(x) - \beta S(\hat{x}))/(\kappa T))$, $x = \hat{x}$; end;

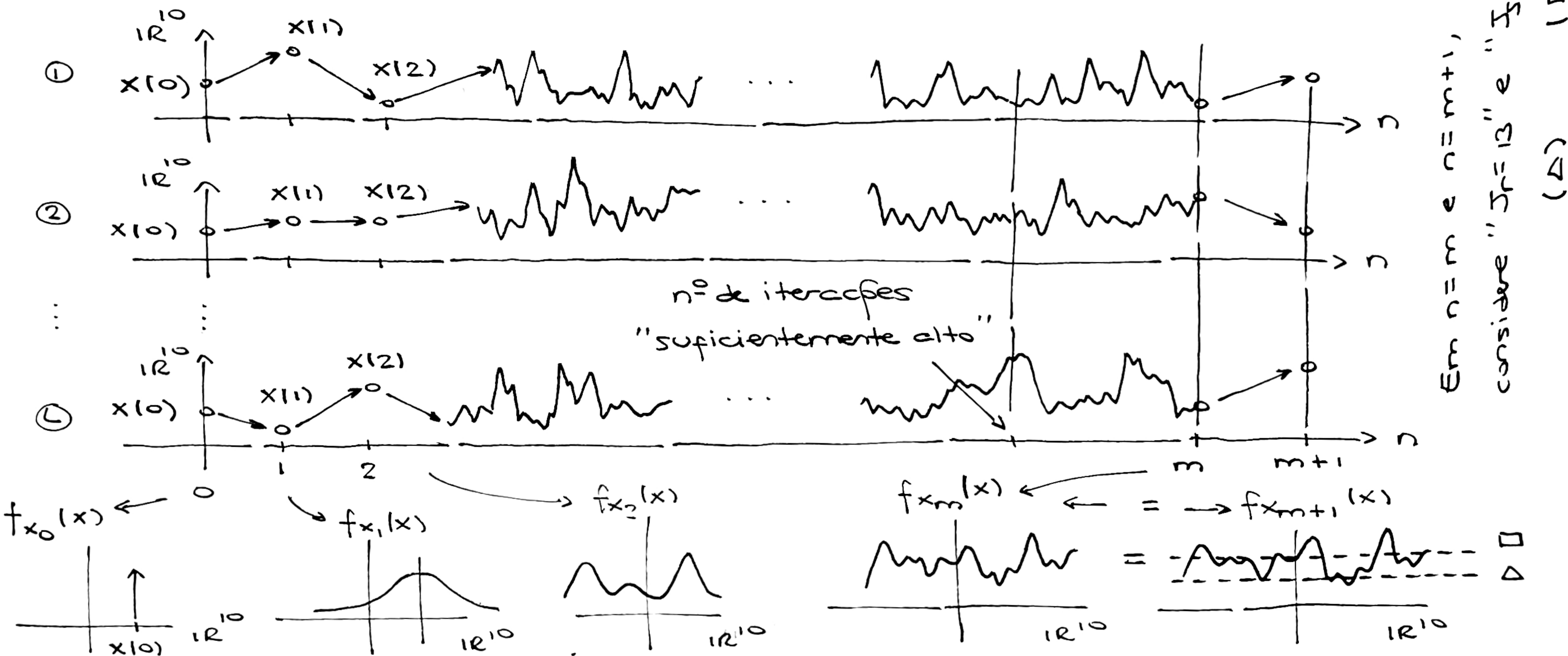
if $n > (N-M)$, $\hat{F} = \hat{F} + F(x)$; end;

$$n = n + 1;$$

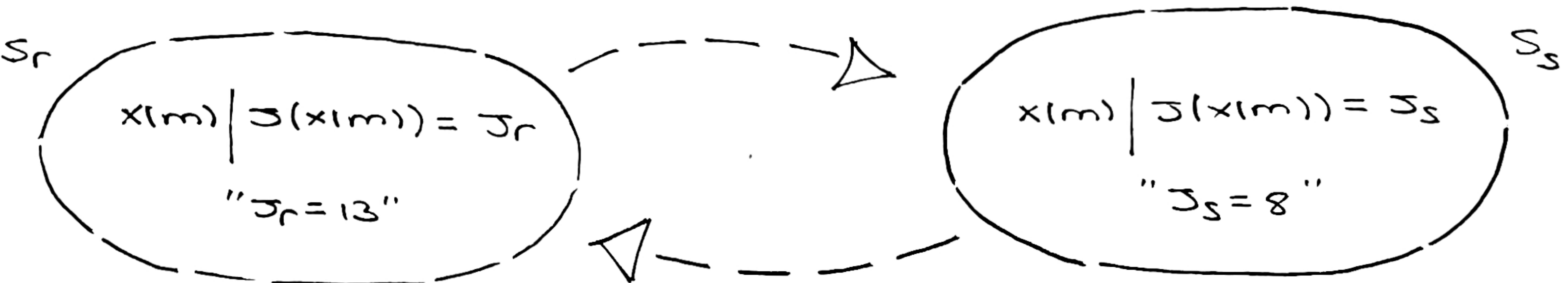
end;

$$\hat{F} = \hat{F}/M;$$

Análise da Convergência (lembre: no exemplo das 5 partículas, $x \in \mathbb{R}^{10}$)



Algoritmo de Metropolis — Análise da Convergência



cardinalidade do
conjunto S_r : J_r

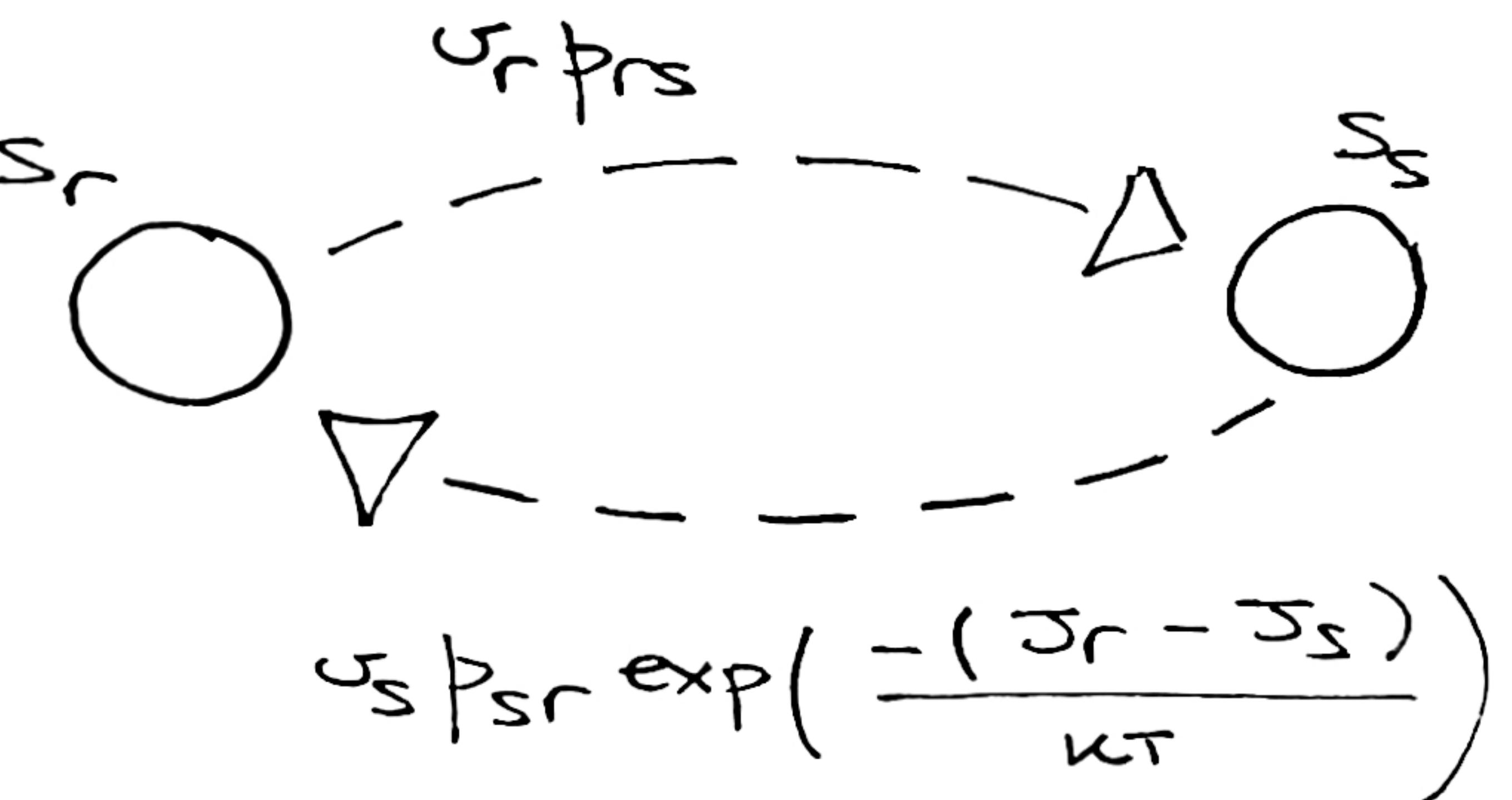
cardinalidade do
conjunto S_s : J_s

As setas tracejadas representam dois fluxos de estados, que acontecem à medida em que o tempo avança de $n = m$ para $n = m + 1$. Um fluxo de estados $x(m)$ com $J(x(m)) = J_r$ para $x(m+1)$ com $J(x(m+1)) = J_s$ e outro "de J_s para J_r ".

Algoritmo de Metropolis — Análise de Convergência

Como o gerador de perturbações aleatórias R (no Algoritmo de Metropolis) é simétrico (mais exatamente, tem média zero), as probabilidades de sortear um destino candidato $\hat{x}_{m+1} \in S_S$ a partir de $x_m \in S_r$ e de sortear $\hat{x}_{m+1} \in S_r \mid x_m \in S_S$, chamadas de p_{rs} e p_{sr} , são iguais.

(Equilíbrio: $f_{x_m} = f_{x_{m+1}}$)



$$u_r p_{rs} - u_s p_{sr} \exp\left(-\frac{(J_r - J_s)}{kT}\right) = 0$$

$$u_s p_{sr} \left(\frac{u_r}{u_s} - \exp\left(-\frac{(J_r - J_s)}{kT}\right) \right) = 0$$

$$\frac{u_r}{u_s} = \frac{\exp(-J_r/(kT))}{\exp(-J_s/(kT))} = \frac{\omega(x \in S_r)}{\omega(x \in S_S)}$$