LA-007

最大クリーク抽出の より高速なアルゴリズム

A Faster Algorithm for Finding a Maximum Clique

森田 昭広† 富田 悦次‡ 亀田 宗克§
Akihiro Morita Etsuji Tomita Toshikatsu Kameda

1. はじめに

現実問題の多くは, グラフ上の問題として定式化して解決することが出来る. 更にこの内の幾つかの問題群は, 無向グラフ中の最大クリーク抽出問題に帰着して解くことが出来る.

無向グラフ中の最大クリークを抽出する問題は典型的なNP困難問題で難しいにもかかわらず, その重要性より, これまでに幾多の研究がなされている（文献[1]〜[10]. 応用例に関しては, 文献[6], 7. Selected Applications 参照）. 筆者らも, 従来よりも格段に高速な厳密最大クリーク抽出アルゴリズムMCQを提唱し[9], それにより, バイオインフォマティクス[11]〜[13], 画像処理[14], 量子論理回路設計[15], DNA配列設計[16]などへの新しい応用の成果が得られている. これらの応用を更に発展させるためには, 最大クリーク抽出アルゴリズムの高速化が非常に重要な基本となる. そこで本論文では, 先に発表した最大クリーク抽出アルゴリズムMCQ[9]を基にして一層の高速化を図ったアルゴリズムMCQ′を提唱し, 実験的にその優位性を検証した結果を示す.

2. 表記法と定義

本論文における表記法などは, 特に記述のない限り以下のように扱うものとする[1].

V を節点 (vertex) の集合, $E \subseteq V \times V$ を枝 (edge) の集合としたときに, $G = (V, E)$ を対象となる単純無向グラフとする. ここで節点集合 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. また2節点 i, j 間の枝は非順序対 $(i, j) = (j, i)$ で表し, このとき節点 i と j は隣接していると言う. ただし, 枝としてループや多重枝は含まないものとする.

グラフの表現としては $|V| \times |V|$ 対称行列である隣接行列 (adjacency matrix) $A_G = (a_{ij})_{(i,j)\in V\times V}$ を用いる. 隣接行列の要素は $(i, j) \in E$ のとき $a_{ij} = 1$, $(i, j) \notin E$ のときは $a_{ij} = 0$ である. グラフ G において節点 v に隣接する節点の集合を $\Gamma(v) = \{j \in V \mid a_{vj} = 1\}$ で表し, その節点の数を $deg(v) = |\Gamma(v)|$ と書き, v の次数 (degree) と呼ぶ. グラフ G の節点の次数のうち, 最大の値を G の最大次数と呼び, $\Delta(G)$ で表す. 全ての節点の次数が一致しているグラフは正則グラフと呼ぶ.

$G = (V, E)$ において, V の部分集合 S によって誘導される部分グラフ (induced subgraph) を $G(S) = (S, E \cap (S \times S))$ と表す. 本論文で部分グラフと記述した場合, それは誘導部分グラフを指す. V の部分集合 C において $G(C)$ の任意の2節点が隣接しているとき, グラフ $G(C)$ は完全 (complete) であると言い, このとき, $G(C)$(あるいは C) をクリーク (clique) と呼ぶ. それが他のクリークの真の部分グラフ (集合) でなければ極大クリークと呼ぶ. 最も節点数の多い極大クリークは最大クリークと呼び, そのサイズ (節点数) を $\omega(G)$ または $\omega(V)$ で表す.

†電気通信大学 大学院 情報システム学研究科, UEC-IS
‡電気通信大学 大学院 電気通信学研究科, UEC-EC
§（株）オリンピア, OLYMPIA Corp.

3. 最大クリーク抽出アルゴリズム

本論文で対象とするのは, グラフのデータ (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個見出して出力するアルゴリズムである. これより, いくつかの最大クリーク抽出アルゴリズムとその効率化手法について述べる.

3.1 基礎アルゴリズム[1]

基礎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節点と隣接している節点を新たに付け加え, 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.

具体的には, まず初期設定として, 探索途中で見出されたクリーク Q を $Q := \emptyset$(空集合), 現在の段階で探索の対象として考えている順序付き節点集合 (候補節点集合と呼ぶ) を $R := V$(全節点集合) とする.

そして, R 中のある節点 p を選び出してこれをクリークに付け加え ($Q := Q \cup \{p\}$), R 中で p に隣接している節点の集合を求めて ($R_p := R \cap \Gamma(p)$), 得られた節点集合 R_p を新たな候補節点集合とみなして同様の操作を再帰的に適用していく (この再帰手続きをEXPANDと呼ぶ). これにより, 各段階の候補節点集合はその時点の Q のいずれの節点とも隣接している節点の集合となっている.

このようにしていって $R_p = \emptyset$ となったとき, そのとき得られているクリーク Q は極大である. ここから更にクリーク探索を続けるために, R から p を削除したものを改めて R と置き直し, Q からも p を削除してから, R 中の新たな節点を改めて p として選択し, 再び同様の操作を進めていく.

この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 を頂点 (node) として, R と $R_p(= R \cap \Gamma(p))$ の関係にあるものを親子関係の枝で結んで得られる探索木 (search tree) として表現することができる. 従って, 探索が深さ i の段階にあるときには, Q には節点数 i のクリークが格納されていることになる. 探索の終了条件は, 初期設定として与えられた候補節点集合 (V) が空になることである.

上記の手順を進めていくと極大クリークを全て列挙することができる. ここでは最大クリークを1個抽出することを目的としているので, 最初に $Q_{max} := \emptyset$ と設定し, より大きな極大クリークが抽出されるたびに Q_{max} を更新していき, 最終的に Q_{max} に記憶されている節点集合から成る部分グラフを最大クリークとする.

しかし, これではすべての組合せについて考慮することになり効率が悪い. そこで次のような方法で効率化を

行う.

3.2 MCQ[9]

探索木における枝を分枝 (branch), その総枝数を分枝数と呼ぶが, アルゴリズムの効率化に当たっては解の探索領域を小さくすることが重要である. そのために何らかの方法を用いて余分な分枝を制限することが分枝限定法のねらいである.

簡単な分枝限定として, 新たな R が生成されたときに $|Q| + |R| \leq |Q_{max}|$ が成立する場合には, そこから先の探索で現在保持している以上のサイズの極大クリークは存在しないことが明らかであるので, それ以上探索を行わない, とする方法が考えられる. 最大クリーク抽出問題の効率的なアルゴリズムを開発するために, 本論文では基本的にこのような分枝限定法を用いる [1].

本論文の基礎となっている最大クリーク抽出アルゴリズム MCQ[9] では, 更に近似彩色手続きを用いてより強力な分枝限定を行っている. グラフ彩色とは隣接している節点が同じ色にならないよう, 各節点に色を付与していく問題であるが, クリークを構成する全節点は互いに隣接しているという性質上, この彩色数をクリークサイズの上界として用いることができる. MCQ[9] で用いられているのは NUMBER-SORT という逐次的近似彩色法であり, 以下の条件を満たすよう, 各候補節点に対して色に相当する正整数を付与していくものである.

1. $a_{ij} = 1$, $(i \neq j)$ であれば, $N[i] \neq N[j]$ である.
2. $N[i] = k > 1$ であるとき, $N[j] = 1, 2, \ldots, k-1$ なる節点 $j \in \Gamma(i)$ がそれぞれ少なくとも一つ存在する.

この番号付けが NUMBER であり, ここで付与された番号の昇順に彩色済み候補節点集合を整列する作業が SORT である. 図 1 にそのアルゴリズムを示す.

NUMBER-SORT(N-S と略記する) によって与えられた色番号の最大値 $\mathrm{Max}\{N[p]\}$ が $\omega(R) \leq \mathrm{Max}\{N[p] \mid p \in R\}$ $(\leq |R|)$ という条件を満たすため, $|Q| + \mathrm{Max}\{N[p] \mid p \in R\} \leq |Q_{max}|$ であれば探索木中の R の子孫における探索で最大クリークが更新されることはなく, これより先の探索を中断することができる.

さらに MCQ[9] では, 文献 [1](アルゴリズム#1) で確認された効率化手法に基づいて, 全探索開始前に全節点集合を次数によって整列し, N-S を適用せず, 整合性のために簡単な番号を付与するのみで手続き EXPAND に進むという方法により効率化を達成している (文献 [9], Tables 1, 3 参照).

3.3 MCQ′[17]

筆者らは, 更に効率の良い全探索開始前の処理を文献 [17] で提唱した. これは, 実際の探索に即した整列を行うために, 整列の手続きとして未整列節点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, という方法である. 同時にその節点と隣接している節点間の枝を切り, 隣接節点の次数をそれぞれ 1 減少させる. この様な整列を行っていくと, その過程で未整列節点集合中の最小次数節点が複数個存在する場合があるため, それらの節点と隣接している要素における次数の総和を計算し, その最小値を持つ節点を最優先対象要素とする. この様にして, その節点を探索するときの分枝数をより小さくすることができる.

```
procedure NUMBER-SORT(R, N)
begin
{NUMBER}
    maxno := 0;
    C_1 := ∅;
    while R ≠ ∅ do
        p := the first vertex in R;
        k := 1;
        while C_k ∩ Γ(p) ≠ ∅
            do k := k + 1 od
        if k > maxno then
            maxno := k;
            C_{maxno+1} := ∅
        fi
        N[p] := k;
        C_k := C_k ∪ {p};
        R := R − {p}
    od
{SORT}
    i := 1;
    for k := 1 to maxno do
        for j := 1 to |C_k| do
            R[i] := C_k[j];
            i := i + 1
        od
    od
end {of NUMBER-SORT}
```

図 1: 手続き NUMBER-SORT

更に, 未整列節点集合中の最小次数節点集合が未整列節点集合全体において等しく, 正則グラフとなる場合がある. この時, 次数を基準にして整列を行っても全く意味が無いため, 整列過程において未整列節点集合のグラフが正則グラフとなった時点で, それ以降次数を基準に整列することを止める. そして, これらの未整列節点集合には近似彩色手続き N-S を適用し, この時の整列結果を整列済み節点集合に連接する.

番号付けに関しては, この未整列節点集合に対して行った手続き N-S から得られた彩色数 $(\mathrm{Max}\{N[p]\})$ に則し, 整列済み節点集合に対しては, その先頭から $\mathrm{Max}\{N[p]\} + 1$, $\mathrm{Max}\{N[p]\} + 2$, ... と, $\Delta(G) + 1$ を比べて, 小さい方を付与していく. 以上の操作が完了した上で探索手続き EXPAND を適用する, としてアルゴリズム MCQ′ は構成される.

3.4 近似彩色アルゴリズム

ここまで述べてきた最大クリーク抽出アルゴリズムにおいて, 近似彩色による効率化ということは共通していた. 先に述べたクリークサイズと近似彩色数の関係から, その精度が高ければ効果的な分枝限定を実現できることは明らかであるが, グラフ彩色問題自体も NP 困難のクラスに属しているため, 実行時間の増大が問題となる. MCQ や MCQ′ が高速であるのは, N-S の高速性に因るところが大である. ここで, 幅広い対象グラフにおいて, N-S 部分が 50 %以上の実行時間を占有していることが確認されている [17]. そこで, より高速な近似彩色アルゴリズムを考案することにより最大クリーク抽出の高速化を目指した.

本論文で提唱する近似彩色アルゴリズムは, N-S と全く同じ結果をより短時間で得ることを目的としている. その方法として, 使用するメモリ領域を局在化させることにより, CPU に搭載されているキャッシュメモリの効果を最大限に発揮させることを図る. そのために, N-S では節点に逐次着目して色番号を付与しているが, これを逆に色番号に着目して, 色番号順に彩色するようにす

る．具体的には，まず候補節点集合を連結リストとして構成し，1 つ目の色番号を付与できる節点を一通り選んでいく．この色番号の付与と同時に，選ばれた節点を彩色済み候補節点集合へ順に格納していき，この操作を残りの全節点に色番号が付与されるまで逐次続ける．以上の逐次的近似彩色アルゴリズムを ABSORB(ABS と略記) と名付け，図 2 にそのアルゴリズムを示す．N-S では全候補節点に対して色番号が付与されるまで，各節点を保持していなければならなかったため，$n \times n$ の領域が必要であった．それに対し，ここで提唱した ABS は連結リストを構成するための $n \times 2$ と，現時点で着目している彩色集合を保持するための $n \times 1$ の領域が確保されていればよいため，キャッシュメモリ内で処理可能となる割合が高まることが期待できる．実際に，各ランダムグラフに対して近似彩色のみを行った結果，最も効果を得ている部分では表 1 程度の差が観察された (n：節点数，p：枝密度 (2 節点間に枝の存在する確率 [2]))．

```
procedure ABSORB(R, N)
begin
    ColorN := 1;
    i := 1;
    List := R
    while List ≠ ∅ do
        p :=the first vertex in List;
        C := ∅;
        while p ≠nil do
            if C ∩ Γ(p) = ∅ then
                C := C ∪ {p};
                R[i] := p;
                i := i + 1 :
                List := List − {p};
                N[p] := ColorN;
            fi
            p :=the next vertex of p in List;
        od
        ColorN := ColorN + 1;
    od
end {of ABSORB}
```

図 2: 手続き ABSORB

表 1: NUMBER-SORT と ABSORB の比較

Graph		N-S	ABS	実行時間比
n	p	[msec.]	[msec.]	ABS/N-S[%]
300	0.1	0.261	0.115	44.1
	0.5	0.412	0.244	59.2
	0.9	0.589	0.287	48.7
	0.95	0.734	0.313	42.6

3.5 MCQ″

前節の近似彩色アルゴリズム ABS を N-S の代わりに MCQ′ 中に埋め込んだ最大クリーク抽出アルゴリズムを MCQ″ とする．

4. 計算機実験

上記アルゴリズム MCQ″ を評価するため，計算機上に実働化し，種々のグラフに対して実行した．実働化に用いたプログラミング言語は C であり，使用した計算機環境は CPU：Pentium4 2.2GHz(キャッシュメモリ 512KB)，OS：Linux，コンパイラ：gcc -O2 である．

先ず，実行対象のグラフは，節点数 n，枝密度 p について乱数の種が異なるランダムグラフを 10 個ずつ作成したものとし，それらの計算結果の平均値を表 2 に掲載する．但し，長大実行時間を要する為に，乱数の種 1 個のみに対して行った結果は，数値右肩に * 印をつけて区別してある．次に，DIMACS ベンチマークグラフ [2] に対する実行結果を表 3 に掲載する．表 2, 3 において，他の各アルゴリズムの実行時間は，本実験環境に合わせて校正 [2] した値である (空白欄は，参考文献中に実験結果が示されていない部分．Tar/5 は推定参考値)．この内で，ABS の効果が顕著に表れている例として，MANN_a45 に対して MCQ′ から MCQ″ では 1/2.9 に減少，hamming10-2 に対して MCQ′ から MCQ″ では 1/2.1 に減少している．MCQ に対する MCQ″ の実行時間としては，例えば san200_0.9_3 に対しては 1/86.7 に減少，san400_0.9_1 に対しては 1/16.1 に減少 ($\star$ 印)，などである．

対象グラフよっては MCQ″ よりも MCQ の方が高速である場合もあるが，その場合の違いは少なく，かつ他所から発表されたどのアルゴリズムよりも元々MCQ が既に非常に高速となっている場合が多い．

以上より，本論文で提唱した MCQ″ は近年他所から発表されたどのアルゴリズムと比べても，全般的に一段と高速化されていることが確認された．

5. むすび

本論文では，より高速化した最大クリーク抽出アルゴリズム MCQ″ を提唱し，その高速性を実証した．具体的問題の一例として，タンパク質スレッディング問題から得られるある実データについて，MCQ では 1,084 秒かかった実行時間が MCQ″ では 47 秒と 1/23 に短縮された．本最大クリーク抽出アルゴリズムは，効率的な極大クリーク全抽出アルゴリズム [12] と合わせ，本論文冒頭で示した応用の他にも，薬剤設計問題 [19], [20] 等，更に広い応用が考えられ，この様な実問題へのより大きい寄与が期待出来る．

謝辞 データを提供していただきました京大・阿久津達也教授に御礼申し上げます．また，本論文における実験において多大な協力をいただいた大学院生の中村知倫氏，卒研生の卜部昌平氏，有益なコメントをいただいた査読者の方々に感謝します．なお，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] D. S. Johnson and M. A. Trick, ed., “Cliques, Coloring, and Satisfiability,” DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 26, American Mathematical Society (1996).

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

[4] 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 [2] (1996).

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

[6] 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 vol. A, Kluwer Academic Publishers, pp.1-74 (1999).

表 2: ランダムグラフに対する平均実行時間 [sec.]

Graph			dfmax	MCQ	MCQ′	MCQ″	New	COCR
n	p	ω	[6]	[9]	[17]		[8]	[5]
100	0.8	19-21	0.22	0.026	0.019	**0.016**	0.10	0.24
	0.9	29-32	5.97	0.066	0.059	**0.047**	1.04	0.31
	0.95	39-44	40.94	0.023	0.019	**0.015**	0.31	
200	0.6	14	0.46	0.14	0.12	**0.11**	0.27	0.82
	0.7	18-19	6.18	1.13	0.93	**0.80**	4.75	2.59
	0.8	24-27	314.92	20.19	17.63	14.81	231.54	**13.66**
300	0.5	12-13	0.59	0.24	0.196	**0.185**	0.32	1.78
	0.6	15-16	7.83	2.28	1.85	**1.52**	5.50	7.83
	0.7	19-21	233.69	37.12	30.89	**27.10**	179.71	
	0.8	28-29	48,280.6*	2,063.3	1,790.0	**1,514.9**		
500	0.4	11	1.02	0.48	0.44	**0.42**	0.94	
	0.5	13-14	14.45	5.40	4.52	**4.27**	11.40	27.41
	0.6	17	399.22	96.34	79.94	**73.71**	288.10	
1,000	0.4	12	51.92	22.98	19.49	**19.15**	36.46	
	0.5	15	1,766.85	576.36	482.52	**461.94**		
	0.6	20	160,028.33*	34,496.35*	27,095.67*	**25,290.03***		

表 3: DIMACS ベンチマークグラフに対する実行時間 [sec.]

Graph Name	dfmax [2]	MCQ [9]	MCQ′ [17]	MCQ″	New [8]	χ+DF [7]	COCR [5]	MIP [4]	SQU [3]	Tar/5 [10]
brock200_1	23.96	2.84	2.39	**2.05**	19.05	112.2		1,471		35.0
brock200_2	0.05	0.016	0.015	**0.014**	0.018	0.37	0.54	60	847.3	0.10
c-fat200-5	444.03	**0.0021**	0.0025	0.0023	2.74	0.008		1.7	60.5	
hamming8-2	>24hrs.	0.0205	0.0036	**0.0030**	0.014	0.088			0.05	
hamming10-2	>24hrs.	1.82	0.33	**0.16**	0.88	6.23		1.0		
johnson8-4-4	0.0071	0.0009	0.0006	**0.0004**	0.0035	0.032		0.4	0.004	
johnson16-2-4	1.21	0.34	0.21	0.18	0.095	9.65		777	**0.003**	
MANN_a9	0.062	0.0002	0.00018	**0.00014**	0.0035	0.008				
MANN_a27	>24hrs.	8.49	4.22	**3.40**	>3,500	12,488.6	4.33	1,524		
MANN_a45	>24hrs.	7,107.83	6,212.62	**2,115.91**	>3,500	>17,000				
p_hat500-2	219.37	6.22	4.99	**3.63**	150.5	246.1		5,331		>2,160
p_hat1000-1	1.66	0.86	0.75	**0.74**	2.05	19.83				
p_hat1000-2	>24hrs.	4,104.15	3,512.07	**2,862.37**		>17,000				
san200_0.9_3	>24hrs.	16.41	0.24	⋆ **0.19**		235.27		431	23.60	>2,160
san400_0.7_1	>24hrs.	**1.72**	2.34	1.97	>3,500	514.06				12.1
san400_0.7_2	>24hrs.	1.58	0.45	**0.39**	177.6	192.74			786.8	56.6
san400_0.9_1	>24hrs.	72.69	5.32	⋆ **4.51**		8,712.2		3,240		
sanr400_0.5	3.50	1.47	1.12	**1.06**	2.32	27.87				

[7] T. Fahle, "Simple and fast: Improving a branch-and-bound algorithm for maximum clique, " European Symp. on Algorithms 2002, LNCS 2461, pp.485-498 (2002).

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

[9] 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).

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

[11] D. Bahadur K. C. , T. Akutsu, E. Tomita, T. Seki, and A. Fujiyama, "Point matching under non-uniform distortions and protein side chain packing based on an efficient maximum clique algorithum," Genome Informatics, 13, pp.143-152 (2002).

[12] T. Akutsu, M. Hayashida, E. Tomita, J. Suzuki, and K. Horimoto: Protein threading with profiles and constraints, Proc. IEEE Symp. on Bioinformatics and Bioengineering, pp.537-544 (2004).

[13] D. Bahadur K.C., E. Tomita, J. Suzuki, and T. Akutsu, "Protein side-chain packing problem: A maximum edge-weight clique algorithmic approach," J. Bioinformatics and Computational Biology (to appear).

[14] 堀田一弘, 富田悦次, 高橋治久, "最大クリーク抽出に基づく向きの変化に依存しない人物の顔検出," 情報処理学会論文誌 数理モデル化と応用, vol.44, no.SIG14 (TOM9), pp.57-70 (2003).

[15] 名久井行秀, 西野哲朗, 富田悦次, 中村知倫, "節点重み最大クリーク抽出に基づく量子論理回路の深さ最小化," 京大数解研講究録 1325, pp.45-50 (2003).

[16] S. Kobayashi, T. Kondo, K. Okuda, and E. Tomita, "Extracting globally structure free sequences by local structure freeness," Proc. DNA9, p.206 (2003).

[17] 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).

[18] E. Tomita and A. Tanaka, and H. Takahashi, "The worst-case time complexity for generating all maximal cliques," Proc. COCOON 2004, LNCS 3106, pp.161-170 (2004).

[19] H. Xu and D. K. Agrafiotis, "Retrospect and prospect of virtual screening in drug discovery," Current Topics in Medicinal Chemistry, vol.2, no. 12, pp.1305-1320 (2002).

[20] P. Willett, "Similarity-based approaches to virtual screening," Biochem. Soc. Trans. vol.31, pp.603-606 (2003).