A-005

符号化を必要としない撹乱順列の線形時間ランダム生成について
Linear Time Generation of Random Derangements without Encoded

三河 賢治† 田中 賢‡
Kenji Mikawa Ken Tanaka

1. まえがき

n 個の自然数からなる集合 $[n]=\{1,2,\dots,n\}$ 上の撹乱順列は，$[n]$ 上の順列 $\delta=\delta_1\delta_2\dots\delta_n$ のうち，不動点を持たないもの，すなわち，$\forall i\in[n]$ に対して $\delta_i\neq i$ であるような順列として定義される．撹乱順列の個数を数え上げる問題は，包除原理の標準の例題として親しまれており，その個数を $!n$ とおくと，$!1=0$ と $!2=1$ を境界条件として，$n\geq 3$ について，

$$!n=(n-1)\big(!(n-1)+!(n-2)\big) \tag{1}$$

のように定義される．様々な $!n$ の定義が知られており，

$$!n=n\times!(n-1)+(-1)^n \tag{2}$$

のように表すこともできる．また，ネイピア数 e を用いると，$!n=\lfloor(n!+1)/e\rfloor$ が成り立つ．

$[n]$ 上の順列に対するランダム生成では，Fisher-Yates シャッフル法が最もよく知られた手法であり，Durstenfeld はこの手法を計算機に実装する洗練されたアルゴリズムを紹介している [1]．Fisher-Yates シャッフル法の素朴な実装はランダム生成に $O(n^2)$ 時間を必要とするが，Derstenfeld のアルゴリズムは線形時間でランダム生成を完了する．Martínez らの提案手法 [2] と Merlini らの提案手法 [3] は，どちらも Fisher-Yates シャッフル法を撹乱順列に応用し，ランダム生成された順列から撹乱順列だけを出力するものであった．ランダム生成された順列が撹乱順列である確率は大雑把に $1/e$ となることが期待されるので，平均的に線形時間で撹乱順列を出力するアルゴリズムを構成できそうである．実際，文献 [2]，[3] は，提案手法が確率的にならし線形時間で撹乱順列をランダム生成できることを示した．しかしながら，彼らの手法は，確率論的にはならし線形時間を実現していたが，最悪の場合，無限ループに陥る．

最近，著者らは撹乱順列を厳密に線形時間で実行する手法を提案した [4]．Martínez らの手法 [2] と著者らの手法 は，どちらも撹乱順列のサイクル構造に着目したものであったが，Martínez らは直接的に撹乱順列をランダム生成しているのに対して，著者らは撹乱順列のサイクル表記をランダム生成した後，サイクル表記の撹乱順列を元に戻しているため，若干のオーバーヘッドが存在していた．撹乱順列を直接的に，ただし，厳密に線形時間でランダム生成することは未解決であった．今回，著者らは，Martínez らの手法を改良し，不要な **repeat** ループを取り除くことにより，撹乱順列を直接的に，かつ，厳密に線形時間でランダム生成するアルゴリズムの開発に成功した．

† 新潟大学情報基盤センター
‡ 神奈川大学理学部

2. Martínez らのアルゴリズム

Martínez らは，Fisher-Yates シャッフル法 [1] と Sattolo の巡回順列に対するシャッフル法 [5] に基づいて，ならし線形時間で撹乱順列をランダム生成する手法を提案した [2]. Martínez らの手法は，各ステップで交換対象となる未使用の要素をランダムに発見しなければいけないため，最悪の場合，無限ループに陥る．Martínez らのアルゴリズムを以下に示す.

A1: **for** $k\coloneqq 1$ **to** n **do begin**
A2: $\delta(k)\coloneqq k$ and closed$(k)\coloneqq$ **false**
A3: **end**
A4: $i\coloneqq n$ and $u\coloneqq n$
A5: **while** $u\geq 2$ **do begin**
A6: **if** closed$(i)=$ **false then begin**
A7: **repeat**
A8: $j\coloneqq \mathrm{rand}(i-1)+1$
A9: **until** closed$(j)=$ **false**
A10: swap$\big(\delta(i),\delta(j)\big)$
A11: $P=\mathrm{rand}\big(!(u-1)+!(u-2)\big)$
A12: **if** $P<!(u-2)$ **then begin**
A13: closed$(j)\coloneqq$ **true** and $u\coloneqq u-1$
A14: **end**
A15: $u\coloneqq u-1$
A16: **end**
A17: $i\coloneqq i-1$
A18: **end**

関数 rand(i) は 0 から$i-1$ までの範囲にある整数の一様乱数を出力する．アルゴリズムは $\delta(n)$ から逆順に $\delta(1)$ まで各要素の値を決定していく．一旦，ステップ i で $\delta(i)$ の値が決定すると，以後のステップで接尾辞 $\delta(i)\dots\delta(n)$ が変更されることはない.

Martínez らのアルゴリズムは，大別すると二つの処理から構成される．一つめの処理を (a) とすると，処理 (a) は，$\delta(1)\dots\delta(i-1)$ からランダムに選択した要素と $\delta(i)$ を交換して巡回順列を生成していく処理で，A7 から A10 に記述されている．二つ目の処理を (b) とすると，処理 (b) は，撹乱順列のサイクル構造に一致するように，サイクル構造を制御する処理で，A11 から A15 に記述されている．処理 (a) では，$\delta(n)$ から逆順に $\delta(1)$ まで各要素の値を決定し，各 $\delta(i)$ の値は自身を含まない $\delta(1)\dots\delta(i-1)$ の中から選択するので，処理 (a) を通して自己置換は起こり得ない．次に，処理 (b) を解説する前に，要素の交換とサイクルとの関係について，簡単に解説する．同一サイクルに含まれる 2 個の要素を交換すると，そのサイクルを分解する．異なるサイクルに含まれる要素を交換すると，それらのサイクルを連結する．したがって，要素の交換は，サイクルの伸長と縮退に強く関わっている．

アルゴリズムでは，処理 (a) で選択した要素 $\delta(j)$ と $\delta(i)$ を交換するとき，後述するように，ある一定の確率でサイクルを閉じる．サイクルを閉じると判定した場合，フラグ closed(j) = **true** を立てて，以後のステップで $\delta(j)$ を使用しない．アルゴリズム的には，ステップ i で，$\delta(i)$ と交換する要素 $\delta(j)$ を $\delta(1)\dots\delta(i-1)$ の中からランダムに選択するとき，closed(j) = **false** のフラグが立っている要素（すなわち選択可能な要素）に当たるまでランダム試行を繰り返す．非常にシンプルで分かりやすい方法であるが，最悪の場合，ランダム試行が無限ループに陥る．無限ループを回避する方法は次節に譲り，次にサイクルを閉じる条件について議論しよう．

はじめに，ステップ i で $\delta(1),\dots,\delta(i)$ の中に選択可能な要素が u 個残っていると仮定する．また，ステップ i で $\delta(i)$ と交換される要素 $\delta(j)$ が選択されたと仮定する（この仮定により closed(j) = **false** である）．$\delta(i)$ と $\delta(j)$ の交換後，closed(j) = **true** となる場合と closed(j) = **false** となる場合で，残りの選択可能な要素から生成可能な撹乱順列の個数が異なる．具体的には，closed(j) = **true** の場合，残りの選択可能な要素から $!(u-2)$ 個の撹乱順列を生成可能である．反対に，closed(j) = **false** の場合，$!(u-1)$ 個の撹乱順列を生成可能である．したがって，$\delta(j)$ でサイクルを閉じる事象を A とすると，その確率 $P(A)$ は，次式

$$P(A)=\frac{!(u-2)}{!(u-1)+!(u-2)}. \tag{3}$$

で表される．$!(u-2)$ の値は，(2) 式と，直前のステップで求めた $!(u-1)$ の値から，次式

$$!(u-2)=\frac{!(u-1)-(-1)^{u-1}}{u-1}. \tag{4}$$

で計算できる．アルゴリズム的には，次式

$$\mathrm{rand}\big(!(u-1)+!(u-2)\big)<!(u-2) \tag{5}$$

が成り立つ場合に $\delta(j)$ でサイクルを閉じればよい．これらの処理は A11 から A14 に記述されている．

3. 改良アルゴリズム

著者らは，処理 (a) で，選択可能な u 個の要素の中から直接ランダムに $\delta(j)$ を選択したい．そこで，選択可能な要素を制御するための制御系列 $a=a(1)a(2)\dots a(n)$，要素 x の a 上の位置を返す関数 $a^{-1}(x)$，要素 x の δ 上の位置を返す関数 $\delta^{-1}(x)$ を用意する．制御系列 a は，先頭から u 個の要素が選択可能な要素となるようにしたい．このような制御系列が求まれば，$a(1),a(2),\dots,a(u-1)$ の中から直接ランダムに $\delta(j)$ を選択すればよいだろう．我々のアルゴリズムを以下に示す．

B1: **for** $k\coloneqq 1$ **to** n **do begin**
B2: $\quad\delta(k)\coloneqq\delta^{-1}(k)\coloneqq a(k)\coloneqq a^{-1}(k)\coloneqq k$
B3: $\quad$used$(k)\coloneqq$ **false**
B4: **end**
B5: $i\coloneqq n$ and $u\coloneqq n$
B6: **while** $u\geq 2$ **do begin**
B7: $\quad$**if** used(i) = **false then begin**
B8: $\quad\quad j\coloneqq\mathrm{rand}(u-1)+1$
B9: $\quad\quad t\coloneqq\delta(i)$
B10: $\quad\quad$**if** $\delta(i)=a(j)$ **then** $j\coloneqq u$
B11: $\quad\quad\mathrm{swap}\big(\delta(i),\delta(\delta^{-1}(a(j)))\big)$
B12: $\quad\quad\mathrm{swap}\big(a(u),a(j)\big)$
B13: $\quad\quad P=\mathrm{rand}\big(!(u-1)+!(u-2)\big)$
B14: $\quad\quad$**if** $P<!(u-2)$ **then begin**
B15: $\quad\quad\quad$used$\big(\delta^{-1}(t)\big)=$ **true**
B16: $\quad\quad\quad\mathrm{swap}\big(a(u-1),a(a^{-1}(t))\big)$
B17: $\quad\quad\quad u\coloneqq u-1$
B18: $\quad\quad$**end**
B19: $\quad\quad u\coloneqq u-1$
B20: $\quad$**end**
B21: $\quad i\coloneqq i-1$
B22: **end**

最も重要な部分は，どのようにして制御系列 a の先頭から u 個の要素が選択可能な要素となるのか，であろう．誤解を恐れずに簡単に述べる．

B8 行で，$\delta(i)$ と交換する要素を $a(1),a(2),\dots,a(u-1)$ の中からランダムに選択する．このとき，選択された要素は $\delta\big(\delta^{-1}(a(j))\big)$ であるが，$\delta(i)=a(j)$ が成立する場合には，自己置換となってしまうので，B10 行で $a(u)$ を選択したことにする．B12 行で a に関する交換を行い，選択可能な要素は $a(1),a(2),\dots,a(u-1)$ に残り，使用済となった要素は $a(u),a(u+1),\dots,a(n)$ に締め出される．続いて，B14 行で，サイクルを閉じると判定した場合について述べる．サイクルを閉じない場合，$a(1),a(2),\dots,a(u-1)$ はそのまま選択可能な要素となるので何もしない．一方，サイクルを閉じる場合，$a(1),a(2),\dots,a(u-1)$ の中にいる $a(j)$ は，以後のステップで使用しないので，$a(u-1)$ と交換して，新たに $a(u-1),a(u),\dots,a(n)$ が使用済の要素となる．これらの操作により，本研究では，Martínez らのアルゴリズムから不要な **repeat** ループを取り除き，厳密に線形時間で撹乱順列のランダム生成を実現した．

謝辞

本研究の一部は，日本学術振興会学術研究助成基金（24500076）を受け実施したものである．

参考文献

[1] R. Durstenfeld, “Algorithm 235: Random Permutation”, Commun. ACM, Vol. 7, No. 420 (1964).
[2] C. Martínez, A. Panholzer, H. Prodinger, “Generating Random Derangements”, Proc. ACM-SIAM Workshop on Analytic Algorithms and Combinatorics (2008).
[3] D. Merlini, R. Sprugnoli, M.C. Verri, “An Analysis of a Simple Algorithm for Random Derangements”, Proc. Italian Conf. Theoretical Computer Science (2007).
[4] 三河賢治，田中賢，“撹乱順列の線形時間ランダム生成について”，信学技法，Vol. 112, No. 273 (2012).
[5] S. Sattolo, “An Algorithm to Generate a Random Cyclic Permutation”, Information Processing Letters, Vol. 22, No. 6, 1986.