情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2010-AL-128 No.10
2010/1/26

単純でより高速な最大クリーク抽出アルゴリズム

富田 悦次[†1,†2] 須谷 洋一[†1] 東 貴紀[†1]
高橋 真也[†1] 若月 光夫[†1]

最大クリーク抽出アルゴリズム MCR (Tomita *et al.,* J. Global Optim., 37, 95–111, 2007) は，数多くの問題グラフに対して他よりも非常に高速であることを実験的に確認していた．本稿では，その改良アルゴリズム MCS が MCR や他のアルゴリズムよりも全面的に顕著に高速であることを示す．MCS は特定のグラフに対象を限定したアルゴリズムではないが，枝密度の高い難しいグラフに対しては特により高速である．MCR では 100 日以上かかっても解けない幾つかの超高密度ランダムグラフに対し，MCS は数 10 秒で解を得ることに成功している．

A Simple and Faster Algorithm for Finding a Maximum Clique

Etsuji Tomita ,[†1,†2] Yoichi Sutani,[†1]
Takanori Higashi,[†1] Shinya Takahashi [†1]
and Mitsuo Wakatsuki[†1]

A maximum-clique-finding algorithm MCR (Tomita *et al.*, J. Global Optim., 37, 95–111, 2007) was the fastest among all the existing algorithms in computational experiments for a large number of tested graphs. In this note, it is shown by extensive computational experiments that MCS is remarkably faster than MCR and other algorithms. In particular, it is very much faster than MCR for difficult graphs of very high density, even though MCS is not designed for any particular type of graphs. MCS can find a maximum clique in less than 100 seconds for some extremely dense random graphs while MCR requires more than 100 days for the same graphs.

†1 電気通信大学, The University of Electro-Communications
†2 中央大学研究開発機構, Research and Development Initiative, Chuo University

1. Introduction

A *clique* is a subgraph in which all pairs of vertices are adjacent to each other. Many practical problems can be formulated as maximum clique problems (e.g., see Refs.[2],[3],[4],[1],[6], and others). Therefore, it is required to develop exact maximum-clique-finding algorithms that run very fast in practice.

We developed a *simple* branch-and-bound algorithm that is referred to as MCR [12]; that was successful in reducing the search space with low overhead. It was shown in computational experiments that MCR clearly outperformed other existing algorithms in finding a maximum clique.

In this note, we propose a new approximate coloring that can play a crucial role in the branch-and-bound algorithm. Subsequently, we introduce a new adjunct ordered set of vertices for approximate coloring. Following this ordered set of vertices, we present a new technique for reconstructing the adjacency matrix of a graph. The algorithm that is obtained by introducing these new techniques in MCR is named MCS [13].

While MCS inherits the simplicity of MCR to a large extent, MCS is much more successful in reducing the search space quite efficiently. Consequently, extensive computational experiments have shown that MCS is remarkably faster than MCR and other algorithms. MCS is faster than other algorithms by an order of magnitude for several graphs. In particular, it is faster than MCR for difficult graphs with very high density, even though MCS is not designed for any particular type of graphs. MCR is only briefly described in Sect. 2 due to the page limitation, and the reader is advised to refer to Ref.[12] for further details.

2. Maximum clique algorithm MCR

2.1 Branch-and-bound algorithm

The basic branch-and-bound algorithm MCR [12] begins with a small clique and continues finding larger and larger cliques in a depth-first way until one is found that can be verified to have the maximum size.

2.2 Greedy approximate coloring

In order to prune unnecessary searching, we used *greedy approximate coloring* of the vertices in MCR. That is, each $p \in R$ is *sequentially* assigned a minimum possible

positive integer value $No[p]$, called the *Number* or *Color* of p, such that $No[p] \neq No[r]$ if $(p, r) \in E$. Consequently, we have that $\omega(R) \leq \mathrm{M}ax\{No[p] | p \in R\}$.
Hence, if $|Q| + \mathrm{M}ax\{No[p] | p \in R\} \leq |Q_{max}|$ holds, we need not continue the search for R.

After $Numbers$ ($Colors$) are assigned to all vertices in R, we sort the vertices in ascending order with respect to their $Numbers$. We refer to the numbering and sorting procedure as NUMBER-SORT [12]. In each step, select a vertex p in R, beginning from the last (right) vertex and ending at the first (left) vertex.

2.3 Initial sorting and initial numbering

In the first stage of algorithm MCQ [10], which is a predecessor of MCR, vertices are sorted in descending order with respect to their degrees and are assigned simple initial $Numbers$. At the beginning of MCR, vertices are sorted and assigned initial $Numbers$ in a similar but more sophisticated manner.

3. New algorithm

3.1 New approximate coloring

Approximate coloring is generally quite effectively used in branch-and-bound algorithms for finding a maximum clique. In this note, we propose a new approximate coloring following greedy approximate coloring in Sect. 2.2.

Because of the *bounding condition* mentioned in Sect. 2.2, if $No[r] \leq |Q_{max}| - |Q|$, then it is not necessary to search from vertex r. The number of vertices to be searched can be reduced if the *Number* $No[p]$ of vertex p for which $No[p] > |Q_{max}| - |Q|$ can be made less than or equal to $|Q_{max}| - |Q|$. When we encounter such vertex p with $No[p] > |Q_{max}| - |Q|$, we attempt to change it's $Number$ in the following manner. Let No_p denote the original value of $No[p]$.

[Re-NUMBER p]
0) Let $No_{th} := |Q_{max}| - |Q|$. ($No_{th}$ stands for $No_{threshold}$.)
1) Attempt to find a vertex q in $\Gamma(p)$ such that $No[q] = k_1 \leq No_{th}$, with $|C_{k_1}| = 1$.
2) If such q is found, then attempt to find $Number$ k_2 such that no vertex in $\Gamma(q)$ has $Number$ k_2.
3) If such number k_2 is found, then Re-Number q and p so that $No[q] = k_2$ and $No[p] = k_1$.
(If no vertex q with $Number$ k_2 is found, nothing is done.)

When the vertex q with $Number$ k_2 is found, $No[p]$ is changed from No_p to k_1 ($\leq No_{th}$); thus, *it is no longer necessary to search from p*.

The conventional greedy approximate coloring in Sect. 2.2 followed by the above Re-NUMBER constitutes our new approximate coloring. The new approximate coloring followed by sorting of vertices in ascending order with respect to their $Numbers$ is named Re-NUMBER-SORT. We employ the new procedure Re-NUMBER-SORT instead of the procedure NUMBER-SORT used in MCR in order to make more effective use of the bounding condition.

3.2 Adjunct ordered set of vertices for approximate coloring

The application of *Re-NUMBER*, which is described in Sect. 3.1, changes the $Numbers$ of the vertices, thereby making the vertices disordered with respect to their degrees. We can reduce the search space by *sorting* vertices in R *in descending order with respect to their degrees*. However, the sorting of vertices is a computational burden and reduces the overall running time only for *dense* graphs [9]. So, in addition to the ordered set R of vertices, we simply introduce a particular *adjunct ordered set* V_a of vertices that preserves the order of the vertices sorted in descending order with respect to their degrees *in the first stage*. We apply the procedure Re-NUMBER-SORT to the vertices in V_a, begining from the first (left) vertex and ending at the last (right) vertex. Thus, we can avoid the undesirable effect of Re-NUMBER.

3.3 Reconstruction of the adjacency matrix

Each graph is stored as an adjacency matrix in the computer memory. Sequential numbering in Re-NUMBER-SORT is carried out according to the initial order of vertices in the adjunct ordered set V_a, as described in Sect. 3.2. Taking this into account, we *rename* the vertices of the graph and *reconstruct* the adjacency matrix so that the vertices are *consecutively ordered* in a manner identical to *the initial order of vertices* obtained at the beginning of MCR. The above-mentioned reconstruction of the adjacency matrix results in a more effective use of the cache memory since it facilitates the use of localized memory.

Vol.2010-AL-128 No.10
2010/1/26

3.4 Algorithm MCS

The new algorithm obtained by introducing the techniques described in Sects. 3.1–3.3 in MCR is named MCS.

4. Computational experiments

We carried out computational experiments in order to demonstrate the overall superiority of MCS over MCR. Both MCR and MCS were implemented in exactly the same manner in the programming language C. The computer used, which had a Linux operating system, is described in Appendix. We also executed the DIMACS benchmark program dfmax [5], as a standard. The computation times for other algorithms are calibrated using the ratios as shown in Appendix.

It is confirmed that we are successful in *further reducing the search space* quite efficiently with *low overhead* and hence we have the following results [13].

4.1 Results for random graphs

Random graphs are generated for each pair of n (number of vertices) and p (edge probability) listed in Table 1. The average CPU times [sec] required to solve these graphs when using dfmax, MCR, and MCS are listed in Table 1.

The calibrated CPU time for New [7] is also listed for reference. The boldface entries indicate the fastest time in the row. In Table 1, it is observed that MCS is faster than MCR for all graphs. MCS is particularly faster than MCR for dense graphs. MCS is the fastest for all the random graphs listed in Table 1. For the graphs with $p > 0.99$ in Table 1, MCS is faster than MCR by a factor of greater than 100,000 (10^5 seconds $\simeq$ 1.16 days, and 10^7 seconds $\simeq$ 116 days).

4.2 Results for DIMACS benchmark graphs

Table 2 lists the CPU times required by MCS and other algorithms to solve the DIMACS benchmark graphs [5]. The boldface entries indicate the fastest time among the times obtained within the time limits in the row. From this table, it is confirmed that MCS is almost always faster than MCR and the other algorithm in Table 2.

MCS is almost always considerably faster than COCR, MIPO, SQUEEZE, Target, and ILOG [8] (see Table 4 in Ref.[12]).

Table 1. CPU time [sec] for random graphs

Graph			dfmax	MCR	MCS	New
n	p	ω	Ref.[5]	Ref.[12]	Ref.[13]	Ref.[7]
	0.9	29-32	3.67	0.038	○ **0.013**	0.663
100	0.95	39-48	23.736	0.011	○ **0.003**	0.196
	0.98	56-68	26.5401	0.0012	**0.0009**	
	0.8	23	6.88	0.55	○ **0.23**	
150	0.9	36-39	1058.96	5.26	○ **1.00**	
	0.95	50-59	37,436.79	3.94	★ **0.35**	
	0.98	73-85	$> 10^5$	0.243	★○ **0.006**	
	0.8	24-27	192.7	12.3	○ **4.5**	147.3
200	0.9	40-44	$> 10^5$	647	○ 74	
	0.95	58-66	$> 10^5$	1,272	★○ **59**	
	0.98	90-103	$> 10^5$	30.9	★★ **0.2**	
	0.7	19-21	26,236	23	· **12**	121
	0.8	28-29	$> 10^5$	1,264	○ **394**	
300	0.9	49		1,475,387	★○ **62,607**	
	0.98	120		284,534	★★ **2,623**	
	0.99	154		732.49	★★★ **0.23**	
400	0.99	188		$> 1.8 \times 10^6$	★★★ **1,030**	
	0.6	17	242	63	· **40**	183
500	0.7	22-23	24,998	3,268	○ **1,539**	
	0.994	263	$> 1.5 \times 10^7$	$> 10^7$	★★★★★ **39**	
	0.4	12	33.3	16.1	**13.2**	23.2
	0.5	15	1,107	395	**290**	
1,000	0.6	19-20	106,776	24,986	· **15,317**	
	0.66	23		555,089	○ **275,964**	
	0.998	618		$> 10^7$	★★★★★ **46**	
1,500	0.999	997		$> 1.8 \times 10^6$	★★★★★ **13**	
2,000	0.9995	1,453		$> 10^7$	★★★★★ **61**	
10,000	0.1	7-8	137	100	· **60**	
	0.2	10	9,417	8,055	· **4,389**	

Entries marked ★★★★★, ★★★,★★,★○, ★, ○, and · are respectively at least 100,000, 1,000, 100, 20, 10, 2, and 1.5 times faster than any of the others in the same row.

5. Concluding remarks

Our new algorithm, MCS, retains the *simplicity* of our earlier algorithms and it runs remarkably faster than MCR and the other existing algorithms. Some theoretical analysis of maximum-clique-finding algorithms is on the way based upon Ref.[11].

Table 2. CPU time [sec] for DIMACS benchmark graphs

Graphs				dfmax	MCR	MCS	New
Name	n	*density*	ω	Ref.[5]	Ref.[12]	Ref.[13]	Ref.[7]
brock400_1	400	0.75	27	22,051	1,771	○ **693**	
brock400_2	400	0.75	29	13,519	726	○ **297**	
brock400_3	400	0.75	31	14,795	1,200	○ **468**	
brock400_4	400	0.75	33	10,633	639	○ **248**	
MANN_a27	378	0.990	126	$> 10^5$	2.5	○ **0.8**	$> 2,232$
MANN_a45	1,035	0.996	345	$> 10^5$	3,090	★ **281**	
p_hat300-3	300	0.744	36	779.7	10.8	○ **2.5**	
p_hat500-2	500	0.505	36	132.9	3.1	○ **0.7**	95.7
p_hat500-3	500	0.752	50	$> 10^5$	1,788	★ **150**	
p_hat700-2	700	0.498	44	5,299.9	44.4	● **5.6**	
p_hat700-3	700	0.748	62	$> 10^5$	68,187	★○ **2,392**	
p_hat1000-2	1,000	0.489	46	$> 10^5$	2,434	★ **221**	
p_hat1500-2	1,500	0.506	65	$> 10^5$	722,733	★○ **16,512**	
san200_0.9_1	200	0.900	70	$> 10^5$	1.20	0.22	○ **0.06**
san200_0.9_2	200	0.900	60	$> 10^5$	4.2	○ **0.4**	1.0
san400_0.7_1	400	0.700	40	$> 10^5$	1.76	○ **0.54**	$> 2,232$
san400_0.7_2	400	0.700	30	$> 10^5$	0.33	○ **0.13**	112.97
san400_0.7_3	400	0.700	22	$> 10^5$	3.6	○ **1.4**	
san400_0.9_1	400	0.900	100	$> 10^5$	3.4	★○ **0.1**	
san1000	1,000	0.502	15	$> 10^5$	4.8	2.1	★○ **0.1**
sanr200_0.9	200	0.898	42	86,954	289	● **41**	
sanr400_0.7	400	0.700	21	2,426	379	○ **181**	
gen200_p0.9_44	200	0.900	44	48,262	5.39	★ **0.47**	
gen200_p0.9_55	200	0.900	55	9,281.0	15.0	★ **1.2**	
gen400_p0.9_55	400	0.900	55		5,846,951	★★ **58,431**	
gen400_p0.9_65	400	0.900	65		$> 10^7$	★● **151,597**	
gen400_p0.9_75	400	0.900	75		$> 10^7$	★○ **294,175**	
C250.9	250	0.899	44	$> 10^5$	44,214	★ **3,257**	

Entries marked ★★,★●, ★○, ★, ●, ○, and · are respectively at least 100, 50, 20, 10, 5, 2, 1.5 times faster than any of the others within the time limits in the same row.

Acknowledgements We thank E. Harley for his useful detailed comments. This research was supported in part by Grants-in-Aid for Scientific Research from MEXT, Japan, and a Special Grant for SCOPE Project from MIC, Japan.

References

1) Bahadur, D. K.C., Tomita, E., Suzuki, J., Horimoto, K., Akutsu, T.: Protein threading with profiles and distance constraints using clique based algorithms, J. Bioinformatics and Computational Biology, 4, 19–42 (2006)
2) Bomze, I. M., Budinich, M., Pardalos, P. M., Pelillo M.: The Maximum Clique Problem, In: Du, D.-Z., Pardalos, P.M. (Eds.), Handbook of Combinatorial Optimization, Supplement vol. A, Kluwer Academic Publishers, 1–74 (1999)
3) Butenko, S. , Wilhelm, W.E.: Clique-detection models in computational biochemistry and genomics - Invited Review - , European J. Operational Research, 173, 1–17 (2006)
4) Hotta, K., Tomita, E., Takahashi, H.: A view-invariant human face detection method based on maximum cliques. Trans. IPSJ, 44, SIG14(TOM9), 57–70 (2003)
5) Johnson, D. S., Trick, M. A. (Eds.): Cliques, Coloring, and Satisfiability, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol.26, American Math. Soc. (1996)
6) Matsunaga, T., Yonemori, C., Tomita, E., Muramatsu, M.: Clique-based data mining for related genes in a biomedical database, BMC Bioinformatics, 10 (2009)
7) Östergård, P. R. J.: A fast algorithm for the maximum clique problem, Discrete Applied Math., 120, 197–207 (2002)
8) Régin, J.-C.: Using constraint programming to solve the maximum clique problem, Principles and Practice of Constraint Programming, LNCS 2833, 634-648 (2003)
9) Shindo, M., Tomita, E., Maruyama, Y.: An efficient algorithm for finding a maximum clique, Technical Report of IEC, CAS86-5, 33–40 (1986)
10) Tomita, E., Seki, T.: An efficient branch-and-bound algorithm for finding a maximum clique, DMTCS 2003, LNCS 2731, 278–289 (2003)
11) Tomita, E., Tanaka, A. Takahashi, H.: The worst-case time complexity for generating all maximal cliques and computational experiments (An invited paper in the Special Issue on COCOON 2004), Theoret. Comput. Sci., 363, 28–42 (2006)
12) Tomita, E., Kameda, T.: An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments, J. Global Optim., 37, 95–111 (2007), J. Global Optim., 44, 311 (2009)
13) Tomita, E., Sutani, Y., Higashi, T., Takahashi, Wakatsuki, M.: A simple and faster banch-and-bound algorithm for finding a maximum clique, WALCOM 2010, LNCS 5942, 191–203 (2010)

Appendix: Clique Benchmark Results

Type of Machine: Pentium 4 3.6 GHz, *Compiler and flags used:* gcc -O2.
Our user time (T_1) *for DIMACS benchmark instances:* `r100.5`, `r200.5`, `r300.5`, `r400.5`, and `r500.5` are 2.13×10^{-3}, 6.35×10^{-2}, 0.562, 3.48, and 13.3 seconds, respectively. From Östergård's[7] user time (T_2) for the same instances, we obtained the average value of T_2/T_1 as 4.48[12].