アルゴリズム 75−10
（2000. 11. 10）

ハイパーキューブ上の順列の同形を反復しない網羅的生成

藤井 俊一　　玉木 久夫
明治大学大学院理工学研究科
{tfujii, tamaki}@cs.meiji.ac.jp

概要

ハイパーキューブ上の順列のすべてを、同形を反復することなく生成する問題を考える。部分順列の正規形を適当に定義すると正規形の部分順列全体の集合が、正規形の順列を葉に持つ木を構成することを示す。我々の生成アルゴリズムはこの木に対して深さ優先探索を行なう。このアルゴリズムの生成速度は正規形判定に要する時間に大きく依存している。正規形判定の効率化の手法を開発し、実験により4次元のキューブに対して素朴な方法の約11倍の生成速度が得られることを示す。

Isomorph-free exhaustive generation of permutations on the hypercube

Toshikazu Fujii　　Hisao Tamaki
School of Sience and Technology, Meiji University
{tfujii, tamaki}@cs.meiji.ac.jp

abstract

We consider the problem of generating all permutations on the hypercube without repeating isomorphs. We show that, with an appropriate definition of the canonical forms of partial permutations, the set of all canonical partial permutations forms a tree in which canonical permutations are placed on the leaves. Our generation algorithm simply performs a depth first search on this tree. The generation speed of the algorithm depends heavily on the efficiency of the algorithm that decides if a given partial permutation is canonical. We develop techniques for efficiently deciding canonicality and experimentally show that an improvement by a factor of nearly 11 is achieved for the 4-dimensional cube over the naive method.

1　はじめに

並列計算機内の結合網の理論的な研究において、ルーティングアルゴリズムの研究は大きな部分を占める [3]。ルーティングの理論的なベンチマーク問題として、順列ルーティングが用いられることが多い。すなわち、結合網の頂点集合を V とするとき、ルーティングの要求は、順列 $\pi : V \to V$ によって指定し、各 v から $\pi(v)$ へのメッセージ送信が要求されると解釈する。順列ルーティングに対する理論的な解析を実験によって補完する必要のある場合、順列の生成法が問題になる。

この研究では、ハイパーキューブを対象としてその上の順列の網羅的生成法を考える。直接の動機は Szymanski の予想 [7]([4, 2] も参照) の実験的検証（ないし反証）にある。この予想は、ハイパーキューブを各隣接2頂点間に双方向の辺のある有向グラフとしてモデル化するとき「任意の順列ルーティングは、辺素な有向経路の集まりによって実現される」というものであり、未だ解決されていない。反例を計算機によって探すことはこの問題に対して十分考えられるアプローチである。

すべての順列に対しての網羅的検査を考えるとき、ハイパーキューブの対称性を考慮することが望ましい。すなわち、順列 π を検査したならば、ハイパーキューブの自己同型写像によって π に写される順列はどれもさらに検査する必要はない。そこで、順列のすべてを同形を反復せずに生成するアルゴリズムが求められる。ここで、順列の総数は膨大であり、メモリ中に蓄えることは不可能であること

に注意する必要がある。

組合せ的構造を、メモリ中に蓄えずに列挙する方法の一般的な枠組として Avis と福田の逆探索法 [1] が良く知られており、また、同形を考慮した枠組 [5] が Mckay によって提案されている。これらの枠組の骨子は、列挙の対象となる構造物を頂点とし、それらの構造物の間の変換操作を辺とするグラフに対して頂点のマーク付けなしの木探索を行なうために、局所的に判定できる親子関係を定義することにある。

この報告ではまず、我々の問題が特殊でありこれらの枠組の一般性を必要としないことを示す。すなわち、順列の一般化として部分順列を自然に定義し、部分順列の正規形と、部分順列の拡張操作を適当に定義すると、正規形の部分順列を頂点とし、拡張操作を有向辺とするグラフはそのままで根付木となる。正規形の順列はこの木の葉に位置し、その列挙は単純な深さ優先探索によって行なうことができる。

さらに、このアルゴリズムの生成速度を決定する正規形判定の部分について判定を効率化する手法を開発する。計算機実験によって、効率化を行なったアルゴリズムでは 4 次元のキューブに対して素朴な方法の約 11 倍の生成速度が得られることが示された。

2 定義

d 次元ハイパーキューブ H_d は、頂点集合が $V_d = \{0,1\}^d$ であり、辺集合が $E_d = \{\{u,v\}|u,v \in V, u$ と v のハミング距離が $1\}$ であるようなグラフである。H_d 上の順列とは、V_d からそれ自身への 1 対 1 写像のことを言う。大きさ k の V_d の部分集合から V_d への 1 対 1 写像を k 次の部分順列と呼び、k 次の部分順列全体を Π_d^k と置く。特に、$V_d^k = \{0,1,2,...,k-1\}(0 \leq k \leq 2^d)$ とおいて、V_d^k から V_d への 1 対 1 写像を k 次の詰まった部分順列と呼ぶ。一般に k 次の部分順列 π の定義域を $\mathbf{dom}(\pi)$ で表す。$v \in V_d$ が部分順列 π の定義域に属さないとき、$\pi(v) = *$ と表現する。H_d の自己同形写像とは、H_d 上の順列 ρ で, 全ての $u,v \in V_d$ に対して $\{u,v\} \in E_d \Leftrightarrow \{\rho(u),\rho(v)\} \in E_d$ なるものを言い、H_d 上の自己同形写像の全体を A_d で表す。A_d の大きさは $2^d \times d!$ である [6]。H_d 上の 2 つの部分順列

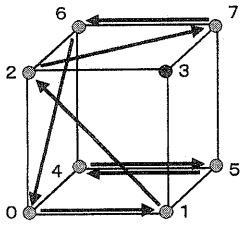

図 1: H_3 上の順列の例

$\pi_1, \pi_2 \in \Pi_d^k$ の間に $\pi_2 = \rho^{-1}\pi_1\rho$ となるような H_d 上の自己同形写像 ρ が存在するとき、π_1, π_2 は同形であると定義する。また、部分順列 π と同形な部分順列の集合を π の同形類と呼び、$[\pi]$ で表す。

以下では、H_d の頂点はそのビット列の表現する整数と同一視し、整数の大小により順序づけられているものとみなす。図 1 は H_3 上の順列の例を図示したものである。

3 順列生成アルゴリズム

同形の反復を避けるために、順列の正規形を考えることは当然であり、また、正規形の定義を部分順列に一般化することも自然である。正規形の定義はいろいろ考えられるが、次のような定義をすると、詰まった部分順列で正規形であるものからなる集合が扱いやすい構造を持つ。

まず、Π_d^k 上の全順序 $\prec$ を部分順列 π を列 $(\pi(0),\pi(1),...,\pi(2^d-1))$ で表現したときの辞書式順序と定義する。ただし、未定義を表す $*$ はどの $i \in V_d$ よりも大きいと見なす。π の正規形は同形類 $[\pi]$ の要素を順序 $\prec$ で比べた中で最小の順列とし、$\hat{\pi}$ で表す。それ自身の正規形であるような順列を、正規形の順列と呼ぶ。

各 k、$0 \leq k \leq 2^d$ に対して、k 次の詰まった部分順列で正規形であるようなものすべてからなる集合を C_k とおく。特に C_{2^d} は正規形の順列の集合であり、この集合の要素を列挙するのが我々の目的である。$\pi \in C_{k+1}$ に対して、$\pi'(i) = \pi(i)(0 \leq i < k)$ によって定義される k 次の詰まった部分順列を π の親と呼び、$\mathrm{parent}(\pi)$ で表す。π と $\mathrm{parent}(\pi)$ の間には、次のような補題が成り立つ。

補題：$\pi \in C_{k+1}$ のとき $\mathrm{parent}(\pi) \in C_k$ である。
証明:π を C_{k+1} に属する任意の順列とする。π は $k+1$ 次の詰まった部分順列であるから、$\pi' = \mathrm{parent}(\pi)$ が k 次の詰まった部分順列であることは明らかである。したがって、π が正規形であることから、π' も正規形であることを導けばよい。その対偶を示すために、π' が正規形でない、すなわち $\tau' \prec \pi'$ であるような $\tau' \in [\pi']$ が存在すると仮定する。$v \in V_d$ を $\tau'(V_d^k)$ に属さない任意の頂点とし、τ を $\tau'(i) = \tau(i), 0 \leq i < k$、$\tau(k) = v$ で定義すると、$0 \leq i < k$ の範囲では $\tau'(i) = \tau(i), \pi'(i) = \pi(i)$ だから、$\tau \prec \pi$ が成り立ち、π は正規形でないことになる。したがって、π が正規形ならば π' も正規形である。■

この親子関係より、根を $\phi \in C_0$ とした頂点集合 $\bigcup_{k=0}^{2^d} C_k$ 上の根付き木が構成される。この木を深さ優先探索することによって、C_{2^d} の要素を列挙することができる。より具体的なアルゴリズムを以下に記す。ここで、k 次の部分順列 π に対して $\pi \cup (k \to v)$ は、$\pi'(i) = \pi(i), 0 \leq i < k$、$\pi'(k) = v$ で定義される $k+1$ 次の部分順列を表す。また、この手続きの、トップレベルの呼び出しは $extend(\phi, 0)$ である。

```
procedure extend(π,k)
    /* π は詰まった k 次の正規形部分順列 */
    if k = 2^d then
        output π;
    else
        for 各 v ∈ V_d \ π(V_d^k) do
            if π ∪ (k → v) が正規形 then
                extend(π ∪ (k → v), k + 1);
            endif
        endfor
    endif
endprocedure
```

4 正規形判定の高速化

3節で述べた順列生成アルゴリズムは正確に全ての順列を反復することなく生成できるが、アルゴリズムの実行は、部分順列が正規形であるかどうかを判定するアルゴリズムの実行速度に大きく依存している。素朴な方法では $|A_d| = 2^d \times d!$ 個の全ての自己同形写像を調べることになるので、多くの時間を要する。そこで、高速な正規形判定法を考える必要がある。

高速化のアルゴリズムを述べる前に素朴な方法のアルゴリズムを以下に記す。

```
procedure canonimal0(π)
    /* π は部分順列 */
    for ∀ρ ∈ A_d do
        τ = ρ^{-1}πρ;
        if τ ≺ π then
            return π は正規形でない
        endif
    endfor
    return π は正規形
endprocedure
```

$u, v \in V_d$ の間のハミング距離を $\mathrm{dist}(u, v)$ で表し、$\pi \in \Pi_d^k$ に対して、π の最小距離 $\mathrm{min\_dist}(\pi)$ を $\mathrm{min\_dist}(\pi) = \min\{\mathrm{dist}(v, \pi(v)) \mid v \in \mathbf{dom}(\pi)\}$ により定義する。最右の c ビットがすべて1で残りが0であるようなビット列を 1_c と表す。整数としてみたとき、$1_c = 2^c - 1$ である。

補題1 u, v を V_d の任意の2頂点とするとき、$\rho(u) = 0, \rho(v) = 1_{\mathrm{dist}(u,v)}$ であるような $\rho \in A_d$ が存在する。(証明略)

補題2 部分順列 π の正規形を $\hat{\pi}$ とすると、$\hat{\pi}(0) = 1_{\mathrm{min\_dist}(\pi)}$ である。

証明:$\pi \in \Pi_d^k$ を固定し、(1)すべての $\tau \in [\pi]$ に対して $\tau(0) \geq 1_{\mathrm{min\_dist}(\pi)}$ であること、および (2)ある $\tau \in [\pi]$ で $\tau(0) = 1_{\mathrm{min\_dist}(\pi)}$ のものがあることを示せばよい。(1) を示すために、$\tau(0) < 1_{\mathrm{min\_dist}(\pi)}$ であるような $\tau \in [\pi]$ があったと仮定すると $\mathrm{dist}(0, \tau(0)) < \mathrm{min\_dist}(\pi)$ である。ρ を $\tau = \rho^{-1}\pi\rho$ であるような H_d の自己同形写像とすると、$\mathrm{dist}(\rho(0), \pi(\rho(0))) = \mathrm{dist}(\rho(0), \rho(\tau(0))) = \mathrm{dist}(0, \tau(0)) < \mathrm{min\_dist}(\pi)$ となって、min_dist の定義に矛盾する。

次に (2) を示す。v を $\mathrm{dist}(v, \pi(v)) = \mathrm{min\_dist}(\pi)$ であるような頂点とすると、補題1より、$\rho \in A_d$ を $\rho(v) = 0, \rho(\pi(v)) = 1_{\mathrm{min\_dist}(\pi)}$ となるように選ぶことができる。$\tau = \rho\pi\rho^{-1}$ とおけば、$\tau(0) = 1_{\mathrm{min\_dist}(\pi)}$ となる。■

補題2より、与えられた π の正規形判定について次のことが言える。

1. $\mathrm{dist}(0, \pi(0)) > \mathrm{min\_dist}(\pi)$ ならば、π は正規

形でないと即座に判定できる。

2. $\mathrm{dist}(0,\pi(0)) = \mathrm{min\_dist}(\pi)$ ならば、$\mathrm{dist}(v,\pi(v)) = \mathrm{min\_dist}(\pi)$ であるような v に対して $\rho(0)=v$, $\rho(1_{\mathrm{min\_dist}(\pi)}) = \pi(v)$ であるような ρ についてのみ、$\rho^{-1}\pi\rho$ と π の辞書式順序の比較を行なえば十分である。

そこで、各 $u,v \in V_d$ に対して、$A_d[u,v] = \{\rho \in A_d \mid \rho(0)=u, \rho(1_{\mathrm{dist}(u,v)}) = v\}$ とおき、次のアルゴリズムを得る。

procedure *canonimal1*(π)
 (if) $\mathrm{dist}(0,\pi(0)) > \mathrm{min\_dist}(\pi)$ **then**
  return π は正規形でない
 for $\forall v \in \mathbf{dom}(\pi)$: $\mathrm{dist}(v,\pi(v)) = \mathrm{min\_dist}(\pi)$
 do
  for $\forall \rho \in A_d[v,\pi(d)]$ **do**
   $\tau = \rho^{-1}\pi\rho$;
   if $\tau \prec \pi$ **then**
    return π は正規形でない
   endif
  endfor
 endfor
 return π は正規形
endprocedure

さらなる高速化のために、$\rho^{-1}\pi\rho$ と π の比較の結果を複数の ρ で共有するためのデータ構造を考える。

まず自己同形写像の集合 $A_d[u,v]$ の各々は、その要素が辞書式順序に並んだ表により表現する。このような表のひとつを T としよう。表 T 中の j 番目の部分順列を $T[j]$ で表す $(0 \le j < |T|)$。二つの自己同形写像 ρ_1 と ρ_2 は、v 以下のすべての i に対して $\rho_1(i) = \rho_2(i)$ を満たすとき、v-等価であると言うことにしよう。T のインデックス j と $v \in V_d$ に対して、整数 $\mathrm{jump}_T(j,v) \le |T|$ を、条件「$j \le j' < \mathrm{jump}_T(j,v)$ なるすべての j' に対して、$T[j]$ と $T[j']$ は v-等価である」を満たすような最大の整数として定義する。各表 T に対して $\mathrm{jump}_T(j,v)$ はあらかじめ計算して表にしておく。

この jump 表は次のように用いる。$\rho = T[j]$ とし、$\pi \preceq \rho^{-1}\pi\rho$ であることが、頂点 u までの比較で決定されたとしよう。すなわち、$\pi(i) = \rho^{-1}\pi\rho(i)$、$0 \le i < u$ かつ $\pi(u) < \rho^{-1}\pi\rho(u)$。ここで、$v = \max\{\rho^{-1}\pi\rho(i) \mid 0 \le i \le u\}$ とおくと、ρ と v-等価な ρ' に対しては明らかに $\pi \preceq (\rho')^{-1}\pi\rho'$ がなりたつ。したがって、T 中のインデックスを $\mathrm{jump}_T(j,v)$ まで進めて検査を続ければ良いことになる。具体的なアルゴリズムを下に示す。

procedure *canonical2*(π, k)
 /* π は k 次の詰まった部分順列 */
 (if) $\mathrm{dist}(0,\pi(0)) > \mathrm{min\_dist}(\pi)$ **then**
  return π は正規形でない
 endif
 for $\forall v \in \mathbf{dom}(\pi)$: $\mathrm{dist}(v,\pi(v)) = \mathrm{min\_dist}(\pi)$
 do
  $T := A_d[v,\pi(d)]$;
  $i := 0$; $v := 0$; $j := 0$;
  while $j < |T|$ **do**
   if $i = k$ **then**
    /* T[j] は π の自己同形なので無視 */
    $i := 0$; $j := \mathrm{jump}_T(j,v)$;
   else
    $\rho := T[j]$; $w := \rho^{-1}\pi\rho(i)$;
    if $w < \pi(i)$ **then**
     return π は正規形でない
    else
     if $i > v$ **then** $v := i$;
     endif
     if $w = \pi(i)$ **then** $i := i+1$;
     else $i := 0$; $j := \mathrm{jump}_T(j,v)$;
     endif
    endif
   endif
  endwhile
 endfor
 return π は正規形である
endprocedure

5 実験結果

実験はランダムに $\pi \in C_k$ を選び、各 π について、それを根とした部分木を全探索するのにかかった時間と部分木の葉の数を調べることによって行う。各部分木に対して、正規形判定を *canonical0*, *canonical1*, *canonical2*(表中ではそれぞれ *cano0*, *cano1*, *cono2* で表記) の 3 通りのアルゴリズムについて行い、それぞれのアルゴリズム

	*cano*0,	*cano*1	*cano*2
時間	9,729	4,311	886
平均生成速度	1,313	2,964	14,426
最大生成速度	1,521	16,059	48,179
最小生成速度	380	761	5,333

表 1: H_4 における生成速度の比較

での生成速度を比較する。表 1 は $\pi \in C_7$ をランダムに 100 個選び、それぞれの π を根とする部分木に対して各アルゴリズムを適用したときの生成速度 (生成数/時間 (s)) を比較したものである。計算機は Sun Ultra60(CPU: UltraSparcII 360MHZ) を使用した。今回用いた各 π からの生成数の合計は 12,782,077 個であり、合計時間は表中の通りである。そのときの各アルゴリズムの生成速度の最大、最小、平均で比較した。これにより、改良後のアルゴリズムでは素朴な方法と比べて、約 11 倍の生成速度が得られたと言える。

参考文献

[1] D.Avis and K.Fukuda. Reverse Search for Enumeration. *Discrete Applied Math*, 6, 23-46 (1996).

[2] Q.P. Gu and H. Tamaki. Routing a permutation in the hypercube by two sets of edge disjoint paths. *Journal of Parallel and Distributed Computing*, 44, 147-152, 1997.

[3] F.T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays·Trees·Hypercubes.* Morgan Kaufmann, 1992.

[4] A. Lubiw. Counterexample to a Conjecture of Szymanski on Hypercube Routing. *Information Processing Letters*, 29, 57-61, 1990.

[5] B.D.Mckay, Isomorph-free Exhaustive Generation, *Journal of Algorithms*, 26(2), 306-324, 1998.

[6] A.Sprague and H.Tamaki Routings for involutions of a hypercube *Dicrete Applied Mathematics*, 48, 175-186, 1994

[7] T. Szymanski. On the permutation capability of a circuit-switched hypercube. In *Proc. International Conference on Parallel Processing (I)*, 103–110, 1989.