Speeding up MCS Algorithm for the Maximum Clique Problem with ILS Heuristic and Other Enhancements

Evgeny Maslov, Mikhail Batsyn, and Panos M. Pardalos

Abstract In this chapter, we present our enhancements of one of the most efficient exact algorithms for the maximum clique problem—MCS algorithm by Tomita, Sutani, Higashi, Takahashi and Wakatsuki (in Proceedings of WALCOM'10, 2010, pp. 191–203). Our enhancements include: applying ILS heuristic by Andrade, Resende and Werneck (in Heuristics 18:525–547, 2012) to find a high-quality initial solution, fast detection of clique vertices in a set of candidates, better initial coloring, and avoiding dynamic memory allocation. A good initial solution considerably reduces the search tree size due to early pruning of branches related to small cliques. Fast detecting of clique vertices is based on coloring. Whenever a set of candidates contains a vertex adjacent to all candidates, we detect it immediately by its color and add it to the current clique avoiding unnecessary branching. Though dynamic memory allocation allows to minimize memory consumption of the program, it increases the total running time. Our computational experiments show that for dense graphs with a moderate number of vertices (like the majority of DIMACS graphs) it is more efficient to store vertices of a set of candidates and their colors on stack rather than in dynamic memory on all levels of recursion. Our algorithm solves p_hat1000-3 benchmark instance which cannot be solved by the original MCS algorithm. We got speedups of 7, 3000, and 13000 times for gen400_p0.9_55, gen400_p0.9_65, and gen400_p0.9_75 instances, correspondingly.

E. Maslov (✉) · M. Batsyn · P.M. Pardalos
Laboratory of Algorithms and Technologies for Network Analysis, National Research University Higher School of Economics, 136 Rodionova, Nizhny Novgorod, Russian Federation
e-mail: lyriccoder@gmail.com

M. Batsyn
e-mail: mbatsyn@hse.ru

P.M. Pardalos
e-mail: pardalos@ufl.edu

P.M. Pardalos
Center of Applied Optimization, University of Florida, 401 Weil Hall, P.O. Box 116595, Gainesville, FL 32611-6595, USA

B.I. Goldengorin et al. (eds.), *Models, Algorithms, and Technologies for Network Analysis*, Springer Proceedings in Mathematics & Statistics 59,
DOI 10.1007/978-1-4614-8588-9_7,

Keywords Maximum clique problem · MCS branch-and-bound algorithm · ILS heuristic · Graph coloring

1 Introduction

The maximum clique problem refers to the problem of finding a clique (a complete subgraph) with the largest number of vertices in a given graph. It has a lot of practical applications (Bomze et al., 1999 [3]; Du and Pardalos, 1999 [6]; Boginski et al., 2003 [2]).

There exists a lot of exact branch-and-bound algorithms for the solving the maximum clique problem: Bron and Kerbosch (1973) [4], Carraghan and Pardalos (1990) [5], Fahle (2002) [7], Tomita and Seki (2003) [18], Tomita and Kameda (2007) [17], Tomita et al. (2010) [19], Li and Quan (2010) [14, 15] and others. According to the published results for DIMACS graphs MCS algorithm by Tomita et al. [19] and the Max-Sat based algorithms by Li and Quan [14, 15] report the best performance among the existing exact methods known to the authors.

Since the maximum clique problem is NP-hard (Garey and Johnson, 1979, [9]), there also exists a number of heuristic approaches which find solutions of high quality: Kopf and Ruhe (1987) [13], Jerrum (1992) [12], Feo and Resende (1995) [8], Glover and Laguna (1997) [10], Singh and Gupta (2006) [16], Grosso et al. (2008) [11] and others. One of the most successful heuristic algorithms is the iterated local search (ILS) developed by Andrade et al. (2012) [1].

In our approach, we apply ILS heuristic (Andrade et al., 2012 [1]) to obtain an initial solution and speed up the original MCS algorithm. This solution provides a good lower bound and many branches having an upper bound not greater than this lower bound are pruned. Our computational results prove the effectiveness of this approach especially for large dense graphs due to the great reduction in the number of the search tree nodes.

Another enhancement we apply follows the ideas of Fahle (2002) [7] to add the vertices adjacent to all candidates to the current clique immediately without branching. We suggest an algorithm of fast detection of such vertices by means of coloring. The next enhancement is using of a full-fledged greedy coloring at the first node of the search tree without reordering of vertices. Finally, we propose to avoid dynamic memory allocation and use only fast stack memory for candidates and their colors on all steps of the algorithm.

The chapter is organized as follows. In the next section, we describe the maximum clique problem and prove three propositions needed fast detection of clique vertices. Section 3 contains a description of our algorithm. Computational results showing a comparison with the original MCS algorithm are presented in Sect. 4.

2 Maximum Clique Problem

Let $G = (V, E)$ be an undirected graph, where $V = \{1, 2, \ldots, n\}$ is the set of vertices, $E \subseteq V \times V$ is a set of edges. The adjacency matrix of $G(V, E)$ is denoted as $A = (a_{ij})$, where $a_{ij} = 1$ if $(i, j) \in E$ and $a_{ij} = 0$ if $(i, j) \notin E$. *A complementary graph* of $G(V, E)$ is the graph $\overline{G}(V, \overline{E})$, where $\overline{E} = (V \times V) \setminus E$. Graph $G(V, E)$ is *complete* if all its vertices are pairwise adjacent, i.e. $\forall i, j \in V,\ (i, j) \in E$. A *clique* C is a subset of V such that all vertices in this subset are pairwise adjacent. A clique which has the maximum size (number of vertices) in a graph is called *a maximum clique*. The number of vertices of a maximum clique in graph $G(V, E)$ is denoted by $\omega(G)$. The maximum clique problem refers to the problem of finding a maximum clique in a given graph.

It is possible to formulate the notion of a *clique* in terms of a chromatic number: a clique is a graph which chromatic number is equal to the number of vertices. It can help to identify a clique by means of coloring (see Proposition 1).

Definition 1 A reasonable coloring is a sequential coloring which would not use a new color for a vertex if it is possible to color it into one of the already used colors.

Proposition 1 *If a graph with k vertices is colored in k colors by a reasonable coloring, then this graph is complete.*

Proof By contradiction: let this graph be incomplete. This means that it has at least two non-adjacent vertices. Let these vertices be i and j, and vertex i is colored before vertex j by our reasonable coloring. Since k vertices are colored in k colors then every vertex has a unique color. But then vertex j will be colored into the same color as vertex i because no other vertex is colored in this color and our coloring is reasonable. This contradicts with the condition that k colors are used. □

Vertices with higher degree are usually contained in larger cliques. For example, a vertex which is adjacent to all other vertices in a graph is contained in all cliques including the maximum one. Such vertices can be found faster after coloring of a given graph (see Propositions 2 and 3).

Proposition 2 *If a vertex is adjacent to all vertices of a graph, then it will have a unique color in any coloring of the graph.*

Proof By definition, another vertex cannot have the same color in a coloring because it is adjacent to this vertex. □

Definition 2 A greedy coloring is a sequential coloring which on every step colors a vertex of a graph in a minimal possible color.

Proposition 3 *For any greedy coloring if a vertex has a unique color then it is adjacent to all vertices with greater color.*

Proof By contradiction: let there be a vertex i with a unique color k and a vertex j with a greater color which is not adjacent to i. Vertex j should be colored in color k or even smaller color because no vertices have color k except i which is not adjacent to j and our coloring always uses a minimal possible color. This contradicts with the condition that j has a greater color than k. □

Following the df algorithm (Fahle, 2002 [7]) we check whether there are candidates which are adjacent to all the other candidate vertices and thus should be immediately added to the current clique without any branching. In contrast with the df implementation, we can check it much faster using Propositions 2 and 3, because we have to check only vertices which have a unique color after coloring. For such a vertex, we should only check if it is adjacent to all vertices with a smaller color.

Following the MCS algorithm of Tomita et al. (2010) [19] for an upper bound in our algorithm we use the same greedy coloring which assigns the minimal possible color number to every vertex one by one.

3 Algorithm Description

The main procedure of our algorithm has the following steps:

1. Find an initial clique with the ILS heuristic (see Andrade et al. (2012) [1] for a detailed description).
2. Perform initial ordering of vertices as follows. Find the vertex with the minimum degree in the graph and put it to the beginning of the current candidates set S. If several vertices have the same minimum degree then take the vertex which has the minimum support (sum of the vertex neighbors degrees) among them. For several vertices having the same minimum support, ties are broken arbitrarily. Then delete it from the graph and repeat this step until the graph has edges. The remaining set of vertices called *Rmin* is put to the beginning of S.
3. Color the current candidates set S with a standard greedy sequential coloring (use minimal possible color on each step) from the beginning (*Rmin* set) to the end.
4. While the current candidates set S has unconsidered candidates call the recursive brand-and-bound procedure described below.

The recursive branch-and-bound procedure has the following steps:

1. Consider the last candidate in S. If its color number plus the size of the current clique is less or equal to the size of the best found clique then:
   - If we are not at the first level of recursion return from recursion because the candidates are ordered by colors and all the remaining candidates have a smaller color number.
   - Otherwise consider the candidate which is previous to this one in S and repeat this step for it.

2. Add this candidate to the current clique Q.
3. Remove it from the current candidates set S.
4. Form a new candidates set S' taking from set S only the neighbors of this candidate. We take these neighbors from S in the initial order obtained on step 2 of the main procedure.
5. Color the new candidates set S' with standard greedy sequential coloring.
6. If the number of colors is equal to the size of the new candidate set S', add all these candidates to the current clique (see Proposition 1). Return from recursion.
7. If there are candidates having a unique color (see Proposition 2), check each of them if it is adjacent to all the candidates with a smaller color (see Proposition 3). If yes, then add such vertices to the current clique and remove them from S'.
8. Sort the new candidates set S' in the ascending order with respect to their color numbers.
9. Recursively call this procedure for the new candidates set S'

To store candidate vertices and their colors on every level of the search tree, we allocate stack memory of a predetermined size, enough for any DIMACS graph with the number of vertices from 100 to 1500 and for any possible search tree depth. We suggest to use a greedy coloring on the first step, but do not reorder the vertices by their color numbers as in MCS algorithm. So on the first level we have to keep in the candidates set those vertices for which their color number plus the current clique size is not greater than the currently best clique size.

4 Computation Results

We test our algorithm on hard DIMACS instances which need more than 10 minutes to be solved. The computational results are presented in Tables 1 and 2. Table 1 shows the number of the search tree nodes for our algorithm and for MCS algorithm. The last two columns contain the size of the maximum clique and the size of the best clique found by the ILS heuristic applied in our algorithm. We run the ILS heuristic with 100000 scans for all the considered instances except *gen400_p0.9_55* and *p_hat1000-3* for which we use 60 millions scans. The greatest reduction of the search tree size in comparison with MCS algorithm is got for *gen* instances: 60 times reduction for *gen400_p0.9_55*, 7000 times for *gen400_p0.9_65* and 240000 times for *gen400_p0.9_75*.

The comparison of the computational time for our algorithm and MCS algorithm is given in Table 2. The best results are obtained for *gen400_p0.9_75* (our algorithm is 13000 times faster for it), *gen400_p0.9_65* (3000 times faster), *gen400_p0.9_55* (7 times faster), and *p_ hat1000-3* graphs. MCS algorithm is unable to solve *p_hat1000-3* instance (at least in 10 days) while our algorithm solves it

Table 1 The number of search tree nodes

Benchmark	Our algorithm	MCS	Maximum clique	Heuristic solution
brock800_1	1095645796	1097174023	23	19
brock800_2	970862419	972110520	24	20
brock800_3	625139200	625234820	25	19
brock800_4	424101537	424176492	26	19
gen400_p0.9_55	55079436	3425049256	55	54
gen400_p0.9_65	822991	6500277298	65	64
gen400_p0.9_75	41445	10140428816	75	75
keller5	10337321299	10339211493	27	27
MANN_a45	219979	221476	345	344
p_hat1000–2	21587044	25209207	46	44
p_hat1000–3	8773710250	–	68	67
p_hat1500–2	607200969	660539819	65	61
p_hat700–2	416003	670369	44	42
p_hat700–3	81631372	98911559	62	60

Table 2 Running time

Benchmark	ILS	Our algorithm	MCS
brock800_1	167	6482	6337
brock800_2	164	5886	5737
brock800_3	166	3994	3830
brock800_4	166	3009	2849
gen400_p0.9_55	4320	5133	39015
gen400_p0.9_65	8	25	77620
gen400_p0.9_75	7	8	110579
keller5	96	78957	78875
p_hat1000–2	172	360	272
p_hat1000–3	54185	214611	>1000000
p_hat1500–2	346	10231	11859
p_hat700–3	44	1303	1529

in 3 days. The total time of solving all the considered DIMACS instances is reduced by 75 %.

Acknowledgements The authors would like to thank professor Mauricio Resende and his co-authors for the source code of their powerful ILS heuristic. The authors are supported by LATNA Laboratory, National Research University Higher School of Economics (NRU HSE), Russian Federation government grant, ag. 11.G34.31.0057.

References

1. Andrade, D.V., Resende, M.G.C., Werneck, R.F.: Fast local search for the maximum independent set problem. J. Heuristics **18**(4), 525–547 (2012). doi:10.1007/s10732-012-9196-4
2. Boginski, V., Butenko, S., Pardalos, P.M.: On structural properties of the market graph. In: Nagurney, A. (ed.) Innovations in Financial and Economic Networks, pp. 29–45. Edward Elgar, Cheltenham Glos (2003)
3. Bomze, I., Budinich, M., Pardalos, P.M., Pelillo, M.: Handbook of Combinatorial Optimization. Kluwer Academic, Dordrecht (1999). Chap. "The maximum clique problem"
4. Bron, C., Kerbosch, J.: Algorithm 457: finding all cliques of an undirected graph. Commun. ACM **16**(9), 575–577 (1973). doi:10.1145/362342.362367
5. Carraghan, R., Pardalos, P.M.: An exact algorithm for the maximum clique problem. Oper. Res. Lett. **9**(6), 375–382 (1990). doi:10.1016/0167-6377(90)90057-C
6. Du, D., Pardalos, P.M.: Handbook of Combinatorial Optimization, Supplement, vol. A. Springer, Berlin (1999)
7. Fahle, T.: Simple and fast: improving a Branch-and-Bound algorithm for maximum clique. In: Proceedings of the 10th Annual European Symposium on Algorithms, ESA '02, pp. 485–498. Springer, London (2002)
8. Feo, T.A., Resende, M.G.C.: Greedy randomized adaptive search procedures. J. Glob. Optim. **6**(2), 109–133 (1995). doi:10.1007/BF01096763
9. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, New York (1979)
10. Glover, F., Laguna, M.: Tabu Search. Kluwer Academic, Dordrecht (1997)
11. Grosso, A., Locatelli, M., Pullan, W.: Simple ingredients leading to very efficient heuristics for the maximum clique problem. J. Heuristics **14**(6), 587–612 (2008). doi:10.1007/s10732-007-9055-x
12. Jerrum, M.: Large cliques elude the metropolis process. Random Struct. Algorithms **3**(4), 347–359 (1992). doi:10.1002/rsa.3240030402
13. Kopf, R., Ruhe, G.: A computational study of the weighted independent set problem for general graphs. Found. Control Eng. **12**, 167–180 (1987)
14. Li, C.M., Quan, Z.: Combining graph structure exploitation and propositional reasoning for the maximum clique problem. In: Proceedings of the 2010 22nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI'10, vol. 1,, Arras, France, pp. 344–351. IEEE Press, New York (2010)
15. Li, C.M., Quan, Z.: An efficient branch-and-bound algorithm based on maxsat for the maximum clique problem. In: Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI-10, pp. 128–133. AAAI Press, Atlanta (2010)
16. Singh, A., Gupta, A.K.: A hybrid heuristic for the maximum clique problem. J. Heuristics **12**(1–2), 5–22 (2006). doi:10.1007/s10732-006-3750-x
17. Tomita, E., Kameda, T.: An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. J. Glob. Optim. **37**(1), 95–111 (2007)
18. Tomita, E., Seki, T.: An efficient branch-and-bound algorithm for finding a maximum clique. In: Proceedings of the 4th International Conference on Discrete Mathematics and Theoretical Computer Science, DMTCS'03, pp. 278–289. Springer, Berlin (2003)
19. Tomita, E., Sutani, Y., Higashi, T., Takahashi, S., Wakatsuki, M.: A simple and faster branch-and-bound algorithm for finding a maximum clique. In: Proceedings of the 4th International Conference on Algorithms and Computation, WALCOM'10, pp. 191–203. Springer, Berlin (2010)