An extended comparison of the best known algorithms for finding the unweighted maximum clique

Deniss Kumlander

Department of Informatics, Tallinn University of Technology,
Raja St.15, 12617 Tallinn,Estonia
kumlander@gmail.com
http://www.kumlander.eu

Abstract. This paper conducts an extended comparison of two best known at the moment algorithms for finding the unweighted maximum clique. This test is extremely important from both industry and theoretical perspectives. It will be useful for further developing of those algorithms as clearly demonstrated both algorithms advantages and disadvantages, while industry should consider tests result selecting the best algorithm to be applied in their particular environment.

Key words: maximum clique, vertex colouring, colour classes, branch and bound, graph theory

The paper is published by Springer-Verlag in Communications in Computer and Information Science, vol. 14: Modelling, Computation and Optimization in Information Systems and Management Sciences: Second International Conference MCO 2008. http://www.springer.com/computer/communications/book/978-3-540-87476-8

Reference: D. Kumlander, An extended comparison of the best known algorithms for finding the unweighted maximum clique, In: Communications in Computer and Information Science, vol. 14: Modelling, Computation and Optimization in Information Systems and Management Sciences: Second International Conference MCO 2008, Metz, France - Luxembourg, September 2008. Hoai An Le Thi , Pascal Bouvry, Tao Pham Dinh (eds), Springer-Verlag, 2008, p. 175 - 181

1 Introduction

Let $G = (V, E)$ be an undirected graph, where V is the set of vertices and E is the set of edges. Two vertices are called to be *adjacent* if they are connected by an edge. A *clique* is a complete subgraph of G, i.e. one whose vertices are pairwise adjacent. An *independent set* is a set of vertices that are pairwise nonadjacent. A *complement graph* is an undirected graph $G' = (V, E')$, where $E' = \{(v_i, v_j) | v_i, v_j \in V, i \neq j, (v_i, v_j) \notin E\}$ - this is a slightly reformulated definition provided by Bomze et al 1999 [2]. A neighbourhood of a vertex v_i is defined as a set of vertices, which are connected to this vertex, i.e. $N(v_i) = \{v_1, \ldots, v_k | \forall j : v_j \in V, i \neq j, (v_i, v_j) \in E\}$. A *maximal clique* is a clique that is not a proper subset of any other clique, in other words this clique doesn't belong to any other clique. The same can be stated about maximal independent set. The *maximum clique problem* is a problem of finding maximum complete subgraph of G, i.e. maximum set of vertices from G that are pairwise adjacent. In other words the maximum clique is the largest maximal clique. It is also said that the maximum clique is a maximal clique that has the maximal cardinality. The *maximum independent set problem* is a problem of finding the maximum set of vertices that are pairways nonadjacent. In other words, none of vertices belonging to this maximum set is connected to any other vertex of this set. A *graph-colouring problem* or a *colouring* of G is defined to be an assignment of colours to the graph's vertices so that no pair of adjacent vertices shares identical colours. So, all vertices that are coloured by the same colour are nothing more than an independent set, although it is not always maximal.

All those problems are computationally equivalent, in other words, each one of them can be transformed to any other. For example, any clique of a graph G is an independent set for the graph's complement graph G'. So the problem of finding the maximum clique is equivalent to the problem of finding the maximum independent set for a complement graph.

All those problems are NP-hard on general graphs [7] and no polynomial time algorithms are expected to be found. The maximum clique problem has many theoretical and practical applications. In fact, a lot of algorithms contain this problem as a subtask and this is another important applications area for the problem. The first area of applications is data analyses / finding a similar data: the identification and classification of new diseases based on symptom correlation [3], computer vision [1], and biochemistry [11]. Another wide area of applying the maximum clique is the coding theory [5, 12]. There are many others areas of the maximum clique application that makes this problem to be important.

2 Introduction into Branch and Bound Algorithms

First of all the branch and bound types of algorithms should be introduced in case the reader doesn't have enough knowledge since understanding of those is a crucial element to understanding main point of algorithms to be described later. Branch and bound type algorithms do analyse vertices one by one expanding those by selecting into the next level of analysis all vertices among remaining that are connected to the expanding one. This next level of expansion is normally named a depth, so initially, at the first depth all vertices of a graph are presented, i.e. $G_1 \equiv G$. Then the algorithm is executed and picks up a vertex one by one, so the first one to analyse will be v_{11}, where the indexes indicate that it is a version from the first depth and is also the first version on that depth. The algorithm will form the depth 2 by listening there all vertices connected to the algorithm considers all vertices adjacent to the v_{11} and belonging to G_1. Those vertices form a subgraph G_2. If G_2 is not empty then the first vertex of that depth will be expanded next - v_21. The depth 3 will contain all vertices from G_2 that are adjacent to v_{21} and by the selection logic those will also are adjacent to the vertex expanded on the first depth, i.e. v_{11}. Let v_{d1} be the vertex to be expanded at the depth d and G_d is a subgraph of G on a depth d that contains the following vertices: $V_d = (v_{d1}, v_{d2}, \ldots, v_{dm})$. Then a subgraph on the depth d+1 is $G_d + 1 = (V_{d+1}, E)$, where $V_{d+1} = (v_{(d+1)1}, \ldots, v_{(d+1)k}) : \forall i \;\; v_{(d+1)i} \in V_d$ and $(v_{(d+1)i}, v_{d1}) \in E$.

Continuing this way the algorithm will finally arrive to a depth where no vertices exist. Then the previous depth number is compared to the currently maximum clique size and all vertices expanding at the moment on all previous levels (those are called a forming clique) are saved as the maximum clique if it is larger. Anyway the analysis is returned to the previous level and the next vertex of that level should be expanded now. Let say that we returned to the d-th level and the previous expanded vertex is v_{d1}. Then the next vertex to be expanded will be v_{d2}. This should be continued as long as there are vertices on the depth. The algorithm should stop if all vertices of the first level are analysed.

The branch and bound algorithm by itself is nothing else than an exhaustive search and is very bad from the combinatorial point of view. Therefore it is always used together with a special check allowing to cut branches (cases) that cannot produce any better solution that the current maximum one. This check is called a pruning formula and the classical work [6] in the field of finding the maximum clique suggests using the following:

$$\text{if } d - m + n \leq CBC \tag{1}$$

where d is a depth, m is the current (under analyses) vertex index on a depth, n is the total number of vertices in the depth and CBC is the current maximum clique size. Actually it can be generalised into the following:

$$\text{if } d - 1 + Degree(G_d) \leq CBC \tag{2}$$

where $Degree$ equals to $n + 1 - m$ since d - 1 represents the number of vertices in the forming clique (expanded on previous levels) and $n + 1 - m$ the number of vertices can be potentially included into the clique (and called a degree of the depth/branch).

If this formula holds then the depth is pruned - it is not analysed further and the algorithm immediately returns to the previous depth.

3 Different Levels of Using a Vertex Colouring in Nowadays Algorithms

Here two algorithms to be reviewed that are using vertex colouring for finding the maximum clique on different levels. The first idea is to re-apply a heuristic vertex colouring on each new level of a branch and bound algorithm. Another idea is to apply the colouring only once before the branch and bound routine starts and then use results on the permanent base. There are two representatives of both ways nowadays, which are claimed to be quickest; therefore a comparison of those algorithms is worth to do to identify how different ways affects the performance in different cases to be solved.

3.1 Re-applying a Heuristic Vertex Colouring

This subchapter algorithm is developed by Tomita and Seki [13]. Both this and the next chapter algorithms use the same idea - any colour class is an independent set and therefore no more than one vertex from each colour class can participate in a clique. The pruning formula for the algorithm is still the same:

$$\text{if } d - 1 + Degree(G_d) \leq CBC \tag{3}$$

but *Degree* here represents the number of existing colour classes (independent sets). The number of existing colour classes is obviously much better estimation than the number of remaining vertices. Therefore the number of analysed subgraphs decreased dramatically. Obviously the exact colouring cannot be used as it is a task of the same complexity as the maximum clique finding one. Therefore a heuristic colouring is used. The main difference between this algorithm and the next one is an approach to calculating the number of existing colours. This subchapter algorithm finds the heuristic vertex-colouring on each new depth. Besides, it reorders vertices after finding the colouring by colour index in decreasing order. Therefore, instead of calculating the degree each time a new vertex is expanded, the expanding vertex colour index is used as a degree. The pruning formula is reformulated into the following one:

$$\text{if } d - 1 + colour\_index(m) \leq CBC \tag{4}$$

where d is a depth, m is the current (under analyses) vertex index on a depth, and CBC is the current maximum clique size.

3.2 Re-using a Heuristic Vertex Colouring

Here we present an algorithm that obtains a vertex colouring only once and then re-use during its work. The algorithm was developed by Kumlander [10] independently and simultaneously with the previous one. The first step of the algorithm is to obtain a heuristic vertex colouring and re-order vertices by colour classes, so that colour classes will appear one by one in the new vertices order. The algorithm uses the vertex colouring to apply two pruning rules - the direct one and the backtracking one. The backtracking search described below cannot be used for the previous class of algorithm re-colouring on each depth as the backtracking relies on a fixed vertices ordering. Therefore it is a natural part of algorithms from the re-using class. The direct pruning rule is defined using a degree function, which equals to the number of existing colour classes on a depth. The algorithm prunes also:

$$\text{if } d - 1 + Degree \leq CBC \tag{5}$$

where d is a depth, $Degree$ is the depth (subgraph) degree, which is the number of existing colour classes and CBC is the size of the current maximum clique. This algorithm calculates the degree by examining what colour classes exist on a depth. Actually the degree is calculated only ones when the depth is formed and later only adjusted by decreasing on one when the next vertex to be analysed is from another colour class than the previous one. This improves the performance dramatically.

In fact the fixed ordering lets also to apply here one more technique: the backtracking search. It examines the graph vertices in the opposite to the standard branch and bound algorithm's order. The classical vertex level backtracking considers first of all all cliques that could be built using only v_n, then all cliques that could contain v_{n-1} and v_n, and so forth. The general rule - it considers at the i-th step all cliques that could contain$\{v_i, v_{i+1}, v_{i+2}, \ldots, v_n\}$. The core idea here is to keep in memory the size of the maximum clique found for each i-th step (i.e. i-th vertex at the highest level) in a special array b. So $b[i]$ is the maximum clique for the i-th vertex while searching backward. This allows employing one more pruning formula:

$$\text{if } d - 1 + b[m] \leq CBC \tag{6}$$

Besides the algorithm can stop the backtracking iteration and go to the next one if a new maximum clique is found. Colour classes can improve the backtracking by doing it on the colour classes' level instead of individual vertices. Lets say that vertices are coloured and sorted by colour classes, i.e. $V = \{C_n, C_{n-1}, \ldots, C_1\}$, where C_i is the i-th colour (or we call it the i-th colour class). The algorithm now considers first of all all cliques that could be built using only vertices of the C_1, i.e. of the first colour class, then all cliques that could be built using vertices of C_1 and C_2, and so forth. The general rule - it considers at the i-th step all cliques of $\{C_i, C_{i-1}, \ldots, C_1\}$ vertices. The array b is also used, but the index here is the vertex colour index. So the pruning formula will be:

$$\text{if } d - 1 + b[colour\_index(m)] \leq CBC \tag{7}$$

The stopping condition remains since the maximum clique size of a subgraph formed by $\{C_i, C_{i-1}, \ldots, C_1\}$ is either equal to the maximum clique size of a subgraph formed by $\{C_{i-1}, \ldots, C_1\}$ or is larger on 1.

4 Tests

Here the described algorithms of both classes are analysed on DIMACS graphs, which are a special package of graphs used in the Second DIMACS Implementation Challenge [8, 9] to measure performance of algorithms on graphs having different, special structures.

As it has been mentioned earlier, there is a very simple and effective algorithm for the maximum clique problem proposed by Carraghan and Pardalos [6]. This algorithm was used as a benchmark in the Second DIMACS Implementation Challenge [9]. Besides, using of this algorithm as a benchmark is advised in one of the DIMACS annual reports [8]. That's why it will be used in the benchmarking below and is called the "base" algorithm. Results are presented as ratios of algorithms spent times on finding the maximum clique - so the same results can be reproduced on any platforms. Ratios are calculated using the benchmarking algorithm [6]. The larger ratio is the quicker a tested algorithm works as the ratio shows how much quicker the tested one is. The compared algorithms were programmed using the same programming language and the same programming technique. The greedy algorithm was used to find a vertex-colouring.

TS - time needed to find the maximum clique the base algorithm divided by time needed to find the maximum clique by the algorithm re-applying colour classes [13].

VColor-BT-u - time needed to find the maximum clique the base algorithm divided by time needed to find the maximum clique by the algorithm re-using colour classes [10].

Table 1. Benchmark results on DIMACS graphs

Graph name	Edge density	Vertices	Maximum clique size	*TS*	*VColor-BT-u*
brock200_2	0.50	200	12	2.3	*4.0*
brock200_3	0.61	200	15	*3.3*	3.2
hamming8-4	0.64	256	16	39.9	*7848.3*
johnson16-2-4	0.76	120	8	7.0	*20.9*
keller4	0.65	171	11	6.7	*11.8*
p_hat300-1	0.24	300	8	1.0	*1.3*
p_hat300-2	0.49	300	25	4.8	*6.6*
p_hat500_1	0.25	500	9	0.9	*1.5*
p_hat700_1	0.25	700	11	1.1	*1.9*
sanr400_0.7	0.70	400	21	1.4	*5.6*
2dc.256*	0.47	256	7	6.6	*14.5*

* - An original task for this graph is to find the maximum independent set, so the maximum clique is found from the complement graph.

For example, 4.8 in the column marked*TS* means that Tomita and Seki algorithm [13] requires 4.8 times less time to find the maximum clique than the base one. The quickest result of each row is highlighted by the italic font. Presented results show that the *VColor-BT-u* algorithm [10] outperforms the other in most cases. Both reviewed in the paper algorithms are faster than the benchmarking algorithms.

The next test will be conducted on random graphs from densities from 10% to 90% with a step of 10%. 100 instances of graphs have been generated per density and an average ratio is found per algorithm. Here you can see that *VColor-BT-u*

Table 2. Benchmark results on random graphs

Edge density	Vertices	*TS*	*VColor-BT-u*
0.10	1300	0.8	1.0
0.20	1000	1.4	1.3
0.30	600	1.9	1.5
0.40	500	2.7	1.7
0.50	300	3.3	2.3
0.60	200	5.4	3.5
0.70	150	10.8	5.6
0.80	100	40.9	16.2
0.90	80	200.6	102.1

looses to Tomita and Seki algorithm practically on all densities.

5 Conclusion

In this paper two currently best known algorithms for finding the unweighted maximum clique are described and what is more important are compared on different graph types. Both algorithms are branch and bound and both are using colour classes obtained from a heuristic vertex-colouring to find the maximum clique. The main difference is the method of using the colouring. One of those keep re-colouring the graph for each depth formed during the algorithm work and the second does it only once before the core part of the algorithm is executed. The first looses in spending time each time re-colouring and cannot employ backtracking search, while the second looses in precision of colouring the deeper the depth is. Therefore both algorithms have certain disadvantages been both reported as the best known. That is why the comparison test was interested for the industry and theory. Tests were conducted for both DIMACS graphs representing certain important graph types and for randomly generated graphs. The general result is that the re-using colouring algorithm [10] is the better technique is most cases for DIMACS graphs and the re-applying colouring algorithm [13] have shown superb results on random graphs. This let us to conclude that there is no clear winner and tests conducted in the paper should be carefully revisited selecting one or another algorithm to be applied basing on the particular environment it should happen in.

References

1. Ballard, D.H., Brown, M.: Computer Vision. Prentice-Hall, Englewood Cliffs, NJ (1982)

2. Bomze, M., Budinich, M., Pardalos, P.M., Pelillo, M.: The maximum clique problem. Handbook of Combinatorial Optimization, vol. 4, In D.-Z. Du and P. M. Pardalos, eds. Kluwer Academic Publishers, Boston, MA (1999)
3. Bonner, R.E.: On some clustering techniques. IBM J. of Research and Development 8, 22–32 (1964)
4. Brelaz, D.: New Methods to Color the Vertices of a Graph. Communications of the ACM 22, 251–256 (1979)
5. Brouwer, A.E., Shearer, J.B., Sloane, N.J.A., Smith, W.D.: A new table of constant weight codes. J.IEEE Trans. Information Theory 36, 1334–1380 (1990)
6. Carraghan, R., Pardalos, P.M. An exact algorithm for the maximum clique problem. Op. Research Letters 9, 375–382 (1990)
7. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-completeness. Freeman, New-York (2003)
8. DIMACS, Center for Discrete Mathematics and Theoretical Computer Science. Annual Report, December (1999)
9. Johnson, D.S., Trick, M,A,, eds.: Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol 26. American Mathematical Society (1996)
10. Kumlander, D.: Some practical algorithms to solve the maximum clique problem. Tallinn University of Technology Press, Tallinn (2005)
11. Miller, W.: Building multiple alignments from pairwise alignments. Bioinformatics, 169–176 (1992)
12. Sloane, N.J.A.: Unsolved problems in graph theory arising from the study of codes. Graph Theory Notes of New York XVIII, 11–20 (1989)
13. Tomita, E., Seki, T.: An effcient branch-and-bound algorithm for finding a maximum clique. In: Discrete Mathematics and Theoretical Computer Science, 4th International Conference, DMTCS 2003, pp. 278–289. LNCS, vol. 2731, Springer, Berlin (2003)