2007－MPS－65 (1)
2007／6／25

A More Efficient Algorithm for Finding a Maximum Clique with an Improved Approximate Coloring

Yoichi Sutani, Takanori Higashi, and Etsuji Tomita
Department of Information and Communication Engineering
The University of Electro-Communications
Chofugaoka 1-5-1, Chofu, Tokyo 182-8585, JAPAN

Abstract *We propose a new, practical, and exact algorithm for finding a maximum clique in a graph. The algorithm is a considerably improved version of the MCR algorithm, which is a very efficient branch-and-bound algorithm. The improvement is achieved by devising a sophisticated procedure for approximate coloring as well as by introducing a novel strategy for ordering vertices in the branch-and-bound procedure. It is experimentally demonstrated that the new algorithm is remarkably faster than MCR and other existing algorithms for most benchmark instances.*

1 Introduction

Many important problems can be formulated as maximum clique problems. Therefore, efficient algorithms are strongly required for finding a maximum clique in a graph.

The authors have devised an efficient branch-and-bound algorithm, MCR[11], for finding a maximum clique. In this study, we propose a new algorithm that is more efficient than MCR for finding a maximum clique. The new algorithm is obtained by significantly improving the approximate coloring procedure, which is essentially important in the branch-and-bound method and the ordering of vertices. It is experimentally demonstrated that the new algorithm is remarkably faster than the MCR algorithm and other existing algorithms for most benchmark instances.

A preliminary version of this paper was presented in [8].

2 Definitions and Notation

We are concerned with a simple undirected graph $G = (V, E)$ with a finite set V of vertices and a finite set E of *unordered* pairs (v, w) of distinct vertices called edges. For a vertex $v \in V$, let $\Gamma(v)$ be the set of all vertices that are adjacent to v. We call $|\Gamma(v)|$ the degree of v. In general, for a set S, the number of elements is denoted by $|S|$. For a subset $R \subseteq V$ of vertices, $G(R) = (R, E \cap (R \times R))$ is an *induced* subgraph. An induced subgraph $G(Q)$ is said to be a *clique* if $(v, w) \in E$ for all $v, w \in Q \subseteq V$ with $v \neq w$. In this case, we may simply state that Q is a clique. In particular, a clique with the maximum size is called a *maximum clique*. The number of vertices of a maximum clique is denoted by $\omega(R)$.

3 Maximum Clique Algorithm MCR[11]

3.1 Main Body of MCR

The underlying algorithm MCR begins with a small clique, and grows it or backtracks until one clique is found that can be verified to have the maximum size. We control this growing and backtracking process by applying the branch-and-bound strategy. More precisely, we maintain global variables Q and Q_{max}, where Q consists of vertices of the current clique, and Q_{max} consists of vertices of the largest clique found thus far. Let $R \subseteq V$ consist of vertices (candidates) that may be added to Q. We begin the algorithm by letting $Q := \emptyset$, $Q_{max} := \emptyset$, and $R := V$ (the set of all vertices). We select a certain vertex p from R and add p to Q ($Q := Q \cup \{p\}$). Then, we compute $R_p := R \cap \Gamma(p)$ as the new set of candidate vertices. This procedure is applied recursively while $R_p \neq \emptyset$.

When $R_p = \emptyset$ is attained, Q constitutes a maximal clique. If Q is maximal and $|Q| > |Q_{max}|$ holds, Q_{max} is replaced by Q. We then backtrack by removing p from Q and R. We select a new vertex p from the resulting R and continue the same procedure until $R = \emptyset$ is at-

tained.

3.2 Approximate Coloring

In order to prune unnecessary searching, we employ the so-called *greedy approximate coloring* of vertices, that is, we sequentially assign for each $p \in R$ a minimum possible positive integer value $No[p]$ called the *Number* or *Color* of p so that $No[p] \neq No[r]$ if $(p, r) \in E$.

Consequently, $\omega(R) \leq \mathrm{Max}\{No[p] | p \in R\}$; hence, if $|Q| + \mathrm{Max}\{No[p] | p \in R\} \leq |Q_{max}|$ holds, then we can disregard all r in R.

After *Numbers* are assigned to all vertices in R, we sort these vertices in the ascending order with respect to their *Numbers*. We call this numbering and sorting procedure NUMBER-SORT (Figure 2 in [11]). We select a vertex p in R from the last (right) to the first (left).

Let $\mathrm{Max}\{No[r] | r \in R\} = maxno$, and $C_i = \{r \in R | No[r] = i\}, \ i = 1, 2, \ldots, maxno.$

3.3 Initial Sorting and Numbering

In the first stage of the algorithm MCQ[9], which is a predecessor of MCR, vertices are sorted in the descending order with respect to their degrees and they are assigned simple initial *Numbers* (Figure 3 in [11]). At the beginning of MCR, vertices are sorted and assigned initial *Numbers* in a similar but more extended way than in the MCQ algorithm. This concludes the underlying MCR algorithm (Figure 4 in [11]).

4 Improved Algorithm

4.1 Improved Approximate Coloring

When vertex r is selected, if $No[r] \leq |Q_{max}| - |Q|$ holds then r can be disregarded by the *bounding condition*, as mentioned in 3.2. Thus, for a vertex p such that $No[p] > |Q_{max}| - |Q|$, it is desirable that the *Number* $No[p]$ of p could be changed to be less than or equal to $|Q_{max}| - |Q| (\stackrel{def}{=} No_{th})$. When we encounter such a vertex p with $No[p] \stackrel{def}{=} No_p$, as mentioned above, we try to Re-Number p in the following manner [3].

[Re-NUMBER p]
1) Try 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 a q is found, then try to find a *Number* k_2 such that no vertex in $\Gamma(q)$ has the *Number* k_2.
3) If such a number k_2 is found, then Re-Number q and p so that $No[q] = k_2$ and $No[p] = k_1$.
(If we can find no vertex q with *Number* k_2 as above, no further action is performed.)

The above procedure for a vertex p with $No[p] (= No_p)$ is named Re-NUMBER(p, No_p). We apply Re-NUMBER to a vertex p only when $No[p] = maxno$, since Re-NUMBER is rather time-consuming.

4.2 Improvement in the Order of Vertices

As mentioned in [9] and [11], the ordering of vertices plays an important role in the algorithm. In particular, the procedure NUMBER-SORT strongly depends on the order of vertices, since its main ingredient is a *sequential* coloring. In our new algorithm, we sort the vertices just in the same way as in [11] at the first stage. However, the vertices are *disordered* in the succeeding stages owing to the application of Re-NUMBER. In order to avoid this difficulty, we employ another ordered set V_s of vertices that preserves the order of vertices appropriately sorted in the first stage. Such a technique was first introduced in [7]. Subsequently, we replace the previous procedure NUMBER-SORT in MCR[11] by the new procedure NUMBER-SORT-Re.

4.3 Improved algorithm MCR-Re

Summarizing the above all, we have a new improved algorithm named MCR-Re.

5 Computational Experiments

We have implemented the MCR-Re algorithm in the programming language C (Compiler and flags used: gcc -O2) and have performed computational experiments to evaluate it. The computer used is a Pentium4 3.60GHz CPU, and is operated on a Linux operating system. The computational times of other algorithms are ad-

Table 1: CPU time [sec] for random graphs

Graph			dfmax	MCR	MCR-Re	New	COCR
n	p	ω	[4]	[11]		[5]	[6]
100	0.7	14-16	0.018	0.0047	**0.0035**	0.0067	0.12
	0.8	19-21	0.14	0.014	**0.0078**	0.065	0.15
	0.9	29-32	3.67	0.038	○ **0.013**	0.66	0.20
	0.95	39-48	23.74	0.011	○ **0.0028**	0.20	
	0.98	56-68	26.54	0.0012	**0.00087**		
200	0.7	18-19	3.85	0.68	**0.41**	3.02	1.65
	0.8	24-27	192.68	12.29	**4.55**	147.29	8.69
	0.9	40-44	$> 10^5$	646.94	74.85		○ **36.79**
	0.95	58-66	$> 10^5$	1,272.31	★ **59.03**		
	0.98	90-103	$> 10^5$	30.90	★★ **0.21**		
300	0.5	12-13	0.36	0.15	**0.13**	0.20	1.13
	0.6	15-16	4.88	1.41	**1.01**	3.50	4.98
	0.7	19-21	144.11	22.80	**12.25**	121.02	
	0.8	28-29	26,235.96	1,264.10	○ **402.90**		
500	0.5	13-14	8.99	3.61	**2.89**	7.25	17.43
	0.6	17	242.29	62.57	**42.20**	183.28	
	0.7	22-23	24,998.42	3,267.89	○ **1,599.68**		
	0.75	26-27	$> 10^5$	49,933.40	○ **18,517.74**		
1,000	0.3	9-10	1.98	1.28	**1.19**	1.64	
	0.4	12	33.28	16.05	**13.80**	23.19	
	0.5	15	1,107.70	394.71	**302.83**		
	0.6	20	$> 10^5$	24,985.60	**15,316.85**		

Entries indicated by ★★, ★, and ○ represent those that are more than or equal to 100, 10, and 2 times faster than all the others in the same row, respectively.

justed according to the ratios as shown in the Appendix in [11].

5.1 Results for Random Graphs

For each pair of n (the number of vertices) and p (edge probability) in Table 1, random graphs are generated so that there exists an edge for each pair of vertices with probability p. The average CPU time required to solve these graphs by the MCR-Re algorithm and other algorithms are listed in Table 1. The averages are calculated for 10 random graphs for each pair of n and p. In particular, the computation of the average CPU time for $p \geq 0.95$ are for 100 graphs, since the variations among the graphs are very large. Exceptionally, each " $> 10^5$ " in dfmax is a CPU time for only one graph.

Table 1 shows that the MCR-Re algorithm is considerably faster than the MCR algorithm for dense graphs.

5.2 Results for DIMACS Benchmark Graphs

Table 2 lists the CPU times required by the MCR-Re algorithm and other algorithms to solve the DIMACS benchmark graphs[4]. This table also shows that MCR-Re is decidedly the fastest algorithm, with only few exceptions.

6 Concluding Remarks

Our new algorithm, the MCR-Re algorithm is relatively simple and runs remarkably faster than the other existing algorithms. Hence, it can be useful for solving more practical problems. Our technique can also be effectively applied for generating large maximal cliques[10].

Acknowledgment

We would like to express our gratitude to T. Akutsu and others for their useful discussions and for collaborative studies. This research was partially supported by Grants-in-Aid for Scientific Research Nos. 16300001 and 19500010 from the Ministry of Education, Culture, Sports, Science and Technology, Japan.

References

[1] E. Balas, S. Ceria, G. Cornuéjols, G. Pataki. Polyhedral methods for the maximum clique problem. In [4]: 11–28, 1996.

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

Graph Name	ω	dfmax [4]	MCR [11]	MCR-Re	New [5]	χ + DF [2]	COCR [6]	MIPO [1]
brock400_1	27	22,051	1,772	○ **714**		>10,640		
brock400_2	29	13,519	726	○ **307**		>10,640	>415	
brock400_3	31	14,795	1,200	○ **483**		>10,640		
brock400_4	33	10,633	639	○ **256**		>10,640	>415	
brock800_1	23	$> 10^5$	17,790	**9,799**		>10,640		
brock800_2	24	$> 10^5$	16,048	**8,762**		>10,640	>415	
brock800_3	25	91,031	10,853	**6,008**		>10,640		
brock800_4	26	78,737	7,539	**4,162**		>10,640	>415	
c-fat500-10	126	$> 10^5$	0.024	0.027	0.016	**0.015**		
hamming8-4	16	1.85	0.22	0.20	**0.19**	4.51	1.00	29.13
johnson16-2-4	8	0.75	0.14	0.14	0.060	5.88		★ **0.0017**
MANN_a27	126	$> 10^5$	2.54	○ **0.78**	>2,232	7,647	2.75	
MANN_a45	345	$> 10^5$	3,090	★ **304**		>10,640		
p_hat300-3	36	780	10.82	○ **2.65**		633	5.39	
p_hat500-2	36	133	3.14	○ **0.79**	95.71	151		
p_hat500-3	50	$> 10^5$	1,788	★ **157**		>10,640		
p_hat700-2	44	5,300	44.42	○ **5.98**		1,542	25.44	
p_hat700-3	62	$> 10^5$	68,187	★ **2,504**		>10,640	>415	
p_hat1000-2	46	$> 10^5$	2,434	★ **239**		>10,640		
san200_0.9_2	60	$> 10^5$	4.17	0.42	0.96	1,427		○ **0.15**
san200_0.9_3	44	42,643	0.16	○ **0.064**		144		15.15
san400_0.7_1	40	$> 10^5$	1.76	○ **0.55**	>2,232	315		
san400_0.7_2	30	$> 10^5$	0.33	○ **0.13**	113	118		505
san400_0.7_3	22	$> 10^5$	3.60	○ **1.46**		456		
san400_0.9_1	100	$> 10^5$	3.43	★ **0.12**		5,335		
san1000	15	$> 10^5$	4.82	2.22	★ **0.11**	2,249		
sanr200_0.9	42	86,954	289	● **42**		>10,640		
sanr400_0.7	21	2,426	379	○ **187**		11,767		
gen200_p0.9_44	44	48,262	5.39	○ **0.47**			1.88	13.01
gen200_p0.9_55	55	9,281	15.02	1.25			0.96	● **0.19**
gen400_p0.9_55	55	$> 10^5$	$> 10^6$	**59,653**				
C125.9	34	50.05	0.24	○ **0.058**			0.56	46.60

Entries indicated by ★, ●, and ○ represent those that are more than or equal to 10, 5, and 2 times faster than all the others obtained within the time limits in the same row, respectively.

[2] T. Fahle. Simple and fast: Improving a branch-and-bound algorithm for maximum clique. *ESA 2002, LNCS 2461*: 485–498, 2002.

[3] T. Higashi, E. Tomita. A more efficient algorithm for finding a maximum clique based on an improved approximate coloring. *Tech. Rep. Univ. Electro-Commun., UEC-TR-CAS5-2006*: 2006.

[4] D. S. Johnson, M. A. Trick (Eds.). Cliques, Coloring, and Sat. *DIMACS Series in DMTCS*, vol.26, Amer. Math. Soc.: 1996.

[5] P. R. J. Östergård. A fast algorithm for the maximum clique problem. *Disc. Appl. Math.*, 120: 197–207, 2002.

[6] E. C. Sewell. A branch and bound algorithm for the stability number of a sparse graph. *INFORMS J. Comput.*, 10: 438–447, 1998.

[7] Y. Sutani, E. Tomita. Computational experiments and analyses of a more efficient algorithm for finding a maximum clique. *Tech. Rep. IPSJ*, MPS-57: 45–48, 2005.

[8] Y. Sutani, T. Higashi, E. Tomita. A more efficient algorithm for finding a maximum clique with an improved approximate coloring. *Tech. Rep. Summer LA Symp.*, Hiroshima, Japan: 2006.

[9] E. Tomita, T. Seki. An efficient branch-and-bound algorithm for finding a maximum clique. *DMTCS 2003, LNCS 2731*: 278–289, 2003.

[10] E. Tomita, A. Tanaka, H. Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoret. Comput. Sci.*, 363: 28–42, 2006.

[11] E. Tomita, T. Kameda. An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. *J. Global Optim.*, 37: 95–111, 2007.