最大クリーク発見に関する省メモリアルゴリズム

A Space Efficient Algorithm for Finding A Maximum Clique

小畠 教寛

Norihiro Obata

令和 **2** 年 **2** 月

北海道大学工学部 情報エレクトロニクス学科
情報理工学コース
情報知識ネットワーク研究室

要 旨

無向グラフから最大クリークを 1 つ発見する問題は，NP 困難に属する組み合わせ最適化問題である．現実の様々な問題が，最大クリーク抽出やそれに類する問題として，モデル化できることから，最大クリーク問題は，工学的に重要な問題となっている．最大クリーク抽出のアルゴリズムとして，彩色による分枝限定法を用いた高速なアルゴリズムが，Tomita らによって提案されている．このアルゴリズムは，省メモリ化に対しては，多く研究されていない．本論文では，最大クリークが巨大なグラフに対しても実用上で，解を得られるように最大クリークを発見する省メモリで動作するアルゴリズムを提案する．実験において，提案アルゴリズムは，最大クリークのサイズが，大きなグラフに対して，最大メモリ使用量が 1MB から 7MB 程度に減少することが確認された．

目次

第1章

はじめに

1.1 背景

最大クリークを発見する問題は，NP 困難問題のクラスに属する組合せ最適化問題の一つとして古くから研究されている問題である [1]．現実の様々な問題が，最大クリークの発見やそれに類する問題としてモデル化できることから，最大クリークを発見する問題は，工学的に重要な問題となっている．

たとえば，株式市場におけるマーケットグラフ（market graph) の最大クリークは重要な特徴を表す．マーケットグラフとは，各株の銘柄を頂点とし，株価の相関係数がある閾値を超える頂点どうしを辺で接続したグラフである．閾値を高い値にすると，辺が存在することは，2 つの株に重要な相関関係があることを表す．グラフのクリークに含まれる頂点に対応する株は，似た振る舞いをしていることを表す．したがって，最大クリークは，株式市場における似た振る舞いをする株の最大のグループという特徴を表す [2]．

1.2 関連研究

グラフから最大クリークの厳密解を求めるアプローチのほとんどは，分枝限定法を基にしている．厳密解を求める各アルゴリズムの違いは，主に上界と下界を決定する手法と分岐の戦略である．1990 年に，Carraghan と Pardalos [3] は，2 つの工夫を用いた単純なアルゴリズムを提案している．1 つ目は，常に次数最小の頂点を選べるように頂点を整列させることである．2 つ目は，クリークに全ての頂点を加えても最大クリークを更新できなくなったら探索を打ち切ることである．これは，後の厳密解を求める多くのア

ルゴリズムに影響を与えた．同年，Babel と Tinhofer [4] は，彩色を用いることで最大クリークの上界の決定することによって分枝限定をより効率的に探索するアルゴリズムを提案している．その後，2003 年に Tomita [5] らは，彩色による分枝限定法によるアルゴリズムで MCQ が提案している．さらに，Tomita らはそれを改善したアルゴリズムとして，2007 年に MCR [6] を，2010 年に MCS [7][8] を，2016 年に MCT [9] を提案している．

一方，グラフから最大クリークの近似解を高速に求める手法が提案されている．この手法のアプローチには，貪欲な手法と局所探索法がある．貪欲な手法の 1 つとして，2004 年に Grosso らは，DAGS [10] を提案している．これは，現在のクリークをより有望なクリークに変換することと頂点に重みをつけるという 2 点の工夫を用いたアルゴリズムである．局所探索法の 1 つとして，2005 年に Katayama らは，k-opt Local Search(KLS) [11] を提案している．これは，クリークに頂点を追加・削除することで生成可能な解の集合を近傍と捉えて探索を行うアルゴリズムである．

1.3 目的

MCS は，深さ優先探索に基づく最大クリークを発見するアルゴリズムである．MCS のメモリ使用量のボトルネックは，深さ優先探索の各段階は，各段階の探索が終了まで，頂点集合などの情報をコピーして保持する必要があることである．このため，最大クリークが大きいとき，探索が深くなりやすいので，メモリ使用量も大きくなる．

本論文では，最大クリークが巨大なグラフに対しても実用上で解を得られるように最大メモリ使用量の削減に着目をして取り組んだ．我々は，Tomita らの MCS に基づく，その省メモリ化の手法を提案する．省メモリ化のアイデアとしては，深さ優先探索で子に進む際に捨てる情報のみを保持し，子には，頂点集合の情報をコピーせずに渡すことで省メモリ化を行う．MCT では，部分問題に応じて彩色アルゴリズムを切り替える．彩色アルゴリズムの 1 つは親の彩色を引き継ぐため，その情報を保存する必要がある．このため，省メモリ化のアイデアを適用できないので，MCS を改善した MCT ではなく，MCS を基にした．

提案アルゴリズムと基にした MCS に比較して，空間計算量や時間計算量が，改善したかどうかを調べる．私がみる限り，Tomita らは，MCS アルゴリズムに対して，空間計

算量を記載していない．したがって，MCS と提案アルゴリズムの両方に対して，空間計算量と時間計算量についての解析を行う．さらに，提案手法を実装し，35 個のグラフに対して実験を行う．実験に使用したグラフは，DIMACS ベンチマークセットのグラフ[1]とクリークサイズの大きなグラフから辺をいくつか削除することで作成したグラフ，DIMACS のグラフのいくつかのグラフを合体させて，そのグラフ間に辺をランダムに引くことによって作成したグラフの 3 種類を用意した．

1.4 結果

n はグラフのサイズ，m は辺の本数，ω を最大クリークのサイズとする．

本論文の主な貢献は，以下の 5 点である．

- MCS アルゴリズムの隣接行列表現から隣接リスト表現へ適用した．
- MCS アルゴリズムを基にした省メモリなアルゴリズムを提案をした．
- MCS アルゴリズムと提案アルゴリズムの空間計算量について解析を行った．空間計算量について，まとめたものが，表 1.1 である．提案アルゴリズムの隣接リスト表現の空間計算量 $O(n+m)$ が，MCS アルゴリズムの $O(n\omega+m)$ に対して小さく抑えられることを示した．
- 提案アルゴリズムの再帰処理の時間計算量は，隣接行列表現で $O(n^4)$ 時間，隣接リスト表現で $O(n^4 \log n)$ 時間であることを示した．
- MCS アルゴリズムと提案アルゴリズムの 2 つのアルゴリズムに対して，35 個のグラフでの実験を行った．最大クリークのサイズの大きい 14 個のグラフに対して，最大メモリ使用量の削減を確認し，その効果に対しての考察を行なった．実験に使用したグラフは，DIMACS ベンチマークセットのグラフ[1]とクリークサイズの大きなグラフから辺をいくつか削除することで作成したグラフ， DIMACS のグラフのいくつかのグラフを合体させて，そのグラフ間に辺をランダムに引くことによって作成したグラフの 3 種類を用意した．

[1] http://iridia.ulb.ac.be/~fmascia/maximum_clique/DIMACS-benchmark

1.5　本論文の構成

本論文の構成は，次の通りである．第 2 章では，グラフの基本的な定義やグラフの表現について記述する．第 3 章では，最大クリーク抽出アルゴリズムの基本的なアイデアについて説明する．第 4 章では，MCS アルゴリズムを基づく提案アルゴリズムについて説明し，その空間計算量や時間計算量について記述する．第 5 章では，実験結果を記載して，結果に対して考察する．第 6 章では，本論文のまとめと今後の課題について述べる．

表 1.1: アルゴリズムの空間計算量

	MCS アルゴリズム	提案アルゴリズム
隣接行列	$O(n^2)$	$O(n^2)$
隣接リスト	$O(n\omega + m)$	$O(n + m)$

n はグラフのサイズ $\|V\|$，m はグラフの辺の数 $\|E\|$，ω はグラフの最大クリークのサイズ

第 2 章

準備

本章では，本論文で用いる基本的なグラフに関する定義と記法を導入する．まず，2.1 節では，グラフの基本的な定義をする．次に，2.2 節では，グラフの表現方法について説明する．

2.1　グラフ

この節では，グラフの諸定義を行う．本論文では，$[A]^k := \{X \subseteq A | |X| = k\}$ とする．ここで，V を有限集合，$E \subseteq [V]^2$ とする．

無向グラフ

無向グラフは，対 $G = (V, E)$ である．この集合 V を G の頂点集合と呼び，集合 E を G の辺集合と呼ぶ．V の各要素は，頂点と呼ぶ．E の各要素は，辺と呼ぶ．

無向グラフの例として，

$$
\begin{aligned}
V &= \{1, 2, 3, 4, 5, 6\} \\
E &= \{\{1, 2\}, \{1, 4\}, \{1, 5\}, \{2, 3\}, \{2, 4\}, \{2, 5\}, \{3, 4\}, \{3, 5\}, \{3, 6\}, \{4, 5\}, \{5, 6\}\}
\end{aligned}
$$

である無向グラフ $G = (V, E)$ を考える．このとき G は，図 2.1 を示している．

今後，無向グラフのことを単にグラフと呼ぶ．

グラフの基本要素

グラフのサイズとは，頂点集合のサイズのことをいう．頂点 $u, v \in V$ について，$\{u, v\} \in E$ を満たすとき，u, v が隣接しているという．

頂点 $v \in V$ の隣接している頂点の集合を $\Gamma(v)$ と表し，これを隣接頂点集合という．隣接する頂点の数を頂点の次数と呼ぶ．頂点 $v \in V$ の次数を，$deg(v)$ で表す．本論文では，グラフ $G = (V, E)$ の辺密度 $Dens(G)$ は，$\frac{2|E|}{|V|(|V|-1)}$ と定義する．

たとえば，図 2.1 での頂点 1 と 3 の隣接頂点集合は，$\Gamma(1) = \{2, 4, 5\}$ と $\Gamma(3) = \{2, 4, 5, 6\}$ である．頂点の次数は，$(deg(1), deg(2), deg(3), deg(4), deg(5), deg(6)) = (3, 4, 4, 4, 5, 2)$ である．グラフの辺密度は，$\frac{2 \times 11}{6 \times 5} = 0.73$ である．

グラフ $G = (V, E)$ に対して，$V' = V, E' = [V]^2 - E$ を満たすグラフ $G' = (V', E')$ を G の補グラフという．図 2.1 の補グラフは，図 2.2 に示す．

グラフ $G = (V, E)$ の頂点の部分集合 $S \subseteq V$ に対して，$E' = \{\{u, v\} | \forall u, v \in S, \{u, v\} \in E\}$ としたとき，$G' = (S, E')$ を S による誘導部分グラフという．

頂点集合 V 中の任意の 2 頂点が隣接しているとき，グラフ G を，完全グラフという．図 2.3 は完全グラフの例である．

グラフ $G = (V, E)$ に対して，V の部分集合 U に対する誘導部分グラフ $G[U]$ が，完全グラフであるとき，誘導部分グラフ $G[U]$ をクリークという．本論文では，U から誘導されるクリークと U を同一視する．グラフに含まれるクリークで，最も頂点数が多いクリークを，最大クリークとする．最大クリークのサイズを，ω と表す．図 2.1 での最大クリークは，$\{1, 2, 4, 5\}$ および，$\{2, 3, 4, 5\}$ である．

以降では，グラフのサイズを n，辺の本数を m と表記する．

2.2 グラフの表現方法

この節では，代表的なグラフの表現方法である隣接行列表現と隣接リスト表現について説明する．

2.2.1　隣接行列

頂点数 n のグラフに対して，次の条件を満たす $n \times n$ の行列 A を G の隣接行列表現という．A の i 行目 j 列目の要素 a_{ij} に対し，$(i, j) \in E$ なら 1，そうでないなら 0 である．

隣接行列は $n \times n$ 行列なので，空間計算量 $O(n^2)$ で構成することができる．隣接行列でグラフを表した場合，任意の 2 頂点が隣接しているかは，$O(1)$ 時間で判定できる．図 2.1 に関しての隣接行列は，図 2.4 になる．

2.2.2　隣接リスト

隣接リストとは，グラフの隣接関係を表すリストである．グラフ $G = (V, E)$ の隣接リストを A とすると，A_u は，頂点 u の隣接頂点集合 $\Gamma(u)$ となる．隣接リストの空間計算量 $O(n + m)$ で構成することができる．任意の 2 頂点が隣接しているかは，$O(m)$ 時間で判定できる．もし，隣接リストをあらかじめソートして保存することができるなら，二分探索によって，頂点 u との隣接関係を調べるのは，$O(\log |\Gamma(u)|)$ 時間で判定できるように改善できる．図 2.1 に関する隣接リストは表 2.1 のとおりである．グラフ G に対し，その補グラフ G' の隣接リストを持つことでも，同様に確認できる．図 2.1 に関しての補グラフの隣接リストは，表 2.2 のとおりである．

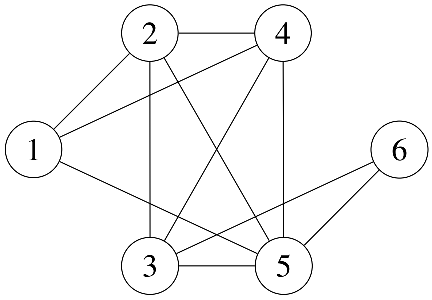

図 2.1: 無向グラフの例

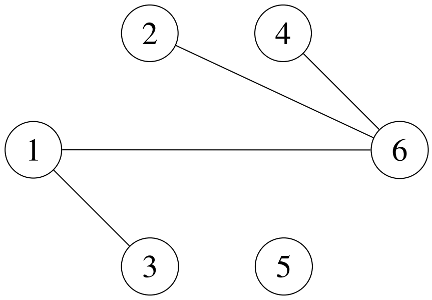

図 2.2: 図 2.1 の補グラフの例

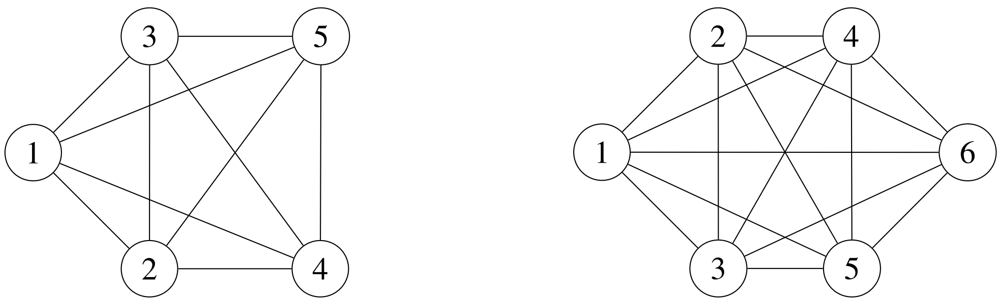

図 2.3: 完全グラフの例

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

図 2.4: 図 2.1 の隣接行列

表 2.1: 図 2.1 の隣接リスト

頂点番号	隣接頂点
1	2,4,5
2	1,3,4,5
3	2,4,5,6
4	1,2,3,5
5	1,2,3,4,6
6	3,5

表 2.2: 図 2.1 の補グラフの隣接リスト

頂点番号	隣接頂点
1	3,6
2	6
3	1
4	6
5	
6	1,2,4

第3章

既存手法

この章では，既存手法で用いられている深さ優先探索に基づく最大クリーク発見のアルゴリズムの説明を行う．3.1 節では，厳密解を発見するナイーブアルゴリズムを与える．3.2 節では，アルゴリズムを高速に動かすための改善方法について説明する．3.3 節では，本論文で提案するアルゴリズムの基になっている MCS アルゴリズム [7][8] について，具体的な説明を行う．

3.1 ナイーブアルゴリズム

このナイーブアルゴリズムは，深さ優先探索に基づき最大クリークを求める．ナイーブアルゴリズムの擬似コードを，Algorithm1 に示す．ある時点で保持しているクリークを Q，その Q の全ての頂点と隣接している頂点の集合を候補点集合 R とする．以下に動作を説明する．まず，R の中から頂点を 1 つ選び R から取り除く．その頂点を p とする．p を Q に加え，R_p を R と p の隣接集合の積集合とする．R_p を R としてこの操作を繰り返し行っていく．R が空になったときグラフの極大クリークが得られる．このとき得られた極大クリークのサイズが，見つかった極大クリークの中で最大のとき，それを最大クリークの候補 Q_{max} にする．R が空になった後はバックトラックをし，Q から p を削除する．その後は，別の頂点を選んで探索を続ける．全ての探索が終了したときに保持している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，全頂点集合 V を根，各時点での候補点集合 R を頂点として，各候補点集合についての親子関係にあるものを枝で結んだ探索木として表現できる．この探索木における枝を分枝，その総数を分枝数と呼ぶ．

Algorithm 1 最大クリーク抽出のナイーブアルゴリズム

procedure MC-NAIVE($G = (V, E)$)
  $Q := Q_{max} := \emptyset$
  EXPAND-NAIVE(V)
  return Q_{max}
end procedure

procedure EXPAND-NAIVE(R)
  while $R \neq \emptyset$ **do**
    $p :=$ a vertex in R
    $R := R - \{p\}$
    $Q := Q \cup \{p\}$
    $R_p := R \cap \{p\}$
    if $R_p \neq \emptyset$ **then**
      EXPAND-NAIVE(R_p)
    else if $Q > Q_{max}$ **then**
      $Q_{max} := Q$
    end if
    $Q := Q - \{p\}$
  end while
end procedure

3.2　近似彩色による分枝限定

MCQ[5] と，MCR[6]，MCS，MCT[9] などのアルゴリズムでは，近似彩色による分枝限定法によって分枝数を抑えることで高速化されている．

3.2.1　分枝限定法

アルゴリズムを高速化するためには，探索過程での分枝数を削減することが効果的である．これを実現することが分枝限定法である．

探索時に保持しているクリークを Q，最大クリーク候補を Q_{max}，候補点集合を R とする．このとき，

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

を満たすとき，最大クリークが存在しないことは，明らかであるので探索を終了する．

分枝限定をより効果的に適用するために彩色を利用する．彩色とは，グラフの隣接頂点同士には異なる番号になるように番号を振ることである．彩色の目的は，最大クリークのサイズの上界をできるだけ小さく求めることである．

補題 1 (彩色による最大クリークのサイズの上界)**.** グラフが隣接頂点同士には異なる番号になるように番号を振られ，彩色に使われている最大の番号を k_{max} とする．このとき，最大クリークのサイズは k_{max} 以下になる．

証明. グラフが彩色されときの彩色に使われた最大の番号を k_{max} のとき，最大クリークのサイズを ω とする．クリークの任意の 2 頂点は隣り合っている．なので，彩色を行われると，必ずクリーク中にはクリークのサイズ以上の彩色番号 k の頂点が存在する．よって，次の式が成り立つ．$\omega \leq k \leq k_{max}$

頂点 p の彩色番号を $\mathrm{No}[p]$ とする．補題 1 より，彩色番号の大きな頂点 p から探索すれば，分枝限定の条件を次の式にすることができる．

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

EXPAND 毎に近似彩色アルゴリズムを適用すると，上記の式を分枝条限定の条件を使うことができるので効率よく探索を終了することができる．

3.2.2 近似彩色

彩色の最適化は, NP 困難に属する組合せ最適化問題であるので, 近似彩色を行う．MCQ と MCR で用いられている具体的な彩色方法について述べる．彩色は，候補点集合の先頭から行い，各頂点には，隣接する頂点の頂点が持つ番号とは異なる最小の正整数を付与する．このように，彩色を行いその番号の昇順に並べる手続きが NUMBER-SORT である．擬似コードを，Algorithm 2 に示す．C_k には，彩色番号 k を持つ頂点が保存されている．

3.3 MCS アルゴリズム

MCS アルゴリズムは，3.1 節のナイーブアルゴリズムを基づき，後述する NUMBER-SORT-R を彩色アルゴリズムとするアルゴリズムである．MCS の擬似コードを, Algorithm 6 に示す．その具体的な方法について，以下で述べる．さらに，近似解の適用による MCS アルゴリズムの高速化についても述べる．この節の最後には，MCS の空間計算量と時間計算量についての解析を行う．

3.3.1 整列順序

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初期の頂点の整列順序は重要である [12].

探索は頂点の次数の小さいものから，彩色は次数が大きなものから行うのが効率の良いかとが実験的に示されている [6] [12]．また，探索木の根に近い部分では，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難しい．このような場合は，次数の小さい頂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．したがって，根の問題に限り最小次数を優先して探索することにより効率化を達成する．最小次数の頂点が複数ある場合は，その頂点集合を R_{min} とし，$ex\text{-}deg(p) := \sum_{r\in(R_{min}\cap\Gamma(p))} deg(r)$ の値が小さい順に探索を行う．以上のことに即した頂点の整列を探索の前処理として，初期頂点整列アルゴリズム EXTEND-INITIAL-SORT-NUMBER を行う．EXTEND-INITIAL-SORT-NUMBER の擬似コードを，Algorithm 3 に示す．

Algorithm 2 彩色アルゴリズム

procedure NUMBER-SORT(R, No)
    $maxno := 0, C_1 := \emptyset$
    while $R \neq \emptyset$ **do**
        $p :=$ the first vertex in R
        $k := 1$
        while $C_k \cap \Gamma(p) \neq \emptyset$ **do**
            $k := k + 1$
        end while
        if $k > maxno$ **then**
            $maxno := k, C_{maxno} := \emptyset$
        end if
        $\mathrm{No}[p] := k, C_k := C_k \cup \{p\}, R := R - \{p\}$
    end while
    $i := 1$
    for $k := 1$ **to** $maxno$ **do**
        for $j := 1$ **to** $|C_k|$ **do**
            $R[i] := C_k[j], i := i + 1$
        end for
    end for
end procedure

Algorithm 3 初期頂点整列アルゴリズム

procedure EXTEND-INITIAL-SORT-NUMBER(V, Q_{max}, No)
    $i := |V|$ ▷ 後ろから追加していく
    $R := V, V := \emptyset$
    R_{min} := set of vertices with the minimum degree in R
    while $|R_{min}| \neq |R|$ **do**
        if $|R_{min}| \geq 2$ **then**
            p :=a vertex in R_{min} such that $ex\text{-}deg(p) = \min\{ex\text{-}deg(q) \mid q \in R_{min}\}$
        else $p := R_{min}[1]$
        end if
        $V[i] := p, R := R - \{p\}, i := i - 1$
        for $j := 1$ **to** $|R|$ **do** ▷ 次数を直す
            if $R[j]$ is adjacent to p **then**
                $deg(R[j]) := deg(R[j]) - 1$
            end if
        end for
        R_{min} :=set of vertices with the minimum degree in R
    end while
    NUMBER-SORT (R_{min}, No)
    for $i := 1$ **to** $|R_{min}|$ **do** ▷ 残りの要素を追加していく
        $V[i] := R_{min}[i]$
    end for
    $m := \max\{\mathrm{No}[q] \mid q \in R_{min}\}$
    if $m = R_{min}$ **then**
        $Q_{max} := R_{min}$
    end if
    $m := m + 1, i := |R_{min}| + 1$ ▷ $|R_{min}| + 1 \leq i \leq |V|$ の範囲を彩色する
    while $i \leq |V|$ **and** $m \leq |V|$ **do**
        $\mathrm{No}[V[i]] := m, m := m + 1, i := i + 1$
    end while
    while $i \leq |V|$ **do**
        $\mathrm{No}[V[i]] := m, i := i + 1$
    end while
end procedure

3.3.2　再彩色アルゴリズム RE-COLOR

$\mathrm{No}[p] \leq |Q_{max}| - |Q| (=: cutc)$ となる頂点は探索不要であるので，この条件を満たす頂点を多くするような頂点の彩色を行いたい．このような考えに基づき提唱されたのが，再彩色アルゴリズム RE-COLOR である．頂点 p の彩色番号が k として，RE-COLOR は以下の手順で行う．

1. $1 \leq k_1 < cutc$ の範囲で，$|\Gamma(p) \cap C_{k_1}| = 1$ を満たす k_1 を探す．
2. $\Gamma(p) \cap C_{k_1}$ の頂点を q とする．
3. $k_1 < k_2 < cutc$ の範囲で，$|\Gamma(q) \cap C_{k_2}| = 0$ を満たす k_2 を探す．
4. 頂点 p の彩色番号を k_1，q の彩色番号を k_2 に変える．

RE-COLOR の動作例を図 3.1 に示す．RE-COLOR の擬似コードを，Algorithm 4 に示す．

3.3.3　MCS の彩色アルゴリズム

MCS の彩色アルゴリズムは，RE-COLOR を NUMBER-SORT に組み込んだアルゴリズムである．この彩色アルゴリズムを NUMBER-SORT-R とする．NUMBER-SORT-R の擬似コードを，Algorithm 5 に示す．彩色の順番には，探索の前処理として行われた EXTEND-INITIAL-SORT-NUMBER で並べられた初期の頂点の整列順序を用いる．NUMBER-SORT-R では，頂点の整列順序に従い先頭から彩色を行う．その頂点の彩色番号を k とする．RE-COLOR に時間をかけ過ぎないように，

$$k > cutc \text{ かつ } k = \text{（現在の最大彩色番号）}$$

を満たすときのみ，RE-COLOR を適用する．

3.3.4　近似解の適用

最大クリークの下界が既知であるのならば，従来の分枝限定法や RE-COLOR の条件により探索の効率化が期待できる．よって探索前に近似解を求めておくことで，その値を下

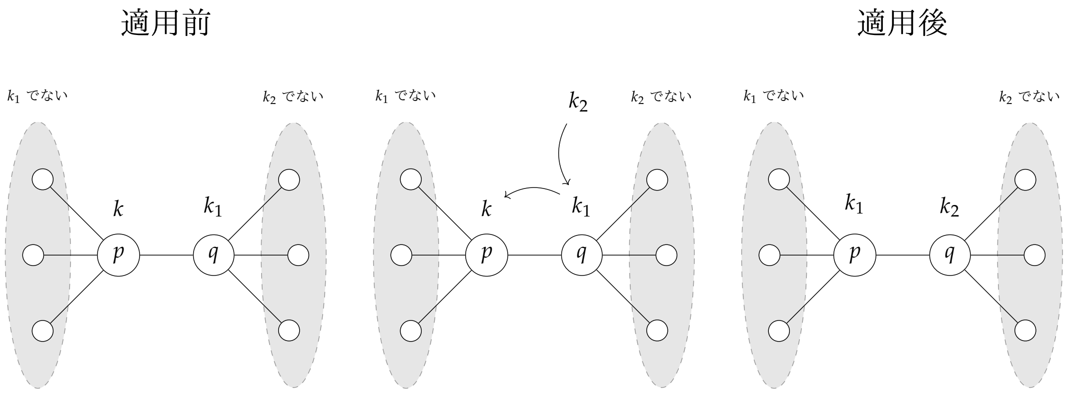

図 3.1: 再彩色アルゴリズムの動作

Algorithm 4 再彩色アルゴリズム

---

procedure RE-COLOR($p, k, cutc$)
    for $k_1 := 1$ **to** $cutc - 1$ **do**
        if $|C_{k_1} \cap \Gamma(p)| = 1$ **then**
            for $k_2 := k_1$ **to** $cutc$ **do**
                if $|C_{k_2} \cap \Gamma(p)| = 0$ **then**
                    $C_k := C_k - \{p\}$
                    $C_{k_1} := (C_{k_1} - \{q\}) \cup \{p\}$
                    $C_{k_2} := C_{k_2} \cup \{q\}$
                    return
                end if
            end for
        end if
    end for
end procedure

Algorithm 5 MCS の彩色アルゴリズム

procedure NUMBER-SORT-R ($V_s, R, \mathrm{No}, cutc$)
  $|R| := |V_s|$, $k_{max} := 1$, $C_1 := \emptyset$
  for $i := 1$ **to** $|V_s|$ **do**
    $p := Vs[i]$, $k := 1$
    while $(C_k \cap \Gamma(p)) \neq \emptyset$ **do**
      $k := k + 1$
    end while
    if $k > k_{max}$ **then**
      $k_{max} := k$, $C_{k_{max}} := \emptyset$
    end if
    $C_k := C_k \cup \{p\}$
    if $(k > cutc)$ **and** $(k = k_{max})$ **then**
      RE-COLOR(p, k)
      if $C_{k_{max}} = \emptyset$ **then**
        $k_{max} := k_{max} - 1$
      end if
    end if
  end for
  $i := |V_s|$
  if $cutc < 0$ **then**
    $cutc := 0$
  end if
  for $k := k_{max}$ **downto** $cutc + 1$ **do**
    for $j := |C_k|$ **downto** 1 **do**
      $R[i] := C_k[j]$
      $\mathrm{No}[R[i]] := k$
      $i := i - 1$
    end for
  end for
  if $i \neq 0$ **then**
    $\mathrm{No}[R[i]] := cutc$
  end if
end procedure

界として探索を効率化する．本論文の近似アルゴリズムには，1.2 節で紹介した KLS を利用する．KLS アルゴリズムとは，局所探索により最大クリークを近似するアルゴリズムである．各探索に対して 1 つの頂点を起点として探索を行う．多くの頂点を起点として，探索する回数を増やすことで近似精度の向上を期待できるが，アルゴリズムの実行時間が増加することはできるだけ避ける必要がある．そのため，探索回数を $20\sqrt{|V|} \times Dens(G)^3$ とする．これは，辺密度が高いグラフでは探索回数が増え，辺密度が低いグラフでは探索する頂点は少なくなる．

3.3.5 MCS アルゴリズムの空間計算量

MCS は，深さ優先探索を行い，図 3.2 のように動作する．したがって，MCS の空間計算量は，以下のことがいえる．

補題 2 (MCS アルゴリズムの探索中の空間計算量)**.** MCS アルゴリズム探索中に使用する空間計算量は，$O(n\omega)$ である．

証明. 探索の各段階で，子が探索を行っているときは，親は探索を終了できないので，候補点集合 R やその彩色番号の情報を持っている．したがって，探索木の根から葉まで合計メモリ使用量を考える．探索木の子の頂点に行くときにクリーク Q に頂点を 1 つ追加しているので，探索木の最大の深さは ω である．

$$|R| \leq |V| = n$$

したがって，探索中の空間計算量は $O(n\omega)$ である． □

補題 3 (MCS アルゴリズムの隣接行列での空間計算量)**.** MCS アルゴリズムの隣接行列での空間計算量は，$O(n^2)$ である．

証明. MCS の探索中のに使用する空間計算量は，定理 2 から $O(n\omega)$ である．2.2.1 で述べたようにグラフの隣接行列の空間計算量は，$O(n^2)$ である．$\omega \leq m$ なので，グラフの隣接行列での MCS アルゴリズムの空間計算量は，

$$O(n\omega) + O(n^2) = O(n^2)$$

となる． □

Algorithm 6 MCS アルゴリズム

procedure MCS ($G = (V, E)$)
  $Q := \emptyset, Q_{max} := \emptyset$
  EXTEND-INITIAL-SORT-NUMBER(V, Q_{max}, No)
  Apply approximation algorithm
  while $|Q| + |V| > |Q_{max}|$ **do**
    $p :=$ the last vertex in V
    if $|Q| + \mathrm{No}[p] > |Q_{max}|$ **then**
      $V_s := V \cap \Gamma(p)$
      EXPAND(V_s)
    else return Q_{max}
    end if
  end while
  return Q_{max}
end procedure

procedure EXPAND (V_s)
  NUMBER-SORT-R (V_s, R, No, $|Q_{max}| - |Q|$)
  while $R \neq \emptyset$ **do**
    $p :=$ the last vertex in R
    if $|Q| + \mathrm{No}[p] > |Q_{max}|$ **then**
      $Q := Q \cup \{p\}$
      $V_p := V_s \cap \Gamma(p)$ ▷ 順序は保存する
      if $V_p \neq \emptyset$ **then**
        EXPAND (V_p)
      else if $|Q| > |Q_{max}|$ **then**
        $Q_{max} := Q$
      end if
    else return
    end if
    $Q := Q - \{p\}$
    $R := R - \{p\}$
    $V_s := V_s - \{p\}$ ▷ 順序は保存する
  end while
end procedure

補題 4 (MCS アルゴリズムの隣接リストでの空間計算量)**.** MCS アルゴリズムの隣接リストでの空間計算量は，$O(n\omega + m)$ である．

証明. MCS の探索中のに使用する空間計算量は，定理 2 から $O(n\omega)$ である．2.2.2 で述べたように隣接リストの空間計算量は，$O(n+m)$ である．$\omega \leq n$ なので，グラフの隣接リストでの MCS の空間計算量は，

$$O(n\omega) + O(n+m) = O(n\omega + n + m) = O(n\omega + m)$$

となる． □

3.3.6 MCS アルゴリズムの時間計算量

MCS の分枝数は指数的に増加する可能性がある．したがって，全体の時間計算量は，指数時間である．再帰処理の EXPAND にかかる時間計算量について解析をする．

補題 5 (NUMBER-SORT-R の時間計算量)**.** NUMBER-SORT-R の時間計算量は隣接行列では，$O(n^3)$ 時間である．隣接リストでは，$O(n^3 \log n)$ 時間である．

証明. Algorithm 5 の 3–18 行目以外は，$O(n)$ 時間で行える．3–18 行目の中で最も時間がかかるのは，13 行目の RE-COLOR アルゴリズムである．隣接行列の頂点の隣接関係を調べるのは，$O(1)$ 時間，隣接リストでは，$O(\log n)$ 時間で行える．RE-COLOR アルゴリズムの最悪な場合は，頂点 p, q に対して全ての頂点をみる可能性があるので，隣接行列では，$O(n^2)$ 時間で隣接リストでは，$O(n^2 \log n)$ 時間 である．これを，最大 n 回行う．NUMBER-SORT-R は隣接行列では $O(n^3)$ 時間で行えて，隣接リストでは $O(n^3 \log n)$ 時間で行うことができる．

補題 6 (EXPAND の時間計算量)**.** EXPAND の時間計算量は隣接行列では，$O(n^3)$ 時間である. 隣接リストでは，$O(n^3 \log n)$ 時間である．

証明. Algorithm 6 の EXPAND の 4–17 行目までの処理で再帰の EXPAND 以外は, $O(n \log n)$ 時間 で行える．EXPAND アルゴリズムで最も時間がかかるステップは，NUMBER-SORT-R である．これを，最大で n 回繰り返される．したがって，EXPAND アルゴリズムの時間計算量は隣接行列では，$O(n^3)$ 時間である．隣接リストでは，$O(n^3 \log n)$ 時間である．

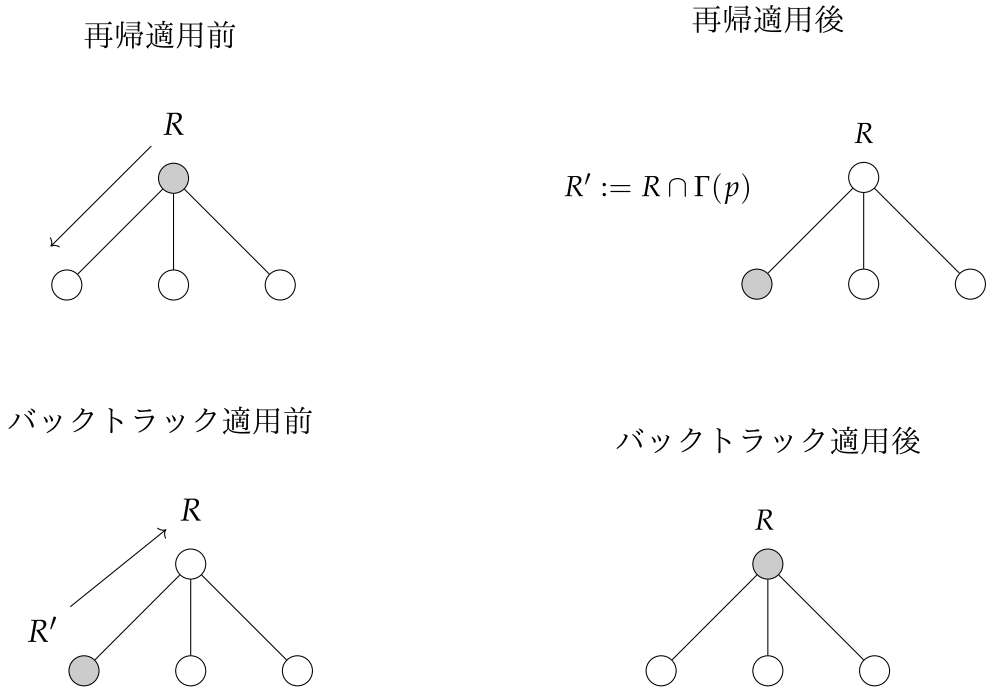

図 3.2: MCS の動作

R, R' は，候補点集合とし，p は探索を行う頂点とする．

第 4 章

提案手法

提案する省メモリ化最大クリーク抽出アルゴリズムの概要について 4.1 節で述べる．次にその空間計算量について 4.2 節で述べる．最後にその時間計算量について 4.3 節で述べる．

4.1　アルゴリズムの概要

MCS アルゴリズムでは，探索中のメモリ使用量のボトルネックは，探索の各段階で持つ候補点集合の情報のサイズである．したがって，たとえグラフを隣接リストなどで実装をしても，グラフとクリークのサイズが大きいとき，メモリ使用量が大きくなりやすい．

提案するアルゴリズムでは，探索の各段階で持つ情報を，親から子に進むときに捨てる頂点集合のみにする．このことで，メモリの使用量を削減するようにする．保持する情報には，頂点が選ばれクリークに追加された頂点集合と頂点を追加するときに隣接しなくなった頂点集合の 2 種類ある．それぞれの頂点集合を V_{d}，V_{e} とする．バックトラックするとき，その頂点集合を使用することで復元をする．具体的な復元の仕方について，以下で説明する．V の順序は，手続き EXTEND-INITIAL-SORT-NUMBER を実行した以降変化しない．よって，適用後に頂点集合の先頭から 1 から頂点の番号を付け直して変える．このことによって，V_{d} や V_{e} を追加した後にソートすることで復元することができる．

たとえば，図 2.1 のグラフで，頂点 1 をクリークに追加したとする．頂点 1 の隣接集合は，$\Gamma(1) = (2, 4, 5)$ である．候補点集合は，V から頂点 1 に隣接していない頂点を削除する．したがって，$V' = V \cap \Gamma(1) = (2, 4, 5)$ になりこれを子に渡す．そして，$V_{\mathrm{d}} = V - V' = (1, 3, 6)$ を保存する．このようにすることで，再帰の各段階で子に進むときに

捨てる頂点集合だけを保持させて，再帰を進めることができる．バックトラックするときの候補点集合を，復元する方法について説明する．まず，子の頂点集合 V' に自分が保存している頂点集合 V_{d} を追加する．したがって，$V = V' \cup V_{\mathrm{d}} = (2,4,5,1,3,6)$ となる．次に，ソートすることで $V = (1,2,3,4,5,6)$ となり元の配列に復元することができる．提案アルゴリズムの動作を図 4.1 に示す．提案アルゴリズムの擬似コードは，Algorithm 7 に示す．

4.2　アルゴリズムの空間計算量

提案アルゴリズムの空間計算量に対して，以下のことがいえる．

定理 7 (提案アルゴリズムの探索中の空間計算量)**.** 提案アルゴリズム探索中に使用する空間計算量は，$O(n)$ である．

証明. EXPAND-PROPOSAL では，保持する主なデータは，V_{e}，V_{d} である．これは子との差分の情報なので，探索木の根からどの探索の段階までの合計もグラフの頂点集合のサイズを超えない．したがって，探索中に使用するの空間計算量は，$O(n)$ となる．　□

定理 8 (提案アルゴリズムの隣接行列の空間計算量)**.** 提案アルゴリズムの隣接行列の空間計算量は，$O(n^2)$ である．

証明. 提案アルゴリズムの探索中に使用する空間計算量は，定理 7 から $O(n)$ である．2.2.1 で述べたようにグラフの隣接行列の空間計算量は，$O(n^2)$ である．グラフの隣接行列での提案アルゴリズムの空間計算量は，

$$O(n) + O(n^2) = O(n^2)$$

となる．　□

定理 9 (提案アルゴリズムの隣接リストの空間計算量)**.** 提案アルゴリズムの隣接リストでの空間計算量は，$O(n+m)$ である．

証明. 提案アルゴリズムの探索中に使用する空間計算量は，定理 7 から $O(n)$ である．2.2.2 で述べたようにグラフの隣接リストの空間計算量は，$O(n+m)$ である．グラフの隣接リ

Algorithm 7 提案アルゴリズム

procedure MCS-PROPOSAL ($G = (V, E)$)
    $Q := \emptyset, Q_{max} := \emptyset$
    EXTEND-INITIAL-SORT-NUMBER(V, Q_{max}, No)
    Apply approximation algorithm
    while $|Q| + |V| > |Q_{max}|$ **do**
        $p :=$ the last vertex in V
        if $|Q| + \mathrm{No}[p] > |Q_{max}|$ **then**
            EXPAND-PROPOSAL($V \cap \Gamma(p)$)
        else return Q_{max}
        end if
    end while
    return Q_{max}
end procedure

procedure EXPAND-PROPOSAL (V_s)
    $V_{\mathrm{e}} := \emptyset$ ▷ 一度クリークに追加された点を保存する
    while $V_s \neq \emptyset$ **do**
        $p, no :=$the last vertex and number in R and No that get NUMBER-SORT-R (V_s, R, No)
        if $|Q| + no > |Q_{max}|$ **then**
            $Q := Q \cup \{p\}$
            $V_{\mathrm{d}} := V_s - \Gamma(p)$ ▷ 隣接していないために削除された点を保存する
            $V_s := V_s \cap \Gamma(p)$
            if $V_s \neq \emptyset$ **then**
                EXPAND-PROPOSAL (V_s)
            else if $Q > Q_{max}$ **then**
                $Q_{max} := Q$
            end if
            $V_s := V_s \cup V_{\mathrm{d}}$，sort to V_s ▷ 隣接していないため削除した頂点を復元する
            $V_s := V_s - \{p\}, V_{\mathrm{e}} := V_{\mathrm{e}} \cup \{p\}$
        else
            $V_s := V_s \cup V_{\mathrm{e}}$，sort to V_s ▷ バックトラックするので，探索した頂点を復元する
            return
        end if
    end while
    $V_s := V_s \cup V_{\mathrm{e}}$，sort to V_s ▷ バックトラックするので，探索した頂点を復元する
end procedure

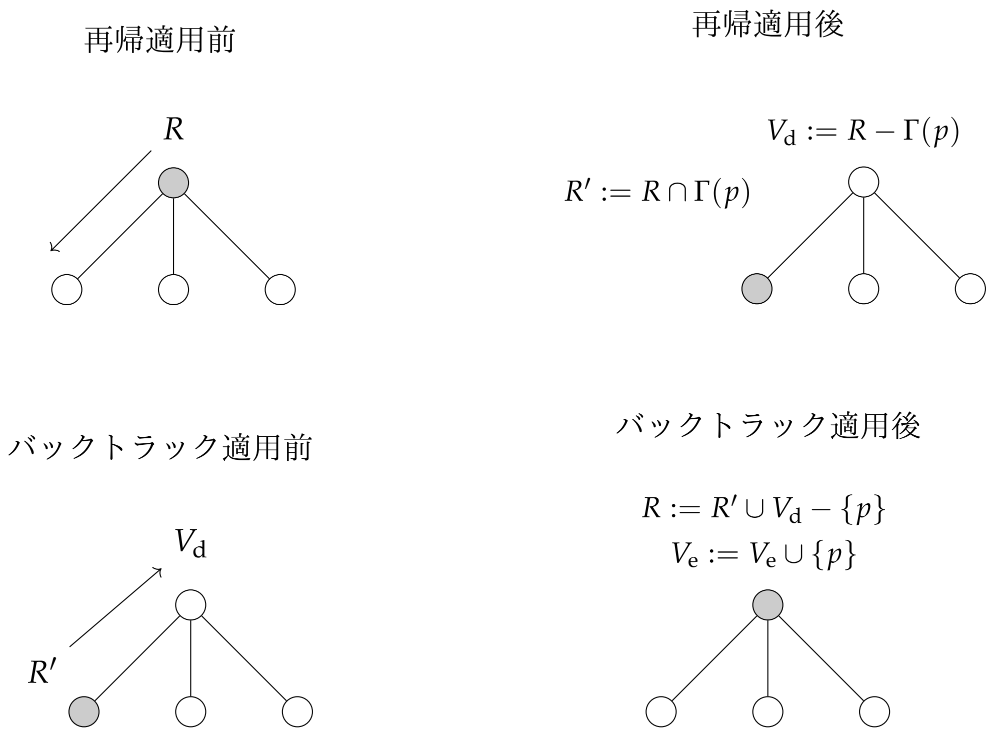

図 4.1: 提案手法の動作

R, R' は，候補点集合とし，p は探索を行う頂点とする．V_{d} は，各再帰で隣接していないので，子に持たせない頂点集合とする．V_{e} は，各再帰の探索済みの頂点の集合とする．

ストでの提案アルゴリズムの空間計算量は，

$$O(n) + O(n + m) = O(n + n + m) = O(n + m)$$

となる． □

4.3 アルゴリズムの時間計算量

提案アルゴリズムの空間計算量に対して，以下のことがいえる．提案アルゴリズムの分枝数は，指数的に増加する可能性がある．したがって，全体の時間計算量は，指数時間である．提案アルゴリズムの分枝の処理 EXPAND-PROPOSAL の時間計算量についても解析をする．

定理 10 (EXPAND-PROPOSAL の時間計算量)**.** EXPAND-PROPOSAL の時間計算量は隣接行列では，$O(n^4)$ 時間で，隣接リストでは，$O(n^4 \log n)$ 時間である．

証明. Algorithm 7 の EXPAND-PROPOSAL の 5–19 行の EXPAND-PROPOSAL を以外の処理と 3–20 行以外の処理は $O(n \log n)$ で全て行うことができる．3–20 行のループで最も時間がかかる処理は，4 行目の処理である．ループの回数は，最大で n 回行う．したがって，EXPAND-PROPOSAL の時間計算量は隣接行列では，$O(n^4)$ 時間で，隣接リストでは，$O(n^4 \log n)$ である．

第 5 章

実験

提案アルゴリズムがMCSに対して，最大メモリ使用量と実行時間がどれだけ増減するのかを確認するために実験を行う．さらに，再帰処理の回数の分枝数や彩色アルゴリズムNUMBER-SORT-Rの実行回数が，実行時間に大きく関わっていると考えられるので，それらの回数も調べる．また，以上の結果に対して考察を行う．

5.1　実験方法

実験方法としては，MCSと提案アルゴリズムを以下の条件に分けて実験を行った．グラフのデータ構造については，隣接行列と隣接リストの2通りについて用意した．辺密度が大きなグラフを多く扱ったので，補グラフでグラフを持つようにした．近似解のアルゴリズムについては，適用をする場合と適用しない場合の2通りについて用意した．計4通りの条件を，MCSと提案アルゴリズムのそれぞれに対して行った．近似解は，3.3.4で述べたKLSアルゴリズムによって得られた解のことする．実験に使用したグラフは，DIMACSベンチマークセットのグラフ[1]とクリークサイズの大きなグラフから辺をいくつか削除することで作成したグラフ，　DIMACSのグラフのいくつかのグラフを合体させて，そのグラフ間に辺をランダムに引くことによって作成したグラフの3種類を用意した．クリークサイズの大きなグラフを作るため，完全グラフから辺をいくつか削除したグラフは2000-1500のように「*a*-*b*」のような名前で表す．「*a*-*b*」は，グラフサイズ*a*の完全グラフから*b*本の辺を削除したグラフである．DIMACSを合体させたグラフは，unionとunion2の2つである．unionは，c-fat200-1とMANNa-45のグラフを組み合わせ

[1] `http://iridia.ulb.ac.be/~fmascia/maximum_clique/DIMACS-benchmark`

たものである．union2 は，c-fat200-1 と，MANNa-45，hamming10-2 を組み合わせたものである．

実験環境は，Intel Core i5-7360U CPU (クロック周波数：2.30GHz，L2 Cached：256 KB，L3 Cached：4 MB)，主メモリ 8 GB の MacOSver.10.14.6 の PC 上で，全てのプログラムは，gcc 8.3.0 -O3 -lm でコンパイルを行った．

5.2 結果と考察

MCS 手法において，近似なし隣接行列を MCS_{0m}，近似なし隣接リストを MCS_{0l}，近似あり隣接行列を MCS_{m}，近似あり隣接リストを MCS_{l} とする．また，提案手法において，近似なし隣接行列を MCSP_{0m}，近似なし隣接リストを MCSP_{0l}，近似あり隣接行列を MCSP_{m}，近似あり隣接リストを MCSP_{l} とする．

実行時間と，最大メモリ使用量，分枝数と NUMBER-SORT-R の実行回数の結果を表 5.1，表 5.2，表 5.3 に示す．fail は，実行に失敗したことを示す．各グラフに対して，時間が最も速いものと最大メモリ使用量が最も小さいものは，太字で示している．NSR 数は，NUMBER-SORT-R の実行回数のことを示している．以下では，結果の比較と考察を行う．

5.2.1 MCS と提案手法の比較

提案アルゴリズムは，MCS に対して実行時間では，2 から 4 倍程度悪化するものが多かった．分枝数は，MCS と提案手法で大きな差がなかった．これは，同じ彩色アルゴリズムを適用しているため，ほぼ同じ探索をしたからだと考えられる．分枝数の僅かな差は，NUMBER-SORT-R を繰り返し適用しているので，RE-COLOR が適用される頂点が増えるために僅かに分枝数が減少したと考察する．NUMBER-SORT-R の実行回数は，MCS の彩色は，再帰の前処理としてのみしか行われないため，MCS では分枝数以下の回数しか行われなかった．また，提案アルゴリズムでは分枝数の 2 倍以下の回数しか行われなかった．これは，提案アルゴリズムの彩色は，子の再帰の前処理と再帰の終了の条件のためにしか行われないため，全体として分枝数の 2 倍以下しか行われなかったと考察をする．MCS での時間計算量が隣接行列で $O(n^3)$ 時間，隣接リストで $O(n^3 \log n)$ 時間で，

提案アルゴリズムでの時間計算量が隣接行列で $O(n^4)$ 時間，隣接リストで $O(n^4 \log n)$ 時間である．提案アルゴリズムの時間計算量が MCS に対して大きく増加しているのに，実行時間が 2 から 4 倍程度に抑えられているのは，分枝数に大きな増加がないことや提案アルゴリズムの彩色の適用回数が分枝数の 2 倍以下であることによるためだと考察する．

解のサイズが大きなグラフに対しては，最大メモリ使用量の減少が確認できた．しかし，解のサイズが小さなものにはほとんど効果が見受けられなかった．これは，MCS の探索木の深さが大きくないとき MCS のメモリ使用量も大きくならず，探索木の深さの最大値は，最大クリークのサイズに等しいからだであると考察する．

fail で表したように実行ができなかったものは，完全グラフから辺をいくつか削除して作られたグラフである．これはクリークサイズが大きいので，近似解を用いない方法では，提案アルゴリズムでなければメモリ使用量が大きくなるため，スタック領域が足りなくなり動かなかったと考える．

5.2.2　隣接行列と隣接リストの比較

実行時間は，隣接リストの方が，隣接行列に比べて 3 から 10 倍程度悪くなった．隣接リストが，隣接行列よりある頂点 p の隣接関係を調べるのに，$O(\log |\Gamma(p)|)$ 倍かかるために実行時間が悪化したと考察する．

最大メモリ使用量は，隣接リストは隣接行列に比べて，辺密度がある程度大きいものはほとんどが最大メモリ使用量が小さくなったが，辺密度が小さいものは，メモリ使用量が増加した．隣接リストを補グラフで保持しているので，辺密度が小さいとき保持するデータが多くなるため，最大メモリ使用量が隣接行列で保持するより大きくなったと考察する．

5.2.3　近似アルゴリズムの適用する場合としない場合の比較

近似アルゴリズムの適用については，一部のグラフに対して分枝数と NUMBER-SORT-R の実行回数の減少し，実行時間の減少が見受けられた．

最大メモリ使用量については，近似解がクリークサイズと一致しているような場合については，メモリ使用量が減少しない傾向があるが，一致していない場合はメモリ使用

量が増加し提案アルゴリズムで大きくメモリ使用量が減少することに成功した．これは，探索において近似解と解が一致しているときには，探索が深くなるとは限らないためにメモリ使用量が減少したと考えられる．

表 5.1: 実行時間 (sec)

入力グラフデータ					従来手法 MCS				提案手法			
					近似なし		近似あり		近似なし		近似あり	
名前	サイズ	辺密度	解	近似解	MCS_{0m}	MCS_{0l}	MCS_{m}	MCS_{l}	MCSP_{0m}	MCSP_{0l}	MCSP_{m}	MCSP_{l}
brock200-1	200	0.74	21	21	0.37	1.52	**0.36**	1.48	1.21	4.03	1.17	3.83
brock200-2	200	0.49	12	11	**0.00**	0.02	0.01	0.03	0.02	0.06	0.02	0.06
brock200-3	200	0.60	15	14	**0.02**	0.07	0.03	0.11	0.07	0.22	0.08	0.26
brock200-4	200	0.65	17	16	**0.07**	0.22	0.07	0.24	0.23	0.63	0.22	0.63
brock400-1	400	0.74	27	25	291.75	1445.32	**267.88**	1311.92	965.72	3847.53	909.17	3492.37
brock400-2	400	0.74	29	24	123.21	611.98	**117.5**	570.68	405.61	1642.91	398.65	1531.95
brock400-3	400	0.74	31	24	**196.84**	959.33	200.2	948.25	644.5	2524.13	672.82	2495.47
brock400-4	400	0.74	33	25	**104.71**	511.61	106.18	499.04	343.96	1372.12	356.05	1339.33
c-fat200-1	200	0.07	12	12	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
c-fat200-2	200	0.16	24	24	**0.00**	**0.00**	**0.00**	0.01	**0.00**	**0.00**	**0.00**	**0.00**
c-fat200-5	200	0.42	58	58	**0.00**	**0.00**	**0.00**	0.02	**0.00**	**0.00**	**0.00**	0.02
c-fat500-1	400	0.03	14	14	**0.00**	0.03	**0.00**	0.03	**0.00**	0.03	0.01	0.03
c-fat500-10	400	0.37	126	126	**0.01**	0.10	0.02	0.25	**0.01**	0.11	0.02	0.24
hamming8-2	256	0.96	128	128	**0.00**	**0.00**	0.16	0.53	0.01	0.01	0.16	0.54
hamming8-4	256	0.63	16	16	**0.13**	0.52	0.14	0.58	0.46	1.38	0.40	1.46
hamming10-2	1024	0.99	512	512	**0.04**	0.16	6.44	26.8	0.15	0.61	6.62	27.21
p-hat500-1	500	0.25	9	8	0.02	0.09	**0.01**	0.11	0.06	0.29	0.07	0.31
p-hat500-2	500	0.50	36	36	0.33	1.98	**0.06**	0.93	1.13	5.49	0.41	1.98
p-hat500-3	500	0.75	50	50	57.40	388.15	**34.40**	228.62	184.76	973.49	115.89	576.51
p-hat700-1	700	0.24	11	9	**0.05**	0.28	0.06	0.30	0.23	1.14	0.23	1.18
p-hat700-2	700	0.49	44	44	2.31	16.57	**1.10**	7.74	8.00	45.77	3.66	20.13
p-hat700-3	700	0.74	62	62	921.42	7083.65	**485.01**	3650.09	2975.85	18243.12	1642.01	9433.96
p-hat1000-1	1000	0.24	10	10	**0.26**	1.46	**0.26**	1.49	1.25	6.84	1.27	6.83
p-hat1000-2	1000	0.49	46	46	90.94	729.06	**51.91**	415.78	307.35	1925.03	182.23	1095.12
p-hat1500-1	1500	0.25	12	11	2.08	10.62	**2.05**	10.44	11.03	53.38	11.04	52.89
MANN-a27	378	0.99	126	126	**0.22**	0.84	0.78	3.33	0.48	1.52	1.09	3.99
MANN-a45	1035	0.99	345	344	**38.93**	161.77	53.07	242.98	69.61	279.24	94.59	347.05
1500-1500	1500	0.99	915	915	99.75	324.54	**91.72**	328.62	126.13	425.53	104.64	341.8
1500-1550	1500	0.99	903	903	291.98	960.68	**156.01**	551.39	375.67	1302.07	198.54	608.78
2000-1500	2000	0.99	1330	1330	**320.64**	934.38	359.64	1110.25	387.92	1175.8	399.91	1135.63
2500-2000	2500	0.99	1629	1629	fail	fail	612.1	1811.04	**313.15**	926.61	671.33	1904.16
3000-1500	3000	0.99	2195	2195	fail	fail	994.29	2698.78	**15.64**	43.05	1016.05	2700.79
3000-2000	3000	0.99	2050	2050	fail	fail	1772.47	4786.92	**1320.74**	3569.42	2082.42	5114.48
union	1235	0.84	347	345	**38.19**	388.02	50.50	519.86	68.81	613.32	91.30	759.34
union2	2259	0.70	522	355	**6.70**	125.74	23.19	402.88	9.66	171.57	27.91	463.39

太字は時間最小を，fail は実行が失敗したことを表す．

表 5.2: 最大メモリ使用量 (kbytes)

入力グラフデータ					従来手法 MCS				提案手法			
					近似なし		近似あり		近似なし		近似あり	
名前	サイズ	辺密度	解	近似解	MCS$_{0m}$	MCS$_{0l}$	MCS$_m$	MCS$_l$	MCSP$_{0m}$	MCSP$_{0l}$	MCSP$_m$	MCSP$_l$
brock200-1	200	0.74	21	21	868	**848**	868	876	896	884	900	884
brock200-2	200	0.49	12	11	872	**868**	872	880	876	888	896	908
brock200-3	200	0.60	15	14	868	**860**	**860**	888	876	880	904	908
brock200-4	200	0.65	17	16	872	**856**	872	880	896	868	900	872
brock400-1	400	0.74	27	25	1004	948	996	**936**	1024	948	1024	952
brock400-2	400	0.74	29	24	996	**924**	1004	956	1020	948	1024	948
brock400-3	400	0.74	31	24	996	**936**	1000	960	1020	948	1024	960
brock400-4	400	0.74	33	25	1004	**936**	1004	**936**	1020	944	1028	948
c-fat200-1	200	0.07	12	12	**872**	928	876	936	880	904	860	928
c-fat200-2	200	0.16	24	24	**868**	908	880	916	884	908	888	888
c-fat200-5	200	0.42	58	58	876	892	880	892	872	888	**860**	896
c-fat500-1	400	0.03	14	14	**1068**	1344	1072	1324	1080	1344	1088	1348
c-fat500-10	400	0.37	126	126	**1080**	1168	1100	1188	**1080**	1160	1088	1168
hamming8-2	256	0.96	128	128	960	888	896	844	896	844	916	**832**
hamming8-4	256	0.63	16	16	912	896	896	896	912	**892**	920	912
hamming10-2	1024	0.99	512	512	2956	1976	1896	**928**	1924	968	1900	**928**
p-hat500-1	500	0.25	9	8	**1068**	1240	1112	1252	1108	1220	1116	1252
p-hat500-2	500	0.50	36	36	**1072**	1108	1100	1120	1108	1120	1104	1120
p-hat500-3	500	0.75	50	50	1100	1004	1088	1004	1108	**992**	1124	996
p-hat700-1	700	0.24	11	9	**1320**	1788	1336	1804	1352	1780	1360	1800
p-hat700-2	700	0.49	44	44	**1320**	1416	1336	1408	1348	1392	1360	1416
p-hat700-3	700	0.74	62	62	1324	1132	1340	1132	1352	**1120**	1368	1136
p-hat1000-1	1000	0.24	10	10	**1828**	2620	1848	2660	1856	2656	1876	2664
p-hat1000-2	1000	0.49	46	46	**1844**	2056	1860	2064	1856	2044	1888	2068
p-hat1500-1	1500	0.25	12	11	**3080**	4560	3080	4576	3096	4584	3116	4600
MANN-a27	378	0.99	126	126	1080	948	1000	**856**	1012	888	1004	876
MANN-a45	1035	0.99	345	344	2700	1688	2624	1628	1960	**932**	1956	936
1500-1500	1500	0.99	915	915	6564	4412	3236	1096	3196	1032	3100	**956**
1500-1550	1500	0.99	903	903	6468	4340	3292	1140	3188	1024	3120	**960**
2000-1500	2000	0.99	1330	1330	11704	7872	5076	1212	4972	1104	4860	**996**
2500-2000	2500	0.99	1629	1629	fail	fail	7284	1232	7232	1180	7076	**1044**
3000-1500	3000	0.99	2195	2195	fail	fail	9848	1140	9976	1276	9776	**1036**
3000-2000	3000	0.99	2050	2050	fail	fail	10048	1352	9960	1256	9780	**1076**
union	1235	0.84	347	345	3200	2276	3192	2272	2408	**1432**	2412	1440
union2	2259	0.70	522	355	6872	5508	6880	5536	5956	**4580**	5960	4584

太字は最大メモリ使用量最小を，fail は実行が失敗したことを表す．

表 5.3: 分枝数と NUMBER-SORT-R の実行回数

入力グラフデータ					従来手法 MCS				提案手法			
					近似なし		近似あり		近似なし		近似あり	
名前	サイズ	辺密度	解	近似解	分枝数	NSR 数	分枝数	NSR 数	分枝数	NSR 数	分枝数	NSR 数
brock200-1	200	0.74	21	21	151757	151752	134082	134080	147459	252543	130159	223336
brock200-2	200	0.49	12	11	2466	2461	1852	1848	2428	4247	1819	3201
brock200-3	200	0.6	15	14	8142	8138	8148	8144	7892	13787	7897	13797
brock200-4	200	0.65	17	16	30753	30747	25620	25617	29969	51872	24893	43350
brock400-1	400	0.74	27	25	89388740	89388732	76595610	76595606	86151316	149481838	73766643	128456968
brock400-2	400	0.74	29	24	33513181	33513175	29357758	29357754	32204691	56077884	28183623	49326562
brock400-3	400	0.74	31	24	65302414	65302404	64264259	64264254	63001066	108590631	61996912	106905890
brock400-4	400	0.74	33	25	30854882	30854873	29641679	29641674	29734877	51696589	28559627	49706160
c-fat200-1	200	0.07	12	12	188	3	188	3	188	3	188	3
c-fat200-2	200	0.16	24	24	176	9	176	9	176	9	176	9
c-fat200-5	200	0.42	58	58	142	26	142	26	142	26	142	26
c-fat500-1	400	0.03	14	14	486	4	486	4	486	4	486	4
c-fat500-10	400	0.37	126	126	374	60	374	60	374	60	374	60
hamming8-2	256	0.96	128	128	128	127	0	0	128	127	0	0
hamming8-4	256	0.63	16	16	31794	31793	31782	31782	27675	49129	27664	49120
hamming10-2	1024	0.99	512	512	512	511	0	0	512	511	0	0
p-hat500-1	500	0.25	9	8	7987	7985	7987	7985	7816	14310	7816	14310
p-hat500-2	500	0.5	36	36	63511	63505	14918	14917	60273	107348	14351	25226
p-hat500-3	500	0.75	50	50	7923418	7923412	4323605	4323604	7527743	13214484	4105879	7249093
p-hat700-1	700	0.24	11	9	22488	22481	22350	22344	22334	41274	22203	41053
p-hat700-2	700	0.49	44	44	339351	339346	125955	125954	321558	573257	119326	213702
p-hat700-3	700	0.74	62	62	88168353	88168349	42753184	42753183	83274421	148567028	40405710	72199507
p-hat1000-1	1000	0.24	10	10	120343	120340	116013	116012	117959	211580	113712	204473
p-hat1000-2	1000	0.49	46	46	12617898	12617893	6646177	6646176	11943160	21326187	6292680	11265227
p-hat1500-1	1500	0.25	12	11	820412	820405	772252	772247	784174	1462328	738262	1376371
MANN-a27	378	0.99	126	126	9118	9116	8843	8843	9118	13481	8843	13178
MANN-a45	1035	0.99	345	344	224555	224551	223056	223055	224555	338520	223056	336749
1500-1500	1500	0.99	915	915	57388	57380	17860	17859	54627	70058	17242	23148
1500-1550	1500	0.99	903	903	191775	191767	64726	64725	183320	244611	61740	84183
2000-1500	2000	0.99	1330	1330	84633	84624	42262	42261	84633	101732	42262	51408
2500-2000	2500	0.99	1629	1629	fail	fail	40549	40548	62913	78754	40549	53132
3000-1500	3000	0.99	2195	2195	fail	fail	964	963	5375	5380	964	967
3000-2000	3000	0.99	2050	2050	fail	fail	103414	103413	128922	165510	103414	136903
union	1235	0.84	347	345	241503	241498	240669	240666	241503	364850	240669	363923
union2	2259	0.7	522	355	20529	20523	29126	29119	20400	26489	28873	38304

NSR 数は NUMBER-SORT-R の実行回数を，fail は実行が失敗したことを表す．

第6章

おわりに

6.1 まとめ

隣接行列での空間計算量は，MCS も提案アルゴリズムも $O(n^2)$ である．隣接リストでは，MCS での空間計算量は $O(n\omega + m)$ であるが，提案アルゴリズムの空間計算量が $O(n + m)$ で抑えられることを示した．これは，最大クリークのサイズが大きな場合でも，最大メモリの使用量が大きくならずに抑えれていることを表す．

提案アルゴリズムでは，ほとんどのグラフに対して実行時間が増加した．解が大きな場合について，最大メモリ使用量が小さくなった．近似解を利用した場合で近似解と解が一致した場合は，探索木の深さが，クリークのサイズより小さくなる可能性があり．最大メモリ使用量が小さくなることがある．また，MCS ではメモリ使用量が大きくなり動かないが，提案アルゴリズムでは動くようなグラフがあることも確認できた．

6.2 今後の課題

提案アルゴリズムは，クリークサイズが小さいようなときはメモリ使用量の恩恵が少なく，実行時間が遅いだけである．そのため，探索の木に近い部分では提案手法を，根にある程度近づくと MCS の手法に切り替えることで，メモリ使用量を抑えつつ十分に効率的な時間で終了できるな工夫を考えることが今後の課題として考えられる．また，MCT などの他のより効率的なアルゴリズムに対しての提案手法を考え方を組み込むことによるメモリ使用量を改善することも今後の課題として考えられる．

謝辞

本論文の執筆にあたり，多くのご助力をいただきました．北海道大学大学院情報科学研究科情報知識ネットワーク研究室の有村博紀教授，喜田拓也准教授御両名には，本論文を執筆をするのにあたり数多くのご助力，ご協力をいただき深く感謝いたします．

加えて，本論文の題材となった貴重な MCQ，MCS，MCT アルゴリズムのソースコードの提供やご助力をしていただいき，電気通信大学先進アルゴリズム研究ステーションの富田悦次名誉教授へ，心より感謝しております．

情報知識ネットワーク研究室のみなさん，秘書の真鍋由布さんには，日頃より大変お世話になっています．研究生活を送るうえで，多くのサポートや刺激をもらいました．ありがとうございます．特に，博士 3 年の栗田和宏さん，修士 1 年の瀧澤涼介さんには，お忙しい中，論文執筆のご助力をいただき感謝しております．最後に，今まであらゆる面で支えてくれた祖父母，両親に深く感謝します．

参考文献

[1] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations*, The IBM Research Symposia Series, pp. 85–103. Plenum Press, New York, 1972.

[2] Vladimir Boginski, Sergiy Butenko, and Panos M. Pardalos. Mining market data: A network approach. *Computers & Operations Research*, Vol. 33, No. 11, pp. 3171 – 3184, 2006. Part Special Issue: Operations Research and Data Mining.

[3] Randy Carraghan and Panos M Pardalos. An exact algorithm for the maximum clique problem. *Operations Research Letters*, Vol. 9, No. 6, pp. 375–382, 1990.

[4] L. Babel and G. Tinhofer. A branch and bound algorithm for the maximum clique problem. *Zeitschrift für Operations Research*, Vol. 34, No. 3, pp. 207–217, May 1990.

[5] Etsuji Tomita and Tomokazu Seki. An efficient branch-and-bound algorithm for finding a maximum clique. In *International Conference on Discrete Mathematics and Theoretical Computer Science*, pp. 278–289. Springer, 2003.

[6] Etsuji Tomita and Toshikatsu Kameda. An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. *Journal of Global optimization*, Vol. 37, No. 1, pp. 95–111, 2007.

[7] Etsuji Tomita, Yoichi Sutani, Takanori Higashi, Shinya Takahashi, and Mitsuo Wakatsuki. A simple and faster branch-and-bound algorithm for finding a maximum clique. In *International Workshop on Algorithms and Computation*, pp. 191–203. Springer, 2010.

[8] Etsuji Tomita, Yoichi Sutani, Takanori Higashi, and Mitsuo Wakatsuki. A simple and faster branch-and-bound algorithm for finding a maximum clique with computational experiments. *IEICE TRANSACTIONS on Information and Systems*, Vol. 96, No. 6, pp. 1286–1298, 2013.

[9] Etsuji Tomita, Kohei Yoshida, Takuro Hatta, Atsuki Nagao, Hiro Ito, and Mitsuo Wakatsuki. A much faster branch-and-bound algorithm for finding a maximum clique. In *International Workshop on Frontiers in Algorithmics*, pp. 215–226. Springer, 2016.

[10] Andrea Grosso, Marco Locatelli, and Federico Della Croce. Combining swaps and node weights in an adaptive greedy approach for the maximum clique problem. *J. Heuristics*, Vol. 10, No. 2, pp. 135–152, 2004.

[11] Kengo Katayama, Akihiro Hamamoto, and Hiroyuki Narihisa. An effective local search for the maximum clique problem. *Information Processing Letters*, Vol. 95, No. 5, pp. 503–511, 2005.

[12] 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. 電子情報通信学会論文誌 D, Vol. 68, No. 3, pp. 221–228, 1985.