Nov. 13, 2013 @IBIS

Exploring the Limits of Computation

# 確率と計算

## ～アルゴリズム理論の視点から

来嶋秀治

九州大学 大学院システム情報科学研究院

# 研究の興味 [(確率的)アルゴリズム論, 離散数学]

敬称略

## 確率的アルゴリズム

- 完璧サンプリング法 (MCMC法) (w/松井)
- ランダムウォークの脱乱択化 (w/牧野, 古賀, 梶野, 白髪, 山内, 山下)
- データストリーム中の頻出アイテム検知 (w/緒方, 山内, 山下)
- 置換多面体上の点の乱択丸め (w/畑埜, 瀧本, 安武, 末廣, 竹田, 永野)
- メディアン安定結婚問題 (w/根本)

## グラフアルゴリズム

- 区間グラフの部分グラフ同型性判定問題はP? NP-c? (w/斎藤, 大舘, 宇野)
- 区間グラフ上の支配集合数え上げ問題はP? #P-c? (w/岡本, 宇野)
- 最大次数4の剛性サーキットは，いつでも
  2つの互いに疎なハミルトン閉路に分解できるか？ (w/谷川)

## 分散アルゴリズム

(w/ 溝口, 小野, 徐, 山内, 山下)

- ポピュレーションプロトコルのリーダー選挙問題の領域計算量

# 「確率と計算」本講演のねらい

「乱択アルゴリズムにおいて、乱数に真に求める性質は何か？」

1. 乱択の威力: ストリーム中の頻出アイテム検知
2. 高度な乱択技法: 組合せ的対象のランダム生成 (MCMC法)
3. 脱乱択化: ランダムウォークの脱乱択化


# 1. ストリーム中の頻出アイテム検知

緒方 正虎，山内 由紀子，来嶋 秀治，山下 雅史

九州大学

θ: "頻出度"パラメータ

Σ: アイテム集合(有限)

# ストリームデータ中の頻出アイテム検知

## 問題: 頻出アイテム検知

Input: $\theta \in (0,1)$ $\boldsymbol{x} = (x_1,\ldots,x_N) \in \Sigma^N$ (順々に)

Find: all $s \in \Sigma$ s.t. $f(s) \geq \theta \cdot N$ ●○○

但し $f(s)$ は $s$ が $\boldsymbol{x}$ 中で出現した回数

事前には、
N (or log Nの近似値)も
わからない。

## 例1. 1日のPOSデータ(@果物屋)

$\Sigma$ ={ 🍎, 🍈, 🍌, …, 🍇 }

$\boldsymbol{x}$ = 🍎, 🍌, 🍇, 🍎, 🍎, 🍈, 🍎, 🍌, 🍌, 🍎, 🍌, ….

## 例. 1日のアクセスIPアドレス

$\Sigma \subseteq$ {0.0.0.0,…,255.255.255.255}

$\boldsymbol{x}$ = 123.45.67.89, 111.11.1.1., 123.45,67,89, 122.122.12.12…

would like to find items appearing w/ frequency more than θ=1% of N.

# 頻出アイテム検知の領域複雑度

定理 [Karp, Shenker, Papadimitriou '03]

頻出アイテム検知を厳密に行うには,

$\Omega(|\Sigma| \log (N/|\Sigma|))$ bits が必要.

($N \gg |\Sigma| \gg 1/\theta$ とする)

定理 [Karp, Shenker, Papadimitriou '03]

頻出アイテム検知に対する

$O((1/\theta) \log N)$ bits の偽陽性(近似)アルゴリズムが存在.

($N \gg |\Sigma| \gg 1/\theta$ とする)

決定的

$o(\log N)$ bitsアルゴリズム？

- e.g. $O(\log \log N)$ bits?

# 単純化: o(log N) bitsで要素数を数えられるか?

**問題: 要素数え上げ**

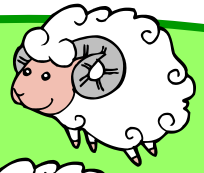
**Input**: $x = (a,\ldots,a) \in \Sigma^N$ (順々に)

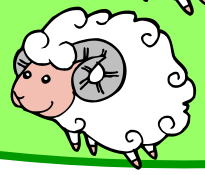
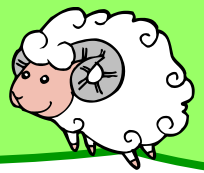
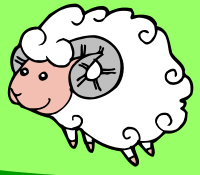
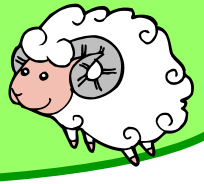
**Find**: N

事前には、
N (or log Nの近似値)も
わからない。

**アルゴリズム: 数え上げ**

0. **Set** n:= 0.
1. **Read** an input. **If** no more input, **goto** 3.
2. n++, **Goto** 1.
3. **Output** n (as N = n).


$\Sigma$ = {  }

X =  ,  ,  ,  , … …

# 単純化: o(log N) bitsで要素数を数えられるか？

問題: 要素数え上げ

Input: $x = (a,\ldots,a) \in \Sigma^N$ (順々に)

Find: N

事前には、
N (or log Nの近似値)も
わからない。

アルゴリズム: 数え上げ

0. **Set** n:= 0.
1. **Read** an input. **If** no more input, **goto** 3.
2. n++, **Goto** 1.
3. **Output** n (as N = n).

O(log N) bits

o(log N) bits近似アルゴリズム？

- e.g. O(log log N) bits?

# 単純化: o(log N) bitsで要素数を数えられるか？

問題: 要素数え上げ

Input: $x = (a,\ldots,a) \in \Sigma^N$ (順々に)

Find: N

事前には、
N (or log Nの近似値)も
わからない。

Remark

- NはO(log N) bitsで表現可能.
  - ✓ N = 1,351,127,649,213
- Nの近似はO(log log N) bitsで表現可能
  - ✓ $N \approx 1.351 \times 10^{12}$

つまり、

指数部(=log N)がo(log N) bits領域で近似計算できるか？ということ。

表現はO(log log N) bitsで可能

# 単純化: o(log N) bitsで要素数を数えられるか？

問題: 要素数え上げ

Input: $x = (a,\ldots,a) \in \Sigma^N$ (順々に)

Find: N

事前には、
N (or log Nの近似値)も
わからない。

アルゴリズム: 数え上げ

0. **Set** n:=0.
1. **Read** an input. **If** no more input, **goto** 3.
2. n++, **Goto** 1.
3. **Output** n (as N = n).

$\Theta(\log N)$ bits

定理. [Flajolet '85, Ogata et al. '11]
決定的アルゴリズムでは$\Omega(\log N)$ bit 必要！

# 確率的数え上げ

**問題: 要素数え上げ**

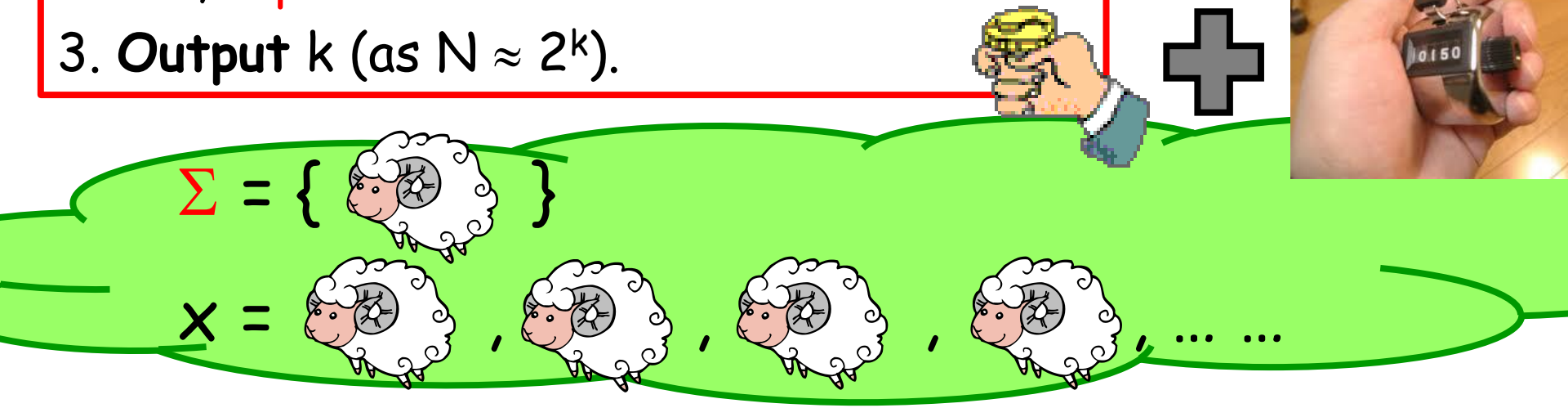
Input: $x = (a,\ldots,a) \in \Sigma^N$ (順々に)

Find: N

事前には、
N (or log Nの近似値)も
わからない。

**アルゴリズム: 確率的数え上げ**

0. **Set** k:=0.
1. **Read** an input. **If** no more input, **goto** 3.
2. k++, w.p. $1/2^k$. **Goto** 1.
3. **Output** k (as $N \approx 2^k$).



# 確率的数え上げ

問題: 要素数え上げ

Input: $x = (a,\ldots,a) \in \Sigma^N$ (順々に)

Find: N

事前には、
N (or log Nの近似値)も
わからない

⇒ key point

"w.p. $1/2^k$ " using
$O(\log K)$ bits on PTM.

アルゴリズム: 確率的数え上げ

0. **Set** k:=0.
1. **Read** an input. **If** no more input, **goto** 3.
2. k++, w.p. $1/2^k$. ~~**Goto**~~ 1.
3. **Output** k (as $N \approx 2^k$).

$O(\log \log N)$ bits

Thm. [Morris '78, Flajolet '85]

$E[2^k] \approx N+1$

because
$N \approx 1+2+4+8+16+\ldots+2^k$

# 確率的数え上げ(改良型; Ogata et al. 2011)

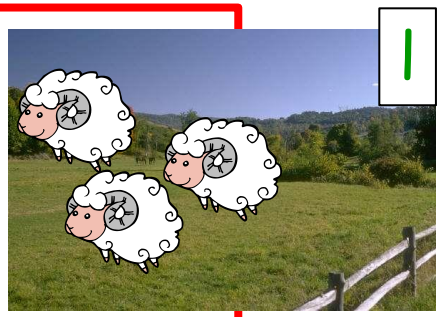
問題: 要素数え上げ

Input: $x = (a,\ldots,a) \in \Sigma^N$ (順々に)

Find: N

事前には、
N (or log Nの近似値)も
わからない。

アルゴリズム: 確率的数え上げ(改良型)




0. **Set** k:=0, l:=0.
1. **Read** an input. **If** no more input, **goto** 4.
2. l++, w.p. $1/2^k$.
3. If l==$2^b$, k++, and **set** l := l' w.p. $\binom{2^b}{l'} \cdot 2^{-2^b}$. **Goto** 1.
4. **Output** l and k (as $N \approx l * 2^k$ ).

Thm. [Ogata, Yamauchi, K., Yamashita '11]

$E[l*2^k] \approx N$.

# 確率的数え上げ(改良型; Ogata et al. 2011)

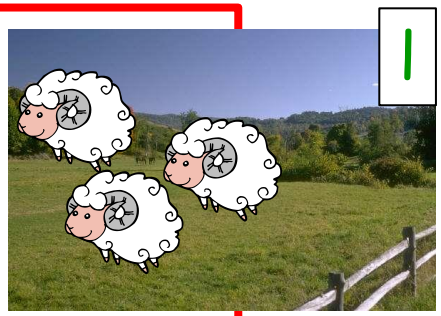
問題: 重複

Find

(直感的に) "l" (< $2^b$) は仮数部分を表す.

i.e., 各ひつじを牧場"l"に(確率$1/2^k$で)捕まえておく.

アルゴリズム: 確率的数え上げ(改良型)




0. **Set** k:=0, l:=0.
1. **Read** an input. **If** no more input, **goto** 4.
2. l++, w.p. $1/2^k$.
3. If l==$2^b$, k++, and **set** l := l' w.p. $\binom{2^b}{l'} \cdot 2^{-2^b}$. **Goto** 1.
4. Output $\hat{n} = l * 2^k$.

各ひつじは, k反復目の時, 確率$1/2^k$で(牧場"l"に)残っている.

なぜなら"exponent"=jの時捕まった確率$1/2^j$で捕まったひつじは"exponent"=k (k > j)の時, 確率$1/2^{k-j}$で牧場に残っている.

⇒ Pr[ひつじ in l] = $1/2^j * 1/2^{k-j} = 1/2^k$.

# 確率的数え上げ(改良型; Ogata et al. 2011)

問題: 要素数え上げ

Input: $x = (a,\ldots,a) \in \Sigma^N$ (順々に)

Find: N

事前には、
N (or log Nの近似値)も
わからない。

アルゴリズム: 確率的数え上げ(改良型)

0. **Set** k:=0, l:=0.
1. **Read** an input. **If** no more input, **goto** 4.
2. l++, w.p. $1/2^k$.
3. If l==$2^b$, k++, and **set** l := l' w.p. $\binom{2^b}{l'} \cdot 2^{-2^b}$. **Goto** 1.
4. **Output** l and k (as $N \approx l * 2^k$ ).

O(log log N) bits

Thm. [Ogata, Yamauchi, K., Yamashita '11]

$E[l*2^k] \approx N$.

"改良型" を使うと、頻出アイテム検知も可能. (詳細略)

# ストリームデータ中の頻出アイテム検知

問題: 頻出アイテム検知

Input: $\theta \in (0,1)$ $\mathbf{x} = (x_1,\ldots,x_N) \in \Sigma^N$ (順々に)

Find: all $s \in \Sigma$ s.t. $f(s) \geq \theta \cdot N$

但し $f(s)$ は $s$ が $\mathbf{x}$ 中で出現した回数

事前には、
N (or log Nの近似値)も
わからない。

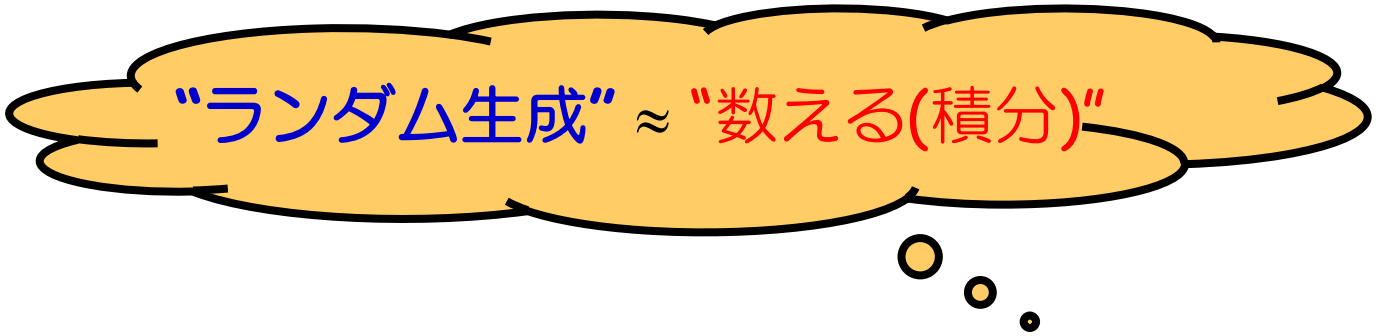
Thm. [Ogata, Yamauchi, K., Yamashita '11]

$O(\log \log N)$ bits 領域の乱択近似アルゴリズムが存在.

詳細略

$o(\log N)$ bitsアルゴリズム？

- e.g. $O(\log \log N)$ bits?




# 2. 組合せ的対象のランダム生成

マルコフ連鎖モンテカルロ法

(MCMC: Markov chain Monte Carlo)

joint with 松井知己 (中央大学)

# 例: 2行分割表のランダム生成

## 2元分割表

- ✓ 各セルには非負整数が入る
- ✓ （与えられた）周辺和を満たす

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | | 12 |
| | | | | | | 18 |
| 5 | 4 | 3 | 7 | 5 | 6 | 30 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 5 | 4 | 3 | 0 | 0 | 0 | 12 |
| 0 | 0 | 0 | 7 | 5 | 6 | 18 |
| 5 | 4 | 3 | 7 | 5 | 6 | 30 |

状態A

| | | | | | | |
|---|---|---|---|---|---|---|
| 4 | 3 | 1 | 3 | 1 | 0 | 12 |
| 1 | 1 | 2 | 4 | 4 | 6 | 18 |
| 5 | 4 | 3 | 7 | 5 | 6 | 30 |

状態B

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 5 | 6 | 12 |
| 5 | 4 | 3 | 6 | 0 | 0 | 18 |
| 5 | 4 | 3 | 7 | 5 | 6 | 30 |

状態C

**問題**

Given: 周辺和

出力: 分割表の一様ランダム生成

# 例: 2行分割表のランダム生成

2元分割表

- ✓ 各セルには非負整数が入る
- ✓ （与えられた）周辺和を満たす

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | | 12 |
| | | | | | | 18 |
| 5 | 4 | 3 | 7 | 5 | 6 | 30 |

与えられた周辺和を満たす2行分割表の**個数を求める問題**

⇒#P完全 (NP困難) ['97 Dyer, Kannan, & Mount]

**問題**

Given: 周辺和

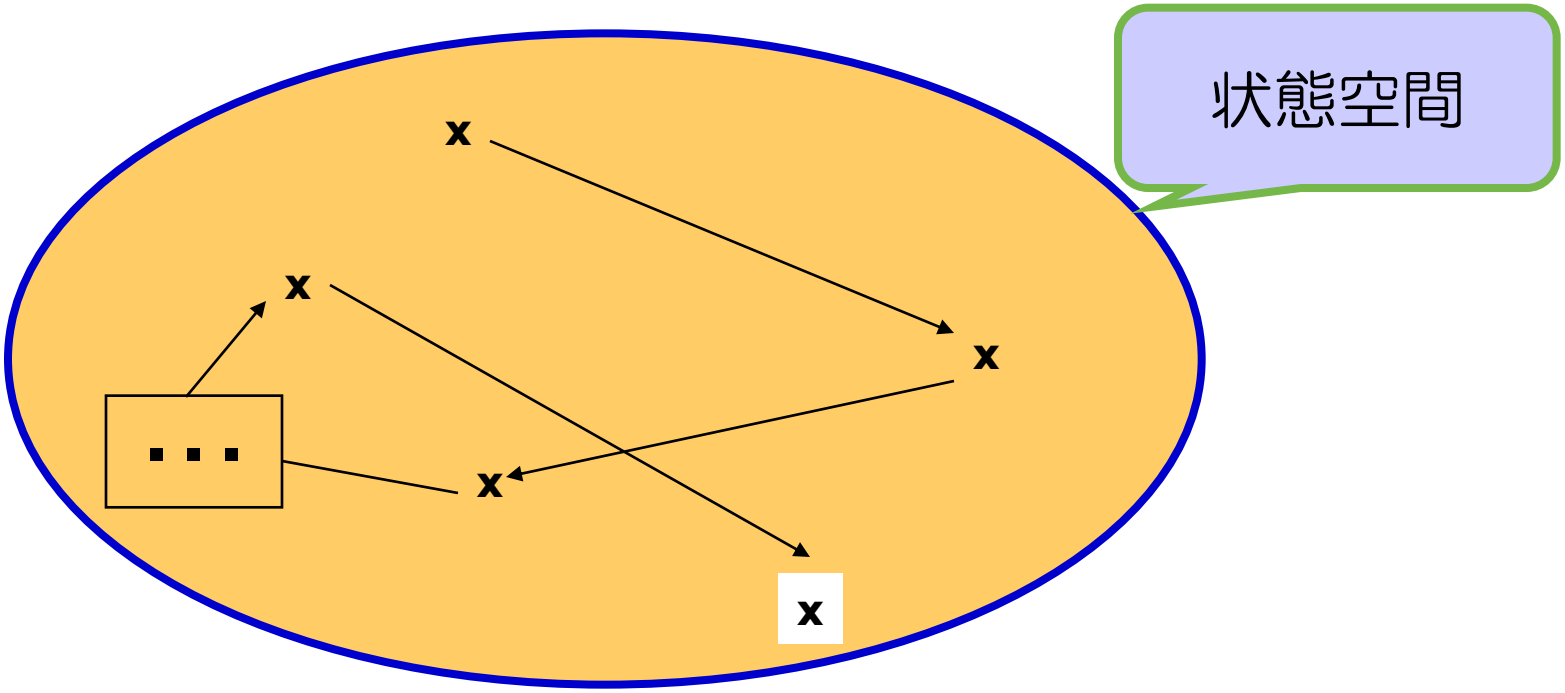
出力: 分割表の一様ランダム生成

マルコフ連鎖を用いた

サンプリング法

# MCMC法のアイデア

1. 所望の分布を定常分布にもつマルコフ連鎖を設計する.
2. 十分な回数推移させて，定常分布からサンプリングする.




⇒ (漸近的に)所望の分布に従うサンプリングを実現

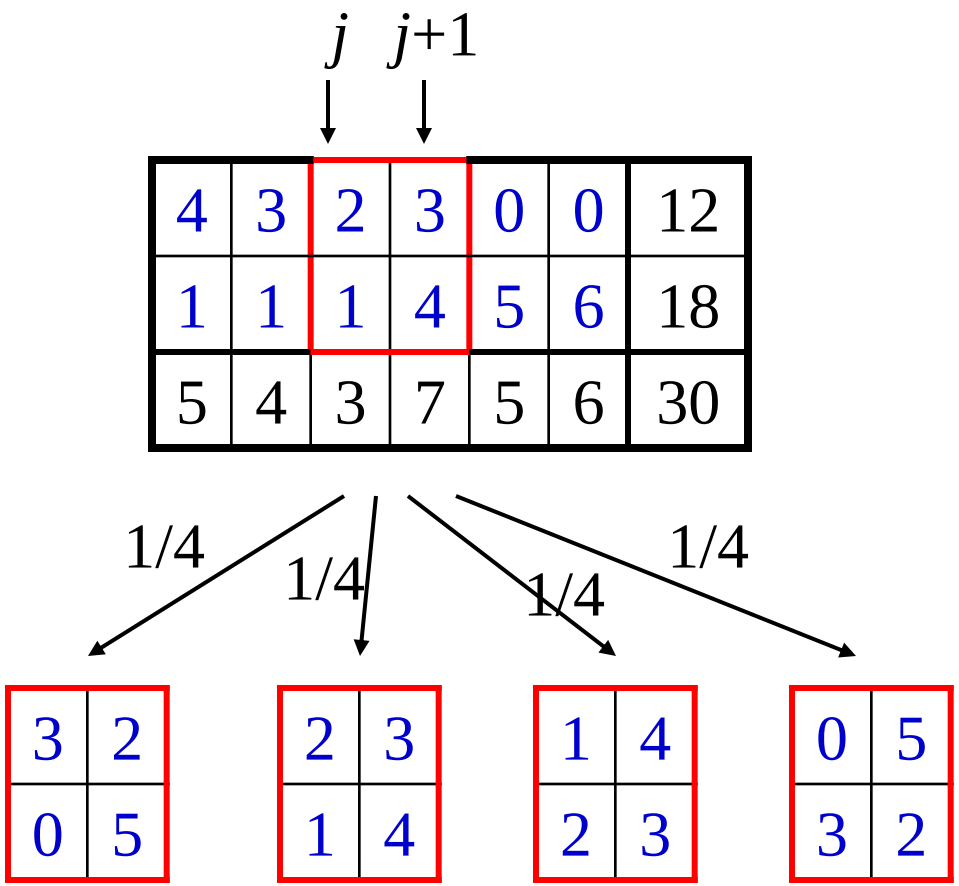
# 例: 2行分割表に対するマルコフ連鎖 [K & Matsui '06]

- $j$ 列目（と$j$+1列目）を $1/(n-1)$の確率で選ぶ。
- $j$ 列目と $j$+1列目に対して 推移可能な状態に 等確率で推移する。

| | | |
|---|---|---|
| 2 | 3 | 5 |
| 1 | 4 | 5 |
| 3 | 7 | 10 |

+

| | |
|---|---|
| $+k$ | $-k$ |
| $-k$ | $+k$ |



# 提案するマルコフ連鎖の特徴

**定理**

提案したマルコフ連鎖 の定常分布は一様分布である.

略証: $\forall(X, Y),\ P(X, Y) > 0 \Rightarrow P(Y, X) > 0$ かつ $P(X, Y) = P(Y, X)$

$X$

| | | | | | | |
|---|---|---|---|---|---|---|
| 4 | 3 | 2 | 3 | 0 | 0 | 12 |
| 1 | 1 | 1 | 4 | 5 | 6 | 18 |
| 5 | 4 | 3 | 7 | 5 | 6 | 30 |

$Y$

| | | | | | | |
|---|---|---|---|---|---|---|
| 4 | 3 | 0 | 5 | 0 | 0 | 12 |
| 1 | 1 | 3 | 2 | 5 | 6 | 18 |
| 5 | 4 | 3 | 7 | 5 | 6 | 30 |

(detailed balance equation $\pi(X)\, P(X, Y) = \pi(Y)\, P(Y, X)$ )

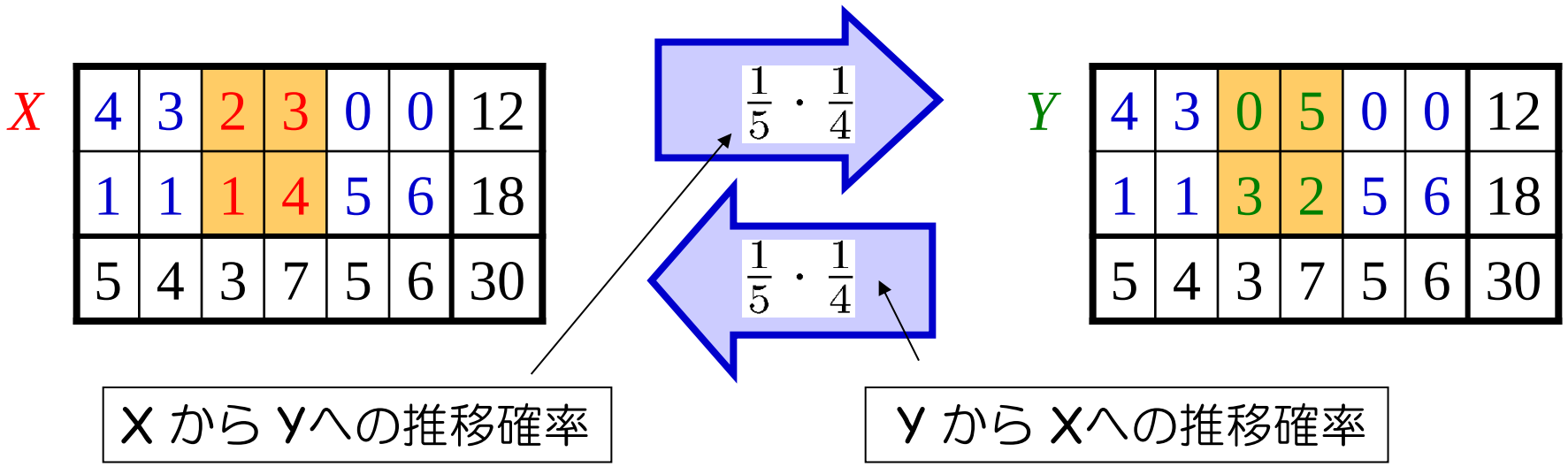
# 提案するマルコフ連鎖の特徴

**定理**

提案したマルコフ連鎖 の定常分布は一様分布である.

一様分布の証明: 詳細釣り合いの式

$$1 \cdot P(x, y) = 1 \cdot P(y, x) \qquad \forall x, y \in \Omega$$



# 提案するマルコフ連鎖の特徴

**定理**

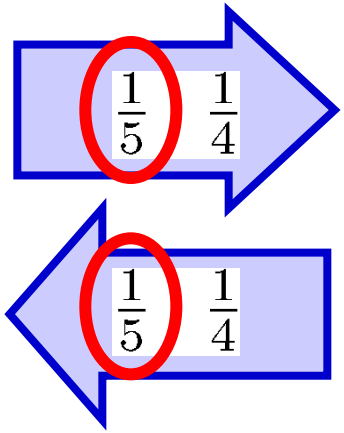
提案したマルコフ連鎖 の定常分布は一様分布である.

一様分布の証明: 詳細釣り合いの式

$$1 \cdot P(x, y) = 1 \cdot P(y, x) \qquad \forall x, y \in \Omega$$

$X$

| | | | | | | |
|---|---|---|---|---|---|---|
| 4 | 3 | 2 | 3 | 0 | 0 | 12 |
| 1 | 1 | 1 | 4 | 5 | 6 | 18 |
| 5 | 4 | 3 | 7 | 5 | 6 | 30 |




$Y$

| | | | | | | |
|---|---|---|---|---|---|---|
| 4 | 3 | 0 | 5 | 0 | 0 | 12 |
| 1 | 1 | 3 | 2 | 5 | 6 | 18 |
| 5 | 4 | 3 | 7 | 5 | 6 | 30 |

隣接２列を一様乱択 (確率 1/(6-1))

# 提案するマルコフ連鎖の特徴

**定理**

提案したマルコフ連鎖 の定常分布は一様分布である.

一様分布の証明: 詳細釣り合いの式

$$1 \cdot P(x, y) = 1 \cdot P(y, x) \qquad \forall x, y \in \Omega$$

$X$

| | | | | | | |
|---|---|---|---|---|---|---|
| 4 | 3 | 2 | 3 | 0 | 0 | 12 |
| 1 | 1 | 1 | 4 | 5 | 6 | 18 |
| 5 | 4 | 3 | 7 | 5 | 6 | 30 |

$\frac{1}{5}$ $\frac{1}{4}$

$\frac{1}{5}$ $\frac{1}{4}$

$Y$

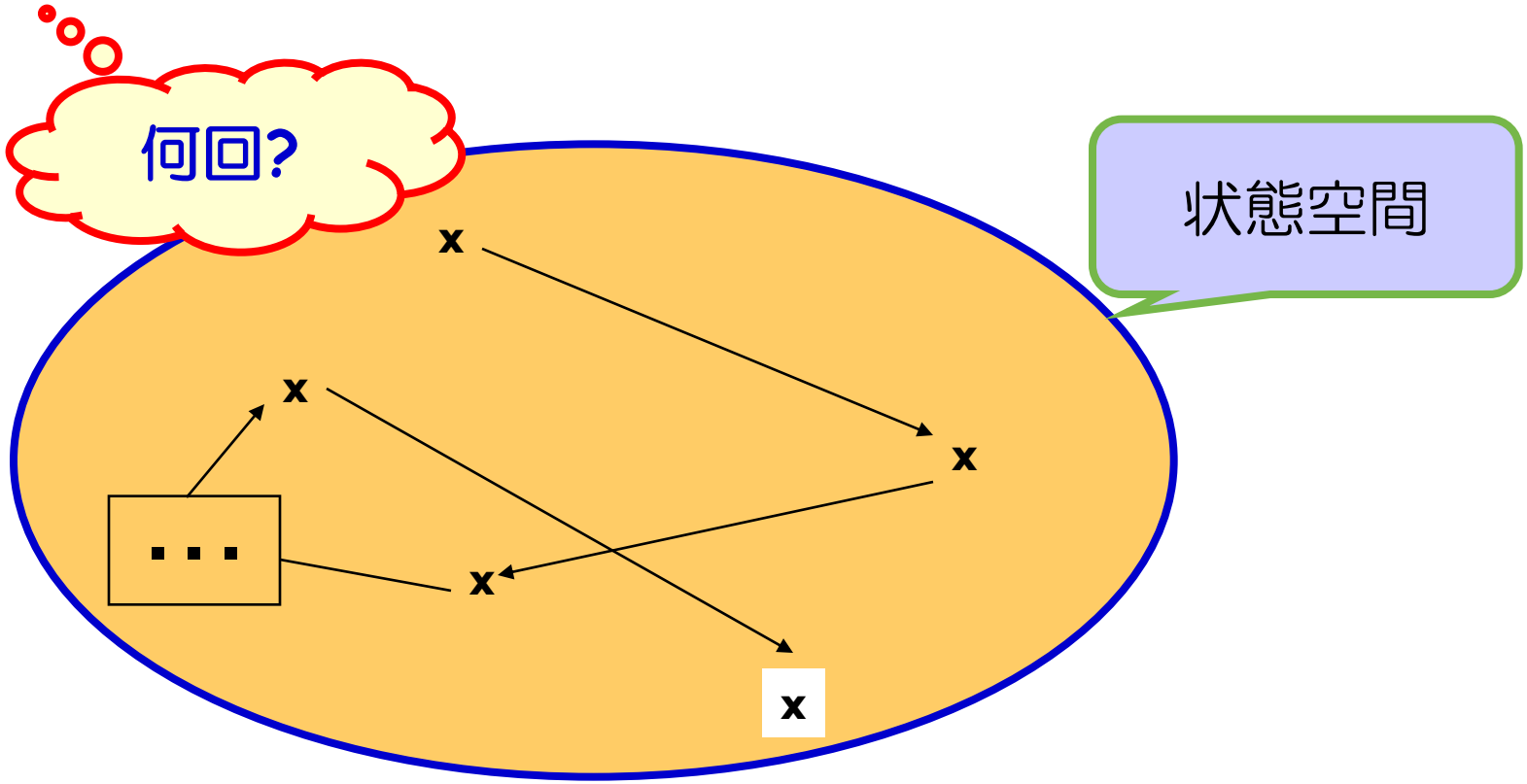
| | | | | | | |
|---|---|---|---|---|---|---|
| 4 | 3 | 0 | 5 | 0 | 0 | 12 |
| 1 | 1 | 3 | 2 | 5 | 6 | 18 |
| 5 | 4 | 3 | 7 | 5 | 6 | 30 |

(3,4) 列が選ばれたとき,
可能な推移先は(共に)4つ

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 2 | 2 | 3 | 1 | 4 | 0 | 5 |
| 0 | 5 | 1 | 4 | 2 | 3 | 3 | 2 |

# MCMC法のアイデア

1. 所望の分布を定常分布にもつマルコフ連鎖を設計する.
2. 十分な回数推移させて，定常分布からサンプリングする.




⇒ (漸近的に)所望の分布に従うサンプリングを実現

## 問題点は何か？

「何回推移させれば十分か？」

### 近似サンプリング法

- ✓ 収束スピードの算定
  - **mixing time**, total variation distance

### 完璧サンプリング法

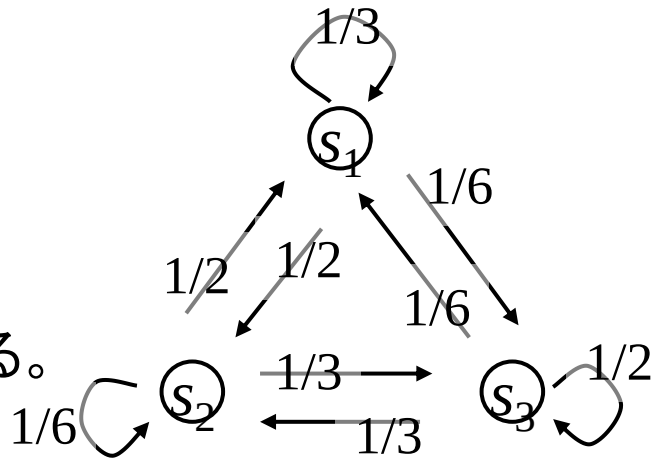
- ✓ マルコフ連鎖の推移シミュレーションを工夫
- ✓ 無限回の推移の結果を出力 (⇒ 定常分布に厳密に従う)
  - **Coupling from the past** [Propp & Wilson 1996]

・・・

単調マルコフ連鎖の設計で効率化

# 更新関数 （update function）

- マルコフ連鎖 $MC$ （エルゴード的）
  - 状態空間: $s_1, s_2, s_3$ ; 有限 （3 状態）
  - 推移規則: 乱数 $\lambda \in \{1,\ldots,6\}$ に対して決まる。

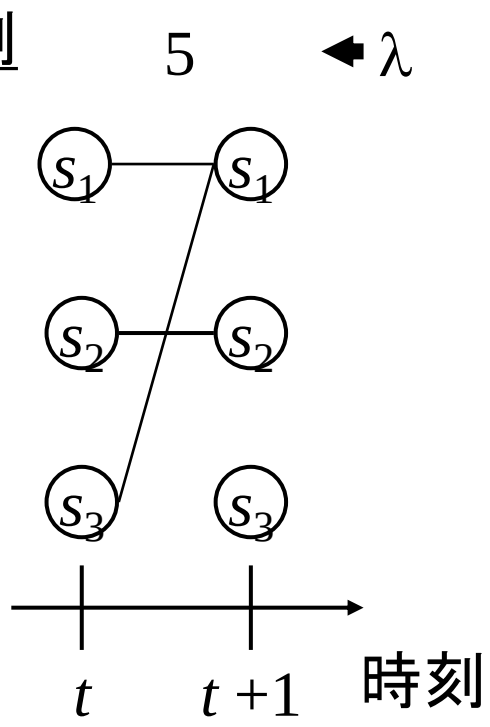



更新関数

現在の状態

| 乱数 | $s_1$ | $s_2$ | $s_3$ |
|---|---|---|---|
| 1 | $s_3$ | $s_1$ | $s_2$ |
| 2 | $s_2$ | $s_3$ | $s_2$ |
| 3 | $s_2$ | $s_1$ | $s_3$ |
| 4 | $s_1$ | $s_1$ | $s_3$ |
| 5 | $s_1$ | $s_2$ | $s_1$ |
| 6 | $s_2$ | $s_3$ | $s_3$ |

例



# CFTPアルゴリズムと定理 [Propp & Wilson 1996]

マルコフ連鎖 $MC$:
- 有限離散の状態空間 $\Omega$
- 更新関数 $\Phi_s^t(x, \lambda)$ ← 時刻 $s$, 状態 $x$ から 時刻 $t$ まで推移 (乱数列 $\lambda$ )
- エルゴード性を満たす。

## CFTPアルゴリズム

1. set $T = -1$; set $\lambda$: 空列;
2. generate $\lambda[T]$ : 乱数; put $\lambda := ( \lambda[T], \lambda[T+1], \ldots, \lambda[-1] )$;
3. $\Omega$ の全ての状態について、時刻 $T$ から 0 まで $\lambda$ を用いて マルコフ連鎖 $MC$ を推移させる。
   a. if coalesce $(\exists y \in \Omega, \forall x \in \Omega, y = \Phi_T^0(x, \lambda)) \Rightarrow$ return $y$;
   b. otherwise, set $T := T - 1$ ; step 2.に戻る;

## CFTP定理

上のアルゴリズムが停止して値を返す時、

その値はマルコフ連鎖 $M$ の定常分布に厳密に従う確率変数を実現する。

# CFTPのアイデア

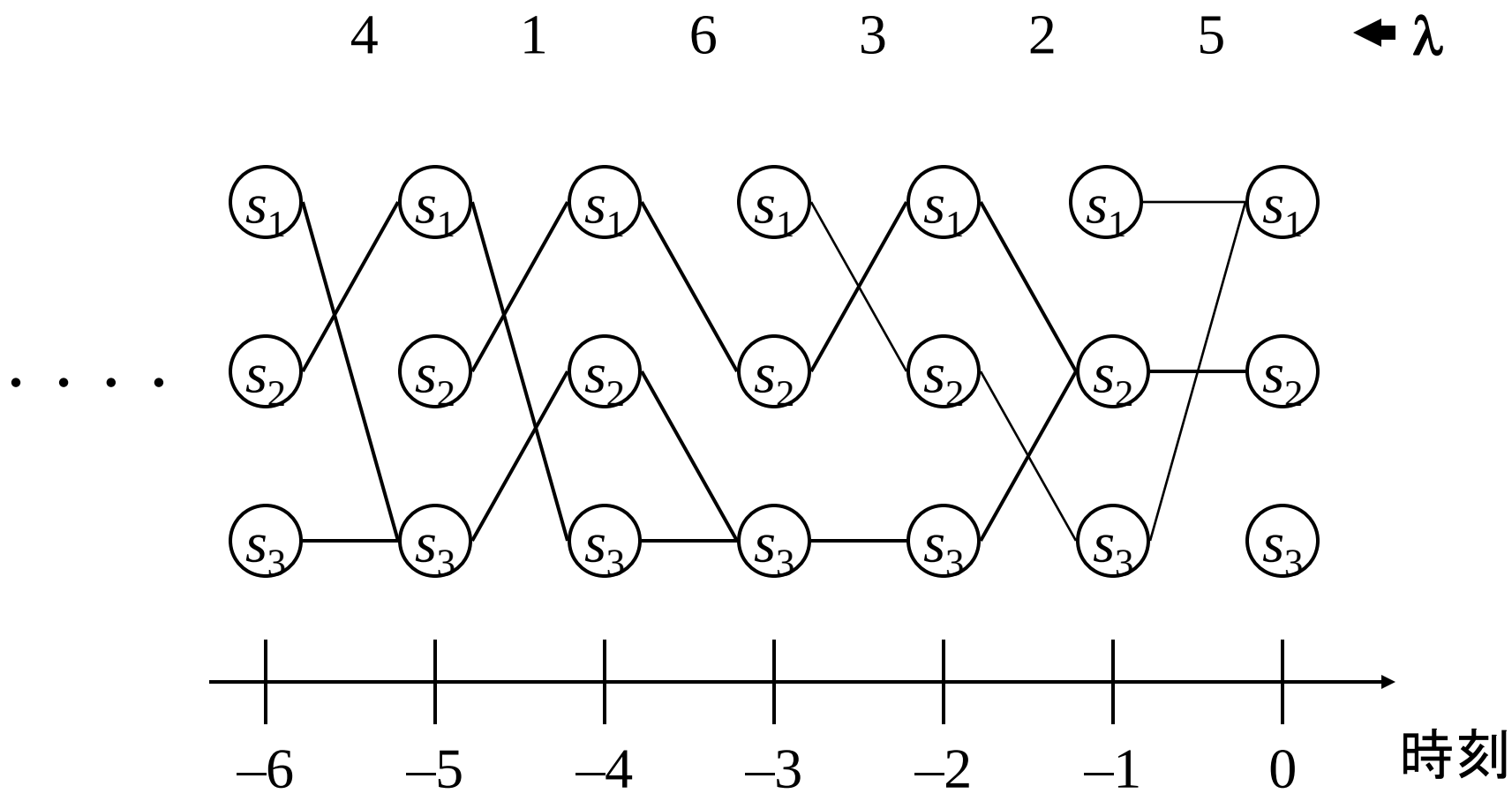
- 仮想的に無限の過去から推移を続けるマルコフ連鎖を考える。
  - 現在（時刻0）の状態は定常分布に厳密に従う。
- 現在（時刻0）の状態は何か？
  - 最近の乱数列から推定する。

    ⇒ 現在の状態を一意に特定する証拠を見つける。

coalescence

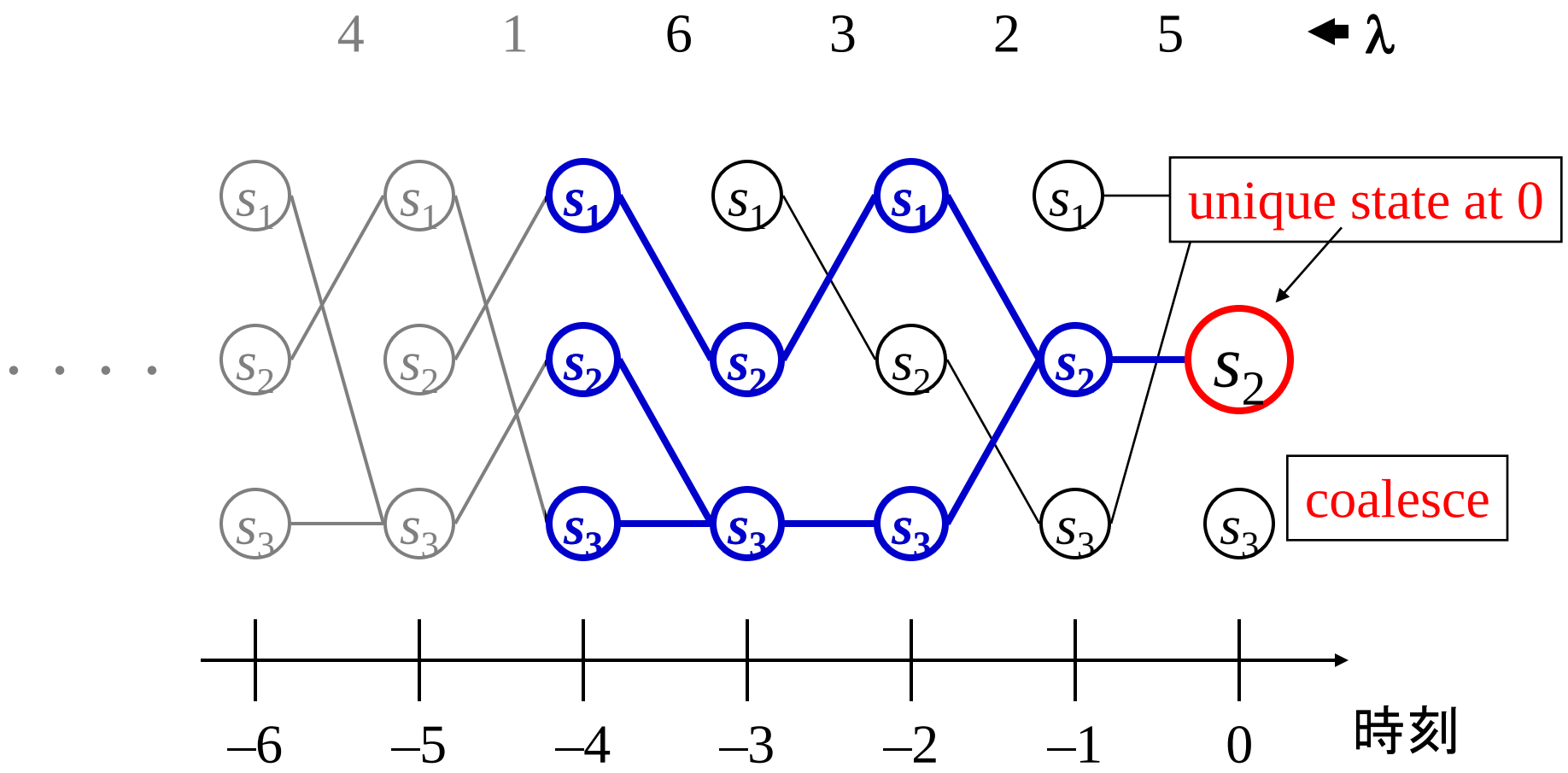
=> 乱数列と対応する推移に依存

# CFTPのアイデア



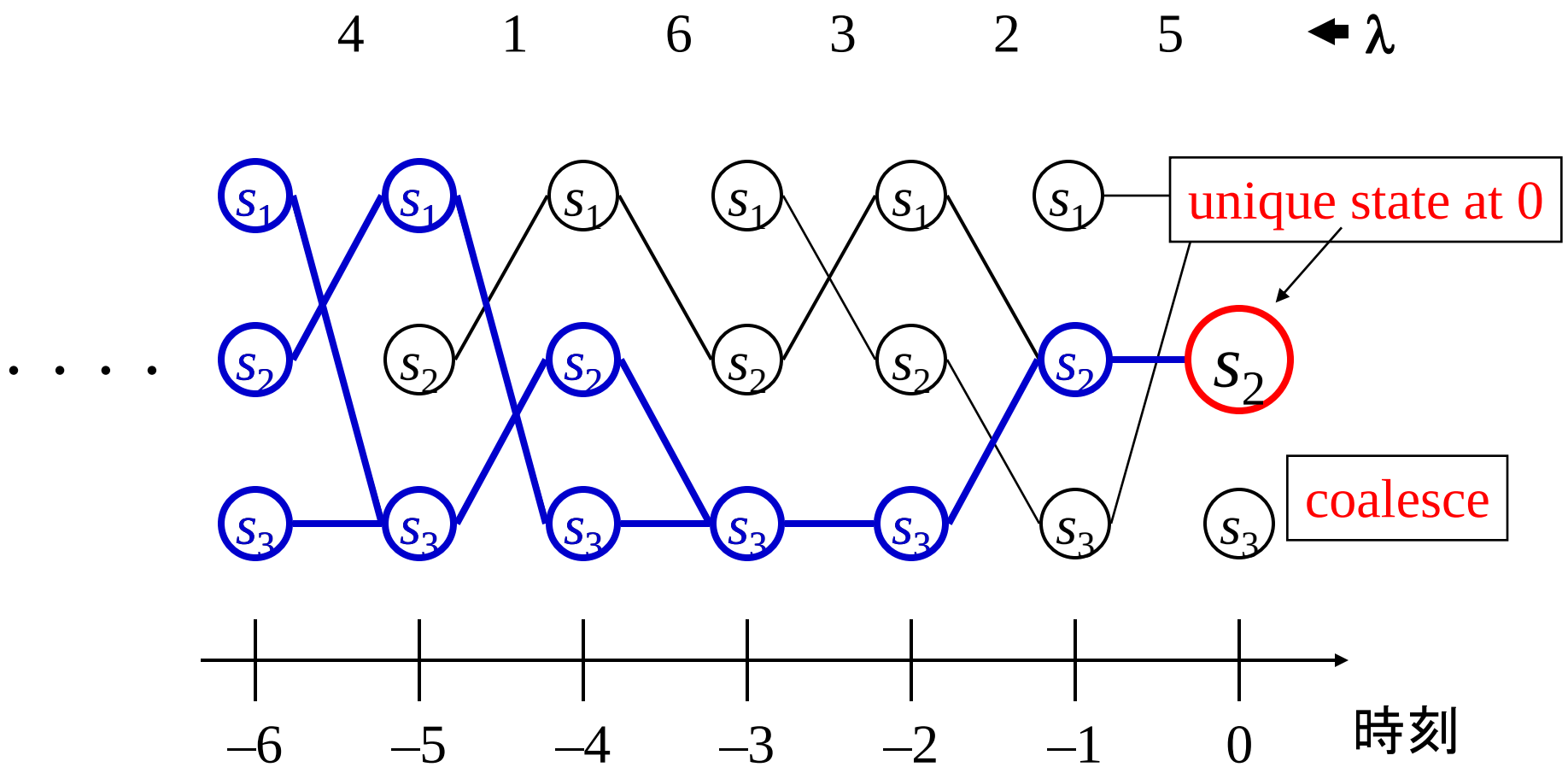
# CFTPのアイデア

初期状態を知る必要がない !!



# CFTPのアイデア

シミュレーションの開始時刻は、時刻0の状態に影響を及ぼさない



# CFTPアルゴリズムと定理 [Propp & Wilson 1996]

マルコフ連鎖 $MC$:
- 有限離散の状態空間 $\Omega$
- 更新関数 $\Phi_s^t(x, \lambda)$ ← 時刻 $s$, 状態 $x$ から 時刻 $t$ まで推移 (乱数列 $\lambda$ )
- エルゴード性を満たす。

## CFTPアルゴリズム

1. set $T = -1$; set $\lambda$: 空列;
2. generate $\lambda[T]$ : 乱数; put $\lambda := ( \lambda[T], \lambda[T+1], \ldots, \lambda[-1] )$;
3. $\Omega$ の全ての状態について、時刻 $T$ から 0 まで $\lambda$ を用いて マルコフ連鎖 $MC$ を推移させる。
   a. if coalesce $(\exists y \in \Omega, \forall x \in \Omega, y = \Phi_T^0(x, \lambda)) \Rightarrow$ return $y$;
   b. otherwise, set $T := T - 1$ ; step 2.に戻る;

## CFTP定理

上のアルゴリズムが停止して値を返す時、

その値はマルコフ連鎖 $M$ の定常分布に厳密に従う確率変数を実現する。

## MCMC完璧サンプリングの設計例

### 単調なマルコフ連鎖

- ✓ Isingモデル
- ✓ Pottsモデル（クラッターモデル） [Propp &Wilson 96]
- ✓ tiling [Wilson 01]
- ✓ 2行分割表 [K & Matsui 06]
- ✓ 離散化Dirichlet分布 [Matsui & K 07]
- ✓ 待ち行列ネットワーク [K & Matsui 08]
- ✓ Q-Ising モデル [Yamamoto, K & Matsui 08]

### それ以外のCFTP

- ✓ 根付き全張木 [Propp &Wilson 98]

### 乱数ベクトルの記憶容量に関する問題

- ➢ Wilson の read onceアルゴリズム [Wilson 00]
- ➢ Fillのinterruptibleアルゴリズム [Fill 98]

# 参考文献


- 来嶋秀治，松井知己，“完璧にサンプリングしよう！”
  オペレーションズ・リサーチ, 50 (2005),
  第一話「遥かなる過去から」, 169--174 (no. 3),
  第二話「天と地の狭間で」, 264--269 (no. 4),
  第三話「終りある未来」， 329--334 (no. 5).
- 玉木久夫, 乱択アルゴリズム, 共立出版, 2008, 3000円


- ✓ 「ORアーカイブ」でダウンロード可能
  http://www.orsj.or.jp/~archive/menu/03_50.html
- ✓ 来嶋のHP
  http://www.kurims.kyoto-u.ac.jp/~kijima/
  - ➢ “資料”からダウンロード可能
  - ➢ 2行分割表の完璧サンプリング法のデモ (JAVAアプレット)

# サンプリングアルゴリズム --最近の動向と今後の課題

解決済み (多項式性)

✓ 高次元凸体上の一様分布/対数凹分布
[Dyer, Frieze, & Kannan 91], [Lovasz & Vempala 07]

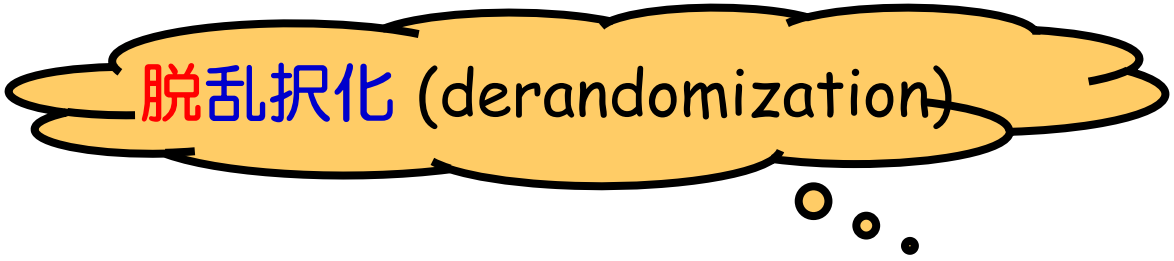
✓ パーマネント/2部グラフの完全マッチング
[Jerrum, Sinclair, & Vigoda 04], [Stefankovic, Vempala, & Vigoda 09]

✓Ising モデル/白黒画像 [Jerrum & Sinclair 93], [Propp &Wilson 96]

未解決問題 (多項式性):
- ✓ **2元分割表** (u.a.r.)
- ✓ **順序イデアル** (u.a.r.)
- ✓ **Tutte多項式**
  - ➢ **マトロイド基** (u.a.r.)
  - ➢ **Pottsモデル**

不可能性
一般の**対数劣モジュラ分布**に対して, [K 09+]
(RP ≠ NPの下) 多項式時間サンプリング法は存在しない.

「乱択アルゴリズムにおいて、乱数に真に求める性質は何か？」



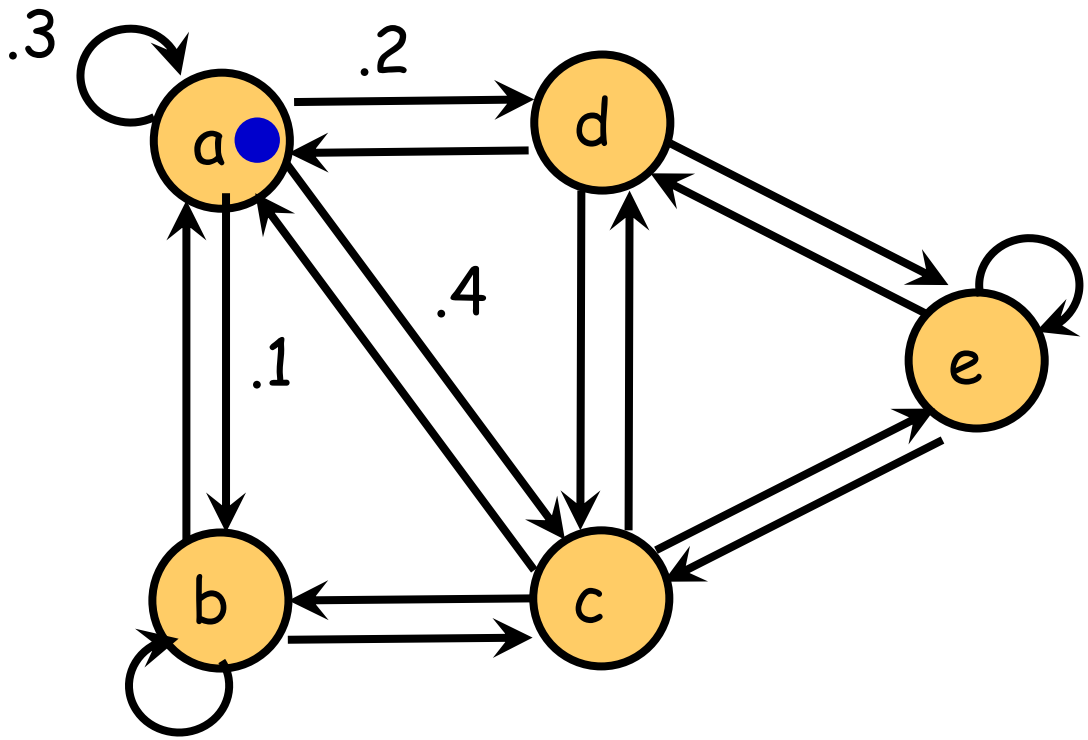

# 3. ランダムウォークの脱乱択化

- ✓ 来嶋, 古賀 健太郎 (FANUC), 牧野 和久 (東大)
- ✓ 梶野 洸 (東大), 来嶋, 牧野 和久 (東大)
- ✓ 白髪 丈晴, 山内 由紀子, 来嶋, 山下 雅史 (九大)

# ランダムウォーク

トークンがグラフ上をランダムウォークする.

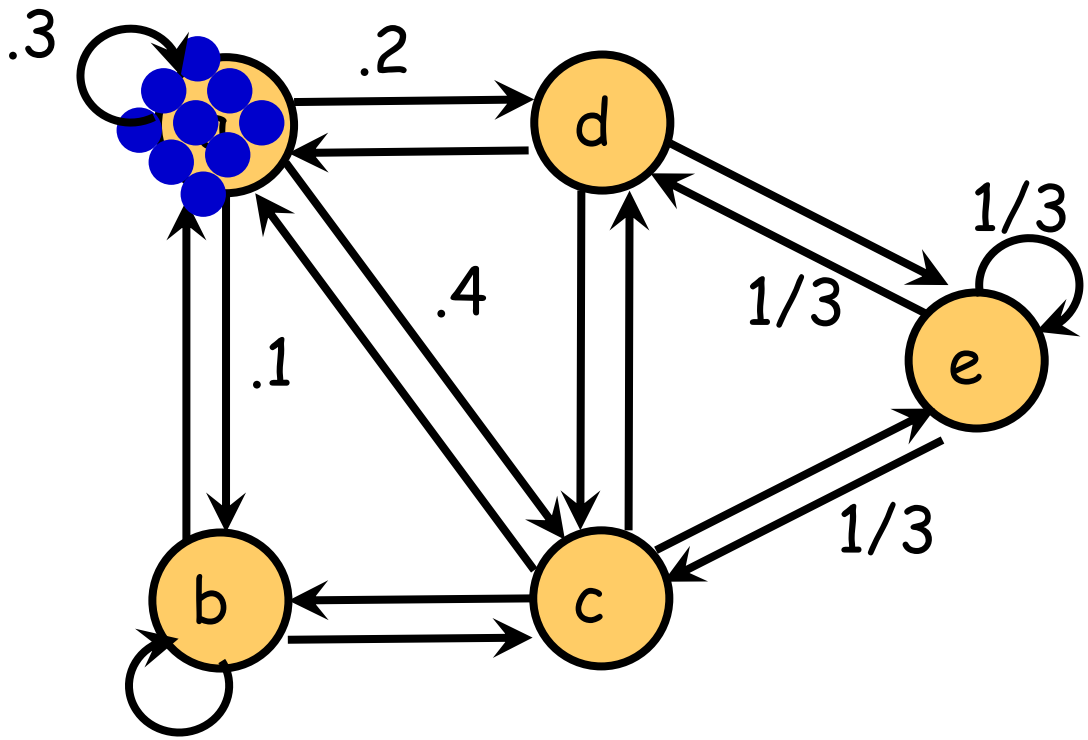
- ✓ $\pi^0$: 初期分布 (トークンは確率$\pi^0_v$で頂点vに居る)
- ✓ P: 推移確率行列 (確率$P_{uv}$で頂点u頂点vに移動)
- ✓ 時刻tの確率分布$\pi^t := \pi^0 P^t$ (頂点vに居る確率 $(\pi^t)_v$ )



# ランダムウォーク (複数トークンによる分布の近似)

N個のトークンがグラフ上を独立にランダムウォークする.

- ✓ $\mu^0$: 初期配置 ($\pi^0 \approx \mu^0/N$)
- ✓ P: 推移確率行列 (確率$P_{uv}$で頂点u頂点vに移動)
- ✓ 時刻tの期待配置$\mu^t := \mu^0 P^t$ ($\pi^t \approx \mu^t/N$)



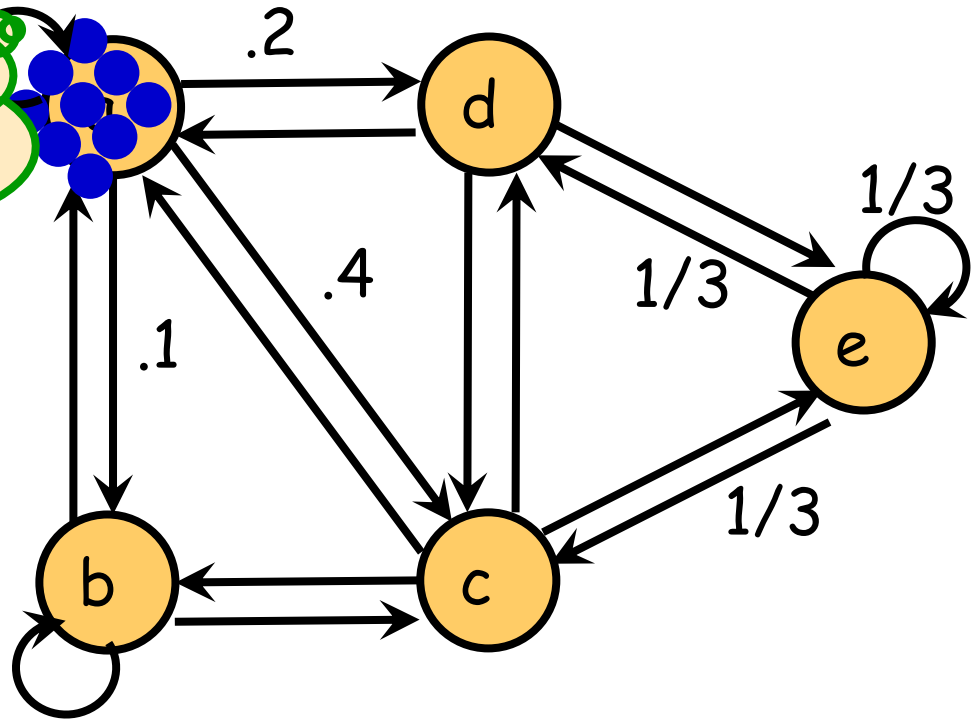
# ランダムウォーク (複数トークンによる分布の近似)

N個のトークンがグラフ上を独立にランダムウォークする.

- ✓ $\mu^0$: 初期配置 ($\pi^0 \approx \mu^0/N$)
- ✓ P: 推移確率行列 (確率$P_{uv}$で頂点u頂点vに移動)
- ✓ 時刻tの期待配置$\mu^t := \mu^0 P^t$ ($\pi^t \approx \mu^t/N$)

(Nが非常に大きければ, )
トークンを約2:4:1:3の割合で
隣接点にばらまいている.

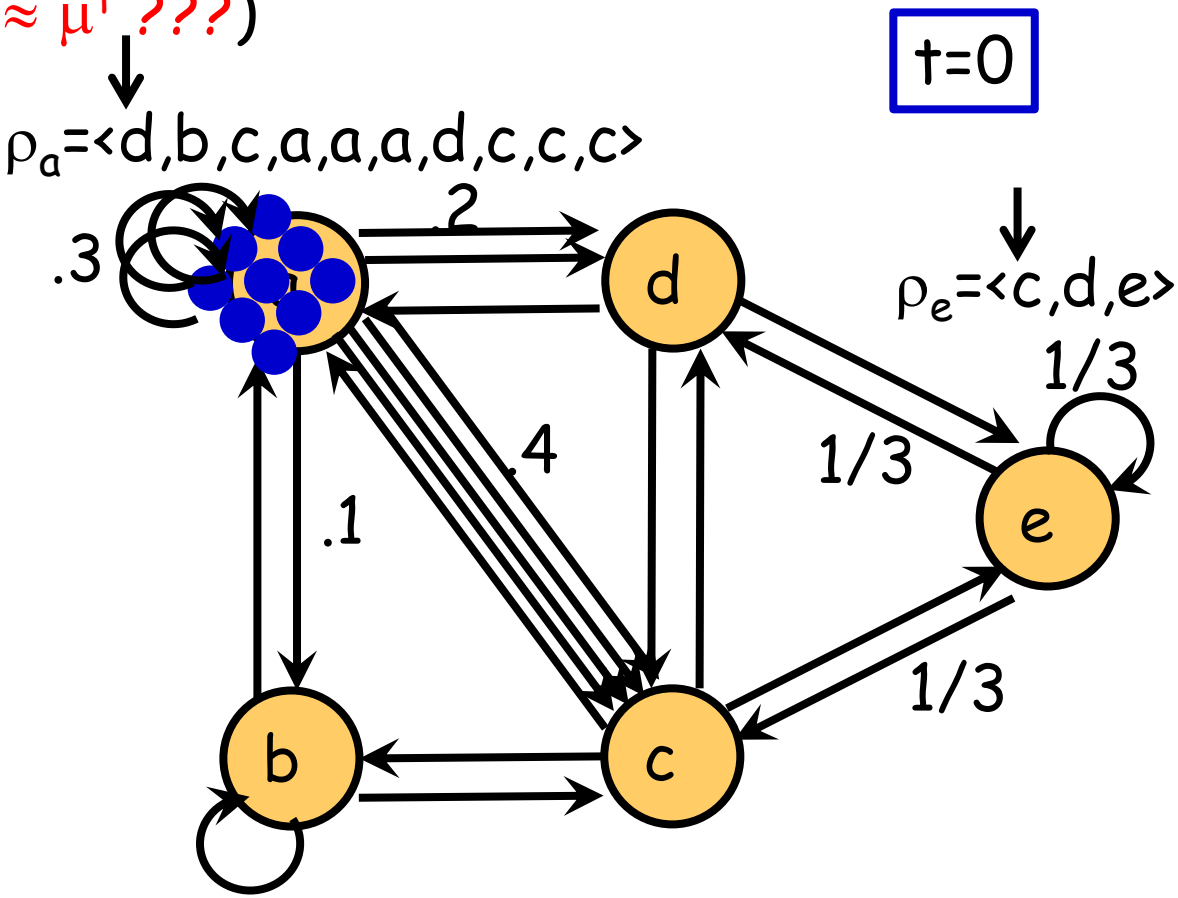
決定的過程でも大差ない?



# deterministic RW (Propp機械; rotor-router)

N個のトークンがグラフ上を移動する.

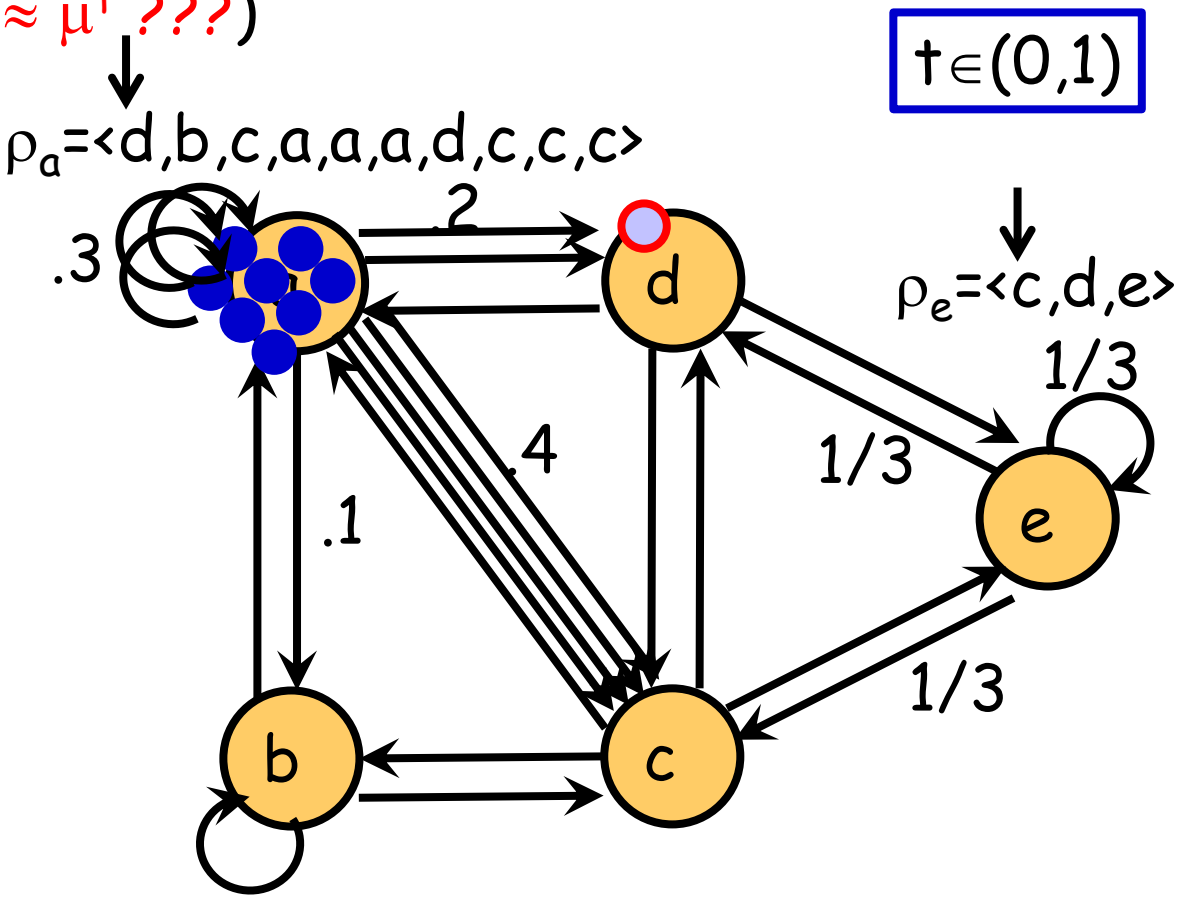
- ✓ $\chi^0$: 初期配置 ($\chi^0 = \mu^0$)
- ✓ $\rho$: "**rotor router**" (比率$P_{uv}$で頂点u頂点vに"順次"移動)
- ✓ 時刻tの配置$\chi^t$ ($\chi^t \approx \mu^t$ ???)



# deterministic RW (Propp機械; rotor-router)

N個のトークンがグラフ上を移動する.

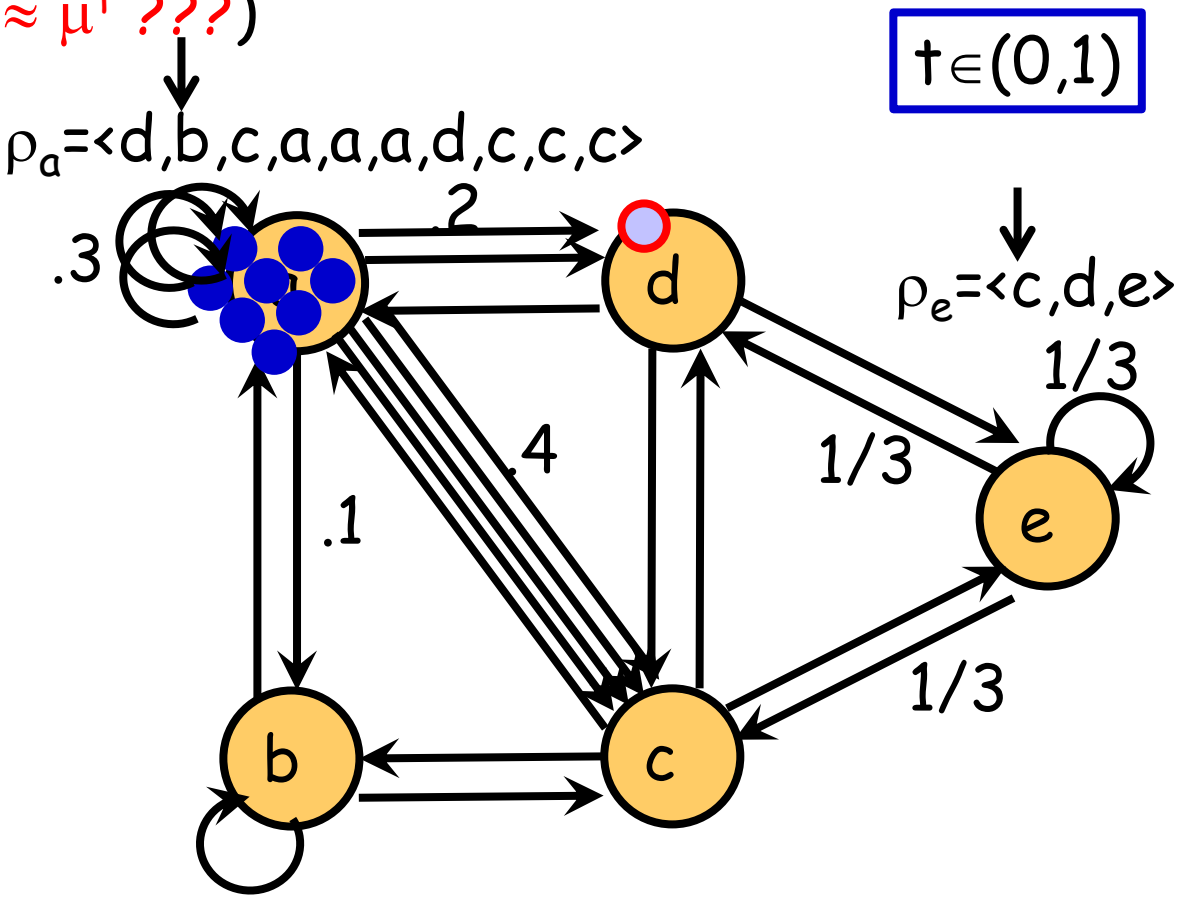
- ✓ $\chi^0$: 初期配置 ($\chi^0 = \mu^0$)
- ✓ $\rho$: "**rotor router**" (比率$P_{uv}$で頂点u頂点vに"順次"移動)
- ✓ 時刻tの配置$\chi^t$ ($\chi^t \approx \mu^t$ ???)



# deterministic RW (Propp機械; rotor-router)

N個のトークンがグラフ上を移動する.

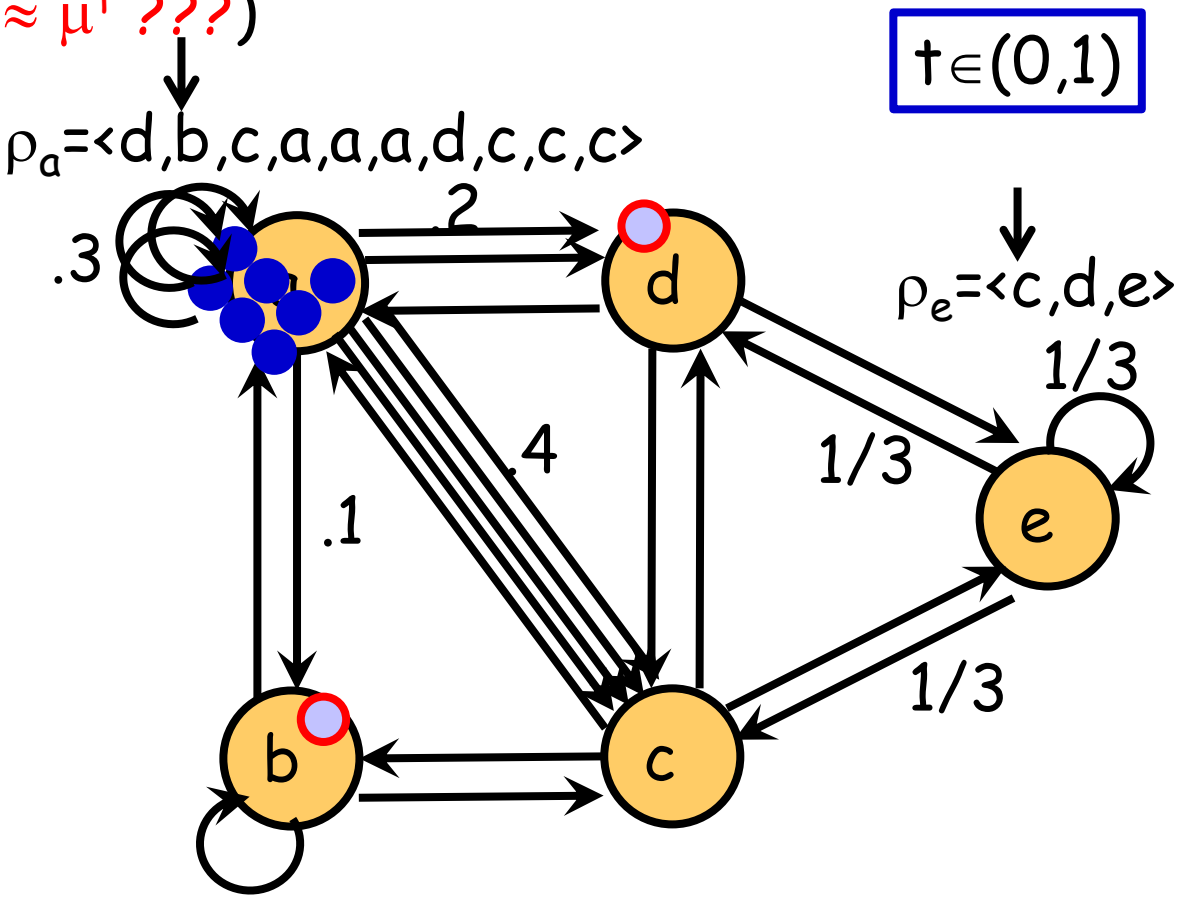
- ✓ $\chi^0$: 初期配置 ($\chi^0 = \mu^0$)
- ✓ $\rho$: "**rotor router**" (比率$P_{uv}$で頂点u頂点vに"順次"移動)
- ✓ 時刻tの配置$\chi^t$ ($\chi^t \approx \mu^t$ ???)



# deterministic RW (Propp機械; rotor-router)

N個のトークンがグラフ上を移動する.

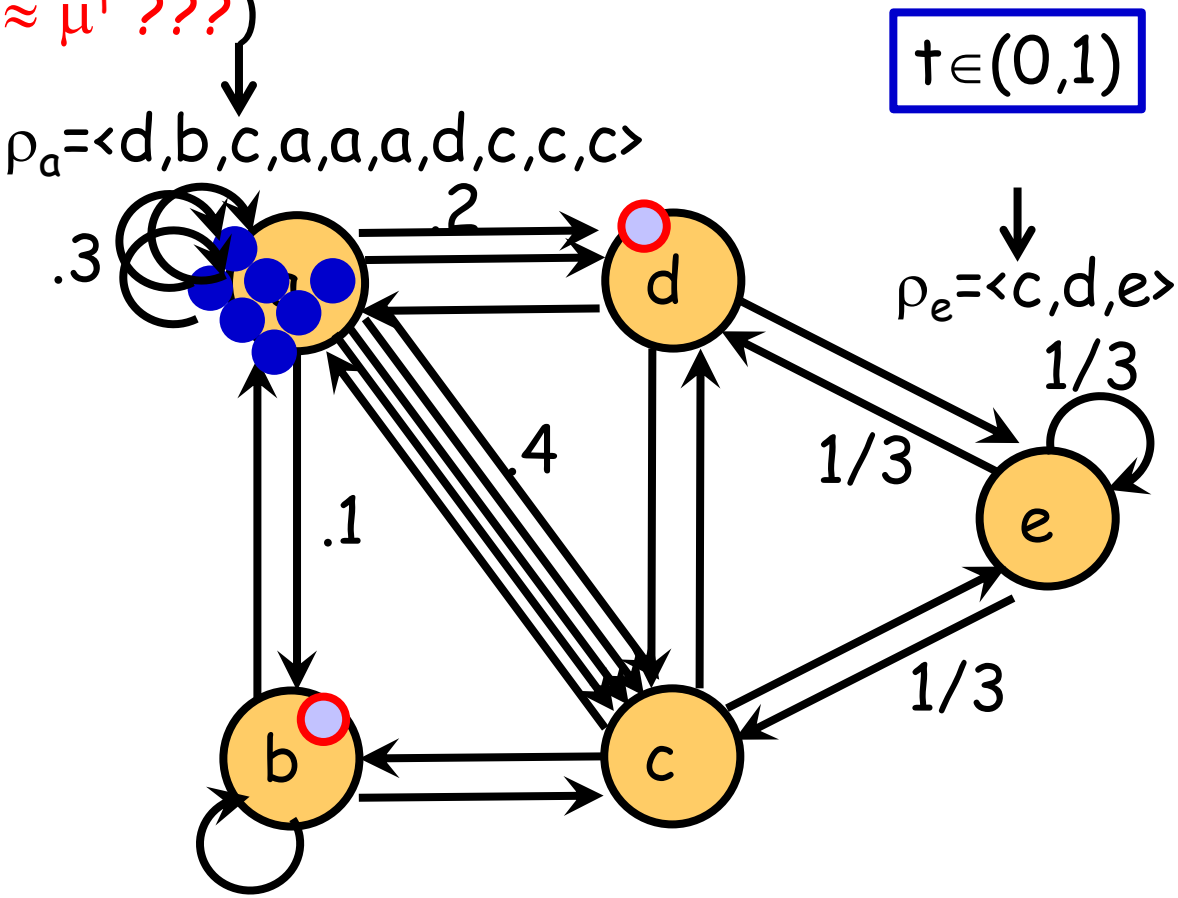
- ✓ $\chi^0$: 初期配置 ($\chi^0 = \mu^0$)
- ✓ $\rho$: "**rotor router**" (比率$P_{uv}$で頂点u頂点vに"順次"移動)
- ✓ 時刻tの配置$\chi^t$ ($\chi^t \approx \mu^t$ ???)



# deterministic RW (Propp機械; rotor-router)

N個のトークンがグラフ上を移動する.

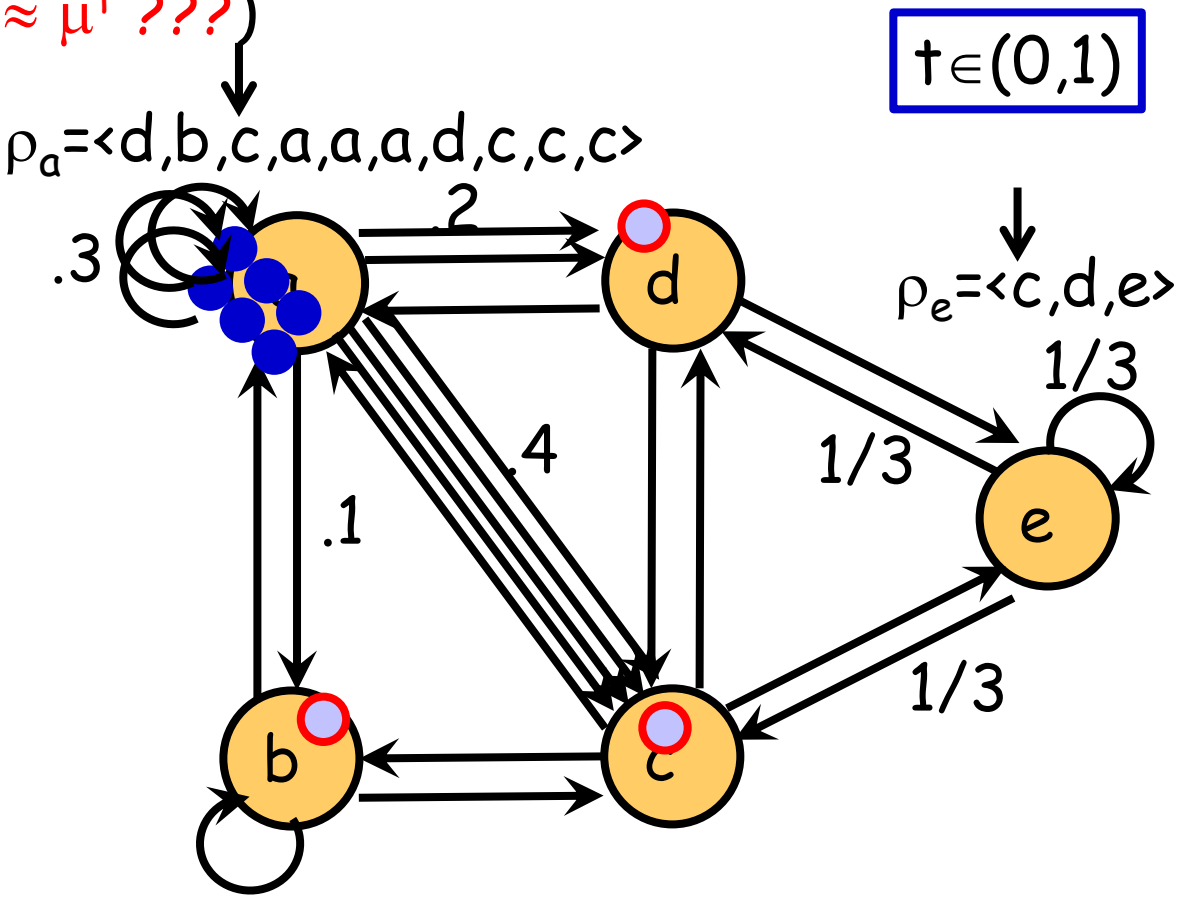
- ✓ $\chi^0$: 初期配置 ($\chi^0 = \mu^0$)
- ✓ $\rho$: "**rotor router**" (比率$P_{uv}$で頂点u頂点vに"順次"移動)
- ✓ 時刻tの配置$\chi^t$ ($\chi^t \approx \mu^t$ ???)



# deterministic RW (Propp機械; rotor-router)

N個のトークンがグラフ上を移動する.

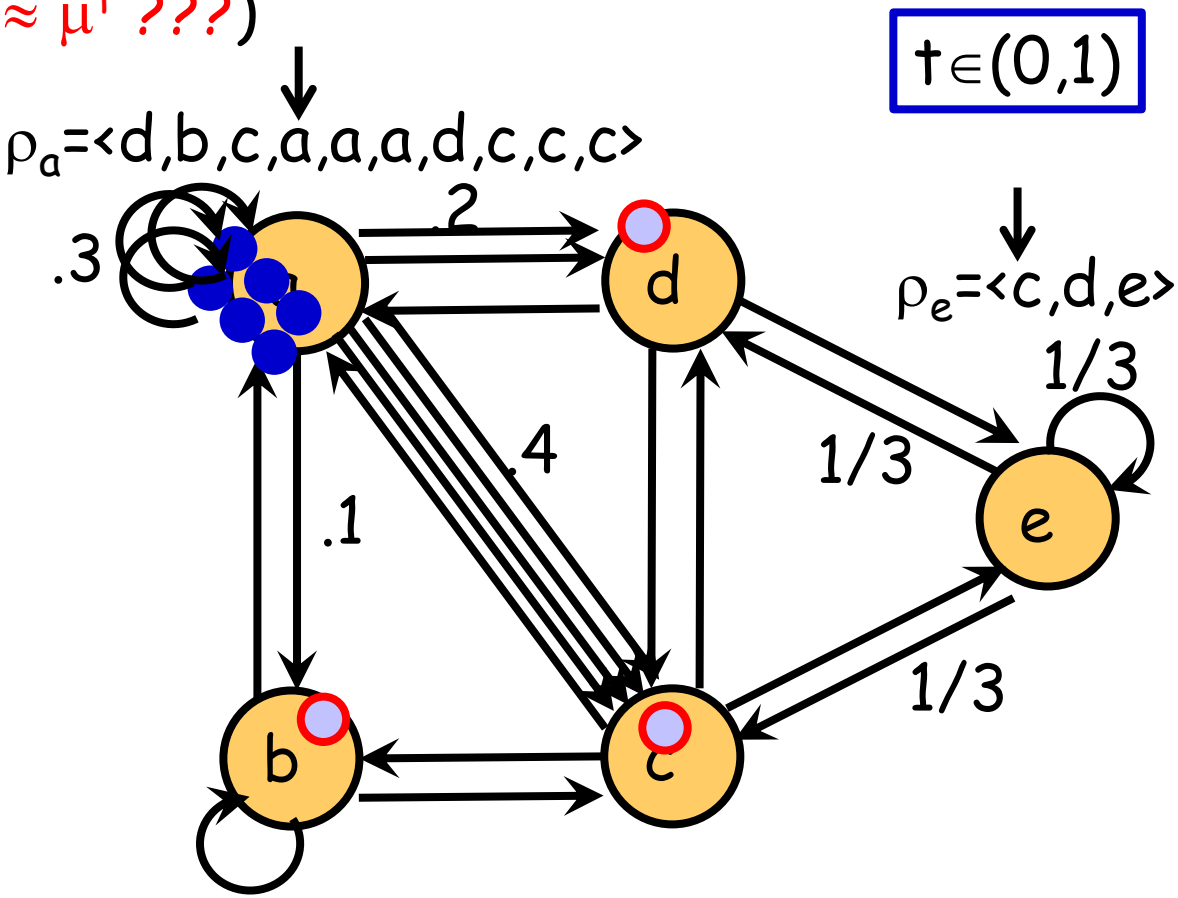
- ✓ $\chi^0$: 初期配置 ($\chi^0 = \mu^0$)
- ✓ $\rho$: "**rotor router**" (比率$P_{uv}$で頂点u頂点vに"順次"移動)
- ✓ 時刻tの配置$\chi^t$ ($\chi^t \approx \mu^t$ ???)



# deterministic RW (Propp機械; rotor-router)

N個のトークンがグラフ上を移動する.

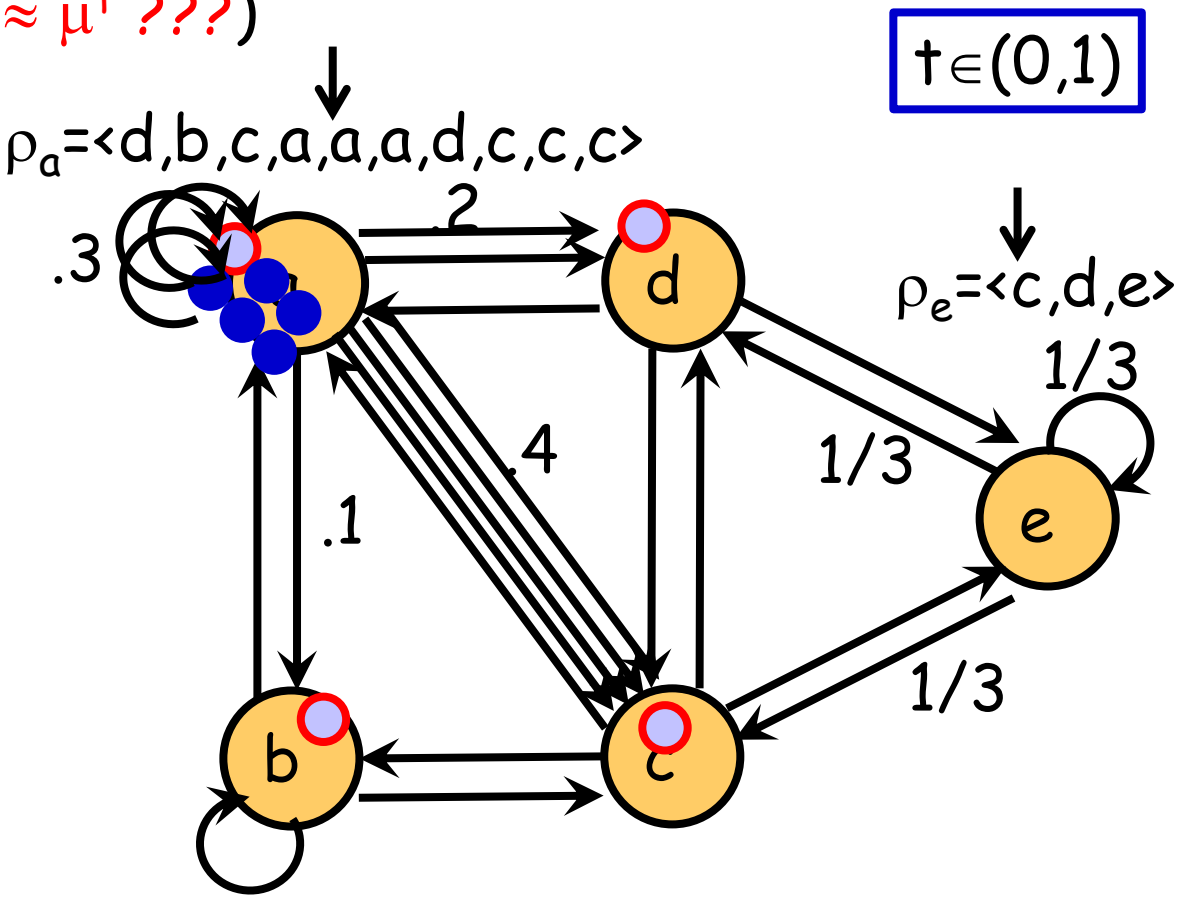
- ✓ $\chi^0$: 初期配置 ($\chi^0 = \mu^0$)
- ✓ $\rho$: "**rotor router**" (比率$P_{uv}$で頂点u頂点vに"順次"移動)
- ✓ 時刻tの配置$\chi^t$ ($\chi^t \approx \mu^t$ ???)



# deterministic RW (Propp機械; rotor-router)

N個のトークンがグラフ上を移動する.

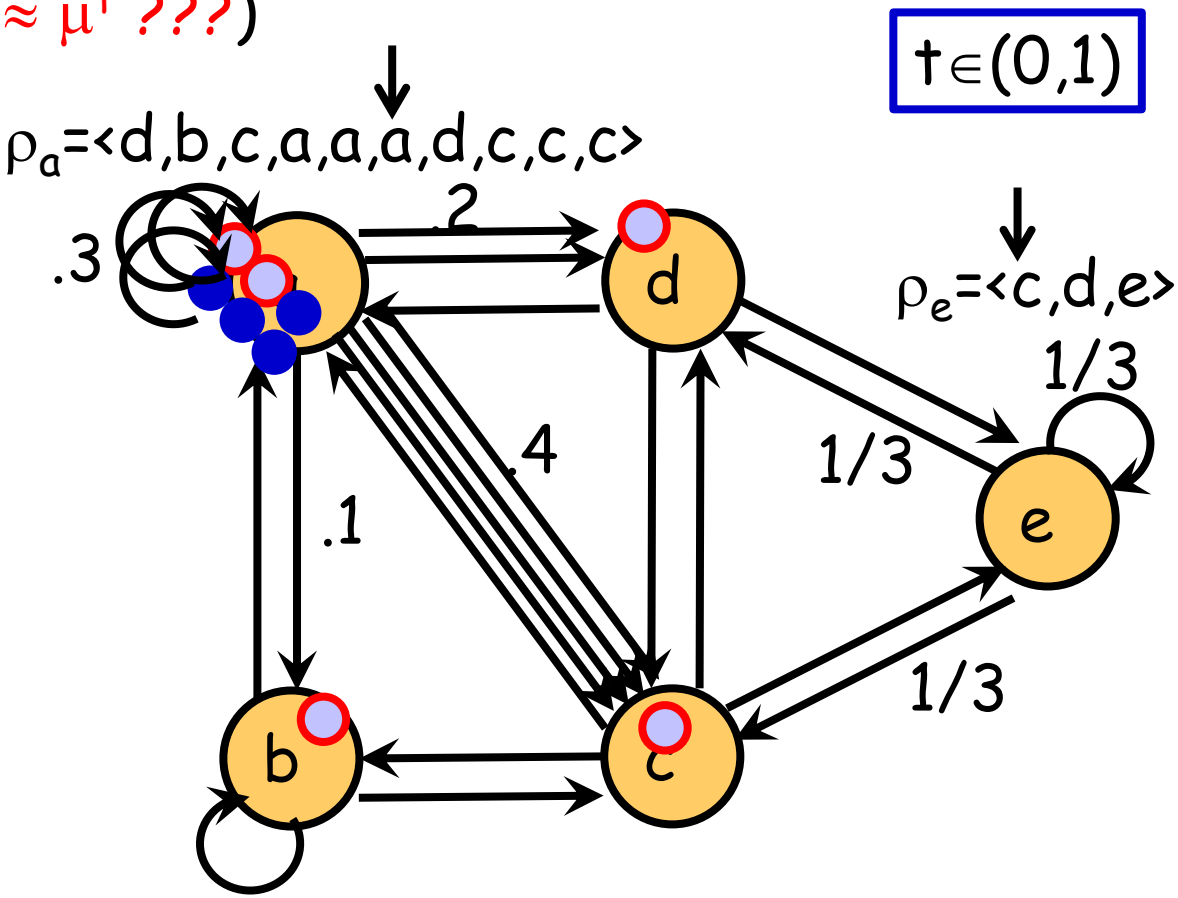
- ✓ $\chi^0$: 初期配置 ($\chi^0 = \mu^0$)
- ✓ $\rho$: "**rotor router**" (比率$P_{uv}$で頂点u頂点vに"順次"移動)
- ✓ 時刻tの配置$\chi^t$ ($\chi^t \approx \mu^t$ ???)



# deterministic RW (Propp機械; rotor-router)

N個のトークンがグラフ上を移動する.

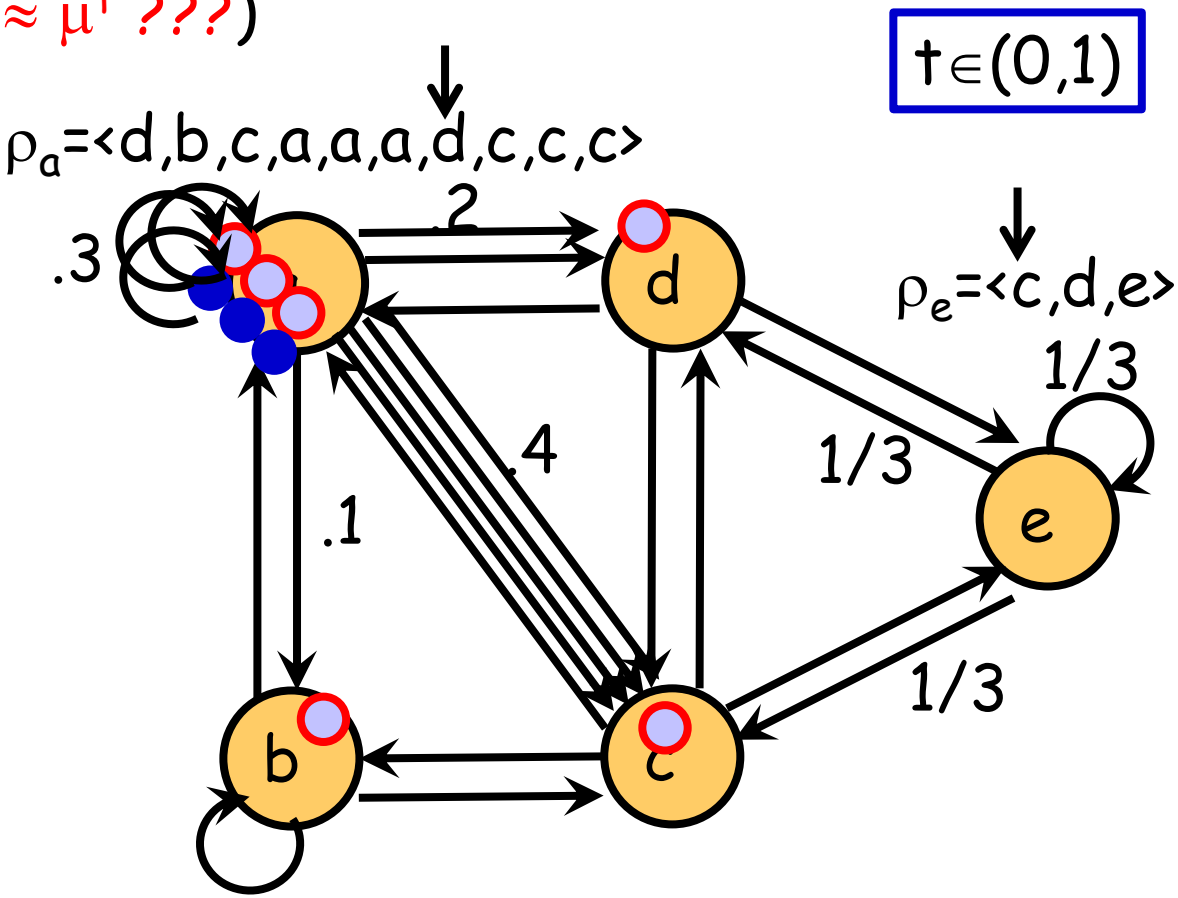
- ✓ $\chi^0$: 初期配置 ($\chi^0 = \mu^0$)
- ✓ $\rho$: "**rotor router**" (比率$P_{uv}$で頂点u頂点vに"順次"移動)
- ✓ 時刻tの配置$\chi^t$ ($\chi^t \approx \mu^t$ ???)



# deterministic RW (Propp機械; rotor-router)

N個のトークンがグラフ上を移動する.

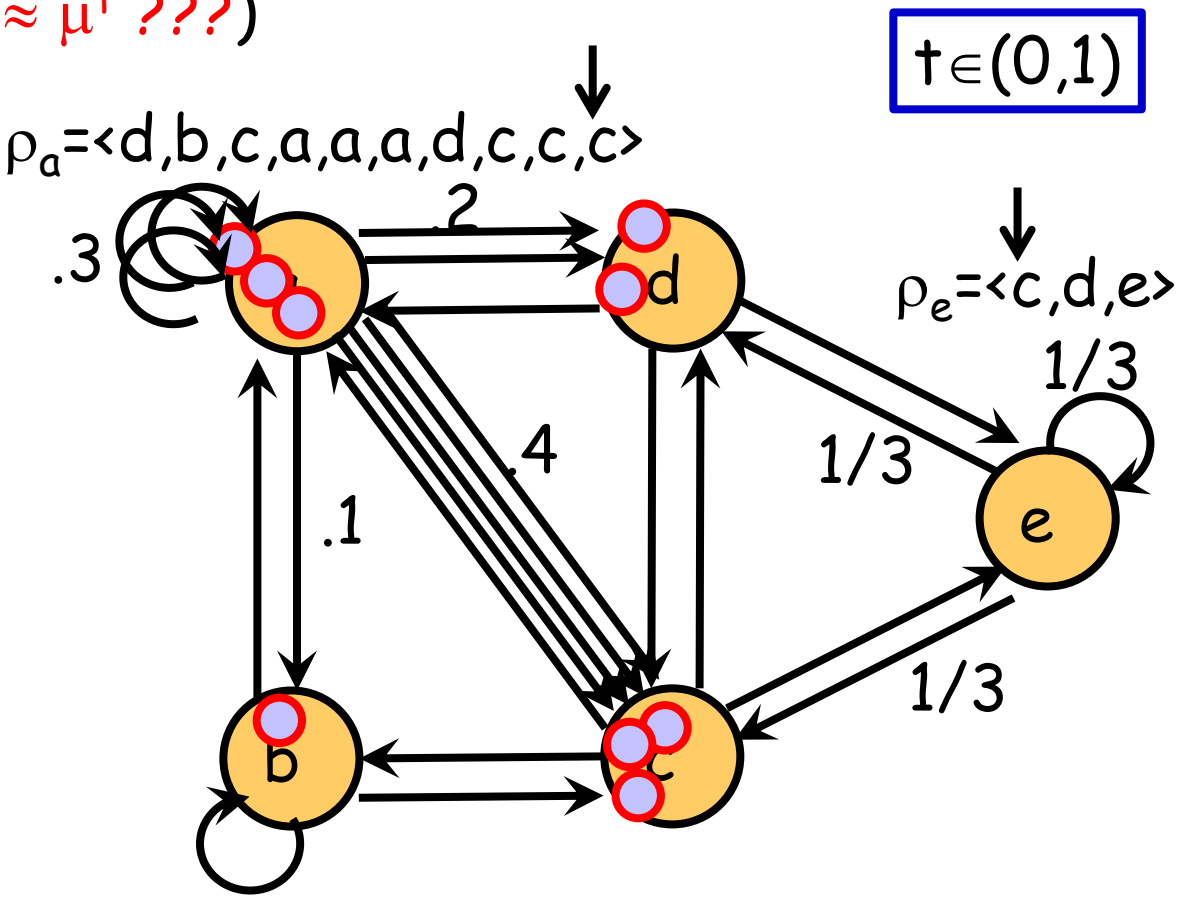
- ✓ $\chi^0$: 初期配置 ($\chi^0 = \mu^0$)
- ✓ $\rho$: "**rotor router**" (比率$P_{uv}$で頂点u頂点vに"順次"移動)
- ✓ 時刻tの配置$\chi^t$ ($\chi^t \approx \mu^t$ ???)



# deterministic RW (Propp機械; rotor-router)

N個のトークンがグラフ上を移動する.

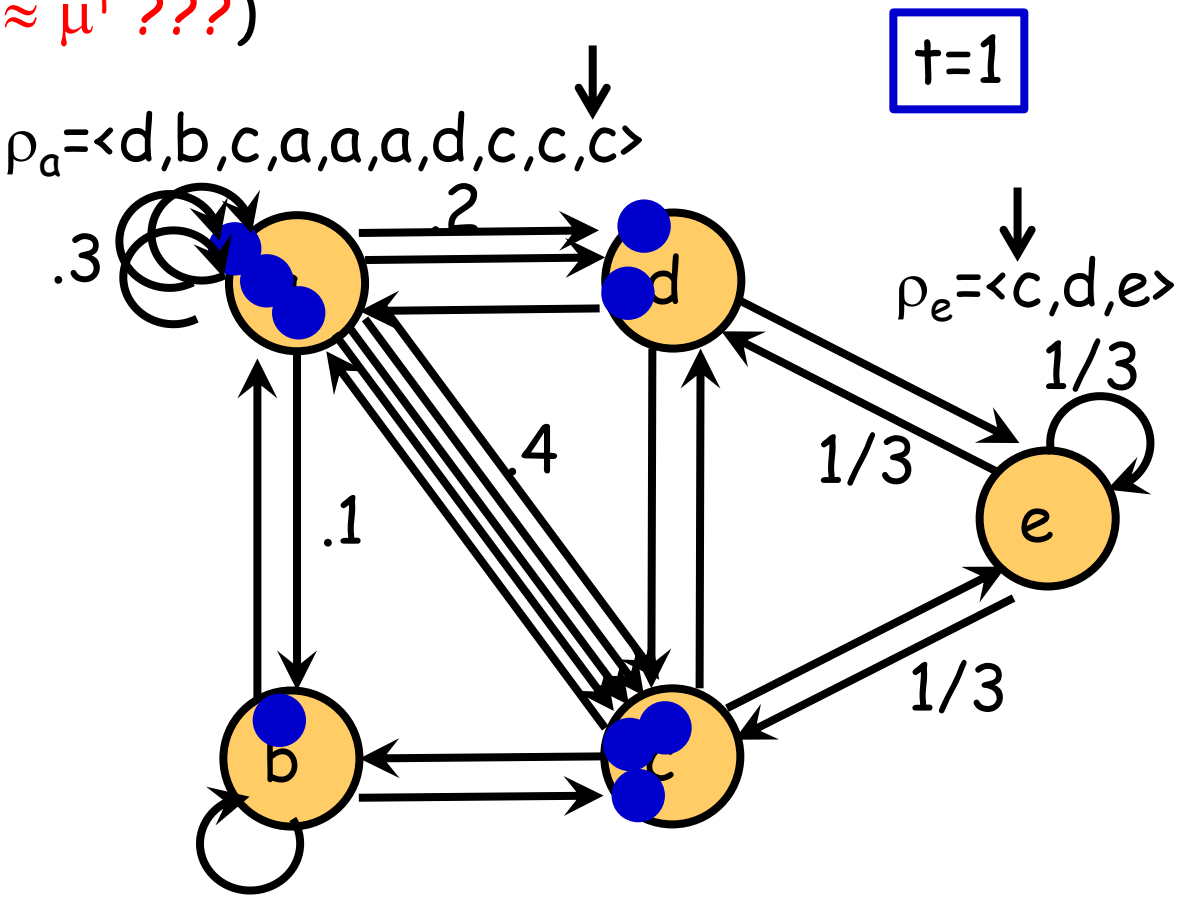
- ✓ $\chi^0$: 初期配置 ($\chi^0 = \mu^0$)
- ✓ $\rho$: "**rotor router**" (比率$P_{uv}$で頂点u頂点vに"順次"移動)
- ✓ 時刻tの配置$\chi^t$ ($\chi^t \approx \mu^t$ ???)



# deterministic RW (Propp機械; rotor-router)

N個のトークンがグラフ上を移動する.

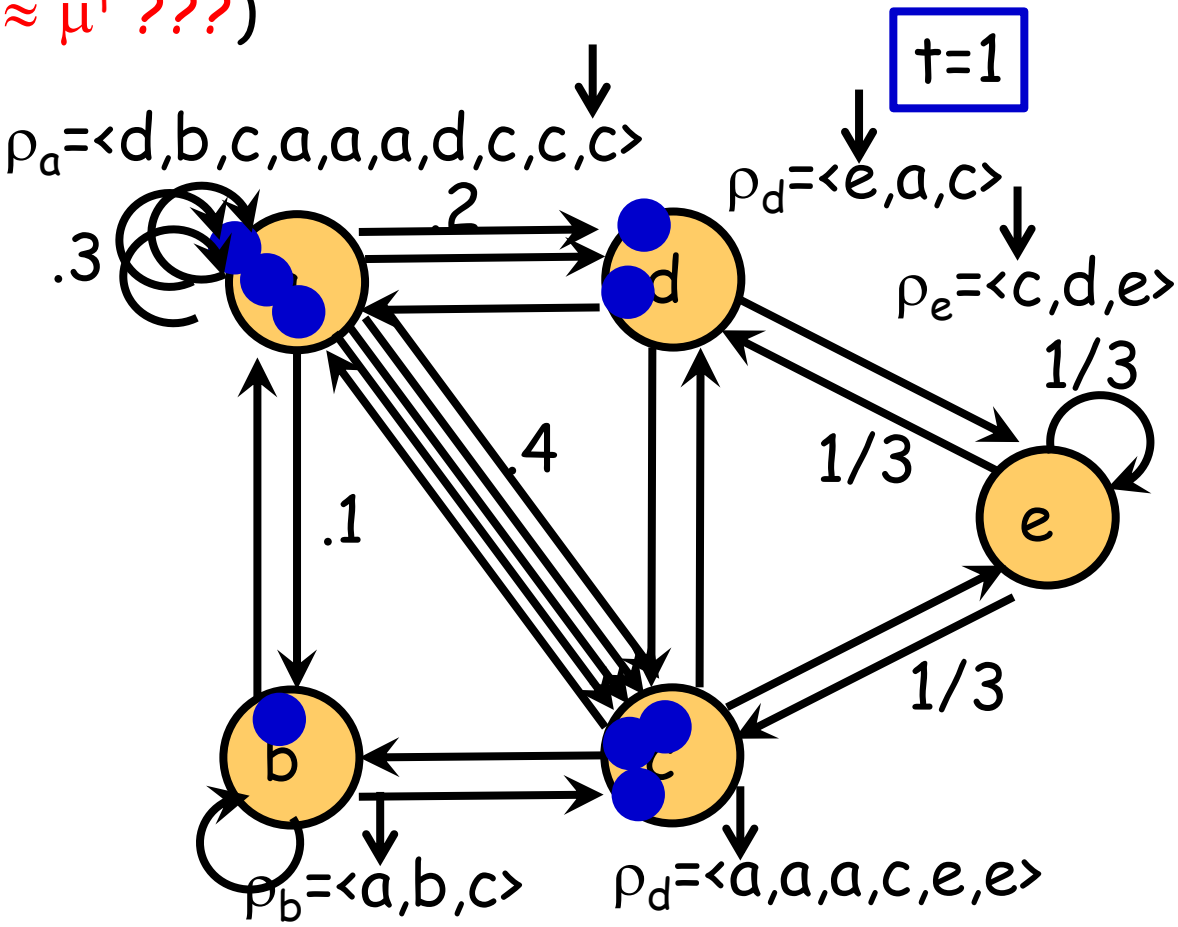
- ✓ $\chi^0$: 初期配置 ($\chi^0 = \mu^0$)
- ✓ $\rho$: "**rotor router**" (比率$P_{uv}$で頂点u頂点vに"順次"移動)
- ✓ 時刻tの配置$\chi^t$ ($\chi^t \approx \mu^t$ ???)



# deterministic RW (Propp機械; rotor-router)

N個のトークンがグラフ上を移動する.

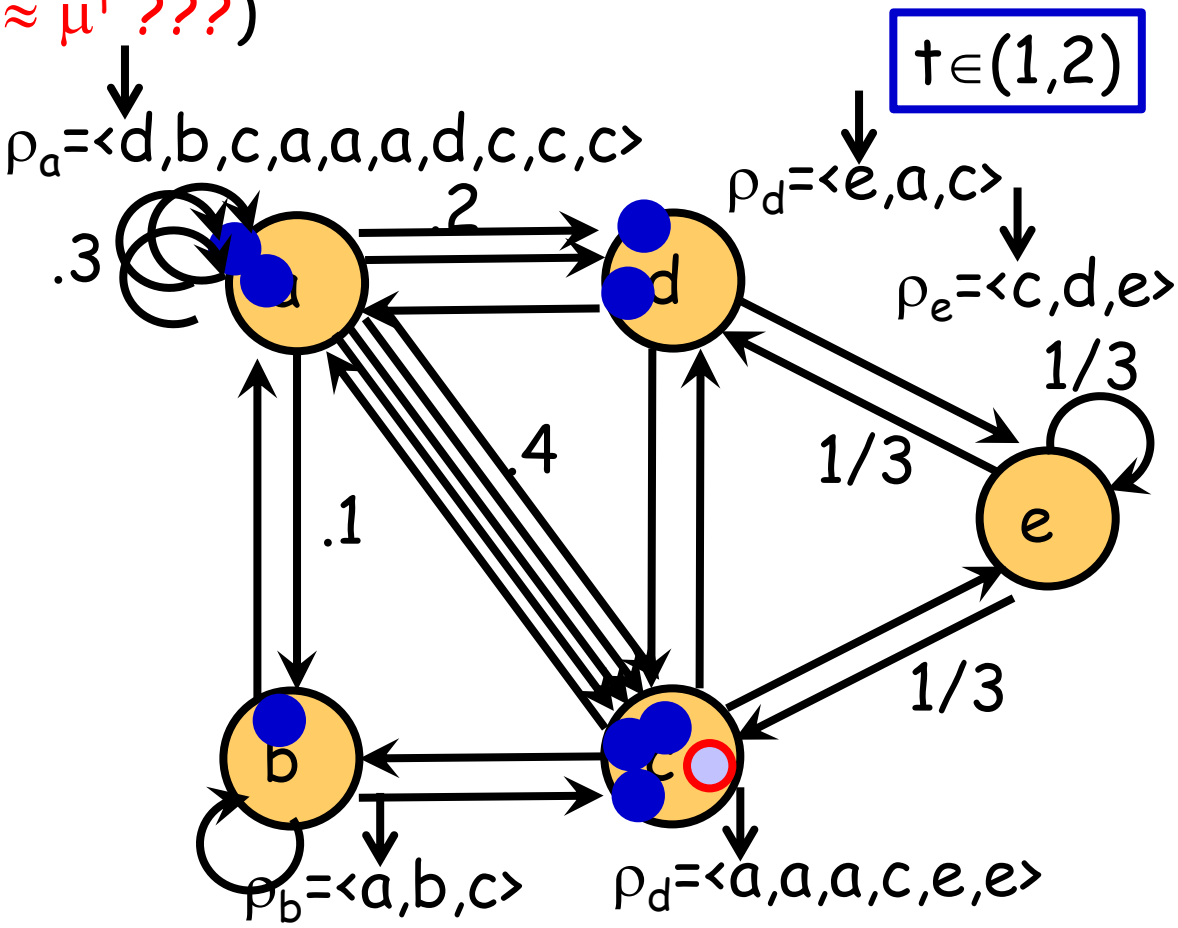
- ✓ $\chi^0$: 初期配置 ($\chi^0 = \mu^0$)
- ✓ $\rho$: "**rotor router**" (比率$P_{uv}$で頂点u頂点vに"順次"移動)
- ✓ 時刻tの配置$\chi^t$ ($\chi^t \approx \mu^t$ ???)



# deterministic RW (Propp機械; rotor-router)

N個のトークンがグラフ上を移動する.

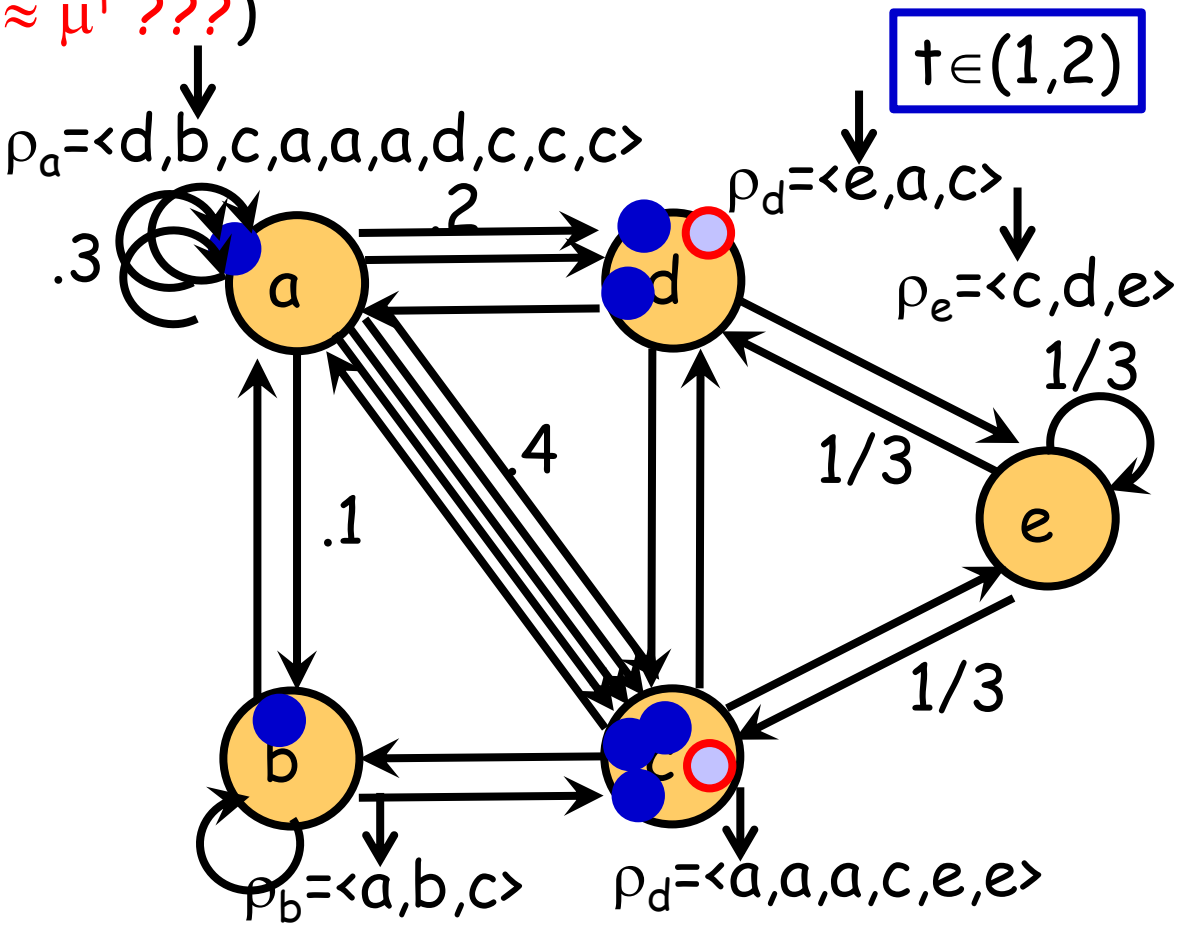
- ✓ $\chi^0$: 初期配置 ($\chi^0 = \mu^0$)
- ✓ $\rho$: "**rotor router**" (比率$P_{uv}$で頂点u頂点vに"順次"移動)
- ✓ 時刻tの配置$\chi^t$ ($\chi^t \approx \mu^t$ ???)



# deterministic RW (Propp機械; rotor-router)

N個のトークンがグラフ上を移動する.

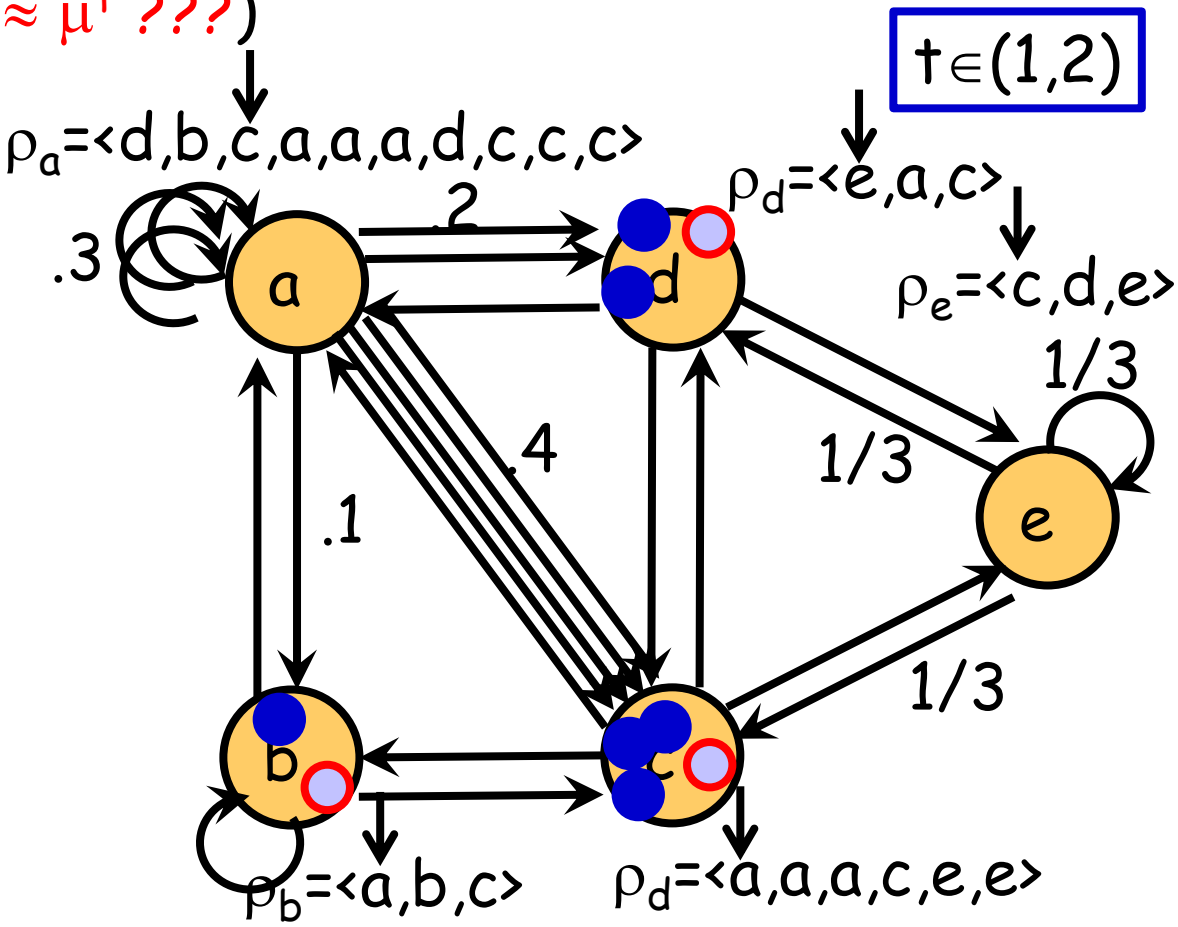
- ✓ $\chi^0$: 初期配置 ($\chi^0 = \mu^0$)
- ✓ $\rho$: "**rotor router**" (比率$P_{uv}$で頂点u頂点vに"順次"移動)
- ✓ 時刻tの配置$\chi^t$ ($\chi^t \approx \mu^t$ ???)



# deterministic RW (Propp機械; rotor-router)

N個のトークンがグラフ上を移動する.

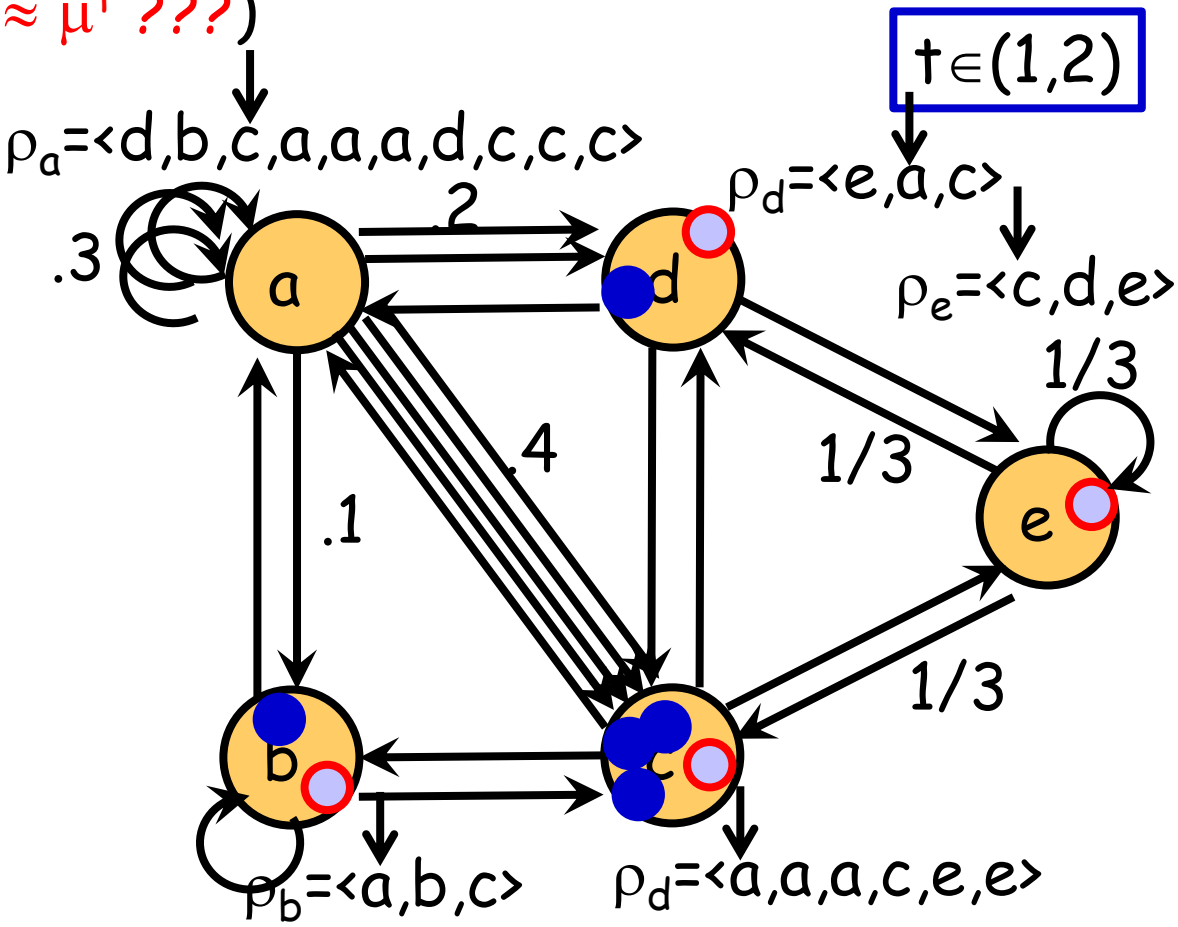
- ✓ $\chi^0$: 初期配置 ($\chi^0 = \mu^0$)
- ✓ $\rho$: "**rotor router**" (比率$P_{uv}$で頂点u頂点vに"順次"移動)
- ✓ 時刻tの配置$\chi^t$ ($\chi^t \approx \mu^t$ ???)



# deterministic RW (Propp機械; rotor-router)

N個のトークンがグラフ上を移動する.

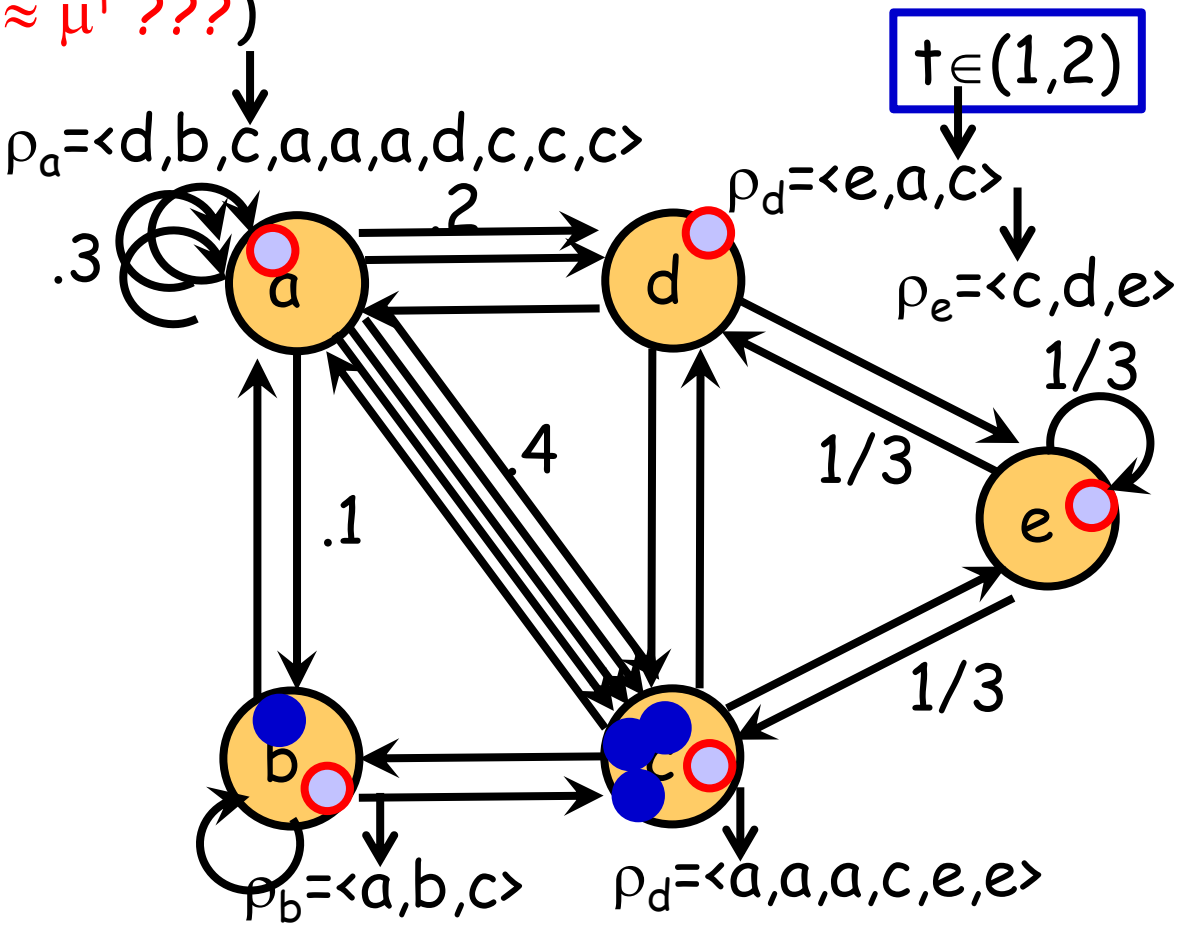
- ✓ $\chi^0$: 初期配置 ($\chi^0 = \mu^0$)
- ✓ $\rho$: "**rotor router**" (比率$P_{uv}$で頂点u頂点vに"順次"移動)
- ✓ 時刻tの配置$\chi^t$ ($\chi^t \approx \mu^t$ ???)



# deterministic RW (Propp機械; rotor-router)

N個のトークンがグラフ上を移動する.

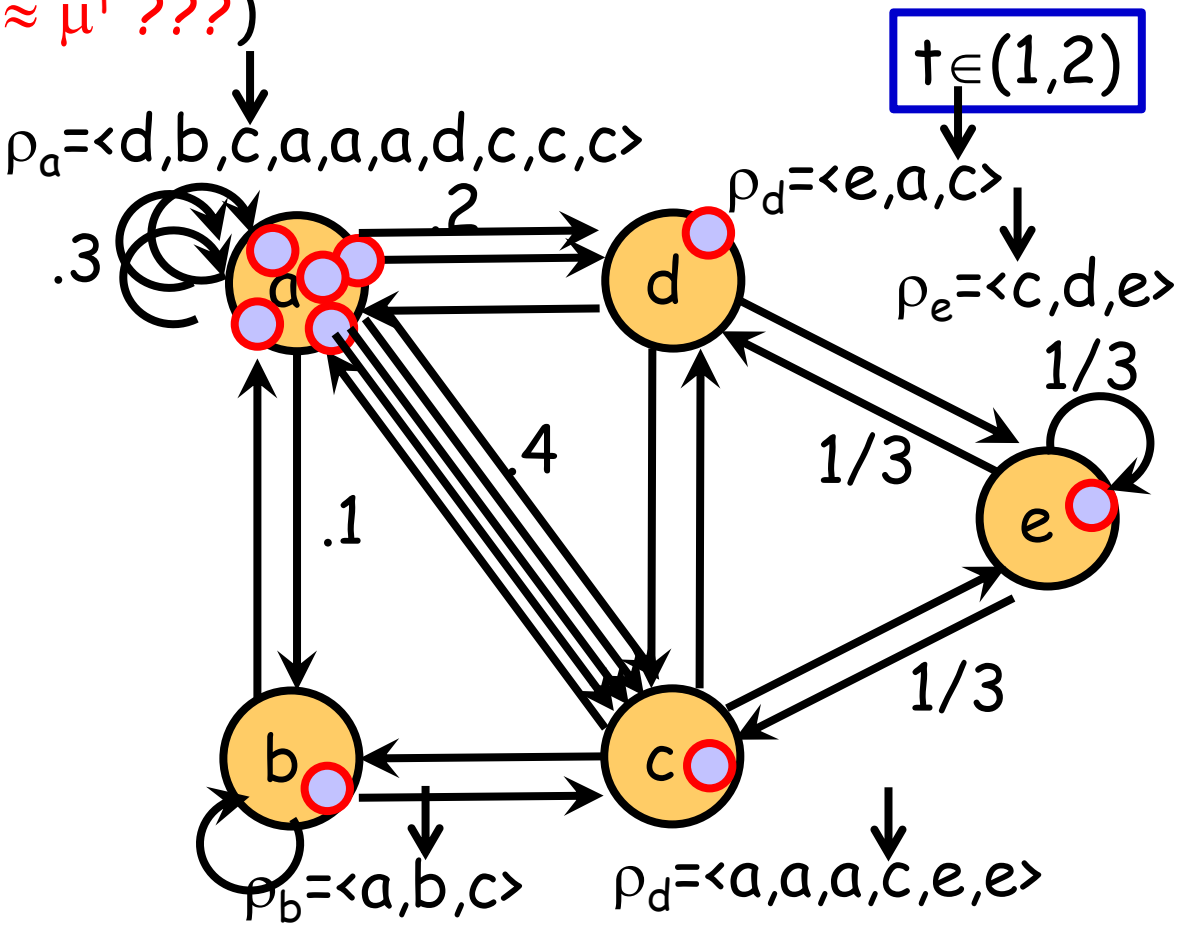
- ✓ $\chi^0$: 初期配置 ($\chi^0 = \mu^0$)
- ✓ $\rho$: "**rotor router**" (比率$P_{uv}$で頂点u頂点vに"順次"移動)
- ✓ 時刻tの配置$\chi^t$ ($\chi^t \approx \mu^t$ ???)



# deterministic RW (Propp機械; rotor-router)

N個のトークンがグラフ上を移動する.

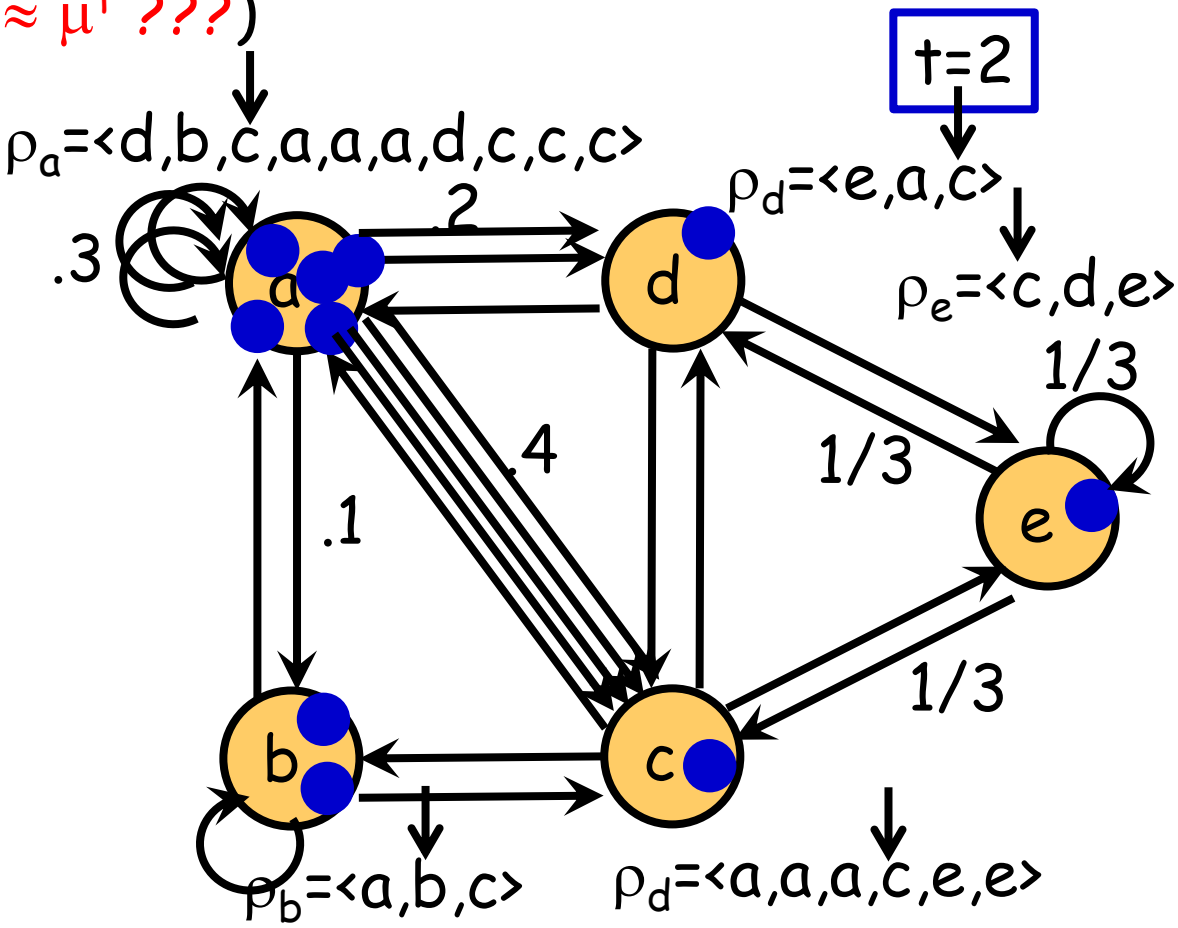
- ✓ $\chi^0$: 初期配置 ($\chi^0 = \mu^0$)
- ✓ $\rho$: "**rotor router**" (比率$P_{uv}$で頂点u頂点vに"順次"移動)
- ✓ 時刻tの配置$\chi^t$ ($\chi^t \approx \mu^t$ ???)



# deterministic RW (Propp機械; rotor-router)

N個のトークンがグラフ上を移動する.

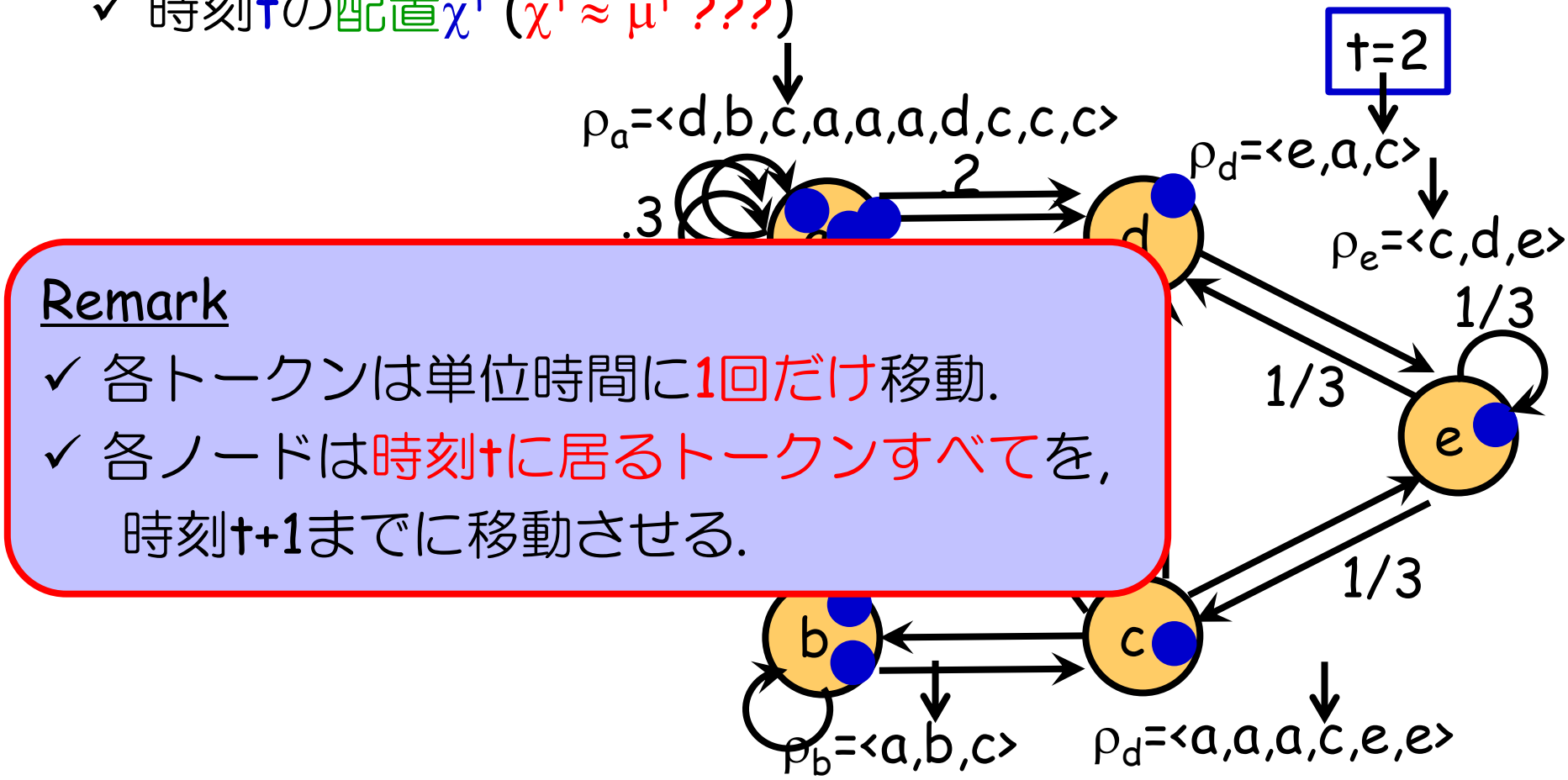
- ✓ $\chi^0$: 初期配置 ($\chi^0 = \mu^0$)
- ✓ $\rho$: "**rotor router**" (比率$P_{uv}$で頂点u頂点vに"順次"移動)
- ✓ 時刻tの配置$\chi^t$ ($\chi^t \approx \mu^t$ ???)



# deterministic RW (Propp機械; rotor-router)

N個のトークンがグラフ上を移動する.

- ✓ $\chi^0$: 初期配置 ($\chi^0 = \mu^0$)
- ✓ $\rho$: "**rotor router**" (比率$P_{uv}$で頂点u頂点vに"順次"移動)
- ✓ 時刻tの配置$\chi^t$ ($\chi^t \approx \mu^t$ ???)



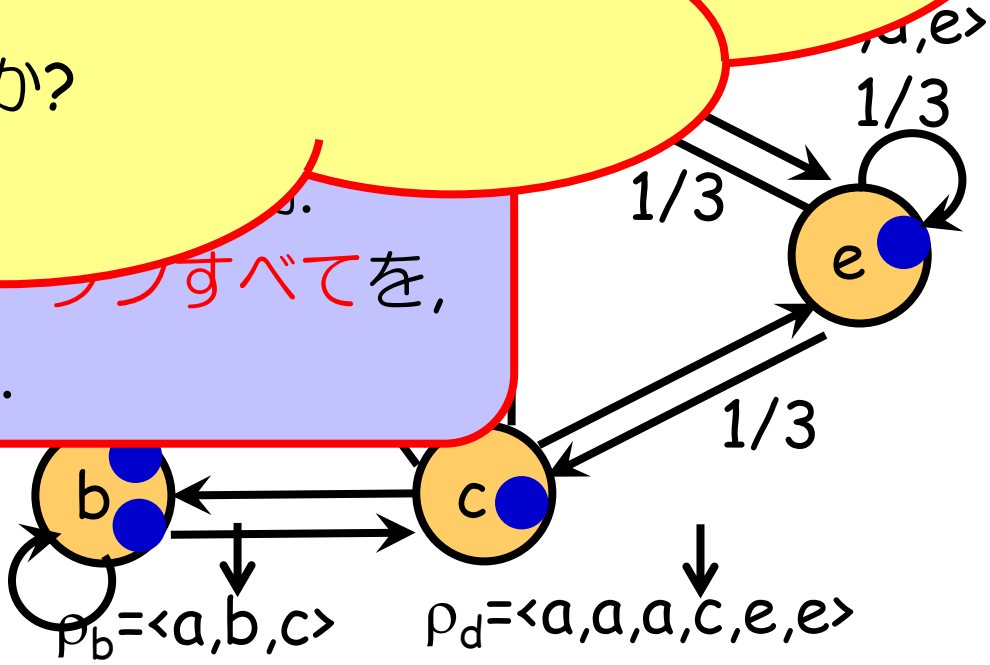

**Remark**

- ✓ 各トークンは単位時間に1回だけ移動.
- ✓ 各ノードは時刻tに居るトークンすべてを, 時刻t+1までに移動させる.

# Propp機械はランダムウォークを模倣できるか?

N個のトークンがグラフ上を移動する.

- ✓ $\chi^0$: 初期配置 ($\chi^0 = \mu^0$)
- ✓ (エルゴード)マルコフ連鎖は定常分布に収束する
  - ➢ "平滑化"される.
- ✓ Propp機械はランダムウォークを模倣できるか?
  - ➢ "平滑化"できるか?
  - ➢ "だま"はできないのか?
- ✓ 各
- ✓ 各ノードは時刻tに居るトークンすべてを,
  時刻t+1までに移動させる.



# 誤差の上界

**定理 [K, Koga, Makino 10+]**

対応する推移確率行列Pの固有値がすべて非負ならば,
任意の多重有向グラフ, 任意の初期状態, 任意のrotor-router,
任意の頂点w, 任意の時刻tについて,

$$\left|\chi_w^{(T)} - \mu_w^{(T)}\right| \leq 4mn + \mathrm{O}(m)$$

但し, n,mは多重グラフの頂点数,枝数.

**Remark**

「推移確率行列Pの固有値がすべて非負」という条件.

- reversible lazy Markov chainはこの条件を満たす.
  - ✓ MCMC法で使われるマルコフ連鎖
- +Pが対称 ⇒ Pは半正定値行列

# 誤差の下界

定理 [K, Koga, Makino 10+]

ある多重有向グラフ$\mathcal{G}$=(V,$\mathcal{E}$),

ある初期状態, あるrotor-routerが存在して,

$$\left|\chi_w^{(T)} - \mu_w^{(T)}\right| \geq \Omega(m)$$

但し, mは多重グラフの頂点数,枝数.

チップ数に依存しない

# 先行研究と本研究の成果

定理 [Cooper & Spencer 2006]

$\mathbb{Z}^d$上で, 任意の初期状態, 任意のrotor-router,
任意の頂点w, 任意の時刻tについて,
(dのみに依存する)定数$c_d$が存在して,

$$\left|\chi_w^{(T)} - \mu_w^{(T)}\right| \leq C_d$$

# 単一頂点誤差に関する研究(1/2)

| | | |
|---|---|---|
| 2006 | Cooper, Spencer | $Z^d$上のローターループーター<br>➢ 誤差 $\leq C_d$ |
| 2007 | Cooper, Doerr, Spencer, Tardos | $Z^1$上のローターループーター<br>➢ $C_1 \leq 2.29$ |
| 2008 | Cooper, Doerr, Friedrich, Spencer | 無限のk正則木上のローターループーター<br>➢ 誤差 $> \Omega(\sqrt{kT})$ at time T |
| 2009 | Doerr, Friedrich | $Z^2$上のローターループーター<br>➢ $C_2 \leq 7.83$ (上右下左)<br>➢ $C_2 \leq 7.29$ (上下左右) |
| 2012 | Kijima, Koga, Makino | 有限多重有向グラフ $G$上のローターループーター<br>➢ 誤差$\leq 4mn+O(m)$<br>$\{0,1\}^d$上のPropp機械<br>➢ 誤差$\leq O(d^3)$ (頂点数のpoly log) |
| 2012+ | Kajino et al. | (後述) |

# 単一頂点誤差に関する研究(2/2)

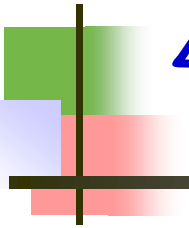
| | | |
|---|---|---|
| 2012<br>(2010) | Kijima, Koga,<br>Makino | 有限多重有向グラフ $G$上のロータールーター<br>➢ 誤差 $\leq 4m^*n+O(m^*)$<br>(P: 有理数 + 既約 + 非周期 + 可逆 + lazy)<br>$\{0,1\}^d$上のPropp機械<br>➢ 誤差 $\leq O(d^3)$ (頂点数のpoly log) |
| 2012+<br>(2011) | Kajino, Kijima,<br>Makino | 有限多重有向グラフ $G$上のロータールーター<br>➢ 誤差 $\leq O\left(\alpha \frac{m^*n^2}{1-\lambda}\right)$<br>(P: 有理数 + 既約)<br>$\{0,1\}^d$上のPropp機械<br>➢ 誤差 $\leq O(d^2)$ (頂点数のpoly log) |

# 単一頂点誤差に関する研究(2/2)

| | | |
|---|---|---|
| 2012<br>(2010) | Kijima, Koga,<br>Makino | 有限多重有向グラフ $G$上のロータールーター<br>➢ 誤差 $\leq 4m^*n+O(m^*)$<br>(P: 有理数 + 既約 + 非周期 + 可逆 + lazy)<br>$\{0,1\}^d$上のPropp機械<br>➢ 誤差 $\leq O(d^3)$ (頂点数のpoly log) |
| 2012+<br>(2011) | Kajino, Kijima,<br>Makino | 有限多重有向グラフ $G$上のロータールーター<br>➢ 誤差 $\leq 0\left(\alpha \frac{m^*n^2}{1-\lambda}\right)$<br>(P: 有理数 + 既約)<br>$\{0,1\}^d$上のPropp機械<br>➢ 誤差 $\leq O(d^2)$ (頂点数のpoly log) |
| 2013+<br>(2012) | Shiraga, Yamauchi,<br>Kijima, Yamashita<br>Brand-new<br>"deterministic sampling" | 有限有向グラフ $G$上の関数ルーター<br>➢ 誤差 $\leq 0\left(\frac{\pi_{\max}}{\pi_{\min}} t^*\Delta\right)$<br>$t^*$: mixing rate, $\Delta$: 遷移グラフの最大次数<br>(P: 実数 + 既約 + 非周期 + 可逆) |

# 今後の課題

- ✓ 上下界の一致. (O(mn), Ω(m))
- ✓ 組合せ構造に由来するグラフに対するpolylogの上界.
- ✓ Blanket time vs Mixing time.
- ✓ MCMC法の脱乱択化.
- ✓ 乱数とは？
  - ➢ 乱択アルゴリズムにおける「乱数」の持つべき性質は？
    - ◆ 準モンテカルロ (quasi Monte Carlo)
    - ◆ カオス系列 (Chaos time series)



# 4. まとめ

1. 乱択の威力: ストリーム中の頻出アイテム検知
   - ✓ O(log log N)領域計算法
2. 高度な乱択技法: 組合せ的対象のランダム生成 (MCMC法)
3. 脱乱択化: ランダムウォークの脱乱択化

# 今後の課題

「乱択アルゴリズムにおいて、乱数に真に求める性質は何か？」

# 今後の課題

「乱択アルゴリズムにおいて、乱数に真に求める性質は何か？」

**講演者の現在**

**確率的アルゴリズムについて、いろいろ研究**

- 自動掃除ロボットは部屋の隅を上手に掃除できるか？
  [with 平原昂樹, 山下雅史]
- 順列集合上のオンライン線形最適化
  [with 安武 翔太, 畑埜 晃平, 瀧本 英二, 竹田 正幸]

iRobot社 Roomba

次の講演の導入

# 置換多面体上の乱択丸め

来嶋秀治 (九州大学)

[共同研究 I]
安武翔太, 畑埜晃平, 瀧本英二, 竹田正幸,
(ISAAC 2011),

[共同研究 II]
末廣大貴, 畑埜晃平, 瀧本英二, 永野清仁,
(ALT 2012),

# 2010年5月のある日

Q. "効率的なアルゴリズムを知りませんか?"

入力: 順列の"平均"* p,

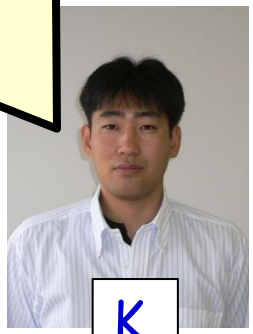
出力: ランダム順列 $X \in S_n$ s.t. $E[X] = p$.

*順列の"平均" $p := (p_1+p_2+\ldots+p_M)/M$

for $p_1=(3,1,4,2)$, $p_2=(2,1,4,3)$, ...

それは "置換多面体上の乱択丸め" ですね.
カラテオドリの定理を使えば,
(たぶん) 多項式時間でできます...

K. Hatano



K.

## 2010年12月のある日

✓ $O(n^3)$時間, and
✓ $O(n)$領域

Q. "効率的なアルゴリズムを知りませんか?"

入力: 順列の"平均"* p,

出力: ランダム順列 $X \in S_n$ s.t. $E[X] = p$.

*順列の"平均" $p := (p_1+p_2+\ldots+p_M)/M$

for $p_1=(3,1,4,2)$, $p_2=(2,1,4,3)$, ...

それは "置換 ... ですね.
カラテ ...
(たぶん)

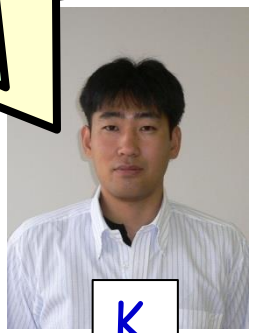
$O(n)$領域!?
$O(n^3)$時間???


E. Takimoto


M. Takeda


K. Hatano


S. Yasutake


K.

オンライン学習グループ

2011年1月1日

✓ $O(n^3)$時間, and
✓ $O(n)$領域

Q. "効率的なアルゴリズムを知りませんか?"

入力: 順列の"平均"* p,

出力: ランダム順列 $X \in S_n$ s.t. $E[X] = p$.

*順列の"平均" $p := (p_1+p_2+\ldots+p_M)/M$

for $p_1=(3,1,4,2)$, $p_2=(2,1,4,3)$, …, $p_M=(3,2,4,1)$.

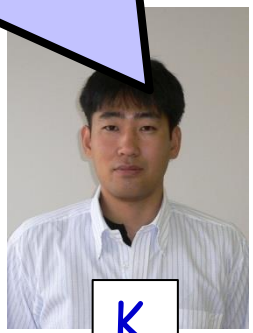
できた!
$O(n^2)$時間, $O(n)$領域!

E. Takimoto

M. Takeda

K. Hatano

S. Yasutake



K.

オンライン学習グループ

# *The end*

*Thank you for the attention.*