アルゴリズム 65－4
（1998. 10. 28）

O(m + n) 時間及びスペースのランダムグラフ生成法

矢農 正紀　　二村 良彦

早稲田大学大学院理工学研究科　早稲田大学理工学部情報学科

〒169–0072 東京都新宿区大久保3–4–1

e-mail : {yanoh,futamura}@futamura.info.waseda.ac.jp

概要

グラフアルゴリズムの精密な評価を行うためには，指定した頂点数と辺数を有する複数のランダムグラフがテストデータとして必要である．本稿では，頂点数nかつ辺数mのラベル付きグラフの一様ランダム生成を$O(m+n)$時間及び同一のスペースで行うアルゴリズムを示す．このアルゴリズムは，理論的な計算量の下限という意味で最適であり，従来の平均$O(m+n)$時間のアルゴリズムに対する理論的な改善を達成している．加えて，コンピュータを用いた実験においても良い結果を得ている．

Random Graph Generation with O(m + n) Time and Space

Masanori Yanoh and Yoshihiko Futamura

Graduate School of Science and Engineering, Waseda University
Department of Computer and Information Science,
Waseda University

3–4–1 Okubo Shinjuku-ku, Tokyo 169–0072 Japan
e-mail : {yanoh,futamura}@futamura.info.waseda.ac.jp

Abstract

This paper proposes a fast algorithm for generating random graphs with n vertices and m edges for given n and m. Random graphs are used to conduct precise evaluation of graph algorithms as test data. Our algorithm can generate a graph with n vertices and m edges uniformly at random in theoretically optimal $O(m+n)$ time and space. We improve the running time from an former algorithm with expected $O(m+n)$ time and $O(m+n)$ space. We also report on experimental results for generating random graphs and show that our algorithm is faster than a former one.

1 はじめに

アルゴリズムの教育及び研究に際して，アルゴリズムが扱う各種データをランダムに供給するランダムデータサーバ(RDS)が必要である．本稿は，グラフに関するランダムデータサーバの基礎技術に関するものである．

グラフを扱う各種アルゴリズムの精密な評価を行うためには，コンピュータ上で複数のランダムグラフをテストデータとして生成し，それらを用いて評価を行う必要がある．実際的な評価を行うためには，少なくとも頂点数nと辺数mを指定することが必要である．

しかしながら，頂点数と辺数を指定しても，グラフの総数は一般に非常に大きくなるので，全てのグラフを列挙して評価を行うことは，現実的に不可能である．

従って，限られた個数のランダムグラフで評価を行って，その上で数学的な検定手法に基づいた評価を行うことが望ましい．その際，テストデータとして用いられるグラフが，偏った確率で生成されていると正当性が保証されないため，理論的に一様な生成が行われている必要がある．

以上の必要性に基づき，我々は頂点数nかつ辺数mのラベル付き連結グラフに対するランダム生成法を開発した[2, 7, 8]．また，頂点数nかつ辺数mの閉路を含まないラベル付き有向グラフ(dag)に対するランダム生成法も示した[9]．

本稿では，各種グラフのランダム生成における基本である，頂点にラベルが付いた頂点数nかつ辺数mのグラフを$O(m+n)$時間及びスペースで一様ランダムに生成するアルゴリズムに関して述べる．これは，Raman [5]が提示した未解決問題の一つに対する実質的な解答である．

ラベル付きグラフの生成では，無向グラフであれば$\binom{n}{2}$本，有向グラフであれば$n(n-1)$本の辺からm本の辺を選び出す．これは，集合から要素数を指定して部分集合を一様ランダムに選び出す組合せ選択問題を解くことに対応する．

集合$\{1,\ldots,r\}$から要素数kの部分集合を選ぶ組合せ選択問題に対しては，Nijenhuis-Wilf [4]がk個の区間の問題に巧妙に分割することで，平均$O(k)$時間かつ$O(k)$空間で生成を行うアルゴリズムを示している(区間分割方式)．これが従来の最適であり，グラフ生成に適用した場合，平均$O(m+n)$時間かつ$O(m+n)$空間となる．

一方，Wormald [6]は，一様乱順列の生成に基づき，$O(r)$時間及び空間の前処理で，毎回$O(k)$時間で生成可能なアルゴリズムを示している(順列方式)．これをグラフ生成に適用することで，$O(m+n)$時間の生成が達成されるが，前処理及び空間に$O(n^2)$の空間が必要であり，nが大きい場合に実現が困難である．

本稿では最初に，順列方式に基づき組合せ選択問題を$O(k+\sqrt{r})$時間及びスペースで解くアルゴリズムを示す(改良順列方式)．それから，無向グラフや有向グラフの生成に組合せ選択アルゴリズムを適用するために，数値と辺の対応付けを行う方法に関して述べる．これにより$O(m+n)$時間及びスペースでのラベル付きグラフ生成が可能である．

加えて，コンピュータ上で改良順列方式と区間分割方式を実装してグラフ生成の実験を行った．改良順列方式を用いた生成は，区間分割方式及びその系である剰余分割方式に対して良い結果を得ている．

以降では，まず第2節で定義や用語に関して説明を行う．続いて，第3節で組合せ選択問題を$O(k+\sqrt{r})$時間及びスペースで解くアルゴリズム及び従来のアルゴリズムを示す．第4節では組合せ選択アルゴリズムを用いてグラフ生成を行うための，数値と辺の対応付けに関して述べる．第5節でコンピュータ上での実験結果と，結果に対する考察を行い，最後に第6節でまとめと今後の課題を述べる．

2 定義と用語

本稿では，計算モデルとして，入力サイズnに対して$O(\log n)$ビット長整数の数値演算を定数時間で処理できるRAM [1]を仮定する．加えて，$O(\log n)$ビット長整数の範囲で，定数時間で一様乱数を生成できる乱数生成器の存在を仮定する．

グラフは，頂点集合$V=\{1,2,\ldots,n\}$と辺集合$E \subseteq E_K = \{(v,w) \mid v,w \in V,\ v \neq w\}$で構成される単純なラベル付きグラフとする．無向グラフでは$(v,w)=(w,v)$が成立するが，有向グラフでは$(v,w) \neq (w,v)$である．

また，頂点数nに対して，無向グラフでは$N=\binom{n}{2}$，有向グラフでは$N=n(n-1)$とNを定義する．すなわち，Nを頂点数nに対する完全グラフの辺数，すなわち存在し得る最大辺数とする．

3 組合せ選択問題

頂点数nかつ辺数mのラベル付きグラフのランダム生成では，N本の辺からm本の辺を選び出す(全$\binom{N}{m}$通り)．これは本質的には，集合から要素数を指定して部分集合を選び出す組合せ選択問題を解くことに対応する．

本節では，要素数rの集合$\{1,2,\ldots,r\}$から要素数kの部分集合を一様ランダムに選び出す組合せ選択問題(全$\binom{r}{k}$通り)に関して述べる．最初に，本稿で示す方式の基本である順列方式に関して述べる．それから，$O(k+\sqrt{r})$時間及びスペースでの生成が可能である改良順列方式を示す．加えて，従来の最適である区間分割方式に関して述べ，各々の比較を行う．最後に，改良順列方式と区間分割方式について、各々の実現で高速化を行うための手法に関して述べる．

3.1 順列方式

一様乱順列を生成する手続きは一般に良く知られている [4]．Wormald [6] による順列方式は，長さrの一様乱順列から取り出したk個の要素が一様ランダムな組合せであることを用いた生成法である．ただし，Wormaldは配列の末尾k個の要素を取り出す手続きを示しているが，ここでは次に述べる改良順列方式との関連から，先頭k個の要素を取り出す手続きを示す．

以下で順列方式の手順を示す．まず，配列$a_1,a_2,\ldots,a_r$を値$1,2,\ldots,r$で初期化する．この初期化は一回だけ行えば良い．

```
for i ← 1 to r do
    a_i ← i;
```

初期化が行われていれば，要素の交換を行う以下の生成手続きで$a_1,\ldots,a_k$に結果が得られる．ただし，$random(i,\ r)$は，区間$[i,r]$内の整数乱数を一様ランダムに生成する関数である．

```
for i ← 1 to k do begin
    x ← random(i, r);
    swap(a_i, a_x);
end;
```

表 1: 順列方式による生成

i	1	2	3	4	5	6	7	8	9
x	4	8	5	4	8	8			
a_i	4	8	5	1	2	3	7	6	9

この生成手続きでは，長さrの一様乱順列の生成をk番目の要素まで行う．$k=r$とすれば一様乱順列の生成となる．ここで，交換する前の配列の値に関係なく一様ランダムな結果が得られることに注意されたい．従って，最初に一回初期化を行えば，以降は生成手続きだけを呼び出せば良い．

計算量については，前処理及び空間に$O(r)$が必要であり，生成手続きの実行時間は$O(k)$である．

表1に$r=9$，$k=6$とした場合の例を示す．ただし，xは各々のiで生成された乱数の値であり，a_iは手続きを実行後の配列の値である．

3.2 改良順列方式

順列方式では，前処理及び空間に$O(r)$を必要とするため，kに比較してrが大きい場合に問題が生じる．これに対して，改良順列方式では，$O(k+\sqrt{r})$時間及びスペースで，前処理を必要としない組合せの生成を行う．本項では最初に，改良順列方式の考え方に関して述べ，それから具体的な手続きを示す．

改良順列方式では，長さkの配列$a_1,\ldots,a_k$と$b_1,\ldots,b_k$を用いる．a_iと交換を行う要素a_xの添字xが，$x\leq k$であれば順列方式と同様に扱う．$x>k$の場合，存在しないa_xの代用としてb_iを用いる．順列方式で交換した後a_xに入るべき値，すなわち実行時点でのa_iをb_iに格納して，a_iには仮の値としてxを代入する．

例として，表1と同一の条件で以上を実行した場合の結果を，表2の上部に示す．

ここまで実行した時点で，$x\leq k$であった各要素については順列方式で実行した場合と同一の値が設定される．$x>k$の場合についても，同じ値xが重複して出現しなければ順列方式でも$a_i=x$であるから仮の値を変更する必要はない．問題となるのは，$x>k$であるxで，同一の値が複数出現する場合である．

表 2: 改良順列方式による生成

i	1	2	3	4	5	6
x	4	8	5	4	8	8
b_i		2			3	6
a_i	4	8	5	1	8	8

⇓

i	1	2	3	4	5	6
ℓ_i					2	5
a_i	4	8	5	1	2	3

ここで，iに対する左最短同値要素ℓ_iを，$j<i$をみたし，かつ$a_j=a_i$である最大の値jと定義する．$j<i$, $a_j=a_i$である値jが存在しない場合は$\ell_i=nil$とする．表2の上部を例とすると，$i=5,6$に対する左最短同値要素はそれぞれ$\ell_5=2$, $\ell_6=5$である．その他の場合は$\ell_i=nil$である．

iに対する左最短同値要素ℓ_iがnilでない場合，a_iに設定すべき正しい値はb_{ℓ_i}に存在する．なぜなら，b_{ℓ_i}には順列方式でのa_xに対応する値を事前に格納しているからである．例えば，表2で正しい結果を得るためには，表2の下部に示すように$a_5 \leftarrow b_2$, $a_6 \leftarrow b_5$とすれば良い．

従って，順列方式と同一の結果を得るためには，左最短同値要素を求める必要がある．そのためには，安定なソーティングアルゴリズムを用いる方法が考えられる．安定なソート結果があれば，左隣を参照することで，容易に左最短同値要素を求めることができる．しかしながら，例えばマージソートのようなソーティングアルゴリズムを用いた場合，$O(k\log k)$時間が必要であり，これは実行時間の増加を引き起こす．

この問題を解決するためには，基数ソートを用いれば良い．基数を$\lceil\sqrt{r}\,\rceil$とすれば2パスでのソートが可能であるから，その実行時間及びスペースを$O(k+\sqrt{r})$で抑えることが可能である．

以下に，改良順列方式の具体的な手続きを2段階に分けて示す．

1. 配列$a_1,\ldots,a_k$の初期化を行ってから，順列方式と同様に乱数xを発生させて，$x\le k$であれば，要素の交換を行う．$x>k$の場合は，前述の通りb_i, a_iを設定する．

```
for i ← 1 to k do
  a_i ← i;
for i ← 1 to k do begin
  x ← random(i, r);
  if x ≤ k then
    swap(a_i, a_x)
  else begin
    b_i ← a_i; a_i ← x;
  end;
end;
```

2. 先頭の部分に示した手続き$radixsort$は基数ソートを行う．具体的には，配列$a_1,\ldots,a_k$に対して，基数を$\lceil\sqrt{r}\,\rceil$としてソートを行い，

$$a_{s_1}\le a_{s_2}\le\cdots\le a_{s_i}\le a_{s_{(i+1)}}\le\cdots\le a_{s_k}$$

が成り立つように，配列$s_1,\ldots,s_k$に結果を返す手続きである．次に，a_{s_i}で値がk以下の部分は変更する必要がないのでスキップする．そして，a_{s_i}で値がkより大きい部分について，隣の要素$a_{s_{(i+1)}}$との比較を行う．もし値が同じであれば，左最短同値要素$\ell_{s_{(i+1)}}$はs_iに等しいから，$a_{s_{(i+1)}}$に正しい値b_{s_i}の設定を行う．

```
radixsort({a_1,...,a_k},{s_1,...,s_k},⌈√r⌉)
j ← 1;
while a_{s_j} ≤ k and j < k do
  j ← j + 1;
for i ← j to k − 1 do
  if a_{s_i} = a_{s_(i+1)} then
    a_{s_(i+1)} ← b_{s_i};
```

この改良順列方式は，基数ソートの部分に最も計算時間及びスペースを必要とする．他の部分が$O(k)$時間及びスペースであることは明らかである．後に述べるグラフ生成に用いる目的で，ここでは基数として$\lceil\sqrt{r}\,\rceil$を選択することで，2回の定数回のパスで基数ソートを実行可能である．従って，全体の計算量は$O(k+\sqrt{r})$時間及びスペースである．

なお，理論的には，ある定数cに対して$r<k^c$であれば，基数をkとして基数ソートのパスを$\lceil c\rceil$回とすることで$O(k)$時間及びスペースでの実行が可能である．しかしながら，グラフ生成の目的に用いる場合は前述の$O(k+\sqrt{r})$時間及びスペースで理論的な最適時間が達成される．それについては次節で述べる．

3.3 区間分割方式

本項では，Nijenhuis-Wilf [4]による区間分割方式について簡単に述べる．その手順は2段階に分けられる．

まず，区間$[1,\ r]$を分割したk個の部分区間$R_1,\ldots,R_k$を定義する．

$$R_l = \left\{ m \;\middle|\; \left\lfloor \frac{(l-1)r}{k} \right\rfloor < m \leq \left\lfloor \frac{lr}{k} \right\rfloor \right\}$$

そして，R_lに対応する部分集合B_lを定義する．このB_lは，$1,2,\ldots,r$から選ばれたk個の整数のうち，R_lに含まれる整数を要素としてもつ集合である．

第1段階では，各区間R_lに対応する個数$|B_l|$のみを求める．$1,2,\ldots,r$の一様乱数xを発生させて，それがすでに選ばれた数で重複していればやり直し，そうでなければ集合B_lに加える．xに対応する区間R_lの添字lは，以下の式により定数時間で求めることができる．

$$l = 1 + \lfloor (xk-1)/r \rfloor$$

選んだ数xが集合B_lにおいて重複していれば，破棄してやり直しを行う．区間R_lに含まれる整数がq個であるとする．そのうち，m個がすでに使用済みであれば，確率m/qで重複する．従って，m/qの確率でxを破棄して乱数生成をやり直せば良い．さもなければxを受容するが，この場合でも，xの値自身は破棄して，1増加させた$|B_l|$の値だけを結果として記録することに注意されたい．

第2段階では，各区間から整数を選び出す．この時点で，各々の$|B_l|$だけがわかっているので，各区間R_lから$|B_l|$個の整数を選び出せば良い．その際，$O(|B_l|^2)$時間で実行する素朴なアルゴリズムを用いるが，個々の$|B_l|$は平均して小さな数になるので，実行時間の増大は発生しない．

空間計算量については，$O(k)$スペースである．加えて，プログラム上の工夫により，余計な配列を用いない実現が可能である．時間計算量については，解析により，平均$O(k)$時間であることが示されている [4].

3.4 各方式の比較

ここでは，組合せ選択問題を解くアルゴリズムに関して，グラフ生成の観点から比較を行う．

表 3: 組合せ選択アルゴリズムの比較

方式	時間計算量	空間計算量
区間分割方式	平均$O(k)$	$O(k)$
順列方式	$O(k)$	$O(r)$
改良順列方式	$O(k+\sqrt{r})$	$O(k+\sqrt{r})$

表3に各方式の計算量を示す．

頂点数nかつ辺数mのラベル付きグラフ生成の場合，rは最大辺数Nに，kは辺数mに対応する．ただし，グラフの表現には，一般的な隣接リストによる表現を用いても$O(m+n)$の空間を必要とする．そして，この空間を設定するために$O(m+n)$時間が必要であるから，グラフ生成の計算量の下限は$O(m+n)$である．

従って，Nijenhuis-Wilf [4]による区間分割方式でグラフ生成を行った場合，計算量は平均$O(m+n)$時間かつ$O(m+n)$空間である．Wormaldによる順列方式であれば$O(m+n)$時間が達成されるが，前処理及び空間に$O(N) = O(n^2)$を必要とする．

これに対して，改良順列方式では，無向グラフと有向グラフのいずれについても$\sqrt{N} < n$が成立するので，$O(m+n)$時間及びスペースでのグラフ生成を行うことが可能である．

ただし，区間分割方式では，生成される組合せが昇順$(a_i < a_{i+1})$で得られるので，生成されるグラフは順序リスト表現となる．これに対して，順列方式，改良順列方式及び剰余分割方式では，生成される組合せの順序が保たれない．しかしながら，グラフ生成に関しては，辺の各頂点を基数とした2パスの基数ソートを用いることで，$O(m+n)$時間及びスペースで区間分割方式と同様に順序リスト表現で結果を得ることができる [3].

また，区間分割方式では，Nijenhuis-Wilfがプログラム上の工夫により，余分な配列を用いることのない実現を行っているが，順列方式，改良順列方式及び後述する剰余分割方式では余分な配列やリストのための空間が必要である．

3.5 改良順列方式の高速化

前項で，ランダムグラフの生成に対して改良順列方式が理論的に最適であることを示した．

しかしながら，実用的な速度での実現を行うには，プログラム上の工夫が必要である．本項では，改良順列方式の実現において，後述の実験においても行った，高速化を行うための手法に関して述べる．

まず，前述の手続きでは，基数ソートを行う処理を別の手続きとして扱っていた．しかし，この処理は別の手続きにせず改良順列方式に一体化させることで高速化を行うことができる．なぜなら，最短左同値要素を求めるためにはソートされた結果は必ずしも必要でなく，2回のパスで処理されたリストをなぞるだけで良いからである．

次に，リストをなぞるとき，長さが1以下の場合を特別扱いする．加えて，基数を$\lfloor\sqrt{N}\rfloor$とする代りに，$\max(m,n)$にする．これにより，メモリー使用量は若干増大するが，リストの長さが平均して短くなるので，特別扱いで処理される部分の割合が大きくなり，高速化を図ることができる．

以上の手法を改良順列方式に組み込むことで，従来の区間分割方式以上に高速なランダムグラフ生成を行うことが可能である．

4 区間分割方式の高速化

本項では，Nijenhuis-Wilfの区間分割方式に対する系として，区間分割方式より高速な生成が可能である剰余分割方式に関して述べる．

コンピュータ上で実現を行う場合，区間で分割を行うと乗算や除算の回数が多く必要であり，結果として実行時間の増大が発生してしまう．この対策として，以下のように$R_1,\ldots,R_k$を再定義する．

$$R_l = \{m \mid m \in \{1,\ldots,r\},\ m \equiv (l-1) \pmod{k}\}$$

すなわち，kで割った剰余で分割を行う．これにより，必要な乗除算の回数を減らすことができる．以降では，ここで再定義したR_lを用いて組合せの生成を行う方式を剰余分割方式と呼ぶ．

なお，生成される組合せの順序は昇順でなく，ランダムでもなくなるが，前述したように，辺の各頂点を基数とした2パスの基数ソートを用いることで，区間分割方式と同様に順序リスト表現で結果を得ることができる．

図 1: 無向グラフの対応付けの例 ($n = 6$)

$1 \to (1,2)$	$2 \to (1,3)$	$3 \to (1,4)$
$4 \to (1,5)$	$5 \to (1,6)$	$6 \to (5,6)$
$7 \to (2,3)$	$8 \to (2,4)$	$9 \to (2,5)$
$10 \to (2,6)$	$11 \to (4,5)$	$12 \to (4,6)$
$13 \to (3,4)$	$14 \to (3,5)$	$15 \to (3,6)$

5 数値と辺との対応付け

頂点数nかつ辺数mのラベル付きグラフの生成に前述の組合せ選択アルゴリズムを用いる場合，集合$\{1,\ldots,N\}$から要素数mの部分集合を選び出して得られた$a_1,\ldots,a_m$をグラフの辺集合に変換する処理が必要である．すなわち，以下に示すような手続きを実行する必要がある．

for $i \leftarrow 1$ **to** m **do**
  $(v_i, w_i) \leftarrow unpair(a_i)$;

このとき，$unpair(a_i)$は，一対一の対応付けに基づく変換である必要がある．また，実行時間の増加を防ぐために，定数時間で実行可能である必要がある．

このような数値を辺に対応付ける方法については，無向グラフに対してRaman [5]が順序を保つ方法を提示しているが，本稿では，各頂点に対する順序を保つ条件に緩めることで，より素朴な方法を示す．

まず，無向グラフでは，$(v,v+1),\ldots,(v,n)$と$(n-v,n-v+1),\ldots,(n-v,n)$を合計すると$n$本の辺となることを利用して，図1に示すように$n$本ずつ数値に辺を対応付ける方法が考えられる．$n$が偶数の場合は，最後は$(n/2,n/2+1),\ldots,(n/2,n)$の$n/2$本である．

従って，上述の$(v_i,w_i) \leftarrow unpair(a_i)$に対応する処理は，以下の手続きのように，$n$で割った商と余りを取り，それを変換することで可能である．

$q \leftarrow \lfloor (a_i-1)/n \rfloor + 1$;
$r \leftarrow \{(a_i-1) \bmod n\} + 1$;
if $r \le n-q$ **then**
  $v_i \leftarrow q,\ w_i \leftarrow q+r$
else
  $v_i \leftarrow n-q,\ w_i \leftarrow r$;

図 2: 有向グラフの対応付けの例 ($n = 4$)

$1 \to (1,2)$	$2 \to (1,3)$	$3 \to (1,4)$
$4 \to (2,1)$	$5 \to (2,3)$	$6 \to (2,4)$
$7 \to (3,1)$	$8 \to (3,2)$	$9 \to (3,4)$
$10 \to (4,1)$	$11 \to (4,2)$	$12 \to (4,3)$

次に，有向グラフでは，辺 (v,w) の各 v について $n-1$ 本の辺が存在するので，無向グラフより問題は単純である．実際には，図2に示すような対応付けを行えばよい．

$(v_i, w_i) \leftarrow unpair(a_i)$ に対応する処理は，以下の手続きのように，$n-1$ で割った商と余りを取り，それを変換することで可能である．

$q \leftarrow \lfloor (a_i - 1)/(n-1) \rfloor + 1;$
$r \leftarrow \{(a_i - 1) \bmod (n-1)\} + 1;$
if $r < q$ **then**
  $v_i \leftarrow q,\ w_i \leftarrow r$
else
  $v_i \leftarrow q,\ w_i \leftarrow r + 1;$

以上で，無向グラフと有向グラフについて数値を辺に対応付ける方法を示したが，この逆の処理である $a \leftarrow pair(v,w)$ についても定数時間で容易に可能であることに注意されたい．

6 実験の結果

ここまでで，改良順列方式を用いた生成が，ラベル付きグラフの生成として理論的に最適であることを示したが，実用上の観点からはコンピュータ上での実行速度に関する検証を行うことも必要である．

本節では，区間分割方式，剰余分割方式，改良順列方式，順列方式の4方式を用いてラベル付きグラフ生成の実験を行った結果を示す．最初に，実験を行った条件を示し，それから結果と考察について述べる．

6.1 実験の条件

コンピュータにはSUN Ultra Enterprise 2 を用いて，C言語でプログラムを制作して実験を行った．コンパイラにはGCCを利用した．

区間分割方式では，Nijenhuis-Wilf [4] のFORTRANプログラムを構造化してC言語上で実現した．条件を同一にするため，乱数生成などに含まれていた実数演算については，整数演算を用いるように変更した．他の方式については，我々が新たにC言語での実現を行った．

実験プログラムでは，隣接リスト表現をデータ構造として用いた．無向グラフの生成と廃棄を繰り返して，それらの合計時間を計測した．生成したグラフの外部記憶への出力は行っていない．

なお，区間分割方式以外では，生成される組合せの順序が昇順ではないため，辺の各頂点を基数とした2パスの基数ソートを用いて，生成されるグラフが順序リスト表現で得られるよう統一している．

6.2 結果と考察

実験の結果として，表4，表5，表6に，それぞれ頂点数 n を5000, 50000, 500000として，各辺数 m を有するラベル付き無向グラフのランダム生成に掛かった時間を示す．表中の単位時間は秒である．また，各々についてグラフは10個生成を行い，その平均時間を示している．

なお，順列方式については $O(n^2)$ スペースが必要であるため，実行可能な頂点数 $n = 5000$ の場合についてだけ結果を示している．また，前処理として必要な，配列の初期化に要した時間は平均で0.72秒であった．

まず，表4に関して述べる．改良順列方式は，素朴な順列方式に比べると速度は若干劣るが，区間分割方式及び剰余分割方式と比較すると高速な結果を得ている．この頂点数が最も小さい結果においては，改良順列方式は辺数 m が大きい場合でも良い結果を得ている．

次に，表5と表6に関して述べる．これらの頂点数が大きなグラフについては，$O(n^2)$ スペースを必要とする順列方式を用いて生成を行うことは困難である．従って，$O(m+n)$ スペースの生成方式に関してだけ結果を示す．これらの実験結果においても，改良順列方式は，区間分割方式や剰余分割方式と比較してより高速なグラフ生成を行っているといえる．

表5において改良順列方式は，区間分割方式に対して約2.3倍から約2.8倍，剰余分割方式に対して約1.3倍から約1.5倍の高速化を達成して

表 4: 頂点数$n = 5000$の実行結果

辺数m	50000	500000	5000000
区間分割	0.99	10.61	119.62
剰余分割	0.50	5.67	66.24
改良順列	0.32	4.08	39.97
順列	0.27	3.20	33.11

表 5: 頂点数$n = 50000$の実行結果

辺数m	50000	500000	5000000
区間分割	1.02	10.91	110.23
剰余分割	0.53	6.02	61.85
改良順列	0.36	4.53	47.93

いる．特に，区間分割方式において，辺数mが大きいほど高速化の割合が小さくなる理由としては，改良順列方式ではグラフの順序リスト表現を得るために，辺の各頂点を基数とした2パスの基数ソートを行っていることが考えられる．

表6においては，頂点数nが大きいため，いずれの辺数mに対してもグラフが疎である．従って，各方式における実行時間の増加はほぼ線形である．改良順列方式は，区間分割方式及び剰余分割方式に対して，それぞれ約1.8倍，約1.1倍の高速化を達成している．表5に比べて高速化の割合が小さい理由としては，グラフが大規模であるために，組合せ選択以外の部分での実行時間が大きいことが考えられる．

7 おわりに

本稿では，頂点数nかつ辺数mのラベル付きグラフを，理論的な計算量の下限という意味で最適な，$O(m+n)$時間及びスペースで生成するアルゴリズムを提案した．コンピュータ上の実験においても，従来の方式と比較して高速な生成を行える実現が可能であることを示した．

表 6: 頂点数$n = 500000$の実行結果

辺数m	50000	500000	5000000
区間分割	1.89	19.52	190.12
剰余分割	1.17	12.03	122.79
改良順列	1.04	10.83	108.52

今後の課題は以下の通りである．

1. 改良順列方式を用いた，グラフに関するランダムデータサーバの実現．
2. 連結性などの制約を付けた場合の，高速なグラフ生成アルゴリズムの開発．
3. 頂点数mが比較的大きなグラフに対する，改良順列方式の高速化．
4. 組合せ選択問題を$O(k)$時間で解くアルゴリズムの開発．

参考文献

[1] Aho, A. V., Hopcroft, J. E. and Ullman, J. D.: *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Massachusetts, 1974.

[2] 二村，高野，矢農：連結グラフの生成方法およびそのプログラム，特願平9-259958, 1997.

[3] van Leeuwen, J.: *Handbook of Theoretical Computer Science Volume A : Algorithms and Complexity*, Elseveir Science Publishers B. V., 1990.

[4] Nijenhuis, A. and Wilf, H. S.: *Combinatorial Algorithms (Second Edition)*, Academic Press, New York, 1978.

[5] Raman, R.: Generating Random Graphs Efficiently, Advances in Computing and Information – ICCI '91 Proceedings, Springer-Verlag, New York, 1991, pp. 149–160.

[6] Wormald, N. C.: Generating Random Regular Graphs, *J. Algorithms*, Vol. 5, 1984, pp. 247–280.

[7] 矢農正紀：頂点数nと辺数mを指定した連結グラフのランダム生成，修士論文，早稲田大学大学院理工学研究科，1998.

[8] 矢農正紀，二村良彦：連結グラフのランダム生成法について，情報処理学会アルゴリズム研究会，98-AL-60, 1998, pp. 1–8.

[9] 矢農正紀，二村良彦：閉路を含まない有向グラフ(dag)のランダム生成法，日本ソフトウェア科学会第15回論文集，B5-3, 1998.