Vol.2016-AL-157 No.8
2016/3/6

最大クリーク抽出アルゴリズムMCSの高速化

吉田 幸平[1]　八田 拓郎[2]　富田 悦次[3]　長尾 篤樹[2]　伊藤 大雄[2]　若月 光夫[2]

概要：　無向グラフ中の最大クリークを 1 つ抽出する問題は, NP 困難に属する基本的な組合せ最適化問題であり, 現実の様々な問題が最大クリーク抽出，あるいはそれに類似した問題としてモデル化できることから，工学的に非常に重要な問題となっている．これに対し, 実働上で効率的に解を得ることを目的とした多くのアルゴリズムが開発されてきており, 逐次的な近似彩色を用いた分枝限定アルゴリズムが効果的とされている．本稿では, 厳密解アルゴリズムの先頭処理として近似解アルゴリズムの適用や, 探索節点の順序変更および, 部分問題に応じて適用する近似彩色アルゴリズムの適応的な切り替えを主に用いて, 従来のアルゴリズム MCS を高速化した．さらに, 計算機実験による評価を与えることにより, 他のアルゴリズムに対する優位性を示した．

An Improved MCS Algorithm for Finding a Maximum Clique

Kohei YOSHIDA[1]　Takuro HATTA[2]　Etsuji TOMITA[3]　Atsuki NAGAO[2]　Hiro ITO[2]
Mitsuo WAKATSUKI[2]

1. はじめに

単純無向グラフ中の最大クリークを１つ抽出する問題に対し，実働上で効率的に解を得ることを目的としたアルゴリズムが活発に開発されていきている [1–3, 5–17]．現実の様々な問題が, 最大クリーク抽出またはそれに関連した問題としてモデル化できることがわかっていることから，最大クリーク抽出問題は工学的に非常に重要問題となっている．クリーク抽出アルゴリズムを適用できる応用例として具体的には，符号理論やバイオインフォマティクス，画像処理，量子論理回路の最適設計，DNA 計算における DNA 系列設計，データマイニングなどが挙げられる．しかし, 最大クリーク抽出問題は NP 困難問題であるため, 問題となるグラフの規模が増大するに従い, 最大クリーク抽出に要する計算量が指数的に増加することが強く予測されているため, 最大クリーク抽出アルゴリズムの高速化が望まれる [17].

この問題に対しこれまで筆者らは, 深さ優先探索を基礎とし, そこに逐次的な近似彩色を用いた分枝限定法などを加えた効率的な最大クリーク抽出アルゴリズム MCQ [13], およびそれを改良した MCR [14], MCS [15, 16] を提案してきた．本稿ではアルゴリズム MCS に対し, 主に近似解を求める前処理や, 部分問題に応じたアルゴリズムの切替など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．なお, 本稿は既に発表した文献 [2] の改良結果である．

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 v の近傍と呼び $\Gamma(v)$ で表し, その要素数を v の次数と呼ぶ．枝の存在割合 $Dens(G)$ は，$Dens(G) = \frac{2|E|}{|V|(|V|-1)}$ で与えられる．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

[1] 電気通信大学 情報理工学部
[2] 電気通信大学大学院 情報理工学研究科
[3] 電気通信大学 先進アルゴリズム研究ステーション

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. 従来のアルゴリズム MCS

本稿では, グラフのデータである隣接行列が入力されたとき, 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を対象とする. 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 [13], MCR [14], MCS [15, 16] 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 [13, 14], 再彩色手続き RE-NUMBER [16] について述べる.

3.1 基本アルゴリズム

基本アルゴリズムは, ある時点で保持しているクリーク Q に, そのクリーク中の全ての節点と隣接している節点 (このような節点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再帰的に繰り返し, クリークサイズを深さとした深さ優先探索を行なう. 従って探索が深さ n の段階であるとき，Q には節点数 n のクリーク，即ちサイズ n のクリークが格納されていることになる．この再帰手続きを EXPAND と呼ぶことにする．EXPAND を再帰的に繰り返すことにより, 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．クリークが極大となった場合, 深さ優先探索の基本に従い, 1 つ前の再帰処理にバックトラックし, 別の候補節点を選んで EXPAND 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, 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．親子関係とは, ある部分問題を親としてその部分問題を EXPAND することにより新たに出来た部分問題を子とする関係を表す．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

先に述べた基本アルゴリズムのような方法では, 全ての組合せについて考慮することになり効率が良いとは言えない. そこで, 効率よく最大クリークを抽出するためには, 探索過程での分枝数を削減することが効果的であるとされている．基本アルゴリズムではグラフに含まれる全クリークを抽出するが, 最大クリーク抽出問題では必ずしも全てのクリークを探索する必要は無い．そのため, 何らかの方法を用いて探索過程での分枝数を削減する, いわゆる分枝限定法がアルゴリズムの効率化に用いられる．最大クリーク抽出において, 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．ここで, $|Q| + |R| \leq |Q_{max}|$ という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.

上述のように, 一般的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．一方で, この処理に多くの時間をかけてしまうことで実行時間全体が逆に増加することがあり得る．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となる．分枝数を大幅に削減するために複雑な処理を行う事で, 枝一本当たりの処理時間が増大し, 総合して実行時間も増大してしまう場合がある. 従って, ただ単に分枝数を削減する事ではなく, 分枝限定のための処理にかかる時間とその分枝数の削減のトレードオフ問題を解決する事が重要となる.

より効率的な分枝限定を行なうために, 以下に述べる近似彩色が一般に用いられる.

3.3 近似彩色

従来の最大クリーク抽出アルゴリズム MCQ, MCR では, 逐次的な近似彩色 (番号付け) を用いることにより，より効率的な分枝限定を実現している．近似彩色では, 各節点に正整数の番号を割り振るが, 隣接する節点同士は同じ番号を持たないようにする．EXPAND を行なう度に候補節点集合に対して近似彩色を行うことにより, 候補節点集合により誘導される部分グラフ中のクリークサイズの上界が求まり, これにより有効に分枝限定が行える.

以下に具体的な彩色方法を述べる．彩色は候補節点集合の先頭の節点から順に行い, 各節点には, 候補節点集合中の近傍の節点が持つ番号とは異なる最小の正整数を付与する. 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 をクリークに加えて EXPAND を行なうとき, この節点の彩色番号 $No[p]$ に対して, 分枝限定条件を

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

とすることにより, 分枝限定をより有効に行っている.

3.4 初期節点整列

最大クリーク抽出アルゴリズムの高速化において，入力で与えられたグラフにおける節点の整列順序が重要であることが確認されている [11, 12]．そのため，従来のアルゴリズム MCR, MCS では 3.3 で述べた近似彩色手続きによる分枝限定に加え，探索の前処理として節点の整列を行っている. MCR, MCS における EXTENDED INITIAL SORT-NUMBER [14] 即ち, 文献 [14] の Fig.4 (Algorithm MCR) において {SORT} から EXPAND の直前までの手

続きを以降では拡張イニシャルソートと呼ぶ．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 [11,14]. そのため, これに即した節点の整列を探索の前処理として行う．以降, 拡張イニシャルソートにより, V の末尾には次数最小の節点が格納されているものと考える．MCR, MCS では探索木の根においては常に次数最小の節点を EXPAND することにより探索を効率化している．また, MCS では候補節点を番号順に保持した集合 R とは別に, この初期節点の順番を保持した集合 V_a を構成し, V_a の先頭の節点から順に近似彩色を行なうことにより分枝数を削減している．

3.5 再彩色アルゴリズム Re-NUMBER

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る事で, 分枝限定の効果が大きくなるため, 効率化につながると考えられてきた．しかし, 実際の探索時間を削減するためには, 精度の良い彩色から小さい最大彩色番号を得る事ではなく, 探索すべき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より, $No[p] > |Q_{max}| - |Q|(= No_{th}$ とする) となる節点が少なくな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 MCS で提唱されたのが, 再彩色アルゴリズム Re-NUMBER(図 1) である．C_k には番号 k を持つ節点が格納されている．このアルゴリズムは, 従来の逐次的な近似彩色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．

```
procedure Re-NUMBER(p, No_p, C_1, C_2,...,C_maxno)
begin
  No_th := |Q_max| − |Q|;
  for k_1 := 1 to No_th − 1 do
    if |C_k1 ∩ Γ(p)| =1 then
      q := the element in (C_k1 ∩ Γ(p)) ;
      for k_2 := k_1 + 1 to No_th do
        if C_k2 ∩ Γ(q) = ∅ then
          C_No_p := C_No_p − {p};
          C_k1 := (C_k1 − {q}) ∪ {p};
          C_k2 := C_k2 ∪ {q};
          return
        fi
      od
    fi
  od
end { of Re-NUMBER}
```

図 1 再彩色アルゴリズム Re-NUMBER

MCS では, 逐次的な近似彩色の過程で節点 p に番号 k が彩色されたとき, $(k > No_{th})$ かつ, k がその時点で最大の番号である場合に限り, p に Re-NUMBER を適用している．これは, Re-NUMBER 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．以下, Re-NUMBER 付きの逐次的近似彩色手続きを NUMBER-R とする．また, MCS で適用する Re-NUMBER における k_2 の範囲は $k_2 := k_1 + 1$ **to** No_{th} であるが, 以降, k_2 の範囲を $k_2 := 1$ **to** No_{th} とする Re-NUMBER を Re-NUMBER1 と呼び, さらに NUMBER-R に対し, Re-NUMBER の適用条件を $\{(k > No_{th})$ かつ k がその時点で最大の番号 $\}$ から $(k > No_{th})$ に変更し, Re-NUMBER の代わりに Re-NUMBER1 を適用する NUMBER-R を NUMBER-R+と呼ぶことにする．

3.6 隣接行列の再構成

MCS における近似彩色は拡張イニシャルソートによる節点の順番で行われるため, 隣接行列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, MCS では拡張イニシャルソートが終了した時点で, 隣接行列の再構成を行なっている．具体的には, 整列後の節点を改めて $1, 2, \ldots, |V|$ と置き直し, それに従って隣接行列を再構築する．このようにすることにより, CPU キャッシュのヒット率の向上につながり, 高速化が達成されている．

4. アルゴリズムの効率化

ここで, 本稿で提案するアルゴリズムの詳細を述べる．本稿では, アルゴリズム MCS に以下の 4 つの改良を加えることにより, 広範なグラフに対して概ね 2〜3 倍の高速化を達成した．

4.1 近似解の利用および初期節点整列順の探索

深さ優先探索を基礎とする最大クリークの探索において, 早い段階で最大クリークサイズの下界が厳密解により近い値を得られていることが重要である．最大クリークサイズの下界が大きいほど No_{th} の値が大きくなり, 分枝限定の対象となる節点をより多くすることができる．それに伴い, Re-NUMBER を適用する節点数の減少が考えられるため, Re-NUMBER によるオーバーヘッドの減少も見込める．実際に, いくつかのグラフにおいて, 探索の早い段階で厳密解に近いサイズのクリークが発見できるかどうかが実行時間に大きく影響することが確認されている．

そこで, 厳密解アルゴリズムの先頭処理として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．最大クリーク問題については, 一般に近似困難性が知られているため, 広範な問題に対して高精度の近似解を得る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．これまでに筆者らは, MCQ を基礎とした単純なアルゴリズム init-lb を MCS の先頭処理として適用したアルゴリズムを発表し, その有効性を確認している [10]. ただし, 厳密解に近い近似解を得たとしても多くの時間を費やしてしまっては, アルゴリズム全体の高速化の目的に反してしまう．そのため, 極力短い時間で, 精度の良い近似解を得る必要がある．

そこで本稿では, init-lb と比較して本目的に対してより有効と考えられた, 文献 [4] の近似解アルゴリズム k-opt local search をアルゴリズムの先頭処理として適用することにより近似解を得, これにより MCS 全体の高速化を図る. k-opt Local Search は厳密解アルゴリズムと比較して非常に短時間で, 多くのグラフに対して厳密解に近い解を得られる近似解アルゴリズムである.

また, この手法に伴い, 節点の探索順序にも変更を加える. MCS までのアルゴリズムでは近似彩色手続きの結果, 彩色番号の減少順に節点を探索していた. これは, 大きい番号を持つ節点は大きいクリークに含まれる傾向にあるため, これにより, 探索の早い段階で最大クリークが発見されることが期待される. 一方, これまでの実験により, 探索する節点の順序は次数の増加順に行なうのが効率的であるということが示されている [11,14]. 彩色番号の減少順, 即ち大きい番号が付けられた節点から探索を行うのは, 必ずしも効率的であるとは言えない.

そこで本稿では, 彩色順序と同様に, 探索順序も拡張イニシャルソートの結果に従うことを考える. 即ち, MCS における, 初期節点整列を保持した候補節点集合 V_a を R とみなし, 節点の近似彩色は R の先頭, 即ち次数の大きい節点から行ない, 探索は末尾, 即ち次数の小さい節点から順に, 彩色番号が No_{th} を超える節点に対して行なう. これにより, 次数の小さい節点から探索することによる効率化が期待できる. ここで, 最大クリークが早期には見つからなくなることにより効率的な探索が行えなくなることが懸念されるが, この問題は k-opt Local Search により比較的良好な近似解を得られていることを考慮するとある程度解決されていると考えられる. また MCS における, V_a への近似彩色終了後に番号順の候補節点集合 R を構成するオーバーヘッドを省くことができる. 当手法を取り入れた場合, 彩色の途中で逐一 No を書き換える方式をとるが, 以降の彩色アルゴリズムにおいてそのことは明記せず省略する.

アルゴリズム k-opt Local Search

アルゴリズム k-opt Local Search も init-lb 同様, 各探索において 1 つの節点を起点として探索を行なう. 但し, 各探索により得られる最大クリークは一つとは限らない. 多くの節点を起点として, 探索回数を増やすほど得られる近似解精度も向上することが期待されるが, そのためにアルゴリズム全体の実行時間が増えてしまうことは避ける必要がある. 本稿においては，探索を開始する節点の数は実験的に $20\sqrt{|V|} \times Dens(G)^3$ としている．このようにすることにより，辺密度が高いグラフでは探索する節点の数が多くなり，逆に辺密度が低いグラフに対しては探索する節点の数は少なくなる.

また, 探索の起点とする節点は, 拡張イニシャルソートの結果に従って V の先頭の節点から順に 1 度ずつ選択している. これは, 次数の大きい節点ほど大きいクリークに含まれる傾向にあり, また MCS の探索では早期に発見されにくいクリークを近似解で得る事が望ましいためである. 以降, V の先頭から $20\sqrt{|V|} \times Dens(G)^3$ 個の節点を起点とした k-opt Local Search の近似解アルゴリズム全体を KLS と呼び, アルゴリズム MCS の先頭処理に KLS を追加したアルゴリズムを MCS_1 と呼ぶ.

適用効果

当手法により, 多くの DIMACS ベンチマークグラフに対して実行時間の減少が見られた. 特に gen400_p0.9_55, 65, 75 に対する効果が大きく, それぞれ 13 倍, 10,000 倍, 100,000 倍程度の高速化となっていた. また, C や p_hat 系列のグラフや sanr200_0.9 に関して概ね 2 倍の高速化が見られた. それ以外のグラフに関しては 1〜2 割の実行時間減少であった. ただし, MANN 系列のグラフ対しては KLS の実行時間とほぼ同じだけ実行時間増加となっていた.

4.2 根に近い部分問題における拡張イニシャルソート

最大クリーク抽出問題の分枝限定をより効率的に行なうためには, 近似彩色の彩色精度を向上させ, 探索する節点数を減少させることが考えられる. しかし実際には, 探索木の根に近い部分問題などはグラフに存在する最大クリークサイズと彩色番号の最大値の差が大きくなる傾向があるため, 彩色精度の向上による効率化が比較的難しい. むしろ, こういった部分問題に関しては, 彩色精度を落としてでも, 次数の小さい節点から順に探索することによる効率化を行なった方が有効な場合がある. 実際に MCR, MCS は根の部分問題に限定し, 通常の番号付けを行なう場合と比較して非常に単純な番号付けを用いているにも関わらず, 常に最小次数の節点を優先して探索することによる効率化を達成している.

そこで本稿では, 探索木の根に近い部分問題においては拡張イニシャルソートを適用することを考える. これに伴い, 拡張イニシャルソートによる彩色精度の低下を少しでも抑えるため, 拡張イニシャルソートを行なった後, R の先頭から順に NUMBER-R を適用する. また, MCS においては, 逐次彩色の途中で最大の番号が付いた節点に限定して Re-NUMBER を適用することによりオーバーヘッドを抑えつつ彩色精度を向上させていたが, 拡張イニシャルソートを適用する部分問題数は探索木全体の部分問題数と比較して僅かとするため, 拡張イニシャルソートを適用した部分問題に関しては, No_{th} を超える番号が付いた節点全てに Re-NUMBER1 を適用し, 多少のオーバーヘッドを許容して彩色精度の向上を考える. 探索は R の末尾, 即ち次数最小の節点から順に行ない, R 中で最大の彩色番号が No_{th} 以下となればバックトラックする. また, 探索木における根以外においては, 節点の順序が大幅に変わることは少ないと考えられるため, 隣接行列の再構築は行わない.

適用効果

5. に後述するパラメータを $Th_1 = 0.4$, $Th_2 = 0$ としたとき, 主に brock 系列のグラフや sanr400_0.7 に効果が見られ, 概ね 3 割程度の実行時間減少となった. その他のグラフに関しては効果が微小であるか, 極僅かに増加していた. 全体として, 辺密度が 0.7〜0.9 のグラフに対して 1〜4 割程度の実行時間減少となった. 以下, $Th_1 = 0.4, Th_2 = 0$ としたアルゴリズムを MCS_2 と呼ぶ.

4.3 葉に近い部分問題における簡易番号付け

探索木中でそれ以上多くの部分問題に展開しない部分問題, 即ち葉に近い部分問題が占める割合は非常に大きいと考えられる. そのため, 探索木の葉に近い部分問題における処理を低減することはアルゴリズム全体の実行時間の削減に大きく寄与すると考えられる.

そこで本稿では, 主に探索木の葉に近い部分問題においては, 一部分の節点にのみ番号を付け直す簡易番号付けを考える. より具体的には, 一つ上の親問題における彩色結果を引き継いだ上で, その番号が No_{th} を超える節点集合のみ逐次彩色および Re-NUMBER を適用する. 全節点の番号を付け直していないために彩色の精度は落ちることが予想されるので, 以下のようにして彩色精度を向上させる. まず, 逐次彩色により No_{th} 以下の番号を付けることが出来ない節点には全て Re-NUMBER1 を適用し, Re-NUMBER 中で k_2 の範囲を従来の $k_1+1 \leq k_2 \leq No_{th}$ から $1 \leq k_2 \leq No_{th}$ へと拡大する. 葉に近い部分問題においては番号を付け直す節点の数は全節点に対して一般に少ないため, これによるオーバーヘッドの増加は限定的であり, 総合的には彩色精度をある程度保ったままオーバーヘッドを削減することが期待できる. このアルゴリズムの詳細を図 2 に記載する. 以下, 簡易番号付けに NUMBER-R+を追加した逐次近似彩色手続きを NUMBER-RL とする.

但し, 簡易番号付けによる彩色精度の低下は分枝数の増加につながり, 特に辺密度の極端に高いグラフに関して分枝数および実行時間の急激な増加が確認された. そのため, 当手法はグラフの辺密度が一定 (0.95 とする) 以下である場合にのみ適用する.

適用効果

5. に後述するパラメータを $Th_1 = 0.4, Th_2 = 0.1$ としたとき, 殆ど全てのグラフに対して実行時間の改善が見られ, 概ね 2〜5 割程度の実行時間削減につながった. 当手法は, 特に辺密度が 0.4〜0.9 のグラフに関して効果が高かった.

5. アルゴリズム MCT

従来の最大クリーク抽出アルゴリズム MCS に, 4.1 〜 4.3 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 MCT とする. MCT においては, 拡張イニシャルソートと簡易番号付けを適用する条件として以下の基準を用いる.

procedure NUMBER-RL($R, No, newNo$)
begin
  for $i := 1$ **to** $|R|$ **do**
    $C_i := \emptyset$;
  od
  $maxno := 1$;
  for $i := 1$ **to** $|R|$ **do**
    if $No[R[i]] \leq No_{th}$ **then**
      $k := No[R[i]]$;
      if $k > maxno$ **then** $maxno := k$ **fi**
      $C_k := C_k \cup R[i]$;
    fi
  od
  for $i := 1$ **to** $|R|$ **do**
    if $No[R[i]] > No_{th}$ **then**
      $p := R[i]$;
      $k := 1$;
      while $C_k \cap \Gamma(p) \neq \emptyset$
        do $k := k + 1$ **od**
      if $k > maxno$ **then**
        $maxno := k$;
      fi
      $C_k := C_k \cup \{p\}$;
      if $(k > No_{th})$ **then**
        Re-NUMBER1$(p, k, C_1, C_2, \ldots, C_{maxno})$;
        if $C_{maxno} = \emptyset$ **then**
          $maxno := maxno - 1$
        fi
      fi
    fi
  od
end { of NUMBER-RL}

図 2 簡易近似彩色手続き NUMBER-RL

$$\frac{No_{th}\text{を超える番号を持つ節点の数}}{\text{候補節点数}} \times \text{グラフの辺密度}$$

ここで, 節点の番号は探索木における親の部分問題の結果を用いるものとする. この値が大きい部分問題は根に近く, 小さい部分問題は葉に近い問題と考えられる. そのため, MCT ではこの値が Th_1 以上の部分問題は拡張イニシャルソートを, Th_2 以下の部分問題は簡易番号付けを適用し, それ以外では通常の NUMBER-R を適用する. 但し, 根の部分問題には必ず拡張イニシャルソートを用いる. また, 一度 NUMBER-R もしくは NUMBER-RL が適用された部分問題以降は NUMBER-R または NUMBER-RL を適用し, 拡張イニシャルソートは行わない. そのために, 親問題において拡張イニシャルソート以外の彩色アルゴリズムを適用したかを保持する変数 $stage$ を用意する.

KLS によって得られた近似解は Q'_{max} に保持し, 根の部分問題に対する拡張イニシャルソートで得られる近似解を Q_{max0} に保持し, これらを比較して大きい近似解を最大クリークサイズの暫定的な下界として用いる.

また, 辺密度が 0.1 以下のグラフに対しては上記の手法では有効性を確認できなかったため, グラフの辺密度が 0.1 以下の場合は単純なアルゴリズム MCS に切替を行なう. アルゴリズム MCT の詳細を図 3 に記載する.

なお, このように適切な条件下でアルゴリズムを適応的に適用して有効性を発揮する手法は, 既に文献 [5,9] におい

て確認した結果に基づいている．

```
procedure MCT(G = (V, E))
begin
global Q := ∅; global Q_max := ∅;
global dens := 2|E|/|V|(|V| − 1);
  if dens ≤ 0.1 then
    MCS(G = (V, E));
  else
    Th_1 := 0.4;      Th_2 := 0.1;
    拡張イニシャルソート (V, Q_max0);
    グラフデータの再構成;
    KLS(V, Q'_max);
    Q_max := max{Q_max0, Q'_max};
    NUMBER-R+(V, No);
    stage := 1;
    EXPAND (V,No,stage);
  fi
  output Q_max      { 最大クリーク }
end { of MCT}

procedure EXPAND(R, No, stage)
begin
  for i := |R| downto 1 do
    p := R[i];
    if (stage = 1 and |Q| + max_{v∈R}{No[v]} > |Q_max|)
    or (stage ≠ 1 and |Q| + No[p] > |Q_max|) then
      Q := Q ∪ {p};
      newR := R ∩ Γ(p);    { 順序は保存する }
      newNo := No;
      if newR ≠ ∅ then
        T := (|{v|newNo[v] > No_th}|)/(|newR|) × dens;
        if stage = 1 and Th_1 ≤ T then
          拡張イニシャルソート (V, Q_max0);
          NUMBER-R+(newR, newNo);
          newstage := 1;
        else if dens > 0.95 or Th_2 ≤ T then
          NUMBER-R(newR, newNo);
          newstage := 2;
        else
          NUMBER-RL(newR, No, newNo);
          newstage := 3;
        fi
        EXPAND(newR, newNo, newstage)
      else if |Q| > |Q_max| then Q_max := Q fi
      fi
      Q := Q − {p};
      R := R − {p};      { 順序は保存する }
    fi
  od
end { of EXPAND }
```

図 3 アルゴリズム MCT と再帰手続き EXPAND

6. 計算機実験

テストグラフは, 最大クリーク抽出アルゴリズムの実行時間比較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．複数の厳密解アルゴリズムの実行結果を実行時間を表 4 に掲載した．MCT のパラメータは $Th_1 = 0.4$, $Th_2 = 0.1$ とした．ランダムグラフは, 節点数 n, 枝存在確率 p について, 乱数のシードが異なるものを 10 個ずつ用意した．表に掲載している値はこれらの平均値となっている．

アルゴリズム BBMCX および MaxCLQ は, 文献 [8] による実行時間を本実験環境に合わせて校正 [3] を行った値 (表 3) を掲載する．その他のアルゴリズムに関しては, 元の文献により各自実装されたアルゴリズム MCS の実行時間と本実験環境による MCS の実行時間をグラフ毎に比較し, その比を乗ずることによりグラフ毎に校正を行なっている．そのため, 元の文献中で実験されていないグラフに加えて, MCS の実行時間が掲載されていないグラフは表中で空欄となっている．なお, BGMP14 は文献 [1] 中の Our algorithm に相当する．また, 改良の各段階の実行時間および分枝数をそれぞれ表 1, 2 に掲載した．

計算機環境

実験に用いた計算機仕様は次の通り．

OS: Linux, CPU: Intel core i7-4790, クロック周波数: 3.6GHz, 主メモリ: 8GB, キャッシュメモリ: 8MB, コンパイラ: gcc -O3.

7. 考察

まず従来のアルゴリズム MCS と, 本稿でその改良を提唱したアルゴリズム MCT を比較する．表 4 より, MCT は実験を行なったグラフのうち大部分に対して 2〜3 倍の高速化となっている．gen 系列のグラフに関しては高速化が著しいが, これは主に KLS による近似解の効果となっており, 加えて簡易番号付けにより 4〜11 倍程度の高速化がなされている．gen400_p0.9_55 は KLS により厳密解が得られておらず, その他の gen 系列グラフと比較して高速化の程度が低いが, 厳密解を与えたうえで MCT を実行した場合, 実行時間は概ね 3 秒程度であった．このように, gen 系列のグラフにおいては厳密解の発見されるタイミングが実行時間全体に大きく影響する．

MANN 系列のグラフに関しては改良効果が見られず, KLS の実行時間の分だけ全体の実行時間が増加している．これは MANN 系列のグラフは探索の初期で厳密解が得られ, その後の探索は根に近い部分問題が大半となる傾向があるため, 本稿で提案した手法では実行時間の改善が見られなかったと考えられる．

また, p_hat 系列のグラフに関しては MCS と比較して概ね 3〜8 倍程度の高速化がなされている．これは, 主に KLS による近似解および簡易番号付けの効果となっている．MCS までのアルゴリズムでは厳密解が求まるタイミングが探索の後半である傾向があるため, 大半の部分問題において No_{th} の値が小さくなってしまい, 分枝限定が効率的に行えていなかった．これに対して KLS を適用することにより, 探索の後半で抽出されていたサイズの大きい極大クリークを探索前に抽出されるため, No_{th} の値が大きくなり分枝限定が効率的に行えるようになった．p_hat 系列の多くのグラフにおいて, MCS と比較して分枝数は概ね 2〜3 倍減少している．さらに, 簡易番号付けにより概ね

2〜3 倍の高速化となっている.

次に, MCS 以外のアルゴリズムと MCT を比較すると, 辺密度が非常に高いグラフは MaxCLQ が高速であるが, それ以外の多くのグラフに関して MCT が最速と言える結果となっており, 特に辺密度が 0.8 以下のグラフに関して最速である場合が多い.

8. おわりに

本稿では, 最大クリーク抽出アルゴリズム MCS に対する幾つかの効率化手法を提案し, 広範な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 MCT を提唱し, アルゴリズム高速化を達成していることを示した. また, 他のアルゴリズムに対する MCT の優位性を示した.

謝辞

ご討論, 支援をいただいた電気通信大学・戸田貴久 助教, 中西裕陽氏, 他関係者に感謝します. 本研究は科研費・基盤研究 (C) および新学術領域研究, 柏森情報科学振興財団, CREST「ビッグデータのための革新的アルゴリズム基盤」による助成を受けている.

参考文献

[1] Mikhail Batsyn, Boris Goldengorin, Evgeny Maslov, Panos M. Pardalos, “Improvements to MCS algorithm for the maximum clique problem,” Journal of Combinatorial Optimization, vol. 27, issue. 2, pp. 397–246 (2014)
[2] 八田拓郎, 富田悦次, 伊藤大雄, 若月光夫, “最大クリーク抽出アルゴリズムの高速化,” 夏の LA シンポジウム, pp.9:1-9:8 (2015)
[3] David S. Johnson, Michael A. Trick, ed., “Cliques, Coloring, and Satisfiability,” DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 26, American Mathematical Society (1996)
[4] Kengo Katayama, Akihiro Hamamoto, Hiroyuki Narihisa, “An effective local search for the maximum clique problem,” Information Processing Letters, vol. 95, isuue 5, pp. 503–511 (2005)
[5] 木幡康弘, 西島大樹, 富田悦次, 藤橋忠悟, 高橋治久, “最大クリーク抽出アルゴリズムの効率化,” 電子情報通信学会技術研究報告, COMP89-113, pp. 1–8 (1990)
[6] Chu-Min Li, Zhe Quan, “An efficient branch-and-bound algorithm based on MaxSAT for the maximum clique problem,” 24th AAAI Conference on Artificial Intelligence, pp. 128–133 (2010)
[7] Evgeny Maslov, Mikhail Batsyn, Panos M. Pardalos, “Speeding up branch and bound algorithms for solving the maximum clique problem,” Journal of Global Optimization, vol. 59, pp. 1–21 (2014)
[8] Pablo San Segundo, Alexey Nikolaev, Mikhail Batsyn “Infra-chromatic bound for exact maximum clique search,” Computer and Operations Research, vol.64, pp. 293–303 (2015)
[9] 新道美喜男, 富田悦次, 丸山 徳 “効率的な最大クリーク抽出アルゴリズム,” 電子情報通信学会技術研究報告, CAS86-5, pp. 33-40 (1986)
[10] 須谷洋一, 東貴紀, 富田悦次, 高橋真也, 仲谷洋幸, “最大クリーク抽出のより高速な分枝限定アルゴリズム,” 情報処理学会研究報告, 2006-AL-108, pp. 79–86 (2006)
[11] 富田悦次, 藤井利昭, “最大クリーク抽出の効率化手法とその実験的評価,” 電子通信学会論文誌 (D), vol. J68-D, no. 3, pp. 221-228 (1985)
[12] Etsuji Tomita, Yasuhiro Kohata, Haruhisa Takahashi, ” A simple algorithm for finding a maximum clique,” Technical Report of the University of Electro-Communications, UEC-TR-C5(1) (1988)
[13] Etsuji Tomita, Tomokazu Seki, “An efficient branch-and-bound algorithm for finding a maximum clique,” DMTCS 2003, LNCS 2731, pp. 278–289 (2003)
[14] Etsuji Tomita, Toshikatsu Kameda, “An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments,” Journal of Global Optimization 2007, 37, pp. 95–111 (2007)
[15] Etsuji Tomita, Yoichi Sutani, Takonori Higashi, Shinya Takahashi, Mitsuo Wakatsuki, “A simple and faster branch-and-bound algorithm for finding a maximum clique,” WALCOM 2010, LNCS 5942, pp. 191–203 (2010)
[16] Etsuji Tomita, Yoichi Sutani, Takanori Higashi, Mitsuo Wakatsuki, “A simple and faster branch-and-bound algorithm for finding a maximum clique with computational experiments,” IEICE Trans. on Information and Systems, vol. E96-D, no. 6, pp. 1286-1298 (2013)
[17] Q. Wu, J.K. Hao, “A review on algorithms for maximum clique problems,” European Journal of Operational research, vol. 242, pp. 639–709 (2015).

表 1 各段階の実行時間 [sec]

Graph	MCS	MCS_1	MCS_2	MCT
brock400_1	288	256	182	115
brock800_4	1,768	1,751	1,256	819
C250.9	1,171	926	774	404
gen400_p0.9_55	22,535	1,651	1,970	167
gen400_p0.9_65	57,384	5.73	6.07	0.735
gen400_p0.9_75	108,298	1.38	1.38	0.327
p_hat700-3	900	456	438	216
p_hat1000-2	85	47	46	28
p_hat1500-2	6,298	2,964	2,832	1,560
DSJC1000.5	141	135	124	93

表 2 各段階の分枝数 [$\times 10^{-3}$]

Graph	MCS	MCS_1	MCS_2	MCT
brock400_1	89,389	76,596	51,784	55,257
brock800_4	380,739	380,408	257,543	269,819
C250.9	254,759	197,321	153,693	186,331
gen400_p0.9_55	2,894,935	180,979	209,568	61,224
gen400_p0.9_65		330	341	131
gen400_p0.9_75		54	52	15
p_hat700-3	88,168	42,753	39,816	53,889
p_hat1000-2	12,618	6,646	6,266	10,049
p_hat1500-2	560,485	253,124	233,749	399,853
DSJC1000.5	51,527	49,344	43,098	44,994

表 3 dfmax に対する実行時間比較 [sec]

Graph	MCT	BBMCX	
	Our	Segundo's	
	T_1	T_2	T_2/T_1
r300.5	0.14	0.189	1.35
r400.5	0.9	1.155	1.28
r500.5	3.44	4.369	1.27

表 4 KLS と厳密解アルゴリズムの実行時間 [sec]

Graph		近似解 (KLS)		MCS	MCT	BBMCX	MaxCLQ	ILS&MAXSAT	BGMP14
	ω	解	Time	[16]		[8]	[6]	[7]	[1]
brock200_1	21	21	0.01	0.359	0.226	**0.185**	0.344	4.42	2.51
brock400_1	27	25	0.08	288	**115**	149	205	187	302
brock400_2	29	24	0.08	123	**51**	67	96	93	131
brock400_3	31	24	0.08	195	**85**	119	160	144	210
brock400_4	33	25	0.08	103	**46**	68	99	72	86
brock800_1	23	21	0.22	4,121	**1,947**	2,689	4,561	3,998	4,215
brock800_2	24	21	0.22	3,682	**1,633**	2,415	4,002	3,462	3,778
brock800_3	25	21	0.22	2,540	**1,105**	1,587	2,510	2,360	2,649
brock800_4	26	20	0.22	1,768	**819**	1,099	1,853	1,684	1,867
C250.9	44	44	0.08	1,171	**404**	713	268		
C2000.5	16	15	0.59	33,898	**21,026**				
gen400_p0.9_55	55	53	0.25	22,535	**167**	19,361		46,504	2,964
gen400_p0.9_65	65	65	0.26	57,384	**0.735**	66,134	36,683	2,129	18
gen400_p0.9_75	75	75	0.28	108,298	**0.327**	47,176	9,984	83	7
MANN_a27	126	126	0.81	0.260	1.05	0.175	**0.159**	1.30	
MANN_a45	345	344	21.5	53	75	32	22	**17**	55
p_hat300-3	36	36	0.06	0.987	**0.277**	0.664	1.16	6	3
p_hat500-2	36	36	0.05	0.295	**0.100**	0.192	0.497	38	12
p_hat500-3	50	50	0.22	57	**17**	33	39	50	59
p_hat700-2	44	44	0.11	2.16	**0.674**	1.53	3.61	58	29
p_hat700-3	62	62	0.46	900	**216**	679	878	552	767
p_hat1000-2	46	46	0.23	85	**28**	73	101	203	112
p_hat1000-3	68	68	1.00	305,145	**38,808**				
p_hat1500-1	12	11	0.03	1.82	**1.40**	1.9462	10	478	422
p_hat1500-2	65	65	0.73	6,298	**1,560**	3,851	8,026	5,346	5,434
san1000	15	10	0.06	1.02	**0.213**	0.682	0.719	449	157
san400_0.7_3	22	18	0.05	0.665	**0.273**	0.391	0.433	26	11
san400_0.9_1	100	100	0.28	**0.0481**	0.252	0.191	0.795	5.18	0.289
sanr200_0.9	42	42	0.06	15.3	4.67	7.38	**4.21**	4.62	10.2
sanr400_0.7	21	21	0.06	77	**40**	44	81	86	81
DSJC1000.5	15	15	0.12	141	**93**	101	265		
keller5	27	27	0.34	82,421	9,999	30,299	4,982	**5,777**	82,507
200_0.8	24-27	24-27	0.028	1.66	**0.783**	0.946	1.08		
200_0.9	40-43	40-43	0.058	28	11	14	**6**		
200_0.95	60-66	60-66	0.095	32	14	30	**2**		
300_0.7	20	19-20	0.034	5.31	2.93	**2.6**	4.69		
300_0.8	28-29	28-29	0.061	161	**60**	89	86		
500_0.5	13-14	12-13	0.026	1.33	1.09	**0.644**	2.09		
500_0.6	17-18	16-17	0.056	17	11	**10**	22		
500_0.7	22-23	21-22	0.101	722	**339**	423	563		
1000_0.4	12	11	0.045	5.99	5.14	**4.52**	14.5		
1000_0.5	15-16	14-15	0.122	134	**91**	103	230		
2000_0.3	10-11	9-10	0.07	15	**13**				
2000_0.4	13-14	12	0.21	459	**366**				
3000_0.2	9	7-8	0.072	3.67	**3.42**	4.34	34		
5000_0.1	7	5-6	0.149	1.17	**1.17**	1.19	68		
5000_0.2	9-10	7-8	0.21	44	**38**	68	578		
10000_0.05	6	3-5	0.571	1.98	**1.98**				
10000_0.1	7	5-6	0.582	13	**13**	20	683		
15000_0.1	8	6	1.299	62	**62**	114	2,749		