数理解析研究所講究録 1325 巻 2003 年 39-44

節点重み最大クリーク抽出アルゴリズム

An Algorithm for Finding a Maximum Clique with Maximum Vertex Weight

中村 知倫, 富田 悦次, 西野 哲朗, 名久井 行秀 (電気通信大学)
Tomonori Nakamura, Etsuji Tomita, Tetsuro Nishino, and Yukihide Nakui
(The University of Electro-Communications)

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, 基本的で重要な組合せ最適化問題の一つであり, NP 困難ではあるが, その効率的な厳密解アルゴリズムをこれまでに発表してきている ([1] 等). 更に最近, 一層効率化を達成したアルゴリズムも得ており [2], Discrete Applied Mathematics 誌 2002 年 8 月号にÖstergård が発表した新しいアルゴリズム [3] に対しても非常に高速であることを確認している [4].

このアルゴリズムを適用することにより, バイオインフォマティクス [5], 画像処理 [6], 分子計算における DNA あるいは RNA 配列の設計 [7] 等の実問題解決の有効な結果を得ている. 更に, この様な実問題処理においては枝に重みを考慮することも重要となり, それに対応した最大クリーク抽出アルゴリズムも開発している [8].

ところで, 量子論理回路の深さを最小化する問題も最大クリーク問題へ還元することが可能であり [9], この場合には節点に重みを考慮することが必要となる. 従って本稿においては, 節点に重みを持ったグラフを対象とし, 最大サイズクリークの中で節点重みの総和が最大であるクリークを抽出する一般的なアルゴリズムを提唱する. 更に, これをランダムグラフ, DIMACS ベンチマークテスト用グラフ, 及び量子論理回路を還元したグラフに適用して, 実験的にその性能評価を行った.

1.1 諸定義及び記法

V を節点の集合, $E \subseteq V \times V$ を枝の集合とするとき, 対象とする無向グラフを $G = (V, E)$ で表す. 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 は隣接しているという. ここで対象とする無向グラフでは, 自己閉路や多重枝は含まないものとする.

グラフ G において, 節点 v に隣接する節点の集合を $\Gamma(v)$ で表す. 一般に集合 S の要素数を $|S|$ で表し, $|\Gamma(v)|$ を節点 v の次数と呼ぶ.

グラフ $G = (V, E)$ 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を $G(S) = (S, E \cap (S \times S))$ と表す. 以後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 の任意の 2 節点が隣接しているとき, 即ち $\forall i, j \in V, i \neq j$ に対して $(i, j) \in E$ であるとき, グラフ G は完全であるという. V の部分集合 C において $G(C)$ が完全であるとき, C をクリークと呼ぶ. なおかつ, 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, 対象とするグラフ G 中で節点数が最大の極大クリークを最大クリークと呼ぶ. グラフ G 中の最大クリークの節点数を $\omega(G)$ 又は単に ω で表し, グラフ G 中の最大クリークの個数を $N(\omega)$ で表す.

V の部分集合 C において任意の 2 節点が互いに隣接していないとき, C を独立節点集合と呼ぶ. グラフ中の全節点を最も少ない数の独立節点集合に分割することを彩色と言い, 同じ独立節点集合に含まれる節点は同じ色であると言う. この彩色を行うと, 隣接する節点同士には必ず異なる色が付与されていることになる. 彩色可能な最も少ない色数を彩色数と呼ぶ.

節点 v に付与された節点重みを $w(v)$, 節点集合 V 内の各節点の節点重みの総和を $w(V)$ で表す.

グラフ G 中の最大クリークの中で, 節点重みの総和が最大となる部分集合を節点重み最大な最大クリークと呼ぶ. 本稿ではこれ以降, 節点重み最大な最大クリークを略して節点重み最大クリークと呼び, 節点重

み最大クリークを抽出する問題を節点重み最大クリーク抽出問題と呼ぶ.

2 アルゴリズム

2.1 最大クリーク抽出アルゴリズム

文献 [1] では, 最大クリーク抽出アルゴリズム MCLIQ(または MC) を提唱している. このアルゴリズムは, 深さ優先探索により最大クリークを 1 個抽出する. この探索を効率的に行うため, 近似彩色に相当する手続き NUMBERING-ARRANGING を行うことで, 各節点ごとのクリークの節点数の上界を高速に求め, その上界により分枝限定を行っている. 本稿では, まずこの MC を基本アルゴリズムとして, 目的とする節点重み最大クリーク抽出アルゴリズムを構築する.

2.2 基本アルゴリズム

```
1:  procedure VWMC(G = (V, E))
2:  begin
3:     global Q := ∅, Q_max := ∅;
4:     global w(Q) := 0, w(Q_max) := 0;
5:     Sort vertices of V in nonincreasing order
6:           with respect to their degrees;
7:     NUMBERING-ARRANGING(V,No);
8:     EXPAND-VWC(V,No);
9:     output Q_max
10: end {of VWMC}
```

図 1: アルゴリズム VWMC

アルゴリズム MC において, 候補節点集合 R における分枝条件は,

$$|Q| + \mathrm{Max}\{No(p)|p \in R\} > |Q_{max}| \tag{1}$$

である. ただし, Q は現在の探索で保持しているクリーク, Q_{max} は現在までに抽出されている最大クリークである. $No(p)$ を近似彩色によって節点 p に付けられた番号とすると, その最大値は候補節点集合内のクリークの節点数の上界である. しかし節点重み最大クリークでは, グラフ中に最大クリークが複数存在する場合, それら全てについて重みの総和の比較を行う必要がある. そこで分枝条件を

$$|Q| + \mathrm{Max}\{No(p)|p \in R\} \geq |Q_{max}| \tag{2}$$

とすることで, Q_{max} と同じ節点数の極大クリークであっても抽出できるようにする. この分枝条件を用いるアルゴリズムを VWMC とする. 図 1, 2 がアルゴリズム VWMC である.

ただしこの方法では, 探索中に最大クリークが見つかるまでは本来探索する必要のない節点も探索されてしまうことになる. そこで ω が既知の場合は, 式 (2) の分枝条件中の $|Q_{max}|$ を ω に変更することで, 無駄な探索が行われずに済む. このようにして探索を行うアルゴリズムを VWMC_ω とする. ω が未知の場合は, 最大クリーク抽出アルゴリズムを用いて ω を求めてから, VWMC_ω を適用する. この方法を用いる場合, MC+VWMC_ω と記す.

2.3 重みの上界による分枝限定

前記のアルゴリズムでは, 対象とする無向グラフ中に複数個存在する最大クリークを全て抽出して, その中で節点重みの総和が最大であるものを 1 個選んでいたが, このような方法を用いると, 最大クリークの個数が多くなると最大クリークを 1 個だけ抽出するよりも抽出に必要な時間が増大してしまう可能性がある. この問題を解決するために, 候補節点のクリーク重みの上界を高速に求める方法を考える.

```
1:  procedure EXPAND-VWC(R,No)
2:  begin
3:    while R ≠ ∅ do
4:      p := a vertex in R
5:        such that No(p) =Max{No(q)|q ∈ R};
6:      if |Q| + No(p) ≥ |Q_max| then
7:        Q := Q ∪ {p};
8:        w(Q) := w(Q) + w(p);
9:        R_p := R ∩ Γ(p);
10:       if R_p ≠ ∅ then
11:         NUMBERING-ARRANGING(R_p,No');
12:         EXPAND-VWC(R_p,No')
13:       else if w(Q) > w(Q_max) then
14:           Q_max := Q;
15:           w(Q_max) := w(Q)
16:           fi
17:       fi
18:     fi
19:     Q := Q − {p};
20:     w(Q) := w(Q) − w(p);
21:     R := R − {p}
22:   od
23: end{of EXPAND-VWC}
```

図 2: 手続き EXPAND-VWC

ある候補節点集合に番号付けを行った結果が $\mathrm{Max}\{No(p)|p \in R\} = maxno$ であり, 付けられた番号ごとの独立節点集合が

$$C_i = \{p \in R | No(p) = i\}, \quad (i = 1, 2, \ldots, maxno) \tag{3}$$

であるとする.

任意の候補節点集合 R 内に番号付けが行われた後, 番号が大きいほうから順に次の深さへの探索が行われていき, その節点における探索でより大きなクリークが見つからない場合 R 内から削除する, という作業を $R = \emptyset$ となるまで繰り返している. 従って, 次に探索する節点の番号が k であるならば, 残りの候補節点の番号は k 以下となる. クリークは異なる番号の独立節点集合から節点が 1 個ずつ選ばれて構成されるので, それぞれの独立節点集合内の重みの最大値の合計値がクリーク重みの上界となる. これを利用すると, 番号が k である節点のクリーク重みの上界は,

$$UWC_k = \sum_{i=1}^{k} \mathrm{Max}\{w(p)|p \in C_i\}, \quad (k = 1, 2, \ldots, maxno) \tag{4}$$

と表すことができ, 式 (4) の重みの上界を用いると, ω が既知の場合, 次のような分枝条件が考えられる.

$$|Q| + \mathrm{Max}\{No(p)|p \in R\} \geq \omega \quad \text{and} \quad w(Q) + UWC_{\mathrm{Max}\{No(p)|p \in R\}} > w(Q_{max}) \tag{5}$$

式 (5) の分枝条件を用いるアルゴリズムを $\mathrm{VWMC}_\omega+$ とする.

3 計算機実験

今回提唱したアルゴリズム VWMC, VWMC_ω, $\mathrm{VWMC}_\omega+$, 更に最大クリーク抽出アルゴリズム MC を C 言語を用いて実働化した. コンパイルは gcc -O2 で行った. 使用した計算機は CPU:Pentium4 2.2GHz, OS は Linux である.

3.1 ランダムグラフ

ランダムグラフは, 各節点に確率 p の一様乱数で枝を張って作成したグラフである. 節点数 n, 枝密度 p の各組み合わせごとに, 乱数の seed として 1 から 10 の正整数を与えることでランダムグラフを 10 個用意し

た. 次に, 作成したランダムグラフの各節点に節点の重みとして, 節点数の 10 分の 1 を重みの最大値として 1 から (重みの最大値) までの整数値を, 各節点に一様乱数的に与える. このような条件で作成したランダムグラフの各節点に重みを付与したものを各グラフ 1 個ずつ用意した. アルゴリズム $VWMC_\omega$, $VWMC_\omega+$ を実行する際には, あらかじめ ω が判明している必要があるので, 前処理として最大クリーク抽出アルゴリズム MC を用いて ω を求めその結果だけを利用した. (即ち, この場合には MC の実行時間は考慮に入れていない.)

アルゴリズム VWMC, $VWMC_\omega$, $VWMC_\omega+$ の実行時間を表 1 に示す. なお, 表には 10 個のグラフの平均値を載せた. 比較のため, 節点重みを考慮しない場合の MC の実行時間も載せた.

表 1: ランダムグラフに対する実行時間 [sec]

グラフ					実行時間			
n	p	ω	$N(\omega)$	w_{max}	MC	VWMC	$VWMC_\omega$	$VWMC_\omega+$
100	0.9	29-32	2-226	159-206	0.070	0.188	0.111	0.113
	0.95	39-46	18-10,714	221-304	0.018	0.129	0.069	0.040
300	0.6	15-16	1-102	189-304	2.23	3.53	3.33	3.44
	0.7	19-21	2-141	278-414	39.62	64.89	56.91	59.26
500	0.5	13-14	1-69	401-475	5.32	7.89	7.71	8.02
	0.6	17	3-22	396-556	97.28	146.77	136.25	147.58
1,000	0.4	12	3-14	589-862	22.14	30.71	28.99	31.46
	0.5	15	10-22	872-1,083	554.56	783.53	760.26	811.04
3,000	0.2	9	1-7	1,370-1,993	28.60	34.21	33.03	34.70
	0.3	11-12	1-88	1,286-2,538	665.58	834.61	805.80	837.34

3.2 DIMACS ベンチマーク用グラフ

DIMACS で提供されているベンチマーク用グラフは節点重みなし無向グラフである. そこで, ランダムグラフの場合と同様に各節点の重みとして, 節点数の 10 分の 1 を重みの最大値として, 1 から (重みの最大値) までの整数値を, 一様乱数に従って与えたものを, それぞれ 1 個ずつ用意した.

DIMACS ベンチマーク用グラフに対する VWMC, $VWMC_\omega$, $VWMC_\omega+$ の実行時間を表 2 に示す.

表 2: DIMACS グラフに対する実行時間 [sec]

グラフ						実行時間			
Name	n	p	ω	$N(\omega)$	w_{max}	MC	VWMC	$VWMC_\omega$	$VWMC_\omega+$
brock200_1	200	0.745	21	2	260	4.39	7.91	5.13	5.40
brock200_2	200	0.496	12	1	122	0.02	0.03	0.02	0.02
c-fat200-1	200	0.077	12	14	158	0.00015	0.00021	0.00037	0.00018
c-fat500-1	500	0.036	14	19	439	0.00068	0.00081	0.0083	0.00077
hamming8-2	256	0.969	128	2	1,678	0.0058	0.0114	0.0113	0.0123
hamming8-4	256	0.639	16	480	290	0.031	0.54	0.85	0.51
johnson8-2-4	28	0.556	4	105	8	0.00003	0.00005	0.00005	0.00003
johnson8-4-4	70	0.768	14	30	66	0.00059	0.00154	0.00152	0.00148
johnson16-2-4	128	0.765	8	2,027,025	89	0.23	1.11	1.10	0.08
keller4	171	0.649	11	2,304	149	0.0455	0.0819	0.0819	0.0605
MANN_a9	45	0.927	16	9,540	56	0.00018	0.00671	0.00657	0.00040
MANN_a27	378	0.990	126	$> 4.2 \times 10^{10}$	3,085	8.03	>24hrs.	>24hrs.	19,860.62
p_hat700-1	700	0.249	11	2	361	0.18	0.24	0.35	0.20
p_hat1000-1	1000	0.245	10	276	715	1.02	1.43	1.42	1.46
san200_0.9_1	200	0.900	70	1	677	0.0231	0.0358	0.0024	0.0026
san400_0.7_1	400	0.700	40	1	891	0.89	584.87	0.26	0.27
san1000	1000	0.502	15	1	763	17.84	46.72	0.65	0.68
sanr200_0.7	200	0.702	18	13	242	0.99	1.70	1.44	1.27

3.3 量子論理回路を還元したグラフ

量子論理回路を還元した節点重み付き無向グラフの実行結果を以下に示す. 無向グラフとして, 文献 [10] の 2 変数論理関数の場合 (表 1), 3 変数論理関数の場合 (表 2) のデータを用いた. 表 1 のグラフを G_1, 表 2 のグラフを G_2 とし, 表の上から順に $G_1(1), G_2(2), \cdots$ のように表す.

実験結果を表 3, 4 に示し, 更に詳細は文献 [10] に示す. ただし, この実験で使用した計算機は文献 [10] と同じ CPU:Pentium4 2.8GHz である.

表 3: 2 変数論理関数の場合に対する実行時間 [sec]

$n = 40,\ p = 0.80000$

$\omega = 12,\ N(\omega) = 32$

グラフ	w_{max}	MC	VWMC
$G_1(1)$	36	0.000018	0.00010
$G_1(2)$	32	0.000019	0.00010
$G_1(3)$	28	0.000020	0.00011
$G_1(4)$	32	0.000020	0.00010

表 4: 3 変数論理回路の場合に対する実行時間 [sec]

$n = 208,\ p = 0.946860$

$\omega = 56,\ N(\omega) = 524,288$

グラフ	w_{max}	MC	VWMC
$G_2(1)$	216	0.00075	7.52
$G_2(2)$	208	0.00075	7.55
$G_2(3)$	200	0.00077	7.54
$G_2(4)$	192	0.00072	7.50
$G_2(5)$	200	0.00072	7.58
$G_2(6)$	208	0.00074	7.61

4 考察

ランダムグラフに関しては, アルゴリズム VWMC, VWMC$_\omega$ の実行時間は MC のそれと比較して, $N(\omega)$ が非常に大きい場合を除いて 2.5 倍程度以内に実行時間が抑えられている. このようなグラフに関して ω が未知の場合は, MC+VWMC$_\omega$ で解を求めるよりも VWMC を用いて解を求める方が, 全体的に見て実行時間が速いことが表より分かる. しかし枝密度が大きく $N(\omega)$ が非常に大きくなると, VWMC, VWMC$_\omega$ では, MC と比較してやはり実行時間は大幅に増大する場合が多い.

アルゴリズム VWMC$_\omega$+ は, $N(\omega)$ が小さい場合にはあまり効率的ではなく, VWMC$_\omega$ と比較して実行時間が増大してしまう場合が多いが, 実行時間の増大は高々1.2 倍程度に抑えられており, 実行時間の増大を極力抑える, という点では成功していると言える. 一方, 枝密度が大きく $N(\omega)$ が大きいグラフでは, クリーク重みの上界による分枝限定が効果的に働き, 実行時間の大幅な短縮ができている場合が多い. しかし, $N(\omega)$ が大きいグラフでも枝密度が小さいグラフの中にはクリーク重みの上界による分枝限定の効果があまり見られないものがいくつかあるが, これは NUMBERING-ARRANGING という手続きは枝密度が大きい方が番号の最大値をより小さく抑えるという性質があるので, 枝密度が小さい方ではクリーク重みの上界が大きくなってしまうためであると考えられる.

DIMACS グラフは枝の張り方に偏りがあるグラフであるが, 実行時間の比較を行うとランダムグラフの時と同様の傾向が見られる. 特に注目すべきは, 枝密度が大きくランダムグラフより $N(\omega)$ が非常に大きいグラフで, このようなグラフでは, MC と比べて VWMC や VWMC$_\omega$ では大幅に実行時間が増大する. このようなグラフに対して, VWMC$_\omega$+ ではクリーク重みの上界による分枝限定が効果的に行われ, 大幅な実行時間の削減に成功している. 特に MANN_a27 というグラフは VWMC や VWMC$_\omega$ では, 24 時間以上かかっても解を抽出することができなかったが, VWMC$_\omega$+ を用いて 6 時間程度で解を抽出することができた.

量子論理回路を還元して生成した無向グラフについては, MC と VWMC の実行時間の比較のみを示した. 表 3 のグラフは $N(\omega)$ が小さいが, VWMC の実行時間が MC の実行時間の 5 倍程度となっている. これは MC の探索過程で最初に発見される極大クリークが最大クリークとなり, 分枝限定が効果的に働きそ

れ以降の探索をしなくてすむのに対して, VWMC では複数個抽出しなければならないためであると考えられる. 表 4 のグラフについては VWMC の実行時間が MC の実行時間の 10,000 倍程度となっている. これは $N(\omega)$ が非常に大きいグラフであるためと考えられる. ただし, この場合には $\mathrm{VWMC}_\omega+$ の大きい有効性が発揮される [10].

5 まとめ

本稿では, 節点重み最大クリークを抽出するアルゴリズムを提唱し, 実働化して実行時間を比較することにより, アルゴリズムの評価を行った. 実験結果から, $N(\omega)$ が非常に大きい場合を除けば, アルゴリズム VWMC, VWMC_ω を用いることで, 最大クリーク抽出アルゴリズムの数倍内程度の実行時間で, 節点重み最大クリークが抽出できることを確認できた. また, VWMC, VWMC_ω では MC と比べて実行時間が大幅に増加してしまうグラフについては, アルゴリズム $\mathrm{VWMC}_\omega+$ が有効であることを実験的に確認した. 更に, 本アルゴリズムを量子論理回路深さ最小化問題に適用した [10].

量子論理回路深さ最小化問題をクリーク問題に還元する場合, 枝密度が非常に大きくなる. 枝密度が大きいグラフに対して効率的な重みなし最大クリーク抽出アルゴリズム MCQd^+[2] 等は既に得ているので, 今後, これらの基本アルゴリズムも考慮し, 枝密度が大きい場合の効率化を目指す.

なお, 本研究は, 電気通信大学研究・教育活性化支援システムの支援を受けている.

参考文献

[1] 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, vol.J79-D-1, no.1, pp.221-228 (1996).

[2] 関 友和, 富田悦次: "分枝限定法を用いた最大クリーク抽出アルゴリズムの効率化," 電子情報通信学会コンピューテーション研究会, COMP 2001-50, pp.101-108 (2001).

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

[4] E. Tomita and T. Seki: "An efficient branch-and-bound algorithm for finding a maximum clique," Proc. Discrete Mathematics and Theoretical Computer Science 2003, LNCS (2003, to appear).

[5] 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).

[6] 堀田一弘, 富田悦次, 関 友和, 高橋治久: "最大クリーク抽出を用いた画像からの対象検出, " 情報処理学会数理モデル化と問題解決研究会, 2002-MPS-42, pp. 49-56 (2002).

[7] S. Kobayashi, T. Kondo, K. Okuda, and E. Tomita: "Extracting globally structure free sequences by local structure freeness," Proc. International Meeting on DNA Based Computers (2003, to appear).

[8] 鈴木純一, 富田悦次, 関 友和: "枝重み最大クリーク抽出アルゴリズム," 2002 年度 夏の LA シンポジウム, no. 17 (2002).

[9] 名久井行秀, 西野哲朗: "量子論理回路深さ最小化問題のクリーク問題への還元," 情報処理学会数理モデル化と問題解決研究会, 2002-MPS-42, pp. 57-60 (2002).

[10] 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," 2002 年度 冬の LA シンポジウム, no. 9.1-9.7 (2003).