社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2006－AL－108（11）
2006／9／27

最大クリーク抽出のより高速な分枝限定アルゴリズム

須谷 洋一[†]　東 貴紀[††]　富田 悦次[††]
高橋 真也[††]　仲谷 洋幸[†]

無向グラフから最大クリークを１つ抽出する問題はグラフ理論における基本的な問題であり, 多くの数理問題に応用できることから重要である. これまで筆者らは, 逐次的な近似彩色を用いた分枝限定法による効率の良い最大クリーク抽出アルゴリズムを発表してきた. 本稿では, それらのアルゴリズムにさらに幾つかの新たな効率化手法を組み合わせることにより, 多くの問題に対してより高速に解を得ることのできる最大クリーク抽出アルゴリズムを提唱する.

A Faster Branch-and-Bound Algorithm for Finding a Maximum Clique

Yoichi SUTANI[†]　Takanori HIGASHI[††]　Etsuji TOMITA[††]
Shinya TAKAHASHI[††]　Hiroyuki NAKATANI[†]

Abstract. Finding a maximum clique is a basic problem in the graph theory, and many practical problems can be formulated as this problem. We have shown efficient branch-and-bound algorithms for finding a maximum clique using an approximate coloring. In this article, we present a faster algorithm which employes an improved approximate coloring and other techniques.

1　はじめに

無向グラフ中の最大クリークを抽出する問題は, NP困難問題のクラスに属す基本的な組合せ最適化問題の一つである. この問題は, その重要性により多くの研究がなされており, 幾つかの高速な最大クリーク抽出アルゴリズムが考案されている [1-13]. その結果として, 多くの応用の成果が得られるに至っている. これらの応用問題をさらに発展させるためにも, 最大クリーク抽出アルゴリズムの高速化は重要である.

これまで筆者らは, 深さ優先探索を基礎とし, それに逐次的な近似彩色を用いた分枝限定法などを加えることにより, 効率的な最大クリーク抽出アルゴリズム MCQ[9] を提唱し, その後, この改良版 MCR[10] および, さらに効率化を行ったアルゴリズムを提案してきた [11-13]. 本稿では, これらのアルゴリズムに幾つかの効率化手法を組み合わせることにより, 最大クリーク抽出アルゴリズム全体のさらなる高速化を実現した.

[†]電気通信大学大学院 電気通信学研究科
Graduate School of Electro-Communications, The University of Electro-Communications
[††]電気通信大学　電気通信学部　情報通信工学科
Department of Information and Communication Engineering, The University of Electro-Communications

2　諸定義

V を節点の集合, $E \subseteq V \times V$ を枝の集合としたときに, $G = (V, E)$ で表される単純無向グラフを対象とする. ここで, 単純無向グラフの定義により, 枝としてループや多重枝は含まないものとする. また, 節点集合 V は順序付きの集合であるものとし, V の先頭から i 番目の要素を $V[i]$ で表す. 節点 $v \in V$ に隣接する節点の集合を $\Gamma(v)$ で表し, その要素数を v の次数と呼ぶ. また, 集合 S の要素数を $|S|$ で表す. このとき枝密度を $|E|/\{|V| \times (|V|-1)/2\}$ とする. 節点集合 V 中の任意の 2 節点が隣接しているとき, グラフ $G = (V, E)$ は完全グラフであるという. V の部分集合 C に対する誘導部分グラフ $G(C)$ が完全であるとき, $G(C)$(あるいは単に C) をクリークと呼ぶ. それが

他のクリークの真の部分集合でなければ極大クリークと呼び, 最も節点数の多い極大クリークのとき最大クリークと呼ぶ. 最大クリークサイズは $\omega(G)$, または単に ω で表す.

3 従来の最大クリーク抽出アルゴリズム

本稿で対象とするのは, グラフのデータが入力されたとき, そのグラフの最大クリークを 1 つ見つけ出力するアルゴリズムである. 以下では, 筆者らがこれまでに発表してきた最大クリーク抽出アルゴリズム, MCQ[9], MCR(=MCQ′)[10] で用いられている分枝限定法と, 分枝限定条件を得るために行う近似彩色手続き NUMBER-SORT[9, 10] について述べる.

3.1 基本アルゴリズム

基本アルゴリズムとしては, ある時点で保持しているクリークに, そのクリーク中の全ての節点と隣接している節点 (この集合を候補節点集合と呼ぶ) を 1 つ新たに付け加え, より大きなクリークにする操作を深さ優先探索によって進めていくという手順をとる. このような操作を再帰的に繰り返すことで, クリークを 1 つずつ大きくしていき, これ以上加えられる節点がなくなった時点で, グラフ中の極大クリークが得られる. この極大クリークのサイズが, それまでの探索で見つかったものの中で最も大きい場合は, これを最大クリークの候補として保存する. 以降は, 深さ優先探索の基本に従い, 1 つ前の再帰処理に戻って別の候補節点を選んで探索を続ける. 最終的に全ての組合せについて調べた時点で保存していた最大クリークの候補が, そのグラフの最大クリークとなる.

クリーク探索の全過程は, 全節点集合 V を根, 各時点における候補節点集合 R を節点として, 各候補節点集合について親子関係にあるものを枝で結んだ探索木として表現することができる. この探索木における枝を分枝, その総数を分枝数という.

3.2 分枝限定法

アルゴリズムを効率化するためには, 探索過程での分枝数を削減することが効果的であり, これを何らかの方法を用いて実現するのが分枝限定法である. 最大クリーク抽出において最も基本的な分枝限定条件は, 現在保持しているクリークを Q, 候補節点集合を R, 現在までに求まっている最大クリークを Q_{max} とすると, 次の式で表される.

$$|Q| + |R| \leq |Q_{max}|$$

このような条件が成り立つときは, これから先の探索を行っても現在までに求まっている最大クリーク Q_{max} よりも大きいサイズの極大クリークが存在しないことが明らかであるため, 探索を打ち切ることができる.

しかし, ここで注意しなければならないことは, 確かに分枝限定を行えば分枝数を削減することは可能であるが, この処理に時間が掛かってしまうと, 実行時間全体は逆に増加することがあり得るということである. そのため, いかに簡単な処理で効率の良い分枝限定を行うかが重要な問題になる.

3.3 NUMBER-SORT

従来の最大クリーク抽出アルゴリズム (MCQ, MCR) では, 分枝限定を効率良く行うために逐次近似彩色手続き NUMBER-SORT[9, 10](以下, N-S) を用いている. これは, 再帰処理ごとに候補節点集合に対し逐次的な近似彩色に相当する番号付けを行うことでクリークサイズの上界を求める手法であり, これにより有効に分枝限定できる. 彩色条件は, 隣接する節点同士は異なる色で彩色する, というのみである. これを実現する手順としては, まず先頭節点に番号“1”を付与し, 以下並べられた順に, いま番号を与えようとする節点及び既に番号が付与された節点について, 彩色条件に矛盾することのない最小の正整数を付与していく. 以上の手順により, 全候補節点に対して番号付けを行い, その番号の昇順に並べ替える手続きが NUMBER-SORT である.

ここで, クリーク中の節点は全て互いに隣接しているので, 上の条件で彩色したとき, その彩色番号は必ずグラフ中のクリークのサイズ以上となる.

以上より, ある節点 p から探索するとき, この節点の彩色番号が $No[p]$ で表されるとき, 分枝限定条件を

$$|Q| + No[p] \leq |Q_{max}|$$

とすることで、分枝限定をより有効にできる.

3.4 アルゴリズム MCR と初期節点整列

従来のアルゴリズム MCR では 3.3 で述べた彩色手続きを用いた分枝限定に加え, 節点の順序が探索に大きな影響を与えることを考慮して, 探索の前処理として節点の整列を行っている. 探索する節点の順序は次数の小さいものから, 彩色は次数の大きいものから, 行うのが効率が良いことが実験的に示されているため, これに即した節点の整列を探索の前処理として行う.

4 アルゴリズムの効率化

4.1 再彩色アルゴリズム RE-COLOR

ここで, 本稿で提案する彩色アルゴリズムについて述べる. これまで, 彩色アルゴリズムが短時間で良い精度が得られれば, 分枝限定の効果が大きくなるため, 効率化につながるはずであると考えられてきた.

しかし, 実際の探索においては, 必ずしも彩色の精度, すなわち最大彩色番号が小さければよいわけではなく, 探索しなければならない節点が少なくなるように彩色を行うことが重要である.

つまり, 前述の分枝限定条件から $No[p] \leqq |Q_{max}| - |Q|(= cutc$ とする) となる節点は探索不要となるので, なるべくこの条件を満たす節点を多くするような節点の彩色を行うことができれば効率化が望める. このような考えに基づき提唱するのが, 再彩色アルゴリズム RE-COLOR(図 1) である. このアルゴリズムは, 従来の彩色アルゴリズム N-S の過程で一度彩色した節点について, ある条件を満たす場合, 彩色をし直すアルゴリズムである. 以下にアルゴリズムのステップを示す [12].

1. 着目する節点 $p \in V$ を N-S で彩色したとき, その彩色番号を $pcol$ とする.
2. 彩色番号 $1 \leqq k_1 \leqq cutc - 1$ に対して 3.〜5. を繰り返す.
3. 彩色済み節点集合 C_{k_1} 中で, $\Gamma(p)$ 中の節点がただ 1 つしか存在しない彩色番号 k_1 と, その節点 q を見つける.
4. 彩色番号 $k_1 + 1 \leqq k_2 \leqq cutc$ に対して 5. を繰り返す.
5. 彩色済み節点集合 C_{k_2} 中で, $\Gamma(q)$ の節点が存在しない彩色番号 k_2 を見つける.

6a. k_2 が見つかったとき, p の彩色番号を k_1, q の彩色番号を k_2 として, アルゴリズムを終了する.

6b. 該当する k_1, k_2 が存在しなかったらそのまま終了する.

4.2 節点の並び順の保存

分枝限定を行うための上界を得る近似彩色手続きには, 問題点として節点の順番が変更されてしまうということがある.

MCR では, 探索の前処理として次数の小さいものから探索できるように節点を並び替えており, この節点の並び順が探索に大きく影響することは先に述べた

```
procedure RE-COLOR(p, pcol)
begin
  for k1 := 1 to cutc − 1 do
    if |C_k1 ∩ Γ(p)| =1 then
      q := the element in (C_k1 ∩ Γ(p)) ;
      for k2 := k1 + 1 to cutc do
        if C_k2 ∩ Γ(q) = ∅ then
          C_pcol := C_pcol − {p};
          C_k1 := (C_k1 − {q}) ∪ {p};
          C_k2 := C_k2 ∪ {q};
          return
        fi
      od
    fi
  od
end { of RE-COLOR}
```

図 1: 再彩色アルゴリズム RE-COLOR

とおりであるが, 近似彩色手続きによってこの並び順が変更されると, 適切な順序での探索が出来なくなる可能性がある. また, RE-COLOR を適用すると, 従来の彩色手続きの場合よりもこの傾向が強くなるのは明らかである.

そこで, この問題を解決するため, 前処理で整列した節点の順番を保存しておき, 近似彩色はこの節点の順番に従って行う, ということを考える [11]. このようにすることで, 近似彩色手続きによって探索するための節点の並びは変化するものの, 次の再帰処理での近似彩色手続きは, 初期節点の順で行われるため, 再度節点の並びが崩れることは起こらず, 探索が進むにつれて徐々に節点の順番が崩れることを防ぐことができる.

これまでの多くの最大クリーク抽出アルゴリズムに RE-COLOR のような手法を組み合わせる場合, 節点の順番が大きく変更されてしまうと, 逆に分枝数が増え, 実行時間が増加してしまうこともありえたが, この方法によって, この問題は解消でき, 加えて従来より効率の良い探索が行えるようになる.

4.3 NUMBER-SORT-R

アルゴリズム RE-COLOR を, 近似彩色手続き N-S に組み込み, さらにこの彩色に際して, 保存しておいた初期節点の順番を用いる新たな近似彩色アルゴリズムを NUMBER-SORT-R（図 2）とする. 本稿では, ある節点 p に対して RE-COLOR を適用する条件と

して,

$(k > cutc)$ かつ (k=(現在の最大彩色番号))

であるときに限定している (k は節点 p が従来方法で彩色された彩色番号). これは, RE-COLOR がそれ自体時間のかかる処理であるので, 必要以上に適用しないようにするためである [13].

また, アルゴリズムの実装においては, 彩色番号が $cutc$ 以下の節点については R および No の値は不要であり, 図 2 の {SORT} 部分における該当の代入演算の実行は省略している. ただし, 彩色番号が $cutc$ となる最初の節点については No のみ参照され, この節点で分枝限定が行われるため, 値として $cutc$ を代入しておくことが必要である.

4.4 グラフデータの再構成

本稿で新たに提案した近似彩色手続き NUMBER-SORT-R では, 彩色は前処理時に決定される節点の順番で行われるため, グラフデータへのアクセスも当然これに伴う. この性質を効率化に活かすため, 前処理の節点の整列が終了した時点で, グラフデータの再構成を行う. 具体的には, 隣接行列に格納されているグラフデータを, 整列された節点の順序で格納できるように節点番号を付け直し, そのうえでグラフデータを再構成する. このようにすることで, CPU キャッシュ効率の向上につながり, 高速化が期待できる.

この手続きの欠点として, グラフデータを再構成する際に余分な時間が掛かることが考えられるが, この時間はほとんど無視できるほど小さいため, 大きな問題とはならない.

4.5 近似解の利用による効率化

最大クリークの探索において, 仮にその問題の最大クリークサイズが分かっていれば, 従来の分枝限定手法や本稿で提案したアルゴリズム RE-COLOR の条件により探索の効率化が望める. 現実の問題において最大クリークサイズが探索の前から分かっているようなことはあり得るが, 基本的には探索前は未知である.

そこで, 探索の前処理として何らかの方法により, 近似解を求めておき, この値をグラフ中のクリークサイズの下界とすることを考える. ただし, この近似解を求める処理に時間を掛けてしまっては, 探索全体の高速化の目的に反するため, なるべく短い時間で, 精度の良い近似解を得られることが理想である.

クリーク問題については, 一般的に近似困難性が知られているため, 多くの問題に対して良い精度を得る

```
procedure NUMBER-SORT-R(V_s, R, No)
begin
  |R| := |V_s|;
  {NUMBER}
    kmax := 1;
    C_1 := ∅;
    for i := 1 to |V_s| do
      p := V_s[i] ;
      k := 1;
      while (C_k ∩ Γ(p)) ≠ ∅
        do k := k + 1 od
      if k > kmax then
        kmax := k;
        C_kmax := ∅
      fi
      C_k := C_k ∪ {p};
      {== RE-COLOR 適用部分 start ==}
      if (k > cutc) and (k = kmax) then
        RE-COLOR(p, k) ;
        if C_kmax = ∅ then
          kmax := kmax − 1
        fi
      fi
      {== RE-COLOR 適用部分 end ==}
    od
  {SORT}
    i := |V_s|;
    if cutc < 0 then cutc = 0 fi
    for k := kmax downto cutc + 1 do
      for j := |C_k| downto 1 do
        R[i] := C_k[j];
        No[R[i]] := k;
        i := i − 1
      od
    od
    if i ≠ 0 then No[R[i]] = cutc fi
end { of NUMBER-SORT-R}
```

図 2: 近似彩色手続き NUMBER-SORT-R

近似アルゴリズムは望み難いが, 比較的良好な近似解を得るヒューリスティクスは多く存在する. 本稿では, 次に述べる MCQ を基礎とした単純なアルゴリズム init-lb を提案し, このアルゴリズム を探索の前処理として実行することにより近似解を得, アルゴリズム全体の効率化を図る.

なお, 探索の前処理で求める近似解は, 最大クリークのサイズに近いほど探索に都合良くなるため, 優れた近似解法が提案されれば, この部分をそのアルゴリズムに変更するだけでアルゴリズム全体の効率化が達成される.

アルゴリズム init-lb

アルゴリズム init-lb は, MCQ のような深さ優先探索アルゴリズムを基としているが, 実行時間を小さく抑えるために, バックトラックを行わず探索領域を非常に小さくしている点が異なる. このようにすることで, ある節点からの探索でただ 1 つの極大クリークが見つかる. 全ての節点からの探索を行うことも当然可能であるが, あくまで短時間で良い精度の解を得ることが目的であるから, 時間と精度のバランスが重要である. そのため探索を開始する節点の数はパラメータ α とし, 本稿においては実験的に α=$2\sqrt{|V|}$ としている. アルゴリズム init-lb のステップを以下に示す.

0_1. 節点集合 V を手続きの引数とする

0_2. 最大クリーク $Q_{max} := \emptyset$ とする (前処理としてこのアルゴリズムを使用する場合は元の値を引継ぐ)

1. 候補節点集合 $R := V$, 探索中のクリーク $Q := \emptyset$ とし, 2a. へ

2a. R の先頭の節点を p とし, 3. へ

2b. R 中の最大次数かつ隣接する節点の次数の和が最大の節点を p とし, 3. へ

3. Q に節点 p を加え, $R := R \cap \Gamma(p)$ とする.

4. $R = \emptyset$ であれば 6. へ

5. (R 中の節点の最大次数)+$|Q| < |Q_{max}|$ であれば 7. へ, そうでなければ 2b. へ

6. $|Q| > |Q_{max}|$ であれば, Q_{max} を Q で更新

7. V の先頭の節点を集合の末尾へ移動し 1. へ
ただし, この操作が定められた回数 α に達すれば終了 (前処理としてこのアルゴリズムを使用する場合, V の節点の順番は元に戻す必要がある).

4.6 アルゴリズム MCS

従来の最大クリーク抽出アルゴリズム MCR に, 4.1 ~ 4.5 で述べた効率化手法を全て組合せたものをアルゴリズム MCS とする. このアルゴリズムの近似解を求める部分は, 他のアルゴリズムに取り替えることが可能であるから, 近似解を前処理として求める部分のみ取り除いたものをアルゴリズム MCS_0 として区別する.

```
procedure MCS(G = (V, E))
begin
  Q := ∅;    Q_max := ∅;
  MCR の節点整列及び節点番号 No の付与;
  グラフデータの再構成;
  init-lb(V);
  EXPAND (V, V, No);
  Q_max の節点番号を
    グラフデータの再構成前の節点番号に戻す;
  output Q_max  { 最大クリーク }
end { of MCS}

procedure EXPAND(V_s, R, No)
begin
  while R ≠ ∅ do
    p := the last vertex in R;
   if |Q| + No[p] > |Q_max| then
     Q := Q ∪ {p};
     V_p := V_s ∩ Γ(p);  { 順序は保存する }
     if V_p ≠ ∅ then
       NUMBER-SORT-R(V_p, newR, newNo);
       {newR, newNo の初期値は意味を持たない }
       EXPAND(V_p, newR, newNo)
     else if |Q| > |Q_max| then Q_max := Q fi
     fi
   else return
   fi
   Q  := Q − {p};
   R  := R − {p};
   V_s := V_s − {p}  { 順序は保存する }
  od
end { of EXPAND }
```

図 3: アルゴリズム MCS と再帰手続き EXPAND

5 計算機実験

本稿で提案したアルゴリズム MCS_0, MCS, およびその基となるアルゴリズム MCR を実働化し, テストグラフについて計算機実験を行うことで, 実行時間について比較評価する. 実行時間はグラフのデータの読み込みなどを含まない, アルゴリズム本体のみの処理時間とする.

テストグラフは, 最大クリーク問題に対するベンチマークグラフとして多く用いられている, DIMACS ベンチマークグラフおよびランダムグラフを用いる. 結果を表 1 および表 3 に示す. 加えて MCS 中の手続きである init-lb についても得られる近似解と, 実行時間

を表 5 に掲載した. ランダムグラフは, 節点数 n, 枝存在確率 p について, 乱数の種が異なるものを 10 個ずつ作成し, それぞれの平均を取ったものを結果として掲載している. 例外として, 枝密度が非常に高く, 値のばらつきが大きいため 10 個の平均では不十分と考えられた n=200 の p=0.95, 0.98 の場合は, 異なる乱数の種 100 個の平均を取った結果を掲載している. 使用した計算機環境は, CPU: Pentium4 3.6GHz, コンパイラ: gcc -O2, OS: Linux である.

また, 表 2 および表 4 は他の文献のアルゴリズムの実行時間を, 本実験環境に合わせて校正 [1] を行った値である (空欄は, 参考文献に実行結果が示されていない部分. Target/5 は推定参考値).

6 考察

表 1 および表 3 において, まず MCR と MCS_0 について比較すると, MCS_0 は多くのグラフに対し分枝数が大幅に削減できており, そのため実行時間も減少している. 特に, DIMACS グラフでは MANN_a, brock, p_hat など, ランダムグラフでは枝密度の高いグラフと時間の掛かる問題に対して高速化が顕著である. これは, 近似彩色アルゴリズムの改良によるもので, 再彩色アルゴリズム RE-COLOR と節点の順番を保存する手法の効果が上手く出ていることを意味する. また, 表 3 のランダムグラフの節点数 5,000 以上の大きいグラフでは, 分枝数の減少はわずかだが実行時間が減少している. これは, グラフデータの再構成によってキャッシュ効率が向上した効果によるものである.

次に, MCS_0 に前処理として近似解を求める手法を加えたアルゴリズム MCS であるが, これについては, DIMACS グラフの最大クリークサイズが大きい問題などで高速化が達成されている. これは, 近似解としてサイズの大きなクリークが見つかることで, 探索の開始時点から RE-COLOR や分枝限定の効果により無駄な探索を省くことができるためと考えられる.

最後に, 本稿で提唱したアルゴリズム MCS と, 他のアルゴリズム (表 2 および表 4) を比較すると, アルゴリズム MCS は広範囲の問題に対して, 最も速く解を得ている. 他の問題についても, 他のアルゴリズムに比べて, 解を得る時間が安定しており, 本アルゴリズムが優位であることが分かる.

7 おわりに

本稿では, 最大クリーク抽出アルゴリズムにおける幾つかの効率化手法を提案し, それを従来のアルゴリズムに組み合わせることにより, 多くの問題に対してより高速に解を得ることのできる最大クリーク抽出アルゴリズム MCS を提唱した. また, 他のアルゴリズムと比較しても, アルゴリズム MCS が優位であることを示した. なお, この深さ優先探索手法を基本とし, 極大クリークを効率良く全列挙することも可能である [14].

謝辞

本研究は科学研究費補助金基盤研究 (B) の支援を受けている.

参考文献

[1] D. S. Johnson and M. A. Trick, ed., “Cliques, Coloring, and Satisfiability,” DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Volume 26, American Mathematical Society (1996).

[2] E. Balas, S. Ceria, G. Cornuéjols, and G. Pataki, “Polyhedral methods for the maximum clique problem,” pp.11-28 in [1] (1996).

[3] J.-M. Bourjolly, P. Gill, G. Laporte, and H. Mercure, “An exact quadratic 0-1 algorithm for the stable set problem,” pp.53-73 in [1] (1996).

[4] E. C. Sewell, “A branch and bound algorithm for the atability number of a sparse graph,” INFORMS J. Comput. 10, pp.438-447 (1998).

[5] I. M. Bomze, M. Budinich, P. M. Pardalos and M. Pelillo, “The Maximum Clique Problem,” in: Ding-Zhu Du and P.M.Pardalos, ed., Handbook of Combinational Optimization, Supplement Volume A, Kluwer Academic Publishers, pp.1-74 (1999).

[6] P. R. J. Östergård, “A fast algorithm for the maximum clique problem,” Discrete Appl. Math. vol.120, pp.197-207 (2002).

[7] T. Fahle, “Simple and fast: Improving a branch-and-bound algorithm for maximum clique,” European Symposium on Algorithms 2002, LNCS2461 (2002).

[8] V. Stix, “Target-oriented branch and bound method for global optimization,” J. Global Optimization, vol.26, pp.261-277 (2003).

[9] E. Tomita, T. Seki, “An efficient branch-and-bound algorithm for finding a maximum clique,” Proc. DMTCS 2003, LNCS 2731, pp.278-289 (2003).

[10] E. Tomita, T. Kameda, “An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments,” Journal of Global Optimization (available online, 2006).

[11] 須谷洋一, 富田悦次, “最大クリーク抽出アルゴリズムの効率化と実験的評価・解析, ” 情報処理学会数理モデル化と問題解決研究会, 2005-MPS-57, pp.45-48 (2005).

[12] 東貴紀, 富田悦次, “近似彩色方法を工夫した最大クリーク抽出アルゴリズムの効率化, ” 電気通信大学テクニカルレポート, UEC-TR-CAS5 (2006).

[13] 須谷洋一, 東貴紀, 富田悦次, “近似彩色の改良による最大クリーク抽出アルゴリズムの効率化, ” 夏の LA シンポジウム (2006).

[14] E. Tomita, A. Tanaka and H. Takahashi, “The worst-case time complexity for generating all maximal cliques and computational experiments, ” Theoretical Computer Science (available online, 2006).

表 1: DIMACS グラフに対する実行時間と分枝数

Graph				実行時間 (sec)			分枝数 $/10^3$		
Name	n	d	ω	MCR	MCS_0	MCS	MCR	MCS_0	MCS
brock200_1	200	0.75	21	1.7	0.86	**0.85**	482	152	**148**
brock200_3	200	0.6	15	0.055	**0.042**	0.046	16	**8.1**	8.2
brock400_1	400	0.75	27	1,772	693	**692**	329,599	89,389	**89,299**
brock400_3	400	0.75	31	726	468	**463**	241,903	65,302	**64,264**
c-fat200-2	200	0.08	24	0.0011	0.0012	0.0022	**0.18**	**0.18**	**0.18**
c-fat500-1	500	0.04	14	**0.0028**	0.0033	0.0058	**0.49**	**0.49**	**0.49**
hamming8-2	256	0.97	128	**0.0022**	0.0025	0.016	0.13	0.13	**0**
hamming10-2	1024	0.99	512	**0.17**	0.19	0.52	0.51	0.51	**0**
johnson8-4-4	70	0.77	14	**0.00039**	0.00041	0.00098	0.14	0.13	**0.11**
johnson16-2-4	120	0.77	8	0.14	0.14	0.14	323	**238**	**238**
keller4	171	0.65	11	0.028	**0.025**	0.028	12	**7.1**	**7.1**
MANN_a27	378	0.99	126	2.5	**0.78**	0.84	38	9.1	**9.0**
MANN_a45	1035	0.99	345	3,090	281	**280**	2,952	225	**223**
p_hat500-3	500	0.75	50	1,788	150	**142**	138,300	7,923	**7,453**
p_hat700-3	700	0.75	62	68,187	2,392	**1,228**	3,733,665	88,168	**42,753**
p_hat1000-2	100	0.49	46	2,434	221	**164**	197,147	12,618	**9,058**
p_hat1500-1	1500	0.25	12	5.1	3.9	**3.8**	1,261	820	**772**
san1000	100	0.50	15	4.8	2.2	2.2	230	85	**84**
san200_0.9_2	200	0.90	60	4.2	0.41	**0.0070**	595	42	**0.11**
san400_0.5_1	400	0.50	13	0.022	0.020	0.017	2.2	1.6	**0.18**
san400_0.9_1	400	0.90	100	3.4	0.12	**0.034**	74	2.3	**0.20**
sanr200_0.7	200	0.70	18	0.57	0.34	**0.30**	185	69	**58**
sanr200_0.9	200	0.90	42	289	41	**34**	40,470	3,471	**2,780**
sanr400_0.5	400	0.50	13	0.89	**0.72**	0.74	301	171	**170**
sanr400_0.7	400	0.70	21	379	181	**179**	89,124	29,632	**29,406**

表 2: 他のアルゴリズムの実行時間 1 (sec)

Graph Name	dfmax [1]	New [6]	χ + DF [7]	COCR [4]	MIPO [2]	SQUEEZE [3]	Target/5 [8]
brock200_1	15	12	69			941	70
brock200_3	0.21	0.098	1.7			124	1.0
brock400_1	22,051		>10,693				>4320
brock400_3	14,795		>10,693				>4320
c-fat200-2	**0.00058**	0.0022	0.0050		7.2	1.8	
c-fat500-1	0.0031	0.016	0.0099			33	
hamming8-2	>100,000	0.0089	0.054		0.029		
hamming10-2	>100,000	0.56	3.8			0.61	
johnson8-4-4	0.0045	0.0022	0.020		0.0029	0.22	
johnson16-2-4	0.75	0.060	5.9		**0.0018**	497	
keller4	0.37	0.11	1.9	0.43	118	152	
MANN_a27	>100,000	>2,232	7,685	2.8		472	
MANN_a45	>100,000		>10,693			>4,800	
p_hat500-3	>100,000		>10,693				>4320
p_hat700-3	>100,000		>10,693				>4320
p_hat1000-2	>100,000		>10,693				
p_hat1500-1	10		89				
san1000	>100,000	**0.11**	2,260				
san200_0.9_2	>100,000	0.96	1,434		0.15	8.1	66
san400_0.5_1	433	**0.0067**	5.0		85		0.80
san400_0.9_1	>100,000		5,361			2,073	>4320
sanr200_0.7	3.1	3.2	19			203	
sanr200_0.9	80,954		>10,693			>4,800	
sanr400_0.5	2.1	1.5	17				
sanr400_0.7	2,426		11,826			>4,800	

表 3: ランダムグラフに対する実行時間と分枝数

Graph			実行時間 (sec)			分枝数 /10^3		
n	p	ω	MCR	MCS_0	MCS	MCR	MCS_0	MCS
100	0.6	11-13	0.0017	**0.0016**	0.0022	0.84	0.48	**0.38**
	0.7	14-16	0.0047	**0.0036**	0.0037	2.2	1.0	**0.74**
	0.8	19-21	0.014	0.0079	**0.0074**	5.6	1.8	**1.4**
	0.9	29-32	0.038	0.013	**0.011**	11	1.9	**1.4**
	0.95	39-44	0.012	**0.0035**	0.0040	2.3	0.36	**0.21**
200	0.6	14	0.092	0.072	**0.068**	37	18	**16**
	0.7	18-19	0.68	0.41	**0.37**	223	85	**75**
	0.8	24-27	12	4.5	**4.3**	3,266	741	**699**
	0.9	40-44	647	74	63	97,627	6,608	**5,527**
	0.95	58-66	1,319	59	**42**	104,801	2,735	**1,930**
	0.98	90-103	31	0.21	**0.10**	2,357	4.3	**1.7**
500	0.4	11	0.35	**0.31**	0.32	125	82	**75**
	0.5	13-14	3.6	**2.8**	**2.8**	1,108	613	**603**
	0.6	17	63	**41**	**41**	15,699	7,312	**7,269**
	0.7	22-23	3,268	1,539	**1,533**	675,344	226,163	**225,271**
1,000	0.2	7-8	**0.13**	**0.13**	0.20	42	**35**	**35**
	0.3	9-10	1.3	**1.2**	1.3	458	325	**323**
	0.4	12	16	**13**	**13**	4,422	2,768	**2,691**
	0.5	15	394	**290**	**290**	90,952	51,255	**51,246**
3,000	0.1	6-7	0.73	**0.60**	1.0	143	135	**134**
	0.2	9	13	**9.5**	10	2,802	2,013	**2,012**
	0.3	11-12	360	**288**	**288**	75,164	50,197	**50,040**
5,000	0.1	7	5.3	**3.3**	4.8	532	**508**	**508**
	0.2	9	197	**135**	138	30,951	26,220	**26,219**
10,000	0.1	7-8	100	**60**	74	5,501	3,899	**3,896**
15,000	0.1	8	511	**327**	342	22,526	15,276	**15,275**

表 4: 他のアルゴリズムの実行時間 2 (sec)

Graph		dfmax	New	COCR
n	p	[1]	[6]	[4]
100	0.6	0.0041	0.0022	0.092
	0.7	0.018	0.0067	0.12
	0.8	0.14	0.065	0.15
	0.9	3.7	0.66	0.20
	0.95	26	0.20	
200	0.6	0.29	0.17	0.52
	0.7	3.9	3.0	1.7
	0.8	193	147	8.7
	0.9	>100,000		**37**
	0.95	>100,000		
	0.98	>100,000		
500	0.4	0.65	0.60	
	0.5	9.0	7.3	17
	0.6	242	183	
	0.7	24,998		
1,000	0.2	0.17	0.21	
	0.3	2.0	1.6	
	0.4	33	23	
	0.5	1,108		
3,000	0.1	0.80		
	0.2	17		
	0.3	631		
5,000	0.1	5.9		
	0.2	242		
10,000	0.1	129		
15,000	0.1	803		

表 5: init-lb の近似解と実行時間 (sec)

Graph		init-lb	
Name	ω	近似解	実行時間
r100.5	9	9	0.00075
r500.5	13	12	0.032
brock200_1	21	20	0.0039
brock200_3	15	11	0.0039
brock400_1	27	23	0.019
brock400_3	31	24	0.019
c-fat200-2	24	24	0.00088
c-fat500-1	14	14	0.0021
hamming8-2	128	128	0.016
hamming10-2	512	512	0.52
johnson8-4-4	14	14	0.00055
johnson16-2-4	8	8	0.0025
keller4	11	11	0.0027
MANN_a27	126	125	0.065
MANN_a45	345	344	1.3
p_hat500-3	50	48	0.035
p_hat700-3	62	62	0.079
p_hat1000-2	46	45	0.20
p_hat1500-1	12	11	0.30
san1000	15	10	0.090
san200_0.9_2	60	60	0.0039
san400_0.5_1	13	13	0.010
san400_0.9_1	100	100	0.017
sanr200_0.7	18	18	0.0039
sanr200_0.9	42	41	0.0036
sanr400_0.5	13	12	0.019
sanr400_0.7	21	20	0.020