An Improved Branch-and-Bound MCT Algorithm for Finding a Maximum Clique

Etsuji Tomita[1,a)] Jiro Yanagisawa[2] Kengo Katayama[3] Kazuho Kanahara[3]
Takahisa Toda[1] Hiro Ito[1,4] Mitsuo Wakatsuki[1] Tetsuro Nishino[1]

Abstract: We improve a branch-and-bound algorithm called MCT（Tomita et al., FAW 2016, LNCS 9711, pp.215–226, 2016) for finding a maximum clique. First, we devise a new efficient approximation algorithm for finding a maximum clique. Second, we employ MIS vertex ordering with an appropriate precondition. Third, we employ a combination of Re-NUMBER and Infra-chromatic bound. Finally, we devise an adaptive change of stages of the search tree. The finally improved MCT algorithm is named MCT*.

It is shown that MCT* algorithm is significantly faster than MCT by extensive computational experiments. In addition, it is shown that MCT* algorithm is faster than the state-of-the-art IncMC2 algorithm（Li et al., INFORMS J. Computing, 30, pp. 137–153, 2018）for many instances.

1. Introduction

Given an undirected graph G, a clique is a subgraph in which all pairs of vertices are mutually adjacent in G. Many important problems can be formulated as maximum clique problems [16].

Algorithms for finding a maximum clique ([16], [14]) in a given graph have received much attention especially recently, since they have applications in many areas. There has been much theoretical and experimental work on this problem [16]. In particular, while finding a maximum clique is a typical NP-hard problem, considerable progress has been made for solving this problem *in practice*. Furthermore, much faster algorithms are required in order to solve many practical problems. Along this line, Tomita et al. developed a series of branch-and-bound algorithms (MCQ [9], MCR [10], MCS [11], [12] and MCT [13] among others) that run fast in practice. It was shown that MCT is very fast for many instances [13].

In this report, we present improvements to MCT in order to make it much faster. First, Kanahara et al. devise a new approximation algorithm named New_IKLS [1] for the maximum clique problem in order to obtain a better initial lower bound on the size of a maximum clique. Second, we introduce MIS vertex ordering [5] with an appropriate precondition. Third, we introduce a combination of Re-NUMBER and Infra-chromatic bound [7], [8]. Finally, we introduce adaptive change of stages of the search tree. The new algorithm obtained from MCT with the above all improvements is named MCT*. It is shown that MCT* is significantly faster than MCT by extensive computational experiments.

The definitions and notation of this report are based on [13]. A more detailed version of this report is to appear in [15].

2. Improved algorithms

2.1 New approximation algorithms for finding a maximum clique

While MCT [13] used the KLS algorithm by Katayama et al. [2], the improved algorithms in this report will use New_IKLS [3], which is an extended version of KLS.

Iterated k-opt Local Search (IKLS) [3] is an Iterated Lo-

[1] The University of Electro-Communications, Chofu, Tokyo 182–8585, Japan
[2] East Nippon Expressway Co. Ltd., Iizakamachi Hirano, Fukushima 960–0231, Japan
[3] Okayama University of Science, Ridaicho, Kita, Okayama 700–0005, Japan
[4] CREST, Chiyoda, Tokyo 102–0076, Japan
a) e.tomita@uec.ac.jp

cal Search based metaheuristic. IKLS consists of a Local Search process in which KLS [2] is employed as a dedicated local search and Kick process that escapes from local optima obtained by KLS. As an additional strategy performed occasionally, Restart is employed to diversify the search by moving to other search points. KLS is an effective local search based on variable depth search (VDS) proposed by Katayama et al. [2]. In KLS, k-opt neighborhood search, consisting of add phase and drop phase, is repeated until a maximal clique is found.

In this report, we devise and employ a further improved approximation algorithm, named New_IKLS, based on [1]. New_IKLS consists of Multi-start KLS (MKLS), IKLS-SFI and simplified Hyper-Heuristic IKLS (HH-IKLS). In New_IKLS, one of these algorithms (MKLS, IKLS-SFI and HH-IKLS) is chosen for the search based on edge density *dens* of a given graph.

Replacing KLS(V, Q'_{max}) in MCT (at line 17 of Fig. 4. of [13]) by New_IKLS(V, Q'_{max}) gives us a new algorithm named $\mathbf{MCT}_1$.

2.2 Vertex ordering at the root of the search tree

It is well known that EXTENDED INITIAL SORT-NUMBER (*degeneracy ordering*) especially at the root of the search tree is effective in general for searching as in MCR [10] and MCS [11].

On the other hand, Li et al. showed that their MIS vertex ordering at the root of the search tree is remarkably effective for searching for some types of graphs as in IncMaxCLQ [4] and IncMC2 [5].

2.2.1 MIS vertex ordering

Given a graph $G = (V, E)$, MIS vertex ordering for V is defined as follows:

First, we extract a MIS in G and name it S_1. Second, we extract a MIS in $G_1 = (V\backslash S_1, E \cap (V\backslash S_1)) = G\backslash S_1$ and name it S_2. Third, we extract a MIS in $G_2 = G_1\backslash S_2$ and name it S_3. ... Until $G_k = G_{k-1}\backslash S_k = \emptyset$. Then in each S_i $(1 \le i \le k)$, we sort the vertices in S_i in nonincreasing order of their degrees. Finally, we obtain the MIS vertex ordering V' for V as $V' = S_1 \cup S_2 \cup S_3 \cup ... \cup S_k$, where the vertices in S_1 appear first in the same order as in S_1, and then the vertices in S_2 follow in the same way, and so on. This completes the definition of MIS vertex ordering.

In order to extract MISs $S_1, S_2, ..., S_k$ from $G, G_1, ..., G_{k-1}$, we employ an algorithm consisting of a simple maximum-clique-finding MCS_1 algorithm [13] to their compliment graphs $\bar{G}, \bar{G}_1, ..., \bar{G}_{k-1}$ successively. Here, MIS vertex ordering is applied only if the density of G is greater than or equal to $0.7+\epsilon$ ($\epsilon = 0.01$) where the complement graph $\bar{G}$ is sparse and the MCS_1 algorithm can be easily carried out. In addition, MIS vertex ordering is applied only if $\max_{v \in V}\{No(v)\} >| Q_{max} |$ because of the bounding condition.

2.2.2 New precondition for MIS vertex ordering

MIS vertex ordering is not effective on all kinds of graphs, and it should be applied very carefully. Li et al. [4] restricted the application of MIS vertex ordering to the case where $|\{i \in \{1, 2, ..., k\} \mid |S_i| = 1\}| \le 1$. A similar precondition is also applied in [5].

First, we applied MIS vertex ordering under the same precondition of [4] or [5], and found that MIS vertex ordering did not work well for some graphs in our environment.

So, we define a new precondition as follows:

Given $G = (V, E)$, let vertices in V have been ordered by EXTENDED INITIAL SORT-NUMBER and let No be the numbers assigned herein. In addition, let the result of MIS vertex ordering be a sequence of maximum independent sets $S_1, ..., S_k$ as above, and let No' be numbers assigned according to this MIS vertex ordering. We pay attention to vertex p $(= V[|V|])$ that is the last vertex after the application of EXTENDED INITIAL SORT-NUMBER, that is to be visited first in this case, and vertex $q = S_k[|S_k|]$ that is the last vertex when MIS vertex ordering is applied, that is to be visited first in this case. Let Q_{max} be a maximum clique so far obtained. Now we consider two integers t_1 and t_2 such that

$$t_1 = |\{v \in V \cap \varGamma(p) \mid No(v) > |Q_{max}| - 1\}|, \text{ and}$$
$$t_2 = |\{v \in V \cap \varGamma(q) \mid No'(v) > |Q_{max}| - 1\}|.$$

Then our new precondition for MIS vertex ordering requires the following (1) or (2) to hold:

$$\frac{t_1}{t_2 + 1} \times \frac{t_1 - t_2}{|V|} > 0.3 \quad (1)$$

$$\max_{v \in R}\{No'(v)\} = |Q_{max}| \quad (2)$$

When equation (2) holds, expansion of the search terminates owing to the bounding condition. When the condition (1) or (2) holds, we apply MIS vertex ordering to V. Otherwise, vertices in V remains to be ordered by EXTENDED INITIAL SORT-NUMBER. We give the procedure of MIS vertex ordering combined with our new precondition the name of **MIS_vertex_ordering**(V, No). The new algorithm based on MCT_1, equipped with this MIS_vertex_ordering(V, No) is named $\mathbf{MCT}_2$.

2.3 Combination of Re-NUMBER and Infra-chromatic bound

Numbering and Re-NUMBERing is very effective to get an upper bound of the size of a maximum clique. But, it is known that the gap between the chromatic number $\chi(G)$ and the size of a maximum clique $\omega(G)$ can be arbitrarily large for $G = (V, E)$ [6]. In order to avoid this difficulty, Li et al. [4], [5] introduced a new upper bound based on MaxSAT. Subsequently, San Segund et al. [7], [8] devised Infra-chromatic bound as its simplified version as follows:

Given a subgraph $G_R = (R, E_R)$ with $R \subseteq V$, let a sequence of independent sets be $C_1, C_2, ..., C_{maxno}$ where $C_1 \cup C_2 \cup ... \cup C_{maxno} = R$, and let a maximum clique found so far and the current clique be Q_{max} and Q, respectively. Let $No_{th} = |Q_{max}| - |Q|$ and $T_a = C_{No_{th}} \cup C_{No_{th}+1} \cup ... \cup C_{maxno}$ where vertices in T_a are ordered as in R and only a vertex in T_a should be expanded. We begin by letting the *forbidden number of vertices* $F := \emptyset$. For each $p_i = T_a[i], \quad i = 1, 2, ..., |T_a|$, try to find $k_1 \in \{1, 2, ..., No_{th}\} \backslash F$ such that $|\Gamma(p_i) \cap C_{k_1}| = 1$. If such k_1 is found then let $q \in \Gamma(p_i) \cap C_{k_1}$. Subsequently, try to find $k_2 \in \{1, 2, ..., No_{th}\} \backslash F \quad (k_2 \neq k_1)$ such that $|\Gamma(p_i) \cap \Gamma(q) \cap C_{k_2}| = \emptyset$. If such k_2 exists then we can prune expansion from p_i owing to *Infra-chromatic bound* [7], [8] and let $T_a := T_a \backslash \{p_i\}$ and $F := F \cup \{k_1, k_2\}$. Such a process as above is repeated in all possible ways. □

We employ a **procedure Re-Ic** that first executes Re-NUMBER if possible and otherwise executes Infra-chromatic bound as in [8]. The new MCT_2 algorithm obtained from MCT_2 by replacing Re-NUMBER at stages 2 and 3 by Re-Ic is named $\mathbf{MCT}_3$ algorithm.

2.4 Adaptive change of stages of the search tree

2.4.1 Stage value T and thresholds Th_1, Th_2 in MCT

We recall the preceding MCT algorithm for its stage value T and thresholds Th_1, Th_2. In MCT, each node of the search tree (subproblem) is classified in stage 1, stage 2, or stage 3 depending on the stage value T.

Let a set of candidate vertices in question be R, $No_{th} := |Q_{max}| - |Q|$, and $R_p = R \cap \Gamma(p)$ for $p \in R$, where R_p is a child of R in the search tree. We define the stage value T for R_p as follows:

$$T = \frac{|\{v \in R_p | No(v) > No_{th}\}|}{|R_p|} \times dens \tag{3}$$

In MCT, we defined that $Th_1 = 0.4$ and $Th_2 = 0.1$, and determined the stages of R_p as follows: The stage of the root of the search tree is in stage 1. If $T \geq Th_1$ and its parent R is in stage 1, then R_p is in stage 1. Otherwise, if $Th_2 \leq T < Th_1$ or $dens > 0.95 + \epsilon$, then R_p is in stage 2. If $T \leq Th_2$ then R_p is in stage 3.

2.4.2 Adaptive change of threshold Th_2

When the stage value T is large, the number of vertices to be expanded becomes large.

If the threshold Th_2 is set to be larger, then the portion of stage 3 becomes larger compared to that of stage 2, where the lightened procedure is carried out at the portion of stage 3. It is considered to be effective if the subproblem is large then we set the threshold Th_2 to be large, otherwise we set the threshold Th_2 to be small. Whether the subproblem is large or not is determined by the threshold value T at depth 1 of the subproblem in question. In our new algorithm, we let $Th_2 := 0.15$ if $T \geq Th_1$ at depth 1 of the subproblem, and we let $Th_2 := 0.05$ otherwise.

By changing the setting of Th_2 as above in MCT_3, we obtain the new algorithm named $\mathbf{MCT}^*$.

3. Computational experiments

We carried out computational experiments in order to demonstrate the effectiveness of the techniques given in the previous section. All of KLS, New_IKLS, MCT_1, MCT_2, MCT_3 and MCT^* algorithms were implemented in C++ language. The computer had an Intel core i7-4790 CPU of 3.6 GHz clock with 8 GB of RAM and 8 MB of cache memory. It worked on a Linux CentOS7 operating system with a compiler g++ 8.2.0 (Option -O3).

Table 1 shows stepwise improvements from MCT to MCT^* for selected graphs. Columns KLS and N_I under *Sol* in Table 1 show the solutions (= the sizes of the nearly maximum clique) obtained by KLS and New_IKLS, respectively. We can confirm the stepwise improvements from this Table 1. Especially, MCT^* is remarkably faster than MCT for frb family and keller5 owing to MIS vertex ordering under our new precondition for it.

Table 1 Comparison of MCT, MCT_1, MCT_2, MCT_3, and MCT* algorithms

N_I is short for New_IKLS

Graph Name	n	*dens*	ω	Sol KLS [2]	Sol N_I [1]	Times [sec] MCT [11]	Times [sec] MCT_1	Times [sec] MCT_2	Times [sec] MCT_3	Times [sec] MCT*	Branches [$\times 10^{-6}$] MCT [11]	Branches MCT_1	Branches MCT_2	Branches MCT_3	Branches MCT*
brock400_3	400	0.748	31	25	31	60.0	17.6	17.7	14.9	13.1	27,096,288	9,583,440	9,583,440	13,624,073	2,935,638
brock400_4	400	0.749	33	25	33	47.2	9.49	9.58	8.02	7.15	21,666,083	4,645,207	4,645,207	1,385,863	1,433,259
gen400_p0.9_55	400	0.900	55	53	55	127	0.07	0.07	0.07	0.07	50,270,918	0	0	0	0
gen400_p0.9_65	400	0.900	65	65	65	0.46	0.24	0.06	0.08	0.07	57,932	57,932	0	0	0
p_hat300-3	300	0.744	36	36	36	0.27	0.24	0.26	0.19	0.23	89,184	114,055	114,055	38,154	29,454
p_hat700-2	700	0.498	44	44	44	0.66	0.66	0.66	0.57	0.47	196,742	196,742	196,742	96,409	52,454
p_hat700-3	700	0.748	62	62	62	203	202	203	153	140	53,847,835	68,845,281	68,845,281	20,648,466	13,031,357
p_hat1000-2	1,000	0.490	46	46	46	28.7	27.7	27.8	23.0	17.7	10,049,314	10,049,314	10,049,314	4,860,345	2,610,385
p_hat1000-3	1,000	0.744	68	68	68	39,201	36,875	36,979	26,876	29,385	9,026,919,909	9,026,919,909	9,026,919,909	3,134,737,026	3,248,375,111
p_hat1500-2	1,500	0.506	65	65	65	1,463	1,487	1,485	1,352	823	399,837,407	451,422,645	451,422,645	190,757,565	88,614,691
san400_0.9_1	400	0.900	100	100	100	0.28	0.06	0.06	0.08	0.08	0	0	0	0	0
sanr200_0.9	200	0.898	42	42	42	4.51	4.5	4.52	3.43	3.7	2,123,667	2,123,667	2,123,667	850,053	663,343
keller5	776	0.752	27	27	27	10,137	10,253	274	211	242	4,494,774,392	199,807,097	131,924,566	83,427,512	84,229,352
hamming10-2	1.024	0.990	512	512	512	7.53	0.93	0.91	0.91	0.88	0	0	0	0	0
frb30-15-1	450	0.824	30	28	29	151	155	0.22	0.22	0.22	83,357,094	85,329,599	112,481	96,126	95,729
frb30-15-2	450	0.823	30	30	30	124	123	0.06	0.06	0.06	64,620,862	64,620,862	0	0	0
frb30-15-3	450	0.824	30	28	29	126	115	0.15	0.15	0.15	73,217,126	65,548,985	59,654	52,363	51,885
frb30-15-4	450	0.823	30	29	30	537	226	0.06	0.06	0.06	314,921,319	120,629,113	0	0	0
frb30-15-5	450	0.824	30	29	29	151	150	0.09	0.09	0.09	85,333,028	85,333,028	19,468	16,735	16,652
frb35-17-1	595	0.842	35	32	34	5,070	4,909	0.26	0.25	0.25	2,556,662,455	2,504,087,791	79,823	68,333	68,308
frb35-17-2	595	0.842	35	33	34	18,846	18,958	4.82	4.65	4.63	9,997,474,079	9,940,818,787	2,604,325	2,251,618	2,251,495
frb35-17-3	595	0.842	35	33	34	4,534	4,324	1.09	1.07	1.07	2,305,493,121	2,196,967,869	559,379	481,089	480,425
frb35-17-4	595	0.842	35	32	34	7,069	6,808	0.26	0.25	0.25	3,493,698,010	3,255,677,248	72,925	63,386	63,356
frb35-17-5	595	0.842	35	33	35	16,717	6,794	0.11	0.11	0.10	8,679,207,631	3,348,191,463	0	0	0
frb40-19-1	760	0.857	40	38	40	>1day	>1day	0.18	0.19	0.19	-	-	0	0	0
frb40-19-2	760	0.857	40	37	39	>1day	>1day	4.34	4.23	4.23	-	-	2,064,228	1,792,513	1,791,884

In addition, we made comparisons between MCT and MCT* for more DIMACS and BHOSLIB graphs and for random graphs. We also added the results of execution of the state-of-the-art IncMC2 algorithm by Li et al. [5] for the same problem. As the results,

MCT* is faster than MCT for many instances tested, and

MCT* is faster than IncMC2 for many DIMACS and BHOSLIB benchmark graphs tested.

MCT* is faster than IncMC2 for all random graphs except for very dense random graphs and r10000.2. See

Table 2. Comparison of Algorithms (for benchmark graphs) in [15], and

Table 3. Comparison of Algorithms (for random graphs) in [15] for the details.

In conclusion, MCT* is significantly faster than MCT. MCT* is faster than IncMc2 and the other algorithms in [5] for many instances. Note that the number of branches in IncMC2 is the smallest in many cases but the CPU time is not necessarily smallest because of the heavy overhead of time in IncMC2. IncMC2 is fast for dense graphs.

Acknowledgements We should like to express our sincere gratitude to S. Matsuzaki and E. Harley for their valuable discussions and comments. This research was supported in part by JSPS KAKENHI Grant Numbers JP17K00006, JP19K12166, and JST CREST Grant Number JPMJCR1402.

References

[1] Kanahara, K., Katayama, K., Tomita, E.: A hyper-heuristic for the maximum clique problem, IEICE Tech. Report, COMP2020-34, 30-37 (2021)

[2] Katayama, K., Hamamoto, A., Narihisa, H.: An effective local search for the maximum clique problem, Information Processing Letters, 95, 503-511 (2005)

[3] Katayama, K., Sadamatsu, M., Narihisa, H.: Iterated k-opt local search for the maximum clique problem, EvoCOP 2007, LNCS 4446, 84-95 (2007)

[4] Li, C.M., Quan, Z.,: Combining graph structure exploitation and propositional reasoning for the maximum clique problem, Proc. IEEE ICTAI, 344–351 (2010)

[5] Li, C.M., Fang, Z., Jiang, H., Xu, K.: Incremental upper bound for the maximum clique problem, INFORMS J. Computing, 30, 137–153 (2018)

[6] Mycielski, J.: Sur le coloriage des graphes, Colloq. Math., 3, 161–162 (1955)

[7] San Segundo, P., Nikolaev, A., Batsyn, M.: Infra-chromatic bound for exact maximum clique search, Computers and Operations Research, 64, 293–303 (2015)

[8] San Segundo, P., Lopez, A., Batsyn, M., Nikolaev, A., Pardalos, P.: Improved infra-chromatic bound for exact maximum clique search, Informatica, 45, 868–880 (2016)

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

[10] 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)

[11] Tomita, E., Sutani, Y., Higashi, T., Takahashi, S., Wakatsuki, M.: A simple and faster branch-and-bound algorithm for finding a maximum clique, WALCOM 2010, LNCS 5942, 191–203 (2010)

[12] Tomita, E., Sutani, Y., Higashi, T., Wakatsuki, M.: A simple and faster branch-and-bound algorithm for finding a maximum clique with computational experiments, IEICE Trans. Information and Systems, E96-D, 1286-1298 (2013) https://www.jstage.jst.go.jp/article/transinf/E96.D/6/E96.D_1286/_article

[13] Tomita, E., Yoshida, K., Hatta, T., Nagao, A., Ito, H., Wakatsuki, M.: A much faster algorithm for finding a maximum clique, FAW 2016, LNCS 9711, 215–226 (2016)

[14] Tomita, E.: Efficient algorithms for finding maximum and maximal cliques - Keynote -, WALCOM 2017, LNCS 10167, 3–15 (2017)

[15] Tomita, E., Yanagisawa, J., Katayama, K., Kanahara, K., Toda, T., Ito, H., Wakatsuki, M., Nishino, T.: An improved branch-and-bound MCT algorithm for finding a maximum clique, Trans. Computational Science & Computational Intelligence, Springer Nature (to appear, 2021)

[16] Wu, Q., Hao, J.K.: A review on algorithms for maximum clique problems, European J. Operational Research, 242, 693–709 (2015)