Vol.2018-AL-166 No.12
2018/1/29

木幅問題の貪欲解法の巨大グラフを対象とした実験的評価

大塚 広夢[1,a)]　杭田 知樹[1,b)]　佐藤 拓人[1,c)]　玉木 久夫[1,d)]

概要：木幅問題には完全削除順序を用いた，頂点選択のためのいくつかの評価基準に基づく貪欲解法が知られている．Bodlaender と Koster は小さなグラフの多くは，貪欲解法で得られる幅と木幅がほとんど変わらないことを示した．本稿では巨大なグラフに対して実験を行い評価する．

キーワード：巨大グラフ，木分解，木幅問題，完全削除順序，貪欲解法

1. はじめに

グラフ理論において，木分解や木幅は多くの NP 困難な問題を解く際に，よく使われる道具の 1 つである．なぜなら，グラフの木幅が小さいとき，多くの NP 困難な問題は，効率的に解けることが知られているからである．

また，現実のグラフの木幅も案外小さい場合があることが知られている．例えば，C 言語の制御フローグラフは，goto 文が存在しなければ木幅は 6 以下である [1] ことや，LIGAND という化合物の木幅はほとんど 3 以下である [2] ことなどが知られている．

しかし，残念なことに，木幅問題は NP 完全であること [3] が知られている．ここで，木幅問題とは，任意のグラフ G と正整数 k を入力としたとき，G が高々 k の木幅を持つか，という判定問題である．

木幅問題に対して，完全削除順序に着目した貪欲解法はよく知られている．論文 [4] では，Bodlaender と Koster は頂点選択のための 6 つの既知の評価基準について実験を行っている．そして，小さなグラフの多くは，貪欲解法で得られる幅と木幅がほとんど変わらないことを示している．しかし，Bodlaender と Koster は巨大グラフに対しては実験を行なっていない．

本稿では，論文 [4] に記載してある，頂点選択のための 6 つの評価基準のうち 5 つと，新たに考えた評価基準の合わせて 6 つに対して，巨大グラフを対象とし実験を行った．

[1] 明治大学
[a)] ohtsuka_yume@cs.meiji.ac.jp
[b)] t_kuida@cs.meiji.ac.jp
[c)] s_takuto@cs.meiji.ac.jp
[d)] tamaki@cs.meiji.ac.jp

2. 準備

グラフ G とし，その頂点集合 $V(G)$，辺集合 $E(G)$ と表す．G の頂点 $v \in V(G)$ に対して，G における v の開隣接頂点集合を $N_G(v) = \{w \in V(G) \mid \{w, v\} \in E(G)\}$，閉隣接頂点集合を $N_G[v] = N_G(v) \cup \{v\}$ とする．頂点集合 $X \subseteq V(G)$ に対して，X の G における開隣接頂点集合を $N_G(X) = (\bigcup_{v \in X} N(v)) \setminus X$，閉隣接頂点集合を $N_G[X] = N_G(X) \cup X$ とする．

グラフ G の**木分解** $(T, \{X_t : t \in V(T)\})$ とは，木 T と T の各ノード $t \in V(T)$ によって定義される頂点集合 $X_t \in V(G)$ の族で，以下の条件を満たすものである．

(1) $\bigcup_{t \in V(T)} X_t = V(G)$，
(2) 任意の辺 $\{v, w\} \in E(G)$ に対して，$v, w \in X_t$ を満たす $t \in V(T)$ が存在し，
(3) 任意の頂点 $v \in V(G)$ に対して，ノードの集合 $\{t : v \in X_t\}$ は T において連結である．

木分解 T の**幅**とは，全ての T のノード $t \in V(T)$ における X_t の大きさの最大値から一減じた値であり，G の**木幅**とは，G の全ての木分解の幅の最小値である．

頂点集合 $C \subseteq V(G)$ が**クリーク**であるとは，C に属するすべての 2 頂点間に辺が存在することである．

コードとは，閉路の 2 頂点間を結ぶ閉路に含まれない辺のことであり，G が**コーダル**であるとは，G の少なくとも長さが 4 のすべての閉路にはコードが存在するグラフである．

G の**三角化** H とは，G を部分グラフとした，$V(H) = V(G)$, $E(G) \subseteq E(H)$ となるコーダルグラフであり，これらの条件において極小であるとき，H を**極小な三角化**と呼ぶ．

G の**削除順序** π とは，$V(G)$ から $\{1, \ldots, |V(G)|\}$ への全

単射である．G の削除順序 π が**完全**であるとは，任意の頂点 $v \in V(G)$ において，$\{w \in V(G) \mid \{v,w\} \in E(G), \pi(w) > \pi(v)\}$ がクリークとなるものである．

G の**不在辺** $\{x,y\}$ とは，$x,y \in V(G)$ かつ，$\{x,y\} \notin E(G)$ を満たす辺である．

3. 木分解と完全削除順序

この節では，G の木分解と完全削除順序の関連性について述べる．そのとき，論じる上で欠かせない概念としてコーダルグラフを用いる．

本稿では，G と G の削除順序 π を入力として，G の木分解と木幅の上界を求めることを目的としている．

まず，G と G の削除順序 π から，π を完全削除順序として持つような，G の三角化 H を求める．

以下の定理は，G の木分解とコーダルグラフに対してよく知られているものである．

定理 1. G が与えられたとき，以下の 3 つは等価である．

(1) G がコーダルグラフである．
(2) G は完全削除順序を持つ．
(3) G の幅最小の木分解 $(T, \{X_t : t \in V(T)\})$ で，任意のノード $t \in V(T)$ に対して，X_t は G のクリークとなるものがある．

定理 2. G と正整数 k が与えられたとき，以下の 3 つは等価である．

(1) G の木幅は高々 k である．
(2) G は，最大クリークサイズが高々 $k+1$ である三角化 H を持つ．
(3) G は，最大クリークサイズが高々 $k+1$ である極小な三角化 H を持つ．

定理 1 は，コーダルグラフと完全削除順序と木分解の関係性について述べたものである．

定理 2 は，G の木幅の上界は，G を三角化または極小な三角化 H の最大クリークサイズの上界と対応することを述べている．

Algorithm 1 G の三角化

Input: グラフ G， 削除順序 π
Output: π を完全削除順序として持つような G の三角化 H

```
H ← G
for i = 1 to |V(G)| do
  π(u) ← i
  for v, w ∈ N_H(u), v ≠ w, π(u) < π(v), π(u) < π(w) do
    if H において v と w が繋がっていない then
      v と w を繋げる
    end if
  end for
end for
return H
```

また，アルゴリズム 1 は，定理 2 の G の三角化をアルゴリズムとして具体的に書いたものである．入力をグラフ G と削除順序 π とし，出力は，π を完全削除順序として持つような G の三角化 H となるアルゴリズムである．定理 1 の（1）と（2）から，削除順序 π が完全であるようにするため，辺を加えている．

次に，H と完全削除順序 π を入力とし，H の木分解を求める．定理 1 の（1）と（3）から，コーダルグラフの木分解を求めることができるため，その部分を具体的に示したものがアルゴリズム 2 である．

Algorithm 2 H の木分解

Input: コーダルグラフ H， 完全削除順序 π
Output: H の木分解 $(T, \{X_t : t \in V(T)\})$

```
for i = |V(H)| to 1 do
  v ← π^{-1}(i)
  V(T) ← V(T) ∪ {v}
  if i = |V(H)| then
    X_v ← {v}
  else
    w = arg min_{{v,π^{-1}(j)}∈E(H), j∈{1...|V(H)|}} j
    X_v ← N_H[v]
    E(T) ← E(T) ∪ {v, w}
  end if
end for
return (T, {X_t : t ∈ V(T)})
```

アルゴリズム 2 は，入力をコーダルグラフ H と完全削除順序 π とし，出力は H の木分解となるアルゴリズムである．

以上のことから，H の木分解が得られ，それが G の木分解となる．よって，G と G の削除順序 π から，G を三角化したグラフ H と H の完全削除順序 π を求めることができた．ゆえに，G の木分解を求めるためには，削除順序 π をどのように求めるかが重要となる．次の節で詳しく説明する．

4. 削除順序 π を求める貪欲解法

この節では，削除順序 π を求める貪欲解法について説明する．そのとき，削除順序 π は，それに基づく三角化の木幅が，G の木幅と比べてできる限り増えないようにしたい．

4.1 貪欲解法

アルゴリズム 3 は，入力をグラフ G，出力は G の三角化 H と，その完全削除順序 π となる，貪欲アルゴリズムである．H_i とは，完全削除順序 π において，i 番目より前の頂点を削除したグラフである．$N_{H_i}(v)$ をクリークにするために辺を加えることで，H の最大クリークサイズが元のグラフ G よりも大きくなる可能性があり，木幅も大きくなることがある．よって，H の最大クリークサイズを小さくするためには，$N_{H_i}(v)$ はなるべく小さいものを選ぶようにしたい．

そこで，表 1 は，π において i 番目の頂点 v_i の 6 つの評価

Algorithm 3 H の完全削除順序 π

Input: グラフ G
Output: G の三角化 H, H の完全削除順序 π
  $H, H_1 \leftarrow G$
  for $i = 1$ to $|V(G)|$ **do**
    最小ヒープから頂点 v を取り出す
    $\pi(v) \leftarrow i$
    H に対して，$N_{H_i}(v)$ をクリークにする
    各評価基準のヒープの更新候補を求める
    その更新候補に対して，ヒープの中身を更新する
    $V_{i+1} = \{\pi^{-1}(j) \in V(G) \mid i+1 \leq j \leq |V(G)|\}$
    $H_{i+1} \leftarrow H[V_{i+1}]$
  end for
  return H, π

表 1 頂点 v_i の評価基準

アルゴリズム	頂点 v_i の評価基準
min(degree)	$v_i = \arg\min_{u \in V(H_i)} \delta_{H_i}(u)$
min(fillin)	$v_i = \arg\min_{u \in V(H_i)} \phi_{H_i}(u)$
min(degree + fillin)	$v_i = \arg\min_{u \in V(H_i)} \delta_{H_i}(u) + \phi_{H_i}(u)$
min(fillin / degree)	$v_i = \arg\min_{u \in V(H_i)} \frac{\phi_{H_i}(u)}{\delta_{H_i}(u)}$
min(degree, fillin)	$v_i = \arg\min_{u \in V(H_i)} \delta_{H_i}(u) + \frac{\phi_{H_i}(u)}{n^2}$
min(fillin, degree)	$v_i = \arg\min_{u \in V(H_i)} \phi_{H_i}(u) + \frac{\delta_{H_i}(u)}{n}$

基準についてまとめたものである．6 つの評価基準とは，論文 [4] に記載してある，GreedySparsestSubgraph を，今回新たに考えた min(fillin / degree) に置き換えた 6 つの評価基準である．これらは，$N_{H_i}(v_i)$ がなるべく小さくなるような頂点 v_i を選びたい，という前提がある．$\phi_{H_i}(u)$ とは，H_i から頂点 u を取り除いた時に $N_{H_i}(u)$ をクリークにするために加える辺の本数，$\delta_{H_i}(u)$ とは，H_i における u の次数を表す．

min(fillin) とは $\phi_{H_i}(u)$ が最小となる頂点 v_i, min(degree) とは $\delta_{H_i}(u)$ が最小となる頂点 v_i を選択する．min(degree + fillin) とは，$\phi_{H_i}(u)$ と $\delta_{H_i}(u)$ の和が最小となる頂点 v_i を選択する．min(fillin / degree) とは，$\frac{\delta_{H_i}(u)}{\phi_{H_i}(u)}$ が最小となる頂点 v_i を選択する．min(degree, fillin) とは，min(degree) で $\delta_{H_i}(u)$ が等しくなる頂点が存在したとき，$\phi_{H_i}(u)$ が小さくなる頂点 v_i を選択する．その逆で，min(fillin, degree) アルゴリズムとは，min(fillin) で $\phi_{H_i}(u)$ が等しくなる頂点が存在したとき，$\delta_{H_i}(u)$ が小さくなる頂点 v_i を選択する．

グラフ H_i において，ある評価基準を使用した評価値を毎回探索し求めるのは時間がかかる．そのため，データ構造は最小ヒープを用いて管理する．

また，各頂点の評価値は変化する．なぜなら，頂点 v_i を選択するとは，$N_{H_i}(v_i)$ をクリークにするために辺を加える操作と，頂点 v_i を削除する操作を行う．すると，$u \in V(H_i)$ において，$\delta_{H_i}(u)$， $\phi_{H_i}(u)$ が変化するため，評価値を更新する必要がある．しかし，評価基準毎に頂点の更新候補が違う．詳しいことは次の節で述べる．

4.2 評価値の更新候補

評価値の更新候補は，$\delta_{H_i}(u)$ と $\phi_{H_i}(u)$ のどちらを使用するかで異なる．

グラフ H_i から頂点 u を選択したとき，$N_{H_i}(u)$ をクリークにし，u を削除する．そのとき，$\delta_{H_i}(u)$ であれば，$N_{H_i}(u)$ の候補を，$\phi_{H_i}(u)$ であれば，$N_{H_i}[N_{H_i}(u)]$ の候補を更新する必要がある．

しかし，$\phi_{H_i}(u)$ の更新範囲は小さくすることができる．$N_{H_i}(u)$ に対して，不在辺を見つける．そのとき，$\bigcup_{x,y \in N_{H_i}(u), \{x,y\} \notin E(H_i)} (N_{H_i}(x) \cap N_{H_i}(y)) \cup N_{H_i}(u)$ が更新候補となる．

ここで，$\phi_{H_i}(u)$ において，$N_{H_i}[N_{H_i}(u)]$ の更新範囲を**二次近傍**，$\bigcup_{x,y \in N_{H_i}(u), \{x,y\} \notin E(H_i)} (N_{H_i}(x) \cap N_{H_i}(y)) \cup N_{H_i}(u)$ の更新範囲を**不在辺共通近傍**と呼ぶ．

また，$\delta_{H_i}(u)$ と $\phi_{H_i}(u)$ の更新候補は，$\delta_{H_i}(u)$ の更新候補が $\phi_{H_i}(u)$ の更新候補の部分集合となっているため，両方使用する場合は $\phi_{H_i}(u)$ の更新候補を採用するので十分である．

不在辺共通近傍を使用したときに，二次近傍に比べて，時間がどのくらい早くなるかについては，5.2 節の実験を通して確認する．

5. 実験

5.1 実験環境

CPU は 3.60GHz IntelCore i7-7700， RAM は 32GB. OS は Ubuntu 14.04.5 LTS．言語は Java1.8，jre1.8.0_144 を用いて実装した．また，ヒープサイズは 30GB 用意した．使用するインスタンスは，PACE（Parameterized Algorithms and Computational Experiments Challenge）2017 のヒューリスティック部門で使用したベンチマークインスタンス 200 個 [5] である． 1 つのインスタンスを解くのに 30 分の制限時間を設けた．

5.2 min(fillin) における評価値の更新範囲の削減

評価基準は min(fillin) とし， 2 つの更新範囲に対してどちらが早くなるかを実験した．グラフは，先ほどと同じくインスタンスの頂点数を $[0, 1000)$, $[1000, 5000)$, $[5000, 50000)$ の 3 つの範囲に分け，作成した．図 1，図 2，および図 3 は，縦軸を時間，横軸を頂点数とし， 2 つの更新範囲に対してプロットし，線を引いた散布図である．対象とするインスタンスは，幅が等しくなり，かつ，両方ともタイムアウトしていないものである．

図 1，図 2，および図 3 から，二次近傍に比べて，不在辺共通近傍を用いた方法が早くなることが分かる．平均時間

は 29 秒早くなり，最大で約 998 秒早く，最小で 31 ミリ秒遅くなる．

このことから，評価値の更新範囲を削減することで，同じ幅を得ることができ，かつ，早くなるといえる．

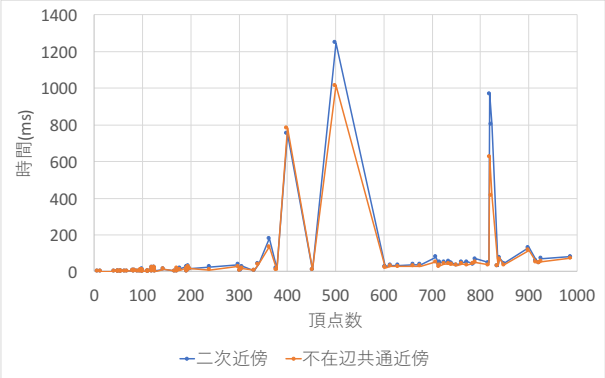

図 1 頂点数が $[0, 1000)$ の二次近傍と不在辺共通近傍の時間比較．両方ともタイムアウトしたインスタンスは除く．

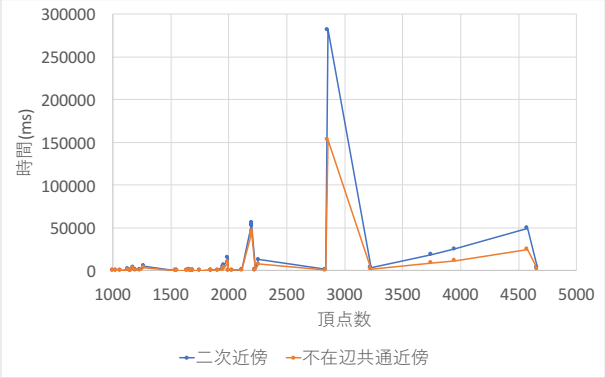

図 2 頂点数が $[1000, 5000)$ の二次近傍と不在辺共通近傍の時間比較．両方ともタイムアウトしたインスタンスは除く．

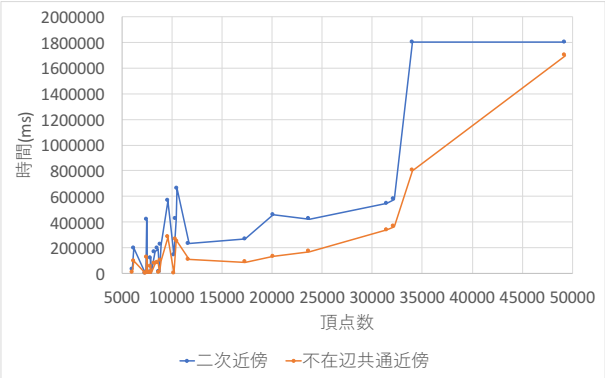

図 3 頂点数が $[5000, 50000)$ の二次近傍と不在辺共通近傍の時間比較．両方ともタイムアウトしたインスタンスは除く．

5.3 解析

グラフや表は，インスタンスの頂点数を 3 つの範囲に分けた．グラフは，インスタンスの頂点数を $[0, 1000)$, $[1000, 5000)$, $[5000, 50000)$ とし，表は，インスタンスの頂点数を $[0, 1000)$, $[1000, 5000)$, $[5000, \infty)$ とし，それぞれ作成した．幅について，図 4，図 5，および図 6 は，縦軸を幅，横軸を頂点数とし，6 つの評価基準ごとにプロットした，縦軸に関する片対数グラフである．一般的に，頂点数の大きいインスタンスから得られる幅は大きくなるため，縦軸のスケールが 3 つの範囲ごとに変化することに注意する．1 つのインスタンスに対して，6 つの評価基準から得られる幅は，横軸が等しい箇所に縦に 6 つプロットされる．解けていないインスタンスに関してはプロットしない．そして，その幅が最小となるインスタンスの個数を数えてまとめたものが，表 2 である．

同様に，時間についてもグラフや表を作成した．先ほどと違う箇所は，グラフの縦軸は時間とし，単位はミリ秒とする．グラフは図 7，図 8，および図 9 となり，表は表 3 となる．

表 4 は，min(degree) と min(fillin / degree) の性能比較を示す度数分布表である．r とは，ある 1 つのインスタンスに対して，min(degree) と min(fillin / degree) で得られた幅の倍率であり，基準は min(fillin / degree) の幅とする．階級は，$[0, 0.9)$，$[0.9, 1.7)$，$[1.7, \infty)$ とし，$[0.9, 1.7)$ の階級の幅は 0.1 とした．また，度数は min(degree) と min(fillin / degree) のどちらか一方で解けるインスタンスの個数とし，解けないものは無限大の幅とみなしている．

また，$\phi_{H_i}(u)$ を使用した評価基準に対する，評価値の更新範囲は全て不在辺共通近傍とする．

5.3.1 幅が最小になる個数が最も多い評価基準

用意したインスタンスにおいて幅が最小になる個数が最も多い評価基準は，表 2 から，min(fillin / degree) であることが分かる．

また，min(fillin / degree)，min(fillin)，min(degree + fillin)，min(fillin, degree)，min(degree, fillin)，min(degree) の順に，幅が最小となるインスタンスの個数が少なくなる．同様に，図 4，図 5，および図 6 からもいえる，つまり，幅について，min(fillin / degree) がはっきりと他よりも良く，$\phi_{H_i}(u)$ を用いた評価基準の方が良い傾向があるといえる．

5.3.2 時間が最小になる個数が最も多い評価基準

用意したインスタンスにおいて時間が最小になる個数が最も多い評価基準は，表 3 から，min(degree) であることが分かる．これは，$\delta_{H_i}(u)$ と $\phi_{H_i}(u)$ の更新候補が，$\delta_{H_i}(u)$ の更新候補が $\phi_{H_i}(u)$ の更新候補の部分集合であり，かつ，$\delta_{H_i}(u)$ と $\phi_{H_i}(u)$ では，圧倒的に $\delta_{H_i}(u)$ を早く求めることができるからである．

また，表 3 から，min(degree)，min(fillin)，min(fillin, degree)，min(degree + fillin)，min(degree, fillin)，min(fillin

表 2　幅が最小となるインスタンスの個数

	min(degree)	min(fillin)	min(degree + fillin)	min(fillin / degree)	min(degree, fillin)	min(fillin, degree)
$[0, 1000)$	19	33	29	73	20	29
$[1000, 5000)$	5	10	11	33	5	10
$[5000, \infty)$	6	9	2	17	2	2
全体	30	52	42	123	27	41

表 3　時間が最小となるインスタンスの個数

	min(degree)	min(fillin)	min(degree + fillin)	min(fillin / degree)	min(degree, fillin)	min(fillin, degree)
$[0, 1000)$	64	24	15	10	17	16
$[1000, 5000)$	32	9	3	0	1	3
$[5000, \infty)$	21	6	1	1	0	1
全体	117	39	19	11	18	20

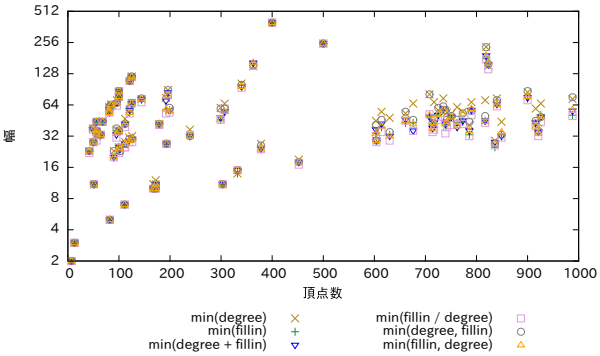

図 4　頂点数が $[0, 1000)$ のインスタンスに対する幅

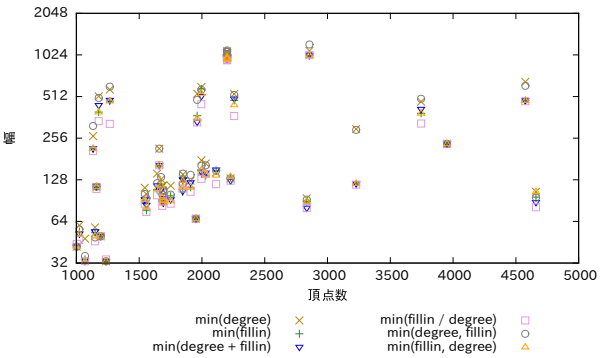

図 5　頂点数が $[1000, 5000)$ のインスタンスに対する幅

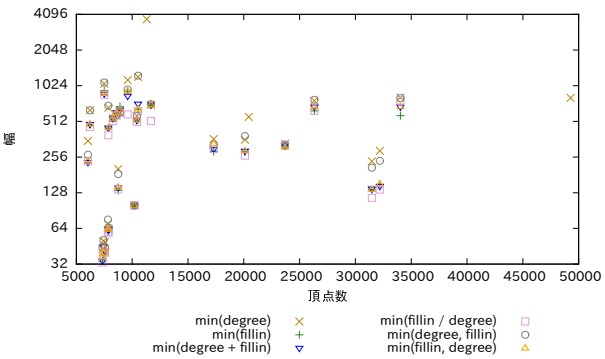

図 6　頂点数が $[5000, 50000)$ のインスタンスに対する幅

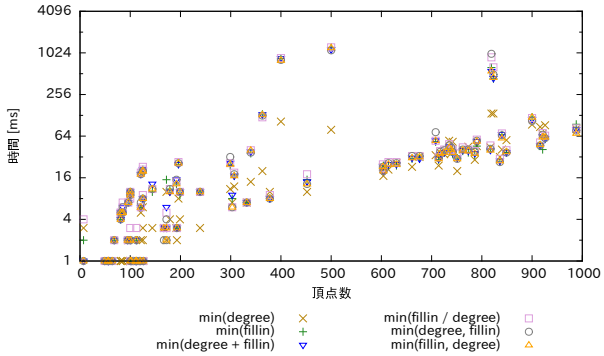

図 7　頂点数が $[0, 1000)$ のインスタンスに対する時間

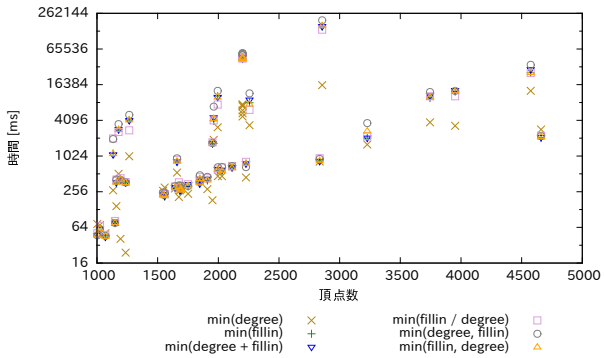

図 8　頂点数が $[1000, 5000)$ のインスタンスに対する時間

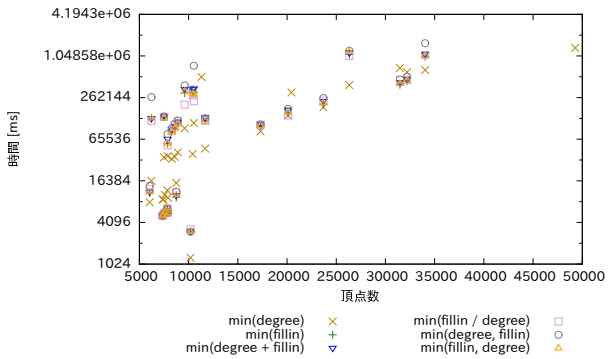

図 9　頂点数が $[5000, 50000)$ のインスタンスに対する時間

表 4　min(degree) と min(fillin / degree) の性能比較.

r	$[0, 0.9)$	$[0.9, 1.0)$	1.0	$(1.0, 1.1)$	$[1.1, 1.2)$	$[1.2, 1.3)$	$[1.3, 1.4)$	$[1.4, 1.5)$	$[1.5, 1.6)$	$[1.6, 1.7)$	$[1.7, \infty)$
個数	4	6	20	27	29	21	19	12	5	4	9

r とは，任意のインスタンスに対して，min(degree) の幅 = min(fillin / degree) の幅 $\times$ r

/ degree) の順に，時間が最小となるインスタンスの個数が少なくなる．同様に，図 7，図 8，および図 9 からもいえる．min(degree) と min(fillin) 以外の残り 4 つの評価基準が遅くなる理由は，$\delta_{H_i}(u)$ と $\phi_{H_i}(u)$ の両方を求めているため，時間がかかる．つまり，時間については，min(degree) が明らかに高速であり，$\phi_{H_i}(u)$ を用いたもの同士の差は小さい傾向があるといえる．

5.3.3　min(degree) と min(fillin / degree) の性能比較

ここで，min(degree) は他の評価基準と比べて，高速で求められることがわかったが，解の質はどうなるだろうか．min(degree) と比較する評価基準は，幅が最小になるインスタンスの個数が最も多くなった，min(fillin / degree) である．

表 4 は，min(degree) と min(fillin / degree) の性能を比較した表である．$[0, 1.0)$ のときが 156 個中 10 個，$(1.0, 1.4)$ のときが 156 個中 96 個あることから，過半数以上のインスタンスで，min(degree) で得られた幅は大きくなる．

min(degree) を使用する際は，高速で求められるが，幅の大きい巨大グラフを対象としているため，多くのインスタンスで幅が大きく，その差も大きいことに注意する必要があるだろう．

6.　まとめ

本稿では，巨大グラフを対象とした，木幅問題の貪欲解法の実験的評価を行った．その結果，今回使用したインスタンスに対して，幅に関しては新たな評価基準 min(fillin / degree) が，時間に関しては min(degree) が良くなることが分かった．また，min(fillin) の更新範囲は，二次近傍より不在辺共通近傍を使用した方が，実験から早くなることが分かった．

今後の課題として，今回は 6 つの評価基準を用いたが，より良い評価基準があるかもしれない．また，使用するインスタンスによって，違った結果が出る可能性があり，さらなる実験をする必要があるだろう．

参考文献

[1] Thorup, M.: All structured programs have small tree width and good register allocation, *Information and Computation*, Vol. 142, No. 2, pp. 159–181 (1998).
[2] Yamaguchi, A., Aoki, K. F. and Mamitsuka, H.: Graph complexity of chemical compounds in biological pathways, *Genome Informatics*, Vol. 14, pp. 376–377 (2003).
[3] Arnborg, S., Corneil, D. G. and Proskurowski, A.: Complexity of finding embeddings in ak-tree, *SIAM Journal on Algebraic Discrete Methods*, Vol. 8, No. 2, pp. 277–284 (1987).
[4] Bodlaender, H. L. and Koster, A. M.: Treewidth computations I. Upper bounds, *Information and Computation*, Vol. 208, No. 3, pp. 259–275 (2010).
[5] : GitHub repositry PACE-2017-TrackA-instances for PACE-challenge on PACE2017 Track A, https://github.com/PACE-challenge/PACE-2017-TrackA-instances.