アルゴリズム 25－3
（1992．1．24）

正則グラフを生成するアルゴリズム

松田 洋一　榎原 博之　中野 秀男　堀内 諭†

大阪大学工学部 通信工学科
†松下電器産業(株)

頂点数 n と次数 d が与えられたとき、任意の d 次正則グラフ $G=(V,E)$ $(|V|=n, |E|=e=dn/2)$ を生成する $O(ne^2)$ 時間のアルゴリズムを提案する。このアルゴリズムは、クリーク挿入法、頂点挿入法および枝交換法の3つの生成方法よりなる。そして、クリーク挿入法と頂点挿入法によって与えられた頂点数と次数を持つ正則グラフが生成できることと、枝交換法を $\lfloor e/2 \rfloor$ 回実行することによって、このグラフを同じ頂点数と次数を持つ任意の正則グラフに変換することができることを示す。

An Algorithm For Generating Regular Graphs

Yoichi Matsuda　Hiroyuki Ebara　Hideo Nakano　Satoshi Horiuchi†

Department of Communication Engineering, Faculty of Engineering, Osaka University
†Matsushita Electric Industrial Co., Ltd.

In this paper, given number of vertices n and degree d, we propose an $O(ne^2)$ algorithm for generating arbitrary d-regular graphs with n vertices. This algorithm consists of three generating methods; *insert-clique method, insert-vertex method and exchange-edges method.* We prove that a regular graph with given (n,d) can be generated by using insert-clique method and insert-vertex method, and that it can be transformed an arbitrary regular graph with the same (n,d) by using exchange-edges method $\lfloor e/2 \rfloor$ times.

1 はじめに

頂点の次数がすべて等しい正則グラフを生成する方法として、確率的グラフ理論の立場からランダムな正則グラフを生成するアルゴリズムが提案されている [3][4]。しかし、これらのアルゴリズムは正則グラフのランダムな生成が主な目的であり、生成できる正則グラフの次数に制限がある。すなわち、頂点数を n としたとき [4] では $O(\sqrt{\log n})$ まで、[3] では $\sqrt{n/2}$ までの次数しか効率良く生成することができない。また、計算量理論の分野において正則グラフの同形判定問題が $\mathcal{NP}$ 完全であるかどうかは未解決であり [1]、この問題の難しさを評価する上でも任意の次数の正則グラフを生成する方法が必要である。

そこで本稿では、頂点数および次数が与えられたとき、任意の正則グラフを生成する $O(ne^2)$ 時間 (n : 頂点数、e : 枝数) のアルゴリズムを提案する。このアルゴリズムでは、クリーク挿入法、頂点挿入法によって任意の頂点数と次数を持つ正則グラフを生成し、枝交換法を $\lfloor e/2 \rfloor$ 回実行することによってこのグラフを同じ頂点数と次数を持つ任意の正則グラフに変換している。

2 準備

頂点 (vertex) の集合 $V = \{1, 2, \cdots, n\}$、枝 (edge) の集合 $E = \{(u_i, v_i) \mid u_i, v_i \in V;\ 1 \leq i \leq e\}$ より構成される無向グラフ (undirected graph)(以後、単にグラフと呼ぶ) $G = (V, E)$ を考える。d 次正則グラフ (d-regular graph) とは、頂点の次数がすべて d となるグラフである。このとき、枝数 e について $e = dn/2$ であるから、次数と頂点数の積 dn は必ず偶数になる。頂点数が n の完全グラフを K_n と表すと、K_n は $(n-1)$ 次正則グラフであるから、d 次正則グラフのうちで頂点数が最小のものは K_{d+1} となる。

また、グラフ G において E に属する枝と属さない枝 (枝のない部分) とを交互に通る辺素な道を G における交互道 (alternating path in G)、それが閉じているとき G における交互閉路 (alternating cycle in G) と定義する。

定義 2.1 (G における交互道 AP_G、交互閉路 AC_G)

$$
\begin{aligned}
AP_G &= \{(\alpha_i, \beta_i) \mid \alpha_i = (u_i, v_i) \in E,\\
&\qquad \beta_i = (v_i, w_i) \notin E;\ \alpha_i \neq \alpha_j,\ \beta_i \neq \beta_j;\\
&\qquad 1 \leq i, j \leq k,\ i \neq j\}\\
&\quad : \ \textit{alternating path in } G\\
AC_G &= \{(\alpha_i, \beta_i) \mid \{(\alpha_i, \beta_i)\} \textit{ is } AP_G;\\
&\qquad 1 \leq i \leq k;\ w_k = u_1\}\\
&\quad : \ \textit{alternating cycle in } G
\end{aligned}
$$

■

交互閉路 AC_G には以下のような性質がある。

性質 2.1 AC_G に含まれる各頂点には E に属する枝 α と属さない枝 β が同数ずつ接続している。

性質 2.2 AC_G の長さの最小値は 4 である。

3 正則グラフの生成

3.1 生成方法について

頂点数 n と次数 d が与えられたときに、頂点数 n の d 次正則グラフ全ての中から任意のグラフを生成する方法について考える。

本稿では、正則グラフを生成する過程を (i) 頂点数が最小の d 次正則グラフ K_{d+1}(複数個) を初期状態とし、正則性を保ちながら頂点数を n まで増加させていき、(ii) (i) で生成したグラフを正則性を保ちながら同じ頂点数と次数を持つ任意の正則グラフに変換するという 2 つの段階に分ける。そして、(i) を実現する方法としてクリーク挿入法と頂点挿入法を、(ii) を実現する方法として枝交換法を提案する。また、次数が 1 の正則グラフの生成方法は自明であるので、次数は 2 以上で考えるものとする。

3.2 生成方法の概略

次数が 2 以上の正則グラフを生成する手順の概略は以下のようになる。

(1) k 個のクリーク K_{d+1} を最初のグラフとする。

(2) 頂点数 n に対して

$$n = k(d+1) + l(d-1) + m$$

が成り立つようにクリーク挿入法の実行回数 l、頂点挿入法の実行回数 m (d が奇数のときは $m/2$) を決める。このときできるだけクリーク挿入法の実行回数が多くなるように m を

$$m = \big(n - k(d+1)\big) \mod (d-1)$$

と定める。

(3) (2) で定めた実行回数でクリーク挿入法と頂点挿入法を実行し、頂点数 n の d 次正則グラフを生成する。

(4) (3) で生成した正則グラフを枝交換法を r 回実行することにより任意の正則グラフに変換する。

次に、クリーク挿入法、頂点挿入法、枝交換法各々の具体的な振る舞いを示す。

- クリーク挿入法

ランダムに $(d-1)$ 本の枝を選び、これらを $e_1 = (x_1, y_1), e_2 = (x_2, y_2), \cdots, e_{d-1} = (x_{d-1}, y_{d-1})$ とする。そして、各々の枝の中間に (図 1) のように新たに頂点を作り、それらがクリーク K_{d-1} を構成するように枝を接続する。

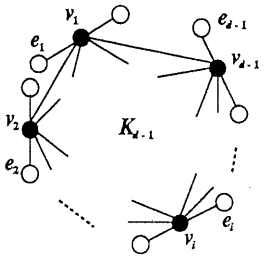

図 1: クリーク挿入法

- 頂点挿入法

クリーク挿入法では、頂点数が $(d-1)$ ずつ増加するため、これだけでは必ずしも所望の頂点数を得ることができるとは限らない。また、頂点の増加数の最小値は d が偶数ならば 1、d が奇数ならば 2 である。そこで、これらの最小増加数を実現する生成方法を提案する。

(i) d が偶数のとき

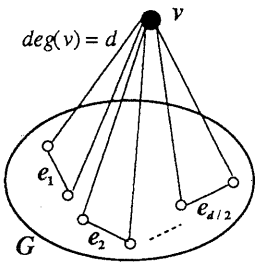

図 2: 頂点挿入法 (d : 偶数)

ランダムに $d/2$ 本の互いに隣接しない枝を選び、これらを $e_1 = (x_1, y_1), e_2 = (x_2, y_2), \cdots, e_{d/2} = (x_{d/2}, y_{d/2})$ とする。次に、これらの枝の端点 x_i, y_i $(1 \leq i \leq d/2)$ を挿入すべき頂点 v と接続する。最後に、$e_1, e_2, \cdots, e_{d/2}$ を除去する。(図 2)

(ii) d が奇数のとき

ランダムに長さ d の点素な道を選び、これを $\{e_1 = (x_0, x_1), e_2 = (x_1, x_2), \cdots, e_d = (x_{d-1}, x_d)\}$ とする。次に、挿入すべき頂点 v_1, v_2 について、v_1 は $x_0, x_1, \cdots, x_{d-1}$ と、v_2 は $x_1, x_2, \cdots, x_d$ と各々接続する。最後に、$e_1, e_2, \cdots, e_d$ を除去する。(図 3)

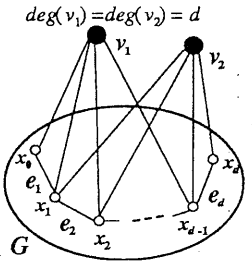

図 3: 頂点挿入法 (d : 奇数)

- 枝交換法

まず長さ 2 の交互道 $AP_G = (\alpha, \beta)$ $(\alpha \in E,\ \beta \notin E)$ を求め、再帰的にこれを延長していく。そして、(図 4) のように AP_G が閉じたとき、これを交互閉路 AC_G とする。その後、AC_G において枝のある部分とない部分とを交換する。

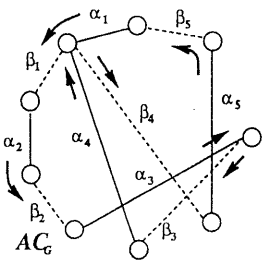

図 4: 枝交換法

3.3 生成アルゴリズム

上述した生成方法に基づく正則グラフの生成アルゴリズムは以下のようになる。

```
algorithm generate_regular_graph; { メインルーチン }
  次数 d, 頂点数 n, K_{d+1} の数 k を入力する;
  if (dn mod 2 ≠ 0  or  d ≥ n
                    or  d ≤ 1 or  k(d+1) > n) then
    もう一度 d, n, k を選び直す;
  end
  e := dn/2;
  m := (n − k(d+1))  mod (d − 1);
  l := (n − k(d+1) − m)/(d − 1);

  for i = 1 to l do
    insert_clique;
  end
  if (d mod 2 = 0) then {d が偶数}
    for i = 1 to m do
      insert_vertex_even;
    end
  else {d が奇数}
    for i = 1 to m/2 do
      insert_vertex_odd;
    end
  end
  if (d < n − 1) then {G は完全グラフでない}
    for i = 1 to r do
      exchange_edges;
    end
  end
end;


{ サブルーチン }
procedure insert_clique; { クリーク挿入法 }
  ランダムに d − 1 本の枝 e_1, e_2, ···, e_{d−1} を選ぶ;
  for i = 1 to d − 1 do
    e_i = (x_i, y_i) の間に v_i を挿入する;
  end
  if (d > 2) then
    create_clique(d − 1);
  end
end;
procedure create_clique(k); { クリーク K_k を作る }
  for i = 1 to k − 1 do
    for j = i + 1 to k do
      v_i と v_j を接続する;
    end
  end
end;

procedure insert_vertex_even; { 頂点挿入法 (d:偶数)}
  select_disjoint_edges(d/2);
  for i = 1 to d/2 do
    SELEDGE から e_i = (x_i, y_i) を選ぶ;
    x_i, y_i を新しい頂点 v と接続する;
    e_i を除去する;
  end
end;
procedure select_disjoint_edges(k);
                    {k本の互いに隣接しない枝を選ぶ }
  EDGESET := E;
  SELEDGE := ∅;
  for i = 1 to k do
    EDGESET からランダムに e_i = (x_i, y_i) を選ぶ;
    e_i および x_i と y_i に接続している枝を
    EDGESET より削除する;
    e_i を SELEDGE に加える;
  end
end;

procedure insert_vertex_odd; { 頂点挿入法 (d:奇数) }
  EDGESET := E;
  SELPATH := ∅;
  i := 1;
  適当な頂点 u を選ぶ;
  select_path(u, d);
  for j = 1 to d do
    SELPATH から e_j = (x_{j−1}, x_j) を選ぶ;
    x_{j−1} を新しい頂点 v_1 と接続する;
    x_j を新しい頂点 v_2 と接続する;
    e_j を除去する;
  end
end;
procedure select_path(v, k); { 長さ k の点素な道を選ぶ }
  v に接続している枝 (v, w) を EDGESET から
  ランダムに選ぶ;
  x_{i−1} := v;
  x_i := w;
  e_i := (x_{i−1}, x_i);
  e_i を SELPATH に加える;
  e_i および x_{i−1} に接続している枝を EDGESET
  より削除する;
  i := i + 1;
  if (i ≤ k) then
    select_path(w, k);
  end
end;

procedure exchange_edges; { 枝交換法 }
  E_0 := E;
  E_α := ∅;
  E_β := ∅;
  E'_α := ∅;
  E'_β := ∅;
  適当な頂点 s を選ぶ;
  alt_path_search(s, 0, forward);
end;
procedure alt_path_search(v, phase, status);
                                { 交互道を見つける }
  if (phase = 0) then { 枝α を探す }
    while (∃α = (v, w) ∈_R E_0 − E_α − E'_α) do
      E より α を削除する;
      E_α に α を加える;
      alt_path_search(w, 1, forward);
      if (status = backward) then
        E に α を加える;
        E_α より α を削除する;
        E'_α に α を加える;
      else
        return(status = completed);
      end
    end
    return(status = backward); {α が見つからない }
  else { 枝β を探す }
    while (∃β = (w, x) ∉_R E_0 ∪ E_β ∪ E'_β) do
```

```
      Eにβを加える;
      Eβにβを加える;
      if (x ≠ s) then
        alt_path_search(x, 0, forward);
        if (status = backward) then
          Eよりβを削除する;
          Eβよりβを削除する;
          E'βにβを加える;
        else
          return(status = completed);
        end
      else { 交互閉路が見つかった }
        return(status = completed);
      end
    end
    return(status = backward); {βが見つからない}
  end
end;
```

■

4 生成アルゴリズムの正当性

本節では、上述した生成アルゴリズムの正当性について、生成方法との対応から**クリーク挿入法および頂点挿入法の正当性、枝交換法の正当性**の順で証明を行なうものとする。

定理 4.1 クリーク挿入法、頂点挿入法の実行によって正則性が失われない。

証明

• クリーク挿入法

挿入されるクリーク K_{d-1} の各頂点 v_i $(1 \leq i \leq d-1)$ によって枝 (x_i, y_i) が (x_i, v_i), (y_i, v_i) に分割されるが、このとき v_i にはこれらの2本の枝の他に K_{d-1} から $(d-2)$ 本の枝が接続している。したがって v_i の次数は $(d-2)+2=d$ となり、正則性は保持される。

• 頂点挿入法

(i) d が偶数のとき

$d/2$ 本の枝 $e_1, e_2, \cdots, e_{d/2}$ は互いに素であり、頂点 v の挿入によって e_i の端点の次数は1増加するが、枝 e_i の除去によって1減少するので不変である。また v の次数も d であるから、正則性は保持される。

(ii) d が奇数のとき

頂点 v_1, v_2 の挿入によって、長さ d の点素な道 $\{e_i = (x_{i-1}, x_i)\ ;\ 1 \leq i \leq d\}$ において始点 x_0 と終点 x_d の次数は $(d+1)$ に、それ以外の点 $(x_1 \sim x_{d-1})$ の次数は $(d+2)$ となるが、e_i を除去することによってこれらの頂点の次数はすべて d となる。また v_1, v_2 の次数も d であるから、正則性は保持される。 ■

定理 4.2 クリーク挿入法、頂点挿入法の実行によって任意の頂点数と次数を持つ正則グラフを生成することができる。

証明

最初のクリーク K_{d+1} の個数を k、クリーク挿入法の実行回数を l、頂点挿入法によって増やすべき頂点数を m とすると、生成する正則グラフの頂点数 n について

$$n = k(d+1) + l(d-1) + m \quad \cdots (*)$$

が成り立つ。

ここで、頂点挿入法の実行について考える。

(i) d が偶数のとき

頂点の増加数は1であるから、$(*)$ より m 回実行すればよい。また、$d \leq n-1$ より、$d/2$ 本の互いに隣接しない枝を選ぶことは可能である。

(ii) d が奇数のとき

このとき、dn は偶数より n は偶数となるが、$(d+1)$, $(d-1)$ もまた偶数なので $(*)$ より明らかに m は偶数となる。したがって、頂点の増加数が2であることより $m/2$ 回実行すればよい。また、生成するグラフの最小連結成分は頂点数が $(d+1)$ であるから、長さ d の点素な道を選ぶことは可能である。

以上より、$(*)$ を満たすような k, l, m を求め、これにしたがって k 個の K_{d+1} を用いてクリーク挿入法を l 回、頂点挿入法を d が偶数なら m 回、奇数なら $m/2$ 回実行すれば任意の頂点数および次数の正則グラフが生成できることが証明される。

■

以上**定理 4.1,4.2** より、クリーク挿入法と頂点挿入法によって正則性を失わずに任意の頂点数と次数を持つ正則グラフを生成できることが証明される。

定理 4.3 枝交換法の実行によって正則性が失われない。

証明

定義 2.1 より、交互閉路 AC_G に含まれる各頂点においては、G の枝である部分 (α) とそうでない部分 (β) が同数ずつ存在するので、α と β を交換しても各頂点の次数は不変であり、正則性が保持される。 ■

定理 4.4 手続き $alt\_path\_search(v, phase, status)$ によって、必ずある頂点 s を始点とする交互閉路 AC_G が構成される。

証明

まず、交互道を探索するための判定条件を

条件1 : $\exists\ \alpha = (u, v) \in_R E_0 - E_\alpha - E'_\alpha$

条件2：$\exists\,\beta=(v,w)\notin_R E_0\cup E_\beta\cup E'_\beta$
と呼ぶことにする。

$alt\_path\_search(v,phase,status)$ は**定義2.1**で示したような交互道 AP_G を探索する再帰的手続きである。ここで、**条件1,2**を満たす交互道 AP_G が必ず存在することは明らかなので、AP_G の探索が必ず始点 s に戻ってくることをいえばよい。

頂点の次数は d であるから、各々の頂点に対してその頂点に接続していない頂点の数は $(n-d-1)$ である。したがって、$d\leq n-d-1$ ならば**条件2**を満たしかつ始点 s に戻ってくるような枝が存在することは明らかである。

次に、$d>n-d-1$ となる場合について考える。このとき、s に戻ってくる枝βが存在しないのは途中で s を通る AP_G がすべて $\{\alpha=(v,s),\beta=(s,x)\}$ となる場合である。ところで、バックトラックの際にα,β は各々E'_α,E'_β に戻されることになるが、戻された枝を (v,w) としたとき、再びこの枝を逆方向に $(w\rightarrow v$ の方向へ$)AP_G$に加えることを許している。したがって s を通る AP_Gが戻されると、(s,x) は $x\rightarrow s$ の方向で再び**条件2**を満たす枝の候補となるので、この枝を通れば AP_G は必ず始点 s に戻ることができる。

以上より、必ず交互閉路 AC_G を構成することができる。 ■

定理4.5 枝交換法を $\lfloor e/2\rfloor$ 回実行することによって、ある頂点数と次数を持つ正則グラフを同じ頂点数と次数を持つ任意の正則グラフに変換することができる。

証明

頂点数 n、次数 d である2つの正則グラフ $G_1=(V,E_1)$, $G_2=(V,E_2)$ が与えられたとき、正則性を失わずに枝交換法の逆変換のみを用いて G_1 を G_2に変換するアルゴリズムが存在することを示せばよい。

まず、E_1のみに属する枝の集合 E'_1と E_2のみに属する枝の集合 E'_2、および次数が0でない頂点の集合からなるグラフ$\tilde{G}$を定義する。

定義4.1 $\tilde{G}=(\tilde{V},E'_1\cup E'_2)$;
$E'_1=E_1-(E_1\cap E_2)$, $E'_2=E_2-(E_1\cap E_2)$;
$deg(v)=0,\ \forall v\in V-\tilde{V}$ ■

また、**定義2.1**と同様に$\tilde{G}$における**交互道**および**交互閉路**を定義する。

定義4.2 ($\tilde{G}$における交互道 $AP_{\tilde{G}}$、交互閉路 $AC_{\tilde{G}}$)

$$
\begin{aligned}
AP_{\tilde{G}} &= \{(\alpha_i,\beta_i)\mid \alpha_i=(u_i,v_i)\in E'_1,\\
&\qquad \beta_i=(v_i,w_i)\in E'_2;\ \alpha_i\neq\alpha_j,\ \beta_i\neq\beta_j;\\
&\qquad 1\leq i,j\leq k,\ i\neq j\}\\
&: \textit{alternating path in } \tilde{G}\\
AC_{\tilde{G}} &= \{(\alpha_i,\beta_i)\mid \{(\alpha_i,\beta_i)\}\textit{ is } AP_{\tilde{G}};\\
&\qquad 1\leq i\leq k;\ w_k=u_1\}\\
&: \textit{alternating cycle in } \tilde{G}
\end{aligned}
$$

■

2つの定義より、**性質2.1,2.2**と同様に$\tilde{G}$について以下のような性質が成り立つ。

性質4.1 $\tilde{G}$の各頂点には E'_1に属する枝 E'_2に属する枝が同数ずつ接続している。

性質4.2 $\tilde{G}$における最小連結成分は頂点数4、枝数4である。これを$\tilde{G}_{min}$と呼ぶ。

ところで、G_1を G_2に変換するためには2つのグラフにおいて枝が異なる部分のみを操作すればよいから、$\tilde{G}$について枝交換法の逆変換を実行すればよいことになる。したがって、G_1を G_2に変換するアルゴリズムは以下のようになる。

```
algorithm transform_G1_to_G2;
(1)  E12 := E1 ∩ E2;
(2)  while (E'1 ≠ ∅) do
(3)    適当なG̃の頂点 s を選ぶ;
(4)    alt_path_search_G̃(s);
(5)    while (last ≠ s) do
(6)      alt_path_search_G̃(last);
(7)    end
(8)  end
end;
procedure alt_path_search_G̃(v);
  α = (u,v) ∈ E'1かつβ = (v,w) ∈ E'2なる
  (α,β) を求める;
  αを E'1より削除する;
  βを E'2より削除し、E12に加える;
  last := x;
end;
```

■

以後、このアルゴリズムの正当性について論じる。

補題1 手続き $alt_path_search_{\tilde{G}}$によって、必ずある頂点 s を始点とする交互閉路 $AC_{\tilde{G}}$が構成される。

証明

$alt_path_search_{\tilde{G}}$が繰り返されると、$\tilde{G}$において頂点 s から出発し、$\alpha\in E'_1$かつ$\beta\in E'_2$なる交互道 $AP_{\tilde{G}}(\alpha,\beta)$ を再帰的に探索していくことになる。ところで**性質4.1**より、始点 s 以外の各頂点においてはαから入りβから出ていく(あるいはその逆)ような AP_Gが必ず存在する。したがってこの探索は s にた

どりつくまで終了しない。$\tilde{G}$において枝の数は有限であることに注意すれば、$last = s$となったときには必ずsを始点とする$AC_{\tilde{G}}$が構成されることになる。

■

補題 2 行(2)～(8)の**while**ループ(L_{out}と呼ぶ)の繰り返しにおいて、$\tilde{G}$の持つ性質は不変である。

証明

補題1より、$last = s$となるときsを始点とする$AC_{\tilde{G}}$が構成できるので、これに含まれる各頂点においてE'_1に属する枝とE'_2に属する枝が各々同数ずつ削除されることになる。したがってL_{out}の繰り返しにおいては常に$\tilde{G}$の性質が保持される。 ■

補題 3 アルゴリズム$transform_G_1_to_G_2$の実行によって、G_1がG_2に変換される。

証明

行(1)で変数E_{12}は$(E_1 \cap E_2)$に初期設定される。いま$G = (V, E_{12} \cup E'_1)$なるグラフ$G$を考えると、初期状態では$E_{12} \cup E'_1 = E_1$より$G = G_1$となる。**補題1**より、$alt_path_search_{\tilde{G}}$によって$AC_{\tilde{G}}$が構成されるが、これには**性質4.2**より$E'_1$に属する枝が少なくとも2つ含まれるので、$E'_1$の要素数は少なくとも2つずつ減少する。また**補題2**より$\tilde{G}$の性質は変化しないので、必ず$E'_1 = \emptyset$となる。このとき$E_{12} = (E_1 \cap E_2) \cup E'_2 = E_2$であるから、$E_{12} \cup E'_1 = E_2$より$G = G_2$となる。

したがって、$transform_G_1_to_G_2$によってG_1がG_2に変換されることになる。 ■

補題 4 L_{out}の繰り返し回数は、高々$\lfloor e/2 \rfloor$回である。

証明

いま、$\tilde{G}$が**性質4.2**における$\tilde{G}_{min}$だけを成分に持つと仮定する。このときeは偶数であり、G_1とG_2の枝がすべて相異なるとすれば、$|E'_1| = |E'_2| = e$より$|E'_1 \cup E'_2| = 2e$である。したがって、$\tilde{G}$における$\tilde{G}_{min}$の個数は$2e/4 = e/2$となる。ところで**補題2**より、L_{out}が1回繰り返されるたびに$\tilde{G}_{min}$が少なくとも1つずつ減少するので、L_{out}は最大$e/2$回繰り返されることになる。

またeが奇数のとき、$\tilde{G}$における$\tilde{G}_{min}$の個数の最大値は$(2e-6)/4 = (e-3)/2$となる。このとき残りの枝によって長さ6の$AC_{\tilde{G}}$が構成されるので、eが偶数の場合と同様にL_{out}は最大$(e-3)/2+1 = (e-1)/2$回繰り返される。

したがって両方の場合を併せれば、L_{out}は高々$\lfloor e/2 \rfloor$回しか繰り返されない。 ■

以上**定理4.3,4.4**および**補題1,2,3**より、枝交換法によって任意の正則グラフが生成できることが証明される。また、**補題4**より、枝交換法の実行回数は高々$\lfloor e/2 \rfloor$回でよいことも証明される。したがってアルゴリズムにおいて、$r = \lfloor e/2 \rfloor$となる。 ■

5 時間計算量の評価

本節では、提案した生成アルゴリズムの時間計算量を評価する。ここで、頂点や枝をランダムに選択するのにかかる時間は$O(1)$であると仮定している。

まず、サブルーチンの各手続きにおける時間計算量は以下のようになる。

- *insert_clique*

$create\_clique(k)$ ではクリーク K_kを作るのに$O(k^2)$時間かかる。したがって、クリークK_{d-1}を挿入するのだから全体では$O(d^2)$の時間がかかることになる。ところで、この手続きによって頂点は$(d-1)$増加するので頂点1つあたりの効率は$O(d)$である。

- *insert_vertex_even*

まず、$select_disjoint_edges(k)$の効率を求める。$EDGESET$に線形リストを用いれば、ランダムに選んだ枝の端点に接続している枝を探して$EDGESET$から削除するのに$O(d)$時間かかる。これをk回繰り返すのだからこの手続きは$O(dk)$時間かかる。したがって、互いに素な枝を$d/2$本選ぶのだから全体では$O(d \cdot d) = O(d^2)$の時間がかかることなる。

- *insert_vertex_odd*

$select_path(v, k)$も$select_disjoint_edges(k)$と同様にランダムに選んだ道に含まれる頂点vに接続している枝を探して$EDGESET$から削除するのだから、この操作には$O(d)$時間かかる。これをk回繰り返すのだから、$select_path(v, k)$の効率は$O(dk)$となる。したがって、長さdの点素な道を選ぶのだから全体では$O(d \cdot d) = O(d^2)$の時間がかかることになる。

ところで、頂点挿入法では頂点は1つ(2つ)しか増加しないので、頂点1つあたりの効率は$O(d^2)$である。したがって頂点挿入法よりクリーク挿入法の方が効率が良いので、できるだけクリーク挿入法の実行回数を多くする方がアルゴリズム全体の効率が良くなることがわかる。実際のアルゴリズムではこれを$m = (n - k(d+1)) \mod (d-1)$とすることで実現している。

- *exchange_edges*

alt_path_searchにおいて、$E_\alpha, E_\beta, E'_\alpha, E'_\beta$に線形リストを用いれば、枝の削除に$O(n)$の時間がかか

る。また、この手続きにおいては G の枝 (u,v) に対して、高々$u \to v$, $v \to u$ の2回しか探索を行なわないので、G の枝すべてを探索してもその回数は高々$2e$ 回である。したがって、手続き全体の効率は $O(n \cdot 2e) = O(ne)$ となる。

以上の結果を用いて、生成アルゴリズムの時間計算量を求める。各々の手続きの実行回数は

$$
\begin{aligned}
&\textit{insert\_clique} &&: l = \frac{n - k(d+1) - m}{d-1} \\
&\textit{insert\_vertex\_even} &&: m = \left(n - k(d+1)\right) \\
& && \qquad mod\ (d-1) \\
&\textit{insert\_vertex\_odd} &&: m/2 \\
&\textit{exchange\_edges} &&: r = \lfloor e/2 \rfloor = \lfloor dn/4 \rfloor
\end{aligned}
$$

である。

ここで、グラフの頂点を増やすのに最も時間がかかる場合について考える。頂点1つあたりの計算量は頂点挿入法の方が多く、$0 \leq m \leq d-2$ であるから $m = d-2$ のとき頂点を増加させるのに最も時間がかかると考えられる。また、最初の K_{d+1} の数が少ないほど l および m が大きくなる。したがって、$k=1$ とすれば $l = (n-2d+1)/(d-1)$ となるから、このようなとき最も時間がかかると考えてよい。

これにしたがってアルゴリズム全体の時間計算量を求めると、

$$O(l \cdot d^2 + m \cdot d^2 + r \cdot ne) = O(dn + d^3 + d^2 n^3)$$
$$= O(d^2 n^3) = O(ne^2)$$

となる。結局、アルゴリズムの実行時間のうちほとんどが枝交換法にかかる時間であることがわかる。

6 むすび

本稿では、頂点数 n および次数 d が与えられたとき、任意の d 次正則グラフを生成する $O(ne^2)$ 時間 ($e = dn/2$: 枝数) のアルゴリズムを提案した。そしてこのアルゴリズムにおいて、クリーク挿入法と頂点挿入法によって任意の頂点数と次数の正則グラフを生成できることと、枝交換法を $\lfloor e/2 \rfloor$ 回実行することによって、このグラフを同じ頂点数と次数を持つ任意の正則グラフに変換できることを示した。

ところで、暗号理論においてはグラフ同形判定問題がその解き難さを安全性の根拠として利用されている [2]。そこで、このアルゴリズムをグラフ同形判定問題における問題例生成アルゴリズムとして応用することが考えられる。この際の安全性に対する問題点として、著者らは [7] において生成される正則グラフが連結であることの必要性を指摘している。

そこで、今後の課題として連結な正則グラフを効率良く生成するようにアルゴリズムを改良することが挙げられる。また、このアルゴリズムを使って正則グラフに対する同形判定問題の難しさを評価することも考えている。

参考文献

[1] M.R.Garey, D.S.Johnson, "Computers and Intractability; A Guide for the Theory of NP-Completeness", *W.H.Freeman & Co.* (1979)

[2] O.Goldreich, S.Micali, A.Wigderson, "Proofs that Yield Nothing But their Validity and a Methodology of Cryptographic Protocol Design", *IEEE Annual Symposium of Foundations of Computer Science*, pp.174-187 (1986)

[3] M.Jerrum, A.Sinclair, "Fast Uniform Generation of Regular Graphs", *Theoretical Computer Science*, Vol.73, pp.91-100 (1990)

[4] N.C.Wormald, "Generating Random Regular Graphs", *Journal of Algorithms*, Vol.5, No.2, pp.247-280 (1984)

[5] M.Behzad, G.Chartrand, L.Lesniak-Foster / 秋山 仁, 西関 隆夫 訳, "グラフとダイグラフの理論", 共立出版 (1981)

[6] 松田, 堀内, 榎原, 中野, "正則グラフの生成に関する一考察", 第14回情報理論とその応用シンポジウム予稿集, pp.5-8 (1991)

[7] 松田, 榎原, 中野, 堀内, "グラフ同形問題のためのランダムな正則グラフ生成について", 信学技報, ISEC-92 (1992)