最大クリーク発見問題の省メモリアルゴリズム

A Space-Efficient Algorithm for the Maximum Clique Problem

小畠 教寛 [1*]　喜田 拓也 [2]
Norihiro Obata[1]　Takuya Kida[2]

[1] 北海道大学
[1] Hokkaido University
[2] 北海学園大学
[2] Hokkai-Gakuen University

Abstract: In this paper, we consider the maximum clique problem (MCP) for undirected graphs. Based on the MCS algorithm, proposed by Tomita *et al.* in 2010, which is the state-of-the-art algorithm for MCP, we propose a space-efficient algorithm, called MCS-SE (*Space-efficient MCS*), for MCP. For an input graph with m edges, n vertices, and with the maximum clique size $\omega \leq n$, the proposed algorithm runs in $O(n+m)$ space and $O(n^4 \log n)$ time per iteration, while the original MCS uses $O(n\omega+m)$ space and $O(n^3 \log n)$ time per iteration. On the computational experiments on DIMACS benchmark datasets, we observed that MCS-SE used less memory than MCS in some graph data sets with large clique sizes.

1　はじめに

最大クリーク発見問題 (the maximum clique problem) は、入力としてグラフ $G = (V, E)$ が与えられたとき、サイズが最大のクリーク Q を見つける問題である。ここに、G の頂点集合 $Q \subseteq V$ が、G の**クリーク** (clique) であるとは、Q に含まれる任意の二つの頂点が隣接していることをいう: $Q \in \mathrm{CLIQUE}(G) \iff \forall u, v \in Q,\ (u, v) \in E$。クリークのサイズは、その頂点数 $|Q|$ と定める。

最大クリーク問題は、重要なグラフの最適化問題であり、経済分野や生命科学分野でのデータ解析に広く用いられている。例えば、経済では、株式市場におけるマーケットグラフ (market graph) ので最大クリークを見つけることで、 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見つけることに使われる。 [1]

2　本研究の目的

2010 年に Tomita *et al.* [2] は、彩色数による探索木の枝刈りを用いた高速な最大クリーク発見アルゴリズムである MCS アルゴリズムを提案している。入力として最大クリークサイズ ω をもつ、m 辺と n 頂点の入力グラフが与えられたとき、MCS アルゴリズムは $O(n\omega + m)$ 領域と 1 回の繰り返し当たり $O(n^3 \log n)$ 時間で、最大クリークを発見する。

MCS は実用的にも極めて高速なアルゴリズムであるが、大きなクリークサイズ ω をもつ入力グラフに対しては、その使用領域量 $O(n\omega+m)$ は無視できないものになる。

そこで、本研究では、MCS アルゴリズムに基づい、その枝刈り等の効果をできるだけ損わずに、そのメモリ量削減の改良を行うことを目指す。主結果として、MCS アルゴリズムが用いている候補集合と彩色情報の領域効率の良い表現を導入し、メモリ効率の良い改良版アルゴリズムである MCS-SE（space-efficient MCS）を提案する。

3　提案手法 MCS-SE

元の MCS アルゴリズムでは、各繰り返しで、候補集合として現在の解候補のクリークの simplicial vertices の集合を保持する。

本研究では、探索木の根から現在の繰り返しまでのパス上で、候補集合が集合の包含関係の下降列になるという性質が成立することに着目し、候補集合の差分表現を用いることで空間使用量を削減しながら探索を行う (図 1) 。

さらに、親繰り返しの彩色情報を保持せず、親への

*連絡先：北海道大学 大学院情報科学院 情報理工学コース
〒 060-0814 札幌市北区北 14 条西 9 丁目, 小畠教寛、有村博紀.
E-mail: {obata,arim}@ist.hokudai.ac.jp

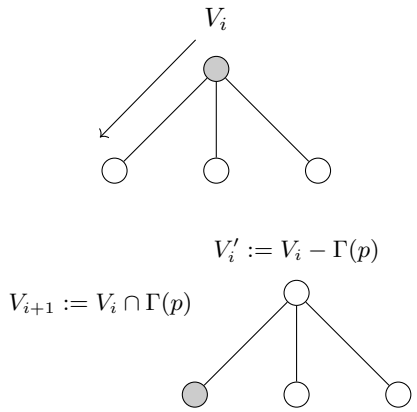

図 1: 差分表現を利用した探索の再帰動作

バックトラックする際に、彩色情報を動的に復元する。この復元は、MCS の効率的な枝刈り戦略を損わないように、元の MCS と完全に同じではないが、できるだけ同じになるように行う。

提案アルゴリズムの擬似コードを，Algorithm1 に示す．ここで、EXTENDED INITIAL SORT-NUMBER は、頂点の次数に基づいて頂点集合 V を整列する副手続きである．NUMBER-SORT-R は、これらの副手続きの詳細については，論文 [2] を参照されたい。

Algorithm 1 提案アルゴリズム

```
 1: procedure MCS-SE (G = (V, E))
 2:     Q := ∅, Q_max := ∅
 3:     EXTENDED-INITIAL-SORT-NUMBER(
    V, Q_max, No )
 4:     Apply approximation algorithm
 5:     while |Q| + |V| > |Q_max| do
 6:         p := the last vertex in V
 7:         if |Q| + No[p] > |Q_max| then
 8:             EXPAND-SE(V ∩ Γ(p))
 9:         else return Q_max
10:         end if
11:     end while
12:     return Q_max
13: end procedure

14: procedure EXPAND-SE ( V_s )
15:     V_e := ∅   ▷ 一度クリークに追加された点を保存する
16:     while V_s ≠ ∅ do
17:         NUMBER-SORT-R(V_s, R, No)
18:         p := R の末尾の頂点
19:         c := No の末尾の彩色番号
20:         if |Q| + c > |Q_max| then
21:             Q := Q ∪ {p}
22:             V_d := V_s − Γ(p)        ▷ 元の情報を保存する
23:             V_s := V_s ∩ Γ(p)   ▷ 隣接頂点以外を削除する
24:             if V_s ≠ ∅ then
25:                 EXPAND-SE (V_s)
26:             else if Q > Q_max then
27:                 Q_max := Q
28:             end if
29:             V_s := V_s ∪ V_d      ▷ 削除した頂点を復元する
30:             V_s の頂点を初期順序に整合するよう整列する
31:             V_s := V_s − {p}, V_e := V_e ∪ {p}
32:         else
33:             V_s := V_s ∪ V_e
34:             V_s の頂点を初期順序に整合するよう整列する
35:             return
36:         end if
37:     end while
38:     V_s := V_s ∪ V_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
39: end procedure
```

4 主結果

最大クリークサイズ ω をもつような、m 辺と n 頂点の入力グラフが与えられたとき、提案アルゴリズムは $O(n+m)$ 領域と 1 回の繰り返し当たり $O(n^4 \log n)$ 時間で、G 中の最大クリークを見つける。これは、MCS が $O(n+m)$ 領域と 1 回の繰り返し当たり $O(n^4 \log n)$ 時間を要するのに対して、メモリを $O(1/\omega)$ に削減している。一方、時間計算量に関しては $O(n)$ 倍程度に増加している。

5 実験結果

実験環境は、PC (CPU Intel Core i5-7360U, 2.30GHz, 8GB memory, MacOSX 10.14.6) 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。初期解を見つけるための近似アルゴリズムには、K-Opt Local Search [3] を使用した。実行時間と、最大メモリ使用料、分枝数の結果を表 1 に示す。

実験のデータセットには、DIMACS ベンチマークセットのグラフ[1]と、それらを組み合わせて作成したグラフ、ランダム生成したグラフの 3 通りを用いた。具体的には、次のデータセットを用いた。

- DIMACS ベンチマークセットのグラフをそのまま使用した実データセットは、brock200-{1, 2, 3, 4} と、hamming{8-2, 10-2}、MANN{a-27, a-45} である。
- クリークサイズを大きくすることと、初期解のための近似アルゴリズムでは解が見つかりにくくするために、DIMACS ベンチマークセットの二個以上のグラフを合体させて、それら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は、union と union2 である。ここに、union は、DIMACS の c-fat200-1 と MANNa-45 の二つのグラフを組み合わせ、union2 は、c-fat200-1 と、MANNa-45、hamming10-2 の三つのグラフを組

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

表 1: 実験結果のまとめ

入力グラフデータ					MCS			MCS-SE		
名前	サイズ	辺密度	解	近似解	実行時間	最大メモリ使用量	分枝数	実行時間	最大メモリ使用量	分枝数
brock200-1	200	0.74	21	21	1.48	876	134082	3.83	884	130159
brock200-2	200	0.49	12	11	0.03	880	1852	0.06	908	1819
brock200-3	200	0.60	15	14	0.11	888	8148	0.26	908	7897
brock200-4	200	0.65	17	16	0.24	880	25620	0.63	872	24893
hamming8-2	256	0.96	128	128	0.53	844	0	0.54	832	0
hamming10-2	1024	0.99	512	512	26.80	928	0	27.21	928	0
MANN-a27	378	0.99	126	126	3.33	856	8843	3.99	876	8843
MANN-a45	1035	0.99	345	344	242.98	1628	223056	347.05	936	223056
union	1235	0.84	347	345	519.86	2272	240669	759.34	1440	240669
union2	2259	0.70	522	355	402.88	5536	29126	463.39	4584	28873
2000-1500	2000	0.99	1330	1330	1110.25	1212	42262	1135.63	996	42262
3000-2000	3000	0.99	2050	2050	4786.92	1352	103413	5114.48	1076	103414

み合わせている。

- DIMACS ベンチマークセットのグラフをそのまま使用した実データセット 2000-1500, 3000-2000 である。ここに、データセット a-b は、グラフサイズ a の完全グラフから b 本の辺を削除して作成されたクリークサイズの大きなグラフである。

DIMACS ベンチマークデータセット等を用いた計算機実験では、大きなクリークサイズ ω をもつグラフに対しては、メモリ量が期待通り削減されることを確認した。繰り返し 1 回当たりの計算時間に関しては、理論的見積もりでは、提案手法は MCS に対して、最悪時に $O(n)$ 倍の計算時間の増加が予測された。しかし、実際のデータセット上の計測では、提案手法の繰り返し 1 回当たりの計算時間は、MCS の計算時間の高々2〜4 倍程度に抑えられることを観察した。

6 おわりに

最大クリーク発見問題に関して、富田による MCS アルゴリズムを考察し、そのメモリ量削減の改良を行った。

謝辞

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からは、お忙しい中、MCS などの実装のソースコードを送っていただきましたことに、心より感謝致します。国立情報学研究所の栗田和宏先生には最大クリーク問題について、北海道大学の有村博紀先生には本研究発表に関して、また、加井丈志さんと瀧澤涼介さんをはじめとする情報知識ネットワーク研究室のメンバーにはさまざまなアドバイスと協力をいただきました。ここに感謝いたします。

参考文献

[1] V. Boginski, S. Butenko, and P.M. Pardalos. Mining market data: A network approach, Part Special Issue: Operations Research and Data Mining, Computers & Operations Research, Vol.33, No.11, pp.317–3184, 2006.
doi:10.1016/j.cor.2005.01.027

[2] E. Tomita, Y. Sutani, T. Higashi, S. Takahashi, M. Wakatsuki, A Simple and Faster Branch-and-Bound Algorithm for Finding a Maximum Clique. In WALCOM 2010. LNCS 5942. Springer, 2010.
doi:10.1007/978-3-642-11440-3_18

[3] K. Katayama, A. Hamamoto, and H. Narihisa, An effective local search for the maximum clique problem. Information Processing Letters, Vol.95, No.5, pp.503–511, 2005.
doi:10.1016/j.ipl.2005.05.010