2005－MPS－57（12）
2005／12／20

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

須谷 洋一,　富田 悦次

電気通信大学 電気通信学部 情報通信工学科
〒 182-8585　東京都調布市調布ヶ丘 1-5-1
E-mail:　{sutani, tomita}@ice.uec.ac.jp

最大クリークを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. そこで, 筆者らは分枝限定法を用い効率良くグラフ中の最大クリークを抽出するアルゴリズムを提唱してきた. 本稿では, これらのアルゴリズムを基にして, より効率良いアルゴリズムを提唱し, この効果を実証した. また, 併せて効率化の根拠を内部動作の解析により検証した.

Computational Experiments and Analyses of a More Efficient Algorithm for Finding a Maximum Clique

Yoichi SUTANI and Etsuji TOMITA

Department of Information and Communication Engineering,
The University of Electro-Communications
Chofugaoka 1-5-1, Chofu, Tokyo 182-8585, Japan
E-mail:　{sutani, tomita}@ice.uec.ac.jp

Abstract. Many practical problems can be formulated as the maximum clique finding problem, and then more efficient algorithms are strongly required for this problem. We present here an improvement of our previous branch-and-bound algorithm for finding a maximum clique. The effect and the evidences of the improvement are shown by computational experiments for random graphs and DIMACS benchmark graphs.

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が, その重要性により多くの研究がなされている. その結果, 近年多くの応用の成果が得られている. これらの応用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な問題である. そこで本稿では, 筆者らが先に提唱したアルゴリズムを改良することによって, 広範囲の問題に対してより優位といえるアルゴリズムを提唱し, 計算機実験によってこの効果を実証した. また, 併せて効率化の根拠を内部動作の解析により検証した.

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. 節点 $v \in V$ に隣接する節点の集合を $\Gamma(v)$ で表し, その節点数を v の次数と呼ぶ. また, 集合 S に含まれる要素数は $|S|$ と表す. 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, また最も節点数の多い極大クリークであれば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. 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. ここで, 基本となる最大クリーク抽出アルゴリズム MCQ[3] とそのアルゴリズム内で用いられている近似彩色手続き, これを改良したアルゴリズム MCQ′[4] について述べ, 続いて今回提唱するアルゴリズムを示す.

3.1 MCQ

MCQ は分枝限定を効率的に行うために, 再帰手続きの前処理として各候補節点集合に対し, 逐次的近似彩色に相当する番号付けを行い, クリークの上界を求める. 彩色において, クリークの節点にはそれぞれ違

う色が割り当てられることから, 彩色数は最大クリークの節点数の上界となる. したがって, この近似彩色数を用いて分枝限定を行うことが可能となっている.

また, MCQ では探索の前処理として, 節点の次数の順に整列し, 次数の小さい節点から優先的に探索を行うようにしている. この操作により, 結果として候補節点集合のサイズが小さくなり, 後回しにした節点が分枝限定効果により探索不要となることで, 分枝数が抑えられることが実験的に示されている [3].

3.2 MCQ′

MCQ′ は, MCQ の探索の前処理として行われる節点の整列を改良し, さらに効率のよい探索順序を実現したアルゴリズムである. これは, 節点を実際の探索に即した順序に並べるために, 整列の手続きを “未整列集合の中で次数が最小となる要素を逐次取り出して並べる” として行うことで実現されている.

3.3 NUMBER-SORT

MCQ, MCQ′ で用いられている逐次近似彩色手続きが NUMBER-SORT[3](以下, N-S) である. これは, 再帰処理毎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, これにより分枝限定を有効にできる. 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とすでに番号を付与された節点のうち, 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までの手続きが NUMBER-SORT である.

3.4 MCQ*

ここで, 本稿で提唱するアルゴリズム MCQ*について述べる. MCQ のような深さ優先探索アルゴリズムにおいて, 初期節点の並び順は問題の解を得るまでの探索回数および実行時間に大きな影響を与えることが知られている. しかし, 従来のアルゴリズムの方式では探索中に上界を得るための近似彩色手続き (N-S) を行う度に, 初めに整列しておいた節点の順番が徐々に崩れてしまう可能性があり, 適切な順序での探索ができないことがあった. 節点の再整列化を探索毎に行えば探索領域の削減はできるが, トータルでの実行時間は増加してしまうことが多い.

この問題を解決するため, MCQ′ の方式により整列された初期の節点の並びを保存しておき, 近似彩色はこの順に従って行う, としたのが今回提唱するアルゴリズム MCQ*である. この方法により, 従来のアルゴリズムに比べてより適切な探索を可能にした. このアルゴリズムの概略を図 1 および図 2 に示す.

```
procedure MCQ*(G = (V, E))
begin
   Q := ∅;     Q_max := ∅;
   節点集合 V を MCQ′ の前処理で整列し,
      各節点に番号 No を付与する;
   EXPAND (V, V, No);
   output    Q_max  { 最大クリーク }
end { of MCQ*}

procedure EXPAND(V_s, R, No)
begin
   while R ≠ ∅ do
      p := 候補節点集合 R の最後尾の要素;
      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(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 1 アルゴリズム MCQ*と再帰手続き EXPAND の改良

```
procedure NUMBER-SORT(Vs, R, No)
begin
    順序付き節点集合 Vs の順に節点を近似彩色し,
      各節点に色番号 No を付与;
    彩色した色番号 No で節点を昇順にソートし,
      順序付き節点集合 R へ格納
end { of NUMBER-SORT}
```

図 2 近似彩色手続き NUMBER-SORT の改良

4. 計算機実験・解析

各アルゴリズムをできる限り共通化して C 言語でプログラムを作成し, 計算機上でテストグラフに対して実行することで比較評価を行った. テストグラフは, ランダムグラフおよび DIMACS ベンチマークグラフとし, 結果は表 1 および表 2 である. 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, それぞれの測定結果の平均を取ったものを掲載している. また, 表 2 中の d は枝密度を表している. 尚, 使用した計算機環境は CPU:Pentium4 2.2GHz, コンパイラ:gcc -O2 である. この環境は [4] における計算機環境と全く同一のものである.

ランダムグラフについては, 全面的に MCQ*の方が速く解を得ている. また, 分枝数の削減効果は枝密度が高くなるにつれて大きくなり, 実行時間が短縮されている. DIMACS グラフでは, グラフによっては MCQ′ と全く同じ結果となるものもあるが, p_hat や san などの時間のかかるグラフでは, MCQ′ と比較しても大

表 1. ランダムグラフにおける実行時間と分枝数

Graph			実行時間 (sec)		分枝数	
n	p	ω	MCQ′	MCQ*	MCQ′	MCQ*
100	0.7	14-16	0.0063	**0.0056**	2,211	**1,768**
	0.8	19-21	0.019	**0.015**	5,603	**3,659**
	0.9	29-32	0.059	**0.033**	10,797	**5,269**
200	0.7	18-19	0.93	**0.71**	223,477	**156,483**
	0.8	24-27	17.63	**9.72**	3,265,800	**1,618,860**
	0.9	40-44	980.36	**238.28**	97,627,472	**20,660,192**
500	0.5	13-14	4.52	**4.06**	1,108,144	**940,382**
	0.6	17	79.94	**64.65**	15,699,473	**12,018,933**
	0.7	22-23	4,134.07	**2,858.49**	675,343,739	**422,929,503**
1,000	0.3	9-10	1.57	**1.56**	458,143	**430,341**
	0.4	12	19.49	**18.33**	4,422,241	**3,944,444**
	0.5	15	482.58	**420.72**	90,952,284	**76,124,618**

表 2. DIMACS ベンチマークグラフにおける実行時間と分枝数

Graph				実行時間 (sec)		分枝数	
Name	n	d	ω	MCQ′	MCQ*	MCQ′	MCQ*
MANN_a27	378	0.99	126	**4.22**	**4.22**	**38,020**	**38,020**
c-fat500-5	500	0.19	64	**0.017**	**0.017**	**436**	**436**
hamming8-4	256	0.64	16	**0.28**	**0.28**	41,604	**36,452**
johnson16-2-4	120	0.77	8	**0.21**	**0.21**	**323,036**	**323,036**
keller4	171	0.65	11	**0.037**	0.040	**11,781**	12,442
brock200_1	200	0.75	21	2.39	**1.63**	482,326	**296,035**
brock400_1	400	0.75	27	2,298.29	**1426.59**	329,598,852	**182,920,257**
p_hat500-2	500	0.51	36	4.49	**1.64**	408,474	**123,804**
p_hat500-3	500	0.75	50	2,662.54	**396.66**	138,299,837	**18,184,523**
p_hat1000-2	1,000	0.49	46	3,512.07	**539.48**	197,146,933	**25,648,472**
p_hat1500-1	1,500	0.25	12	6.32	**5.85**	1,260,981	**1,093,972**
san200_0.7_1	200	0.70	30	0.027	**0.024**	2,983	**1,993**
san200_0.9_2	200	0.90	60	6.22	**1.99**	594,912	**200,523**
san400_0.7_3	400	0.70	22	4.53	**3.22**	409,735	**244,366**
san400_0.9_1	400	0.90	100	5.32	**1.52**	74,008	**19,996**
san1000	1,000	0.50	15	7.14	**6.73**	229,675	**193,364**
sanr200_0.7	200	0.70	18	0.79	**0.59**	184,858	**127,795**
sanr200_0.9	200	0.90	42	433.72	**141.31**	40,470,190	**11,774,521**
sanr400_0.5	400	0.50	13	1.12	**1.02**	300,596	**255,377**
sanr400_0.7	400	0.70	21	502.56	**328.43**	89,123,730	**54,622,436**

幅に実行時間の短縮に成功している.

4.1 探索順序の違いによる効果

MCQ および MCQ′ では, N-S を行うたびに前処理として整列しておいた節点の順序が崩れていく可能性があった. 次数の小さい節点から探索することが分枝数を抑えるのに効果的であることは文献 [3] などから知られているため, 再帰処理での各部分問題においても, 次数の小さい節点から探索する方が望ましいと考えられる. MCQ*では初期の並びを保存することでこの問題を解決している.

これによって得られる効果を検証するため, MCQ′ と MCQ*において “N-S は行うが, 彩色番号による分枝限定を行わない”という変更を加えて, 分枝数の比較実験を行った. この実験の 2 つのアルゴリズムによる分枝数の違いは探索順序の違いのみであるから, 探索効率のよい方が分枝数が少なくなるはずである.

表 3 をみると MCQ*の順序で探索したほうが分枝数が少なくなっている. この結果より, 一般に MCQ* の方が分枝数を抑えるのに効果のある探索順序を実現しているといえる. 特に, 枝密度の高いグラフにおいてこの傾向が大きくなることが確認された.

表 3. 探索順序の違いによる分枝数比較

Graph	MCQ′ の順序	MCQ*の順序
100_0.7	520,170	464,105
100_0.8	6,874,793	6,127,593
1000_0.2	1,463,424	1,450,653
1000_0.3	19,947,915	19,374,539
p_hat300-1	68,901	67,451
brock200_2	366,672	345,910
sanr200_0.7	79,039,432	61,396,165

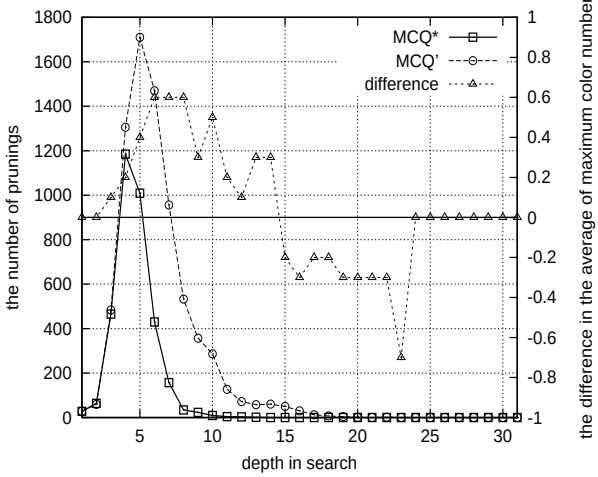

図 3 各深さにおける分枝限定と彩色精度 (ランダム 100_0.9)

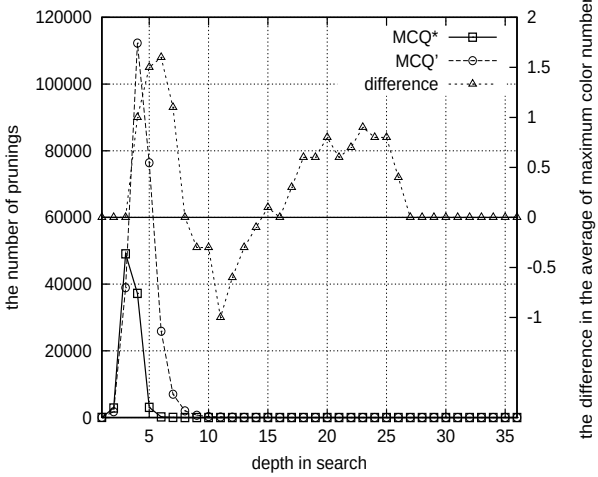

図 4 各深さにおける分枝限定と彩色精度 (p_hat500-2)

4.2 彩色精度の違いによる効果

近似彩色手続き N-S における彩色精度が上昇すれば, 分枝限定効果は高まり, 探索領域の削減に効果がある. N-S では次数の高い節点から順に彩色を行うが, これは文献 [1],[2] で, 逐次的な彩色においてはこのようにすることで彩色精度が上がることが実験的に示されているためである. よって, 探索する順序の逆から彩色すればよく, 効率のよい探索順序は, 即ち効率のよい彩色順序となるので, MCQ*の方式がより彩色の精度がよいことが予想される. これを実際に確認するため, 彩色による分枝限定効果が得られている探索の深さと, この部分においての彩色の精度を比較することでこの効果を検証した.

図 3 および図 4 は, 横軸が探索の深さ, 左縦軸が各深さでアルゴリズムが彩色の効果によって分枝限定を行った回数であり, MCQ′ と MCQ*それぞれに対応している. また, 右縦軸は各深さの再帰処理での最大彩色番号の平均の差を表しており, MCQ*の彩色精度が MCQ′ より良いとき正の値をとる. 図 3 があるランダムグラフ n=100, p=0.9, 図 4 が p_hat500-2 のときの結果である.

図 3 では深さ 5 程度, 図 4 では深さ 3 程度で彩色による分枝限定効果が最も得られており, いずれの場合も, この前後に当たる深さでの彩色精度は MCQ*の方が良くなっている. このため, MCQ*は少ない分枝限定の回数でより多くの探索領域を探索不要とすることができ, 分枝限定の回数自体は MCQ′ より少なく済む. よって, トータルでの分枝数は MCQ*が少なくなり, これが効率化の要因となっていることがわかる. この結果も, 枝密度の高いグラフにより効果が出ることが確認されている.

5. ま と め

本稿では, 最大クリークを効率よく抽出するアルゴリズム MCQ*を提唱した. そして, このアルゴリズムが従来のアルゴリズムと比較して全面的に優位であることを計算機実験によって確認した.

このアルゴリズムは, 既に提案されている近似彩色手続き N-S 部分の改良 ([5],[6]) と組合わせることで, さらに高速化が可能であるが, 本稿ではアルゴリズムの違いによる効果を明確にするため, これは省いた.

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] 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).

[2] 富田悦次, 今松憲一, 木幡康弘, 若月光夫, “最大クリークを抽出する単純で効率的な分枝限定アルゴリズムと実験的評価,” 電子情報通信学会論文誌 (D-I), vol.J79-D-I, no.1, pp.1-8 (1996).

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

[4] 亀田宗克, 富田悦次, “最大クリーク抽出アルゴリズムの高速化と解析・評価,” 情報処理学会研究報告, 2004-AL-93, pp.33-40 (2004).

[5] 森田昭広, 富田悦次, 亀田宗克, “最大クリーク抽出のより高速なアルゴリズム,” 情報科学技術レターズ (FIT2004), pp.19-22 (2004).

[6] 卜部昌平, 富田悦次, 森田昭広, “密なグラフで有効な最大クリーク抽出アルゴリズム,” 情報処理学会第 67 回全国大会, vol.1, pp.231-232 (2005).