アルゴリズム 84－1
（2002．5．23）

グラフ実現問題に対する Bixby-Wagner アルゴリズムの実験的解析

大戸 隆広

東京大学大学院情報理工学系研究科

線形計画問題の制約式の係数行列がグラフの基本サーキット行列であるとき, この問題は輸送問題に帰着され, より高速に解くことができる. そこで, 与えられた {0, 1}-行列が基本サーキット行列となるかを判定する問題 (グラフ実現問題) を高速で解くことが重要となる. グラフ実現問題を効率的に解くためのデータ構造として藤重による PQ-グラフ [4] と Bixby, Wagner[3] による t-分解がある. これらのデータ構造を用いて, グラフ実現問題は入力行列の非零要素数にほぼ比例した時間で解くことができる. 本研究では, t-分解によるアルゴリズムを実装し, グラフ実現問題がほぼ線形時間で解けることを確認する. さらに, ランダムに生成された {0, 1}-行列がグラフ的になる確率を調べ, それに対する考察を行う.

An Experimental Analysis of the Bixby-Wagner Algorithm for Graph Realization Problems

Takahiro OHTO

Graduate School of Information Science and Technology,
The University of Tokyo

A linear programming problem can be solved efficiently by reducing it to a transshipment problem if the constraint matrix is a fundamental circuit matrix of a certain graph. This leads to deciding whether a given {0, 1}-matrix is a fundamental circuit matrix. The PQ-graph (Fujishige[4]) and the t-Decomposition (Bixby, Wagner[3]) are data structures used to solve the problem in almost linear time in the number of ones in the given matrix. This paper reports on implementation of the Bixby, Wagner algorithm. It is confirmed that the actual running time is almost linear. Using this implementation, we further investigate how often a random generated {0, 1}-matrix is graphic.

1 はじめに

グラフ $G = (V, E)$ および全域木 T が与えられたとする. このとき, 辺 $e \in E - T$ を T に追加すると, $G[T \cup \{e\}]$ にはサーキットが 1 つ含まれる. これを基本サーキットという. 基本サーキット系は {0, 1}-行列を用いて表すことができる. この行列を基本サーキット行列という.

線形計画問題の制約式の係数行列が基本サーキット行列となっているとき, この問題を輸送問題に変換することができる. 輸送問題は線形計画問題より速く解くことができるアルゴリズムが知られている. 実際, Bixby, Cunningham[2] によると 1980 年当時のプログラムでは, 輸送問題に変換することによってもとの線形計画問題を 50 倍から 200 倍速く解けている. そこで, {0, 1}-行列が基本サーキット問題であるかをできるだけ速く判定することが重要となる. 現在, この問題をもっとも速く解くアルゴリズムに藤重 [4] によるものと Bixby, Wagner[3] によるものがあり, どちらも入力行列の非零要素数にほ

ぼ比例した時間で終了する. 藤重のアルゴリズムは Ali, Han[1] によって実装されている. そこで本研究では, Bixby, Wanger によるアルゴリズムを Java により実装した. さらに, そのプログラムを使って, このアルゴリズムがほぼ線形時間で解けることを確認した. また, ランダムに生成された $\{0, 1\}$-行列がどれくらいの確率で基本サーキット行列となるかを調べ, それに対する考察を行った.

本稿の構成は以下の通りである. 第 2 節で必要となる用語を説明し, グラフ実現問題を定義する. さらに, グラフ実現問題を解く素朴なアルゴリズムを提示する. 第 3 節で t-分解を定義し, それをもとにしたアルゴリズムの概要を紹介する. また, 具体的な計算時間を紹介する. 第 4 節では実装したプログラムの妥当性を示し, そのプログラムを用いて行った実験の結果を紹介する. 最後に第 5 節でまとめを行い, 今後の課題を述べる.

2 グラフ実現問題

2.1 準備

頂点集合 V, 辺集合 E のグラフを $G = (V, E)$ と表す. 両端が同じ頂点となる辺をループといい, ループではなく同じ両端を持つ辺を並列辺という. A を E の部分集合とする. このとき, A を辺集合, A に含まれる辺の端点集合を頂点集合とする G の部分グラフを $G[A]$ と表す. また, グラフ G の頂点集合, 辺集合をそれぞれ $V(G), E(G)$ と表す.

グラフ G の任意の 2 頂点間にパスが存在するとき, グラフ G は連結グラフである (連結である) という. さらに, グラフ G から任意の $k-1$ 点 $(k \geq 2)$ を除去してできるグラフが常に連結グラフとなるとき, グラフ G を k-連結グラフという. また, 連結グラフ $G = (V, E)$ の辺集合を $E_1, E_2 \neq \emptyset$, $E_1 \cap E_2 = \emptyset$, $E_1 \cup E_2 = E$ を満たすような集合の組 $\{E_1, E_2\}$ に分ける. $|V(G[E_1]) \cap V(G[E_2])| = k$ が成り立つとき, $\{E_1, E_2\}$ をグラフ G の 2-分離という. 3-連結グラフには 2-分離が存在しないことが定義よりすぐに確認できる.

2-連結グラフ G の 2-分離を $\{E_1, E_2\}$ とする. さらに, $V(G[E_1]) \cap V(G[E_2]) = \{u, v\}$ とする. G において, 図 1 のように $G[E_1]$ と $G[E_2]$ のつながりを逆にしたグラフを G' と表す. このとき G' を, $G[E_1]$ を反転させることによって G から得られるグラフという. G'' が G から何回かの部分グラフの反転によって得られるとき, G'' は G の 2-同型グラフであるという. 3-連結グラフでは 2-分離が存在しないため, 新しい 2-同型グラフを作ることは出来ない.

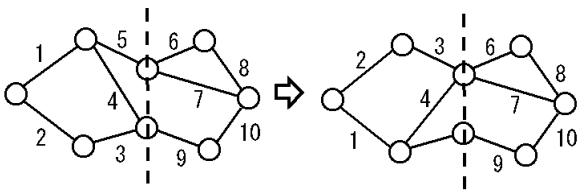

図 1: 部分グラフの反転

2-同型グラフに関して, 以下の重要な定理が成り立つ.

定理 1. G と G' は同じ辺集合を持つ分離不可能グラフとする. このとき, G と G' が 2-同型グラフであることの必要十分条件は, G と G' が同じ閉路集合を持つことである. ■

2.2 問題設定

グラフ $G = (V, E)$ および全域木 T が与えられたとき, 各辺 $e \in E - T$ に関して, グラフ $G[T \cup \{e\}]$ はただ 1 つのサーキットを含む. これを e に関する T の基本サーキットといい, $C(T, e)$ と表す. さらに, 基本サーキット系により $\{0, 1\}$-行列 M を次のように定義できる. まず, M の行集合を T に, 列集合を $E - T$ に対応させ, M の (j, k)-成分 $m_{j,k}$ を,

$$m_{j,k} = \begin{cases} 1 & j \in C(T, k) \text{ のとき,} \\ 0 & \text{その他,} \end{cases}$$

と定義する. このように定義される行列 M を T に関する G の基本サーキット行列という. 例えば, 図 2 のグラフ G において, $T = \{1, 2, 3, 4\}$ とすると, 基本サーキット行列は図 3 で表される.

また逆に, $\{0, 1\}$-行列 M が与えられたとき, M を基本サーキット行列に持つグラフが存在するかを

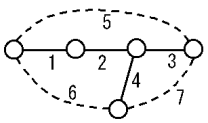

図 2: グラフ G

判定し, 存在する場合にはそのグラフ (の 1 つ) を実現する問題をグラフ実現問題という. グラフ実現可能な行列をグラフ的であるという.

そこで, グラフ実現問題の素朴なアルゴリズムを考える. $r \times c\{0,1\}$-行列 M が与えられたとする. M がグラフ的であろうがなかろうが, M の各行は木辺に, 各列は補木辺に対応していると考える. また, 補木辺が 1 つ決まると基本サーキットが 1 つ決まることより, 各列が基本サーキットに対応していると考えることもできる. よって, M には 0 だけからなる行や列がないと仮定してよい. M の第 1 列から第 k 列による部分行列を M_k と表す. M の第 k 列 $(1 \leq k \leq c)$ の表す基本サーキットに含まれる辺集合を C_k と表す. さらに,

$$P_k = \begin{cases} C_k & k = 1 \text{ のとき}, \\ C_k \cap \{\bigcup_{1 \leq j < k} C_j\} & 1 < k \leq c \text{ のとき}, \end{cases}$$

とする. 任意の k に対し, $P_k \neq \emptyset$ となるとき, M は逐次連結であるという. 任意の $\{0, 1\}$-行列は, 列を並び替えることによって, 逐次連結な小行列のブロックに並び替えることができる (図 4).

そこで, 以下では逐次連結な行列のみを扱う.

グラフ実現問題に関し, 以下の定理が成り立つ.

定理 2. ある k $(1 \leq k < c)$ に対し, M_k がグラフ的であり, それを実現したグラフを G_k とする. このとき, M_{k+1} がグラフ的であることの必要十分条件は, P_{k+1} が G_k のハイポパスとなることである. ■

ただし, グラフ G において, 辺部分集合 P をパスに持つような G の 2-同型グラフが存在するとき, P をハイポパスという.

この定理を利用して, 次のようなアルゴリズムを考えることができる.

(S1) サーキット C_1 だけからなるグラフ G_1 を作る. $i = 1$ とする.

(S2) $i = c$ のとき, アルゴリズムは停止する. このとき M はグラフ的となる.

	5	6	7
1	1	1	0
2	1	1	0
3	1	0	1
4	0	1	1

図 3: 基本サーキット行列

	6	7	8	9	10
1	0	0	0	1	0
2	1	0	0	0	0
3	1	0	0	0	1
4	0	0	1	0	1
5	0	1	0	1	0

↓

	6	10	8	9	7
1	0	0	0	1	0
2	1	0	0	0	0
3	1	1	0	0	0
4	0	1	1	0	0
5	0	0	0	1	1

図 4: 逐次連結な行列への変換

(S3) G_i の 2-同型グラフのうち, P_{i+1} がパスとなるようなグラフ G_i' を見つける. このようなグラフが存在しない場合, アルゴリズムは停止する. このとき M はグラフ的ではない.

(S4) P_{i+1} からなるパスの両端を, $C_{i+1} - P_{i+1}$ からなるパスでつなぐ. こうしてできるグラフを G_{i+1} とする.

(S5) $i \leftarrow i + 1$ として, (S2) にもどる.

このアルゴリズムが正しいことは明らかだが, (S3) を実行するのに 2-同型グラフを表す効率のよいデータ構造が必要となる. そのようなデータ構造として, PQ-グラフ (Fujishige[4]), t-分解 Bixby, Wagner[3]) がある. 今回は, t-分解を用いたアルゴリズムについて説明する.

3 t-分解

3.1 定義

まず, t-分解で必要となる 3 種類のグラフを定義する. 2 頂点とそれらを結ぶ 1 本以上の辺からなるグラフを bond という. 3 本以上の辺を持つ閉路で表される連結グラフを polygon という. 3-連結となるグラフを prime という.

次に分解というデータ構造を定義する. D を 2-連結グラフの有限集合とする. D の要素を頂点とし, 頂点を表すグラフが辺を共有しているときに限り頂

点間に弧が存在するような有向グラフを T とする. T が根付き有向木になっていて, D の任意の 2 要素が高々1 本の辺を共有し, 頂点を全く共有しないとき, D を分解という. $H, K \in D$ に対し $E(H) \cap E(K)$ が存在するとき, その要素 e を H と K のマーカーという. 特に, $K = p(H)$ (H の親) のとき, e を K の子マーカーという. また, e は H の親マーカーともいい, $q(H)$ と表す. また, $r(D) = r(T)$ とする.

$K = p(H)$ のとき, $e = q(H)$ の, H に含まれる両端点を u_1, u_2, K に含まれる両端点を v_1, v_2 とする. このとき, 1 対 1 対応の写像 $f : \{u_1, u_2\} \to \{v_1, v_2\}$ を H の K に対する方向付けという. このような関数 f が与えられたとき, H は K に対して有向であるという. f を変えることを H のマーカーを反転するという. 分解 D のすべての子がその親に対し有向であるとき, D は有向であるという.

有向分解 D が,

1. D のすべての要素が, 3 本以上の辺を持ち, polygon, bond, prime のいずれかである,
2. bond だけが親マーカーに並列な辺を持つ,
3. polygon 同士が隣接することはなく, また bond 同士が隣接することはない,

を満たすとき, D を t-分解という.

$H, K \in D$, $K = p(H)$, $e = q(H)$ とし, H に含まれる e の両端を u_1, u_2 とする. このとき, u_i と $f(u_i)$ $(i = 1, 2)$ を同一視し, 両方のグラフから e を除去して新しいグラフを作ることを H と K を合併するといい, そのグラフを $m(H, K)$ と表す. H, K を $m(H, K)$ に置き換えることにより, 新しい t-分解 D' が得られる. この操作をすべてのマーカーに対して繰り返すことによって, 1 つのグラフを作ることが出来る. これを D の合併グラフといい, $m(D)$ と表す. マーカーを反転させることや polygon の要素を並び替えることによって別の合併グラフが得られるが, こうして作られるグラフはすべて 2-同型となる. これは定理 1 よりすぐに確認できる.

3.2 アルゴリズムの概要

t-分解を用いたアルゴリズムは, 基本的に先ほどの素朴なアルゴリズムと同じで, 2-同型グラフを効率的に表現するために t-分解を用いている. アルゴリズムは大きくわけて, 2 つのステップからなる.

1 つ目のステップは (S3) に対応するもので, HYPOPATH とよばれる. このステップでは入力として, t-分解 D と辺集合 $P \subseteq E(m(D))$ が与えられる. そして出力として, P が $m(D)$ のパスとなり, $m(D)$ が入力の $m(D)$ と 2-同型となるような新しい t-分解 D を返すか, またはグラフ実現不可能と結論を下す.

2 つ目のステップは (S4) に対応するもので, UPDATE とよばれる. このステップでは入力として, t-分解 D, $m(D)$ のパス P, $C \cap E(m(D)) = P$ かつ $C - P \neq \emptyset$ を満たす辺集合 C が与えられる. そして出力として, $m(D)$ に辺集合 $C - P$ を加えて C がサーキットとなるようにしたグラフを表す t-分解 D^* を返す. これら 2 つのステップをもとに, 以下のようなアルゴリズムを記述できる.

(G1) C_1 に含まれる辺の 1 つ (選び方は任意でよい) と辺 r からなる bond を G_0 とする. $D_0 \leftarrow \{G_0\}$ とする. $i \leftarrow 1$ とする.

(G2) $P \leftarrow P_i$ および $D \leftarrow D_{i-1}$ として HYPOPATH を実行する.

(G3) $C \leftarrow C_i$ として UPDATE を実行する. $D_i \leftarrow D^*$ とする.

(G4) $i = c$ のとき, $G \leftarrow m(D_i) - \{r\}$ とし, アルゴリズムは停止する. それ以外のとき, $i \leftarrow i + 1$ とし, (G2) に戻る.

それでは, 2 つのステップ HYPOPATH, UPDATE の概要を説明する. 詳細は元論文 [3] を参照されたい.

HYPOPATH: P の要素を全て含む極小な t-分解を D の簡約 t-分解といい $\widehat{D}$ と表す. $\widehat{D}$ において, $r(\widehat{D})$ から遠いところにある頂点から順に, P の要素がパスになるようにマーカーの反転および polygon 辺の順番の並び替えを行う. そのような変形ができないときはアルゴリズムは停止し, グラフ的ではない. $r(\widehat{D})$ まで変形し終えたら, 変形後の t-分解を返す.

UPDATE: P の両端を u_1, u_2 とし, u_i を含むグラフを K_i とする. K_1 と K_2 を両端とする D 上のパスを R とする. R に含まれる polygon を, P に含まれる辺からなる polygon とそれ以外の辺からなる polygon に分割する. R に含まれるグラフを統合し, u_1, u_2 を新しい辺 f で結ぶ. $\{f\} \cup (C - P)$ からな

	9	10	11	12
1	1	0	0	0
2	1	1	0	1
3	1	0	0	0
4	1	0	1	1
5	0	1	0	0
6	0	1	0	1
7	0	0	1	0
8	0	0	0	1

図 5: 入力された行列

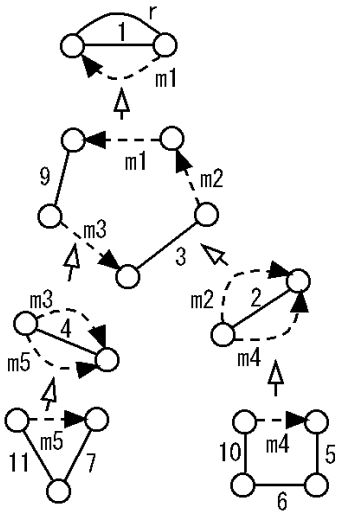

図 6: 列 11 までの t-分解

る polygon を D に追加する. ただし, $|C-P|=1$ のときは u_1, u_2 を $\{C-P\}$ で結び, polygon は追加されない. (以上の変形で必要に応じてマーカーを挿入する.) こうして作られた t-分解を D^* として返す.

例として図 5 で表される行列をアルゴリズムに入力として与える. このとき, 列 14 までを実行してできる t-分解は図 6 となる. 次に, 列 15 の HYPOPATH を実行して出来る t-分解が図 7 であり, これに UPDATE を実行したものが図 8 である. よって, この行列の実現グラフは図 9 となる.

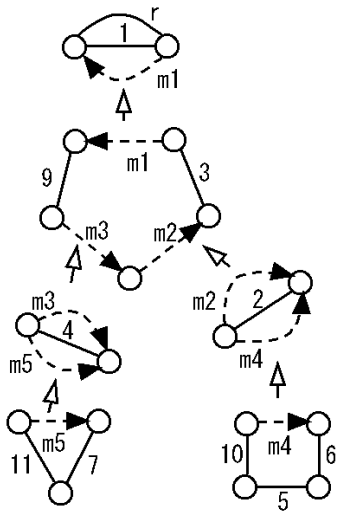

図 7: HYPOPATH 実行後の t-分解

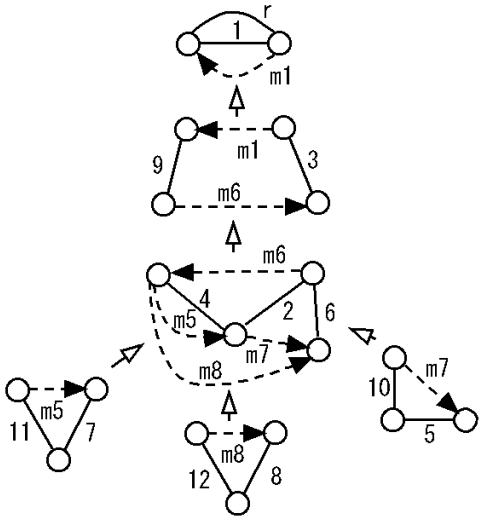

図 8: UPDATE 実行後の t-分解

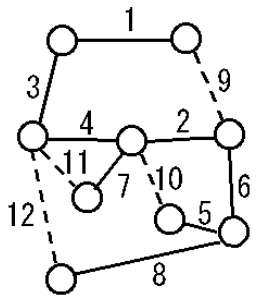

図 9: 実現されたグラフ

3.3 計算時間

n 個の非零要素を持つ $r \times c\{0,1\}$-行列が入力として与えられたとき, このアルゴリズムの実行時間は $O(n\alpha(n,r))$ であることが Bixby, Wagner[3] により示されている. ここで $\alpha(\cdot,\cdot)$ は Ackermann 逆関数と呼ばれる関数で, Ackermann 関数 $A(m,n)$

$$
\begin{aligned}
&A(1,j) = 2^j && (j = 1,2,3,\ldots),\\
&A(i,1) = A(i-1,2) && (i = 2,3,4,\ldots),\\
&A(i,j) = A(i-1,A(i,j-1)) && (i,j = 2,3,4,\ldots),
\end{aligned}
$$

を用いて,

$$\alpha(m,n) = \min\{i \geq 1 \mid A(i,\lfloor m/n \rfloor) > \log n\}$$

として定義される.

$\alpha(m,n)$ は m に関して減少関数, n に関して非常にゆっくりと増加する関数となっている. 実際, $n < 65536$ なら常に $\alpha(m,n) \leq 3$ となり, 現実の問題では定数と見なすことが出来る. よって, グラフ実現問題を入力行列の非零要素数にほぼ比例した時間で解くことが出来る [5].

4 計算機実験

4.1 プログラムの妥当性

t-分解を用いたアルゴリズムを Java により実装した. このプログラムが正しく動くことを確認するために, 以下のテストを行った.

1. グラフ的な行列を作成し, それがグラフ的と判定されることを確認する.
2. {0,1}-行列を入力し, それがグラフ的ならば, 入力された行列が出力されたグラフの基本サーキット行列となることを確認する.

1 番目のテストについて詳細を記述する. まず, 必要な辺数を持つ木をランダムに作成する. その任意の 2 点を選び, それらを結ぶパスに使われる辺により列を決定する. これを必要な列の数だけ繰り返す. こうして作られた行列をプログラムに入力する. 行数 100 以下, 列数 1000 以下の行列をランダムに 100 万個作成し, それらに対し正しく動くことを確認した.

次に 2 番目のテストについて詳細を記述する. まず, 一定の確率にしたがって 1 を出現させた逐次連結行列を作成する. 次にこの行列を入力としてプログラムを実行する. k 列目を付け加えようとしたときグラフ的でなくなったならば, $k-1$ 列目までからなるグラフ的な部分行列を入力として与え, 得られたグラフに i 列 $(1 \leq i \leq k-1)$ に対応するサーキットが含まれることを確認する. 行数 100 以下, 列数 1000, 1 の出現確率 p $(0 < p < 100)$ の行列をランダムに 100 万個作成し, それらに対し正しく動くことを確認した.

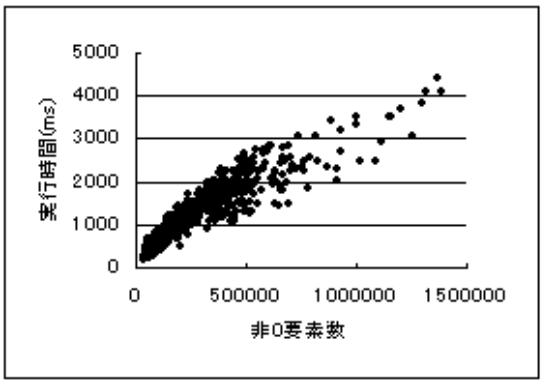

図 10: 実行時間

4.2 実行時間

第 4.1 節で作成したグラフ的行列について実行時間を測定した. 行数が 1000, 2000,..., 5000 の 5 通り, 列数が 5000, 10000,..., 50000 の 10 通りの計 50 通りのサイズの行列についてそれぞれ 10 個の行列 (計 500 個) を調べた. 測定に用いた計算機は PentiumIII 600MHz, 128Mb, WindowsMe のノート PC である. 各行列の非零要素数と実行時間 (ms) をグラフで表したものが図 10 である. これにより, 確かにほぼ線形でグラフ実現問題を解けることが確認された.

4.3 グラフ的可能性

ランダムに {0,1}-行列を作成したとき, それがどれくらいの確率でグラフ的になるかを測定した. 入力行列はランダムな {0, 1}-行列を与える必要があるので, まずその行列を逐次連結となるように列を入れ替える必要がある. そのために, {0, 1}-行列から行, 列のインデックスを頂点集合とし, 成分 が 1

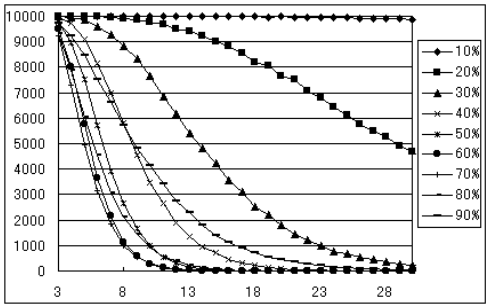

図 11: 行数 5

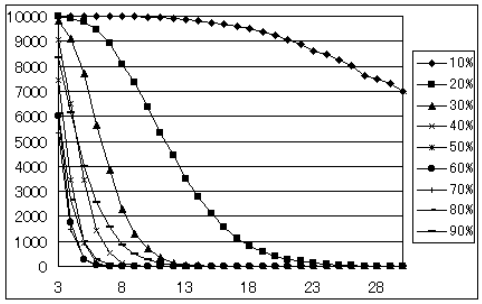

図 12: 行数 10

となる組に辺をつないだ 2 部グラフを作成する. そのグラフを幅優先探索を行うことによって逐次連結に変換した.

測定に用いた行列は, 行数が 5, 10, 50, 列数が 3,4,...,100, 1 の出現確率が 10,20,...,90% であり, それぞれに対し 10000 個の行列を調べた. その結果をグラフにしたものが図 11(行数 5), 図 12(行数 10), 図 13(行数 50) である. ここで, 横軸は入力行列の列数, 縦軸はグラフ的な行列数である. ただし, 列数が多いとほとんどグラフ的とならないため, 列数 30 以下のデータのみを表示した. また, 行数 50 については 10, 50, 90% のみを表示した. このグラフより, 非零要素の密度が高くなっていくにつれてグラフ的となる確率が下がっていき, 70%前後を境目に上がっていくことが読み取れる. これに対する考察を行う.

まず, グラフ的な行列について以下の事実が知られている.

定理 3. グラフ的行列の任意の小行列はグラフ的である. ■

つまり, ある行列が非グラフ的行列を小行列として含んでいるとき, その行列はグラフ的ではない. そこで, 極小な非グラフ的行列を禁止小行列と呼ぶ.

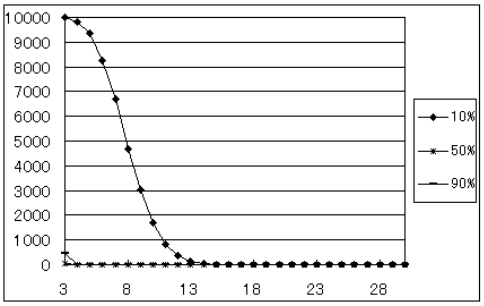

図 13: 行数 50

	4	5	6	7
1	1	1	1	0
2	1	1	0	1
3	1	0	1	1

	5	6	7
1	1	1	1
2	1	1	0
3	1	0	1
4	0	1	1

図 14: 禁止小行列

最も成分数が少ない禁止小行列は図 14 で与えられる. これらの行列の密度は 75%である. そこで, 入力行列の密度が 75%のときこの 2 つの禁止小行列を含む確率が最も高くなると考えられる.

5 まとめ

グラフ実現問題を $O(n\alpha(n,r))$ (n:非零要素数, r:行数) で解くアルゴリズムを紹介した. このアルゴリズムでは 2-同型グラフを効率よく表現するために, t-分解というデータ構造を用いている. さらに, このアルゴリズムを実装し, 非零要素数にほぼ比例した時間で解けることを計算機実験で確認した. また, ランダムに生成された $\{0,1\}$-行列がどれくらいの確率でグラフ的になるかを調べ, それに対する簡単な考察を行った.

今後の課題としては, 禁止小行列を詳しく調べることがあげられる. 今回は最も成分数が少ない 2 つの行列のみを紹介したが, それ以外にもいくつか見つかっている. それらを詳しく調べることによってグラフ的となる確率を最小とする行列の密度を理論的に求められる可能性がある. また, 組合せ的な背景のある線形計画問題に対し, 係数行列がどれくらいの確率でグラフ的になるかを調べたい. さらに, 同じ計算時間を持つ PQ-グラフを用いたアルゴリ

ズムとの比較を行いたい.

参考文献

[1] A. I. Ali and H. -S. Han. Computational implementation of Fujishige's graph realizability algorithm. *European Journal of Operational Research*, Vol. 108 (1998), pp. 452–463.

[2] R. E. Bixby and W. H. Cunningham. Converting linear programs to network problems. *Mathematics of Operations research*, Vol. 5 (1980), pp. 321–357.

[3] R. E. Bixby and D. K. Wagner. An almost linear-time algorithm for graph realization. *Mathematics of Operations Research*, Vol. 13 (1988), pp. 99–122.

[4] S. Fujishige. An efficient PQ-graph algorithm for solving the graph-realization problem. *Journal of Computer and System Sciences*, Vol. 21 (1980), pp. 63–86.

[5] 浅野孝夫. 情報の構造 [上]. 日本評論社, 1994.