A-006

# 撹乱順列の高速なランキングとアンランキングについて
# Fast Ranking and Unranking of Derangements

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 $[n]=\{1,2,\ldots,n\}$ 上の撹乱順列は，$[n]$ 上の順列 $\delta=\delta_1\delta_2\ldots\delta_n$ 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．ランキング関数は，撹乱順列を要素とする集合から $\{0,1,\cdots,!n-1\}$ への全単射であり，アンランキング関数はその逆関数である．

歴史的には，ランクとアンランクを求めるアルゴリズムは，列挙アルゴリズムの副産物として開発されてきた．近年のブレイクスルーとして，Myrvold と Ruskey は，順列をランダム生成する代表的なアルゴリズムである Fisher-Yates シャッフル法を応用し，線形時間で順列のランクとアンランクを求めるアルゴリズムを提案した [6]．彼らは，Fisher-Yates シャッフル法でランダムに交換要素を決定していた部分を剰余で置き換えるという巧妙な技法を導入して，ランダム生成アルゴリズムからアンランキングアルゴリズムに変換する方法を示したと言える．この変換方法を適用することにより，撹乱順列においても，Martínez らのランダム生成アルゴリズム [3] を改良した線形時間ランダム生成アルゴリズムから，線形時間でランク／アンランクを求めるアルゴリズムが提案され [4, 5]，この問題の一応の解決を得た．しかしながら，[4, 5] のアルゴリズムは線形時間を実現するために，計算機への実装が複雑であった．本報告では，より簡素な実装方法を提案し，撹乱順列の高速なランキングとアンランキングを実現する．

$[n]$ 上の順列をランダムに生成するアルゴリズムでは，Fisher-Yates シャッフル法が最もよく知られた方法であり，Durstenfeld はこの方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 [2]．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 らの生成アルゴリズムは，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 [3] では，提案方法が確率的にならし線形時間で撹乱順列をランダムに生成できることを示した．しかしながら，内部に局所的な無限ループの可能性を抱えており，厳密な線形時間の生成アルゴリズムではなかった．Arndt は，Martínez らの方法から不要な **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を提案した [1]．彼のアルゴリズムは，非常にシンプルで簡単に計算機へ実装できる．今回，我々は Arndt のアルゴリズムを Myrvold らの変換方法に則して，シンプルかつ高速な撹乱順列のランキングとアンランキングを実現した．

† 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 らの手法は，各ステップで交換の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，無限ループの可能性を抱える．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: $\quad\delta(k)\coloneqq k$ and closed$(k)\coloneqq$ **false**
A3: **end**
A4: $i\coloneqq n$ and $u\coloneqq n$
A5: **while** $u\geq 2$ **do begin**
A6: $\quad$**if** closed$(i)=$ **false then begin**
A7: $\quad\quad$**repeat**
A8: $\quad\quad\quad j\coloneqq \mathrm{rand}(i-1)+1$
A9: $\quad\quad$**until** closed$(j)=$ **false**
A10: $\quad\quad$swap$\big(\delta(i),\delta(j)\big)$
A11: $\quad\quad P=\mathrm{rand}\big(!(u-1)+!(u-2)\big)$
A12: $\quad\quad$**if** $P<!(u-2)$ **then begin**
A13: $\quad\quad\quad$closed$(j)\coloneqq$ **true** and $u\coloneqq u-1$
A14: $\quad\quad$**end**
A15: $\quad\quad u\coloneqq u-1$
A16: $\quad$**end**
A17: $\quad i\coloneqq i-1$
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 $\delta(i)\ldots\delta(n)$ 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，$\delta(1)\ldots\delta(i-1)$ 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 $\delta(1)\ldots\delta(i-1)$ 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．

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，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，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 $\delta(1)\ldots\delta(i-1)$ 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．サイクルを閉じる条件は次のとおりである．ステップ $i$ で $\delta(1),\ldots,\delta(i)$ の中に選択可能な要素が $u$ 個残っているとする．また，ステップ $i$ で $\delta(i)$ と交換される要素 $\delta(j)$ が選択されたとする．$\delta(i)$ 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．

次に，Arndt のアルゴリズムを以下に示す．

B1: **for** $k\coloneqq 0$ **to** $n-1$ **do begin**
B2: $\quad\delta(k)\coloneqq k$ and $\mathrm{pos}(k)\coloneqq k$
B3: **end**
B4: $u\coloneqq n$
B5: **while** $u\geq 2$ **do begin**
B6: $\quad x_1\coloneqq u-1$ and $r_1\coloneqq\mathrm{pos}(x_1)$
B7: $\quad x_2\coloneqq \mathrm{rand}(u-1)$ and $r_2\coloneqq\mathrm{pos}(x_2)$
B8: $\quad\mathrm{swap}\big(\delta(r_1),\delta(r_2)\big)$
B9: $\quad u\coloneqq u-1$
B10: $\quad P\coloneqq\mathrm{rand}\big(!(u-1)+!(u-2)\big)$
B11: $\quad$**if** $P<!(u-2)$ **then begin**
B12: $\quad\quad\mathrm{swap}\big(\mathrm{pos}(x_2),\mathrm{pos}(u-1)\big)$
B13: $\quad\quad u\coloneqq u-1$
B14: $\quad$**end**
B15: **end**

Arndt のアルゴリズムでは，選択可能な $u-1$ 個の要素の $\delta$ 上の位置を $\mathrm{pos}(0),\mathrm{pos}(1),\ldots,\mathrm{pos}(u-2)$ に格納し，pos を制御することにより，選択可能な要素に当たるまでランダム試行を繰り返す処理を回避している．具体的には，$\delta(r_2)$ でサイクルを閉じるならば，$r_2$ は以降の処理で使用されることはないので退避する（B12 行）．

## 3. アンランキングアルゴリズム

提案アルゴリズムを以下に示す。提案アルゴリズムでは，Myrvold らの変換方法に則して，Arndt の生成アルゴリズムでランダムに交換要素を決定している部分（B7 行）を剰余に置き換える（C9 行）．サイクルを閉じる条件は，式 (5) と同様に，次の要素を決定するためのランク値 $\lfloor r/(u-1)\rfloor$ と $!(u-1)$ を比較して決定できる．

C1: **for** $k\coloneqq 0$ **to** $n-1$ **do begin**
C2: $\quad\delta(k)\coloneqq k$ and $\mathrm{pos}(k)\coloneqq k$
C3: **end**
C4: $u\coloneqq n$
C5: **while** $u\geq 2$ **do begin**
C6: $\quad x_1\coloneqq u-1$ and $r_1\coloneqq\mathrm{pos}(x_1)$
C7: $\quad x_2\coloneqq r \bmod (u-1)$ and $r_2\coloneqq\mathrm{pos}(x_2)$
C8: $\quad\mathrm{swap}\big(\delta(r_1),\delta(r_2)\big)$
C9: $\quad r\coloneqq\lfloor r/(u-1)\rfloor$
C10: $\quad u\coloneqq u-1$
C11: $\quad$**if** $r\geq !(u-1)$ **then begin**
C12: $\quad\quad r\coloneqq r-!(u-1)$
C13: $\quad\quad\mathrm{swap}\big(\mathrm{pos}(x_2),\mathrm{pos}(u-1)\big)$
C14: $\quad\quad u\coloneqq u-1$
C15: $\quad$**end**
C16: **end**

ランキングは，アンランキングの手順を逆順に戻すことで実現できる．以上，本研究では，Arndt のアルゴリズムから，線形時間で撹乱順列のランクとアンランクを求めるアルゴリズムを提案した．

**謝辞**

本研究の一部は，日本学術振興会学術研究助成基金（26330163, 15K00182）を受け実施したものである．

**参考文献**

[1] J. Arndt, “Generating Random Permutations”, Ph.D. Thesis, Australian National University, 2009.
[2] R. Durstenfeld, “Algorithm 235: Random Permutation”, Commun. ACM, Vol.7, No.7, p.420, 1964.
[3] C. Martínez, A. Panholzer, H. Prodinger, “Generating Random Derangements”, Proc. ACM-SIAM Workshop on Analytic Algorithms and Combinatorics, pp.234-240, 2008.
[4] 三河賢治，田中賢，“撹乱順列の線形時間ランキングとアンランキングについて”， 京都大学数理解析研究所講究録，Vol.1849, pp.12-17, 2013.
[5] 三河賢治，田中賢，“符号化を必要としない撹乱順列の線形時間ランキングとアンランキングについて”，電子情報通信学会ソサイエティ大会，p.17, 2013.
[6] W. Myrvold, F. Ruskey, “Ranking and Unranking Permutations in Linear Time”, Inform. Proc. Lett., Vol.79, No.6, pp.281-284, 2001.