On Importance of a Special Sorting in the Maximum-Weight Clique Algorithm Based on Colour Classes

Deniss Kumlander

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

Abstract. In this paper a new sorting strategy is proposed to be used in the maximum weight clique finding algorithm, which is known to be the fastest at the moment. It is based on colour classes, i.e. on heuristic colouring that is used to prune efficiently branches by excluding from the calculation formulae vertices of the same colours. That is why the right ordering before colouring is so crucial before executing the heuristic colouring and consequently the main maximum weight clique searching routine. Computational experiments with random graphs were conducted and have shown a sufficient increase of performance considering the type of application dealt with in the article.

1 Introduction

Let $G = (V, E, W)$ be an undirected graph, where V is the set of vertices, E is the set of edges and W is a set of weights for each vertex. A *clique* is a complete subgraph of G, i.e. one whose vertices are pairwise adjacent. The *maximum clique problem* is a problem of finding maximum complete subgraph of G, i.e. a set of vertices from G that are pairwise adjacent. An *independent set* is a set of vertices that are pairwise nonadjacent. A *graph colouring problem* is defined to be an assignment of colour to its vertices so that no pair of adjacent vertices shares identical colours. The *maximum-weight clique problem* asks for a clique of the maximum weight. The *weighted clique number* is the total weight of weighted maximum clique. It can be seen as a generalization of the maximum clique problem by assigning positive, integer weights to the vertices. Actually it can be generalized more by assigning real-number weights, but it is reasonable to restrict to integer values since it doesn't decrease complexity of the problem. This problem is well known to be NP-hard.

The described problem has important economic implications in a variety of applications. In particular, the maximum-weight clique problem has applications in combinatorial auctions, coding theory [1], geometric tiling [2], fault diagnosis [3], pattern recognition [4], molecular biology [5], and scheduling [6]. Additional applications arise in more comprehensive problems that involve graph problems

H.A. Le Thi, P. Bouvry, and T. Pham Dinh (Eds.): MCO 2008, CCIS 14, pp. 165–174, 2008.

with side constraints. More this problem is surveyed in [7]. In this paper a modification of the best known algorithm for finding the maximum-weight clique is proposed. The paper is organised as follows.

The section 2 describes in details the algorithm to be extended by a new ordering strategy, so readers can understand an essence of the change and the final result. The following section describes the new idea and presents algorithms. The section 4 contains information about conducted tests. The last section concluded the paper and describes open problems.

2 Description of the Algorithm to Be Extended

This section contains a description of an algorithm known to be the best at the moment [12] in finding the maximum weight clique. It is the algorithm that is about to be improved in the paper and therefore it will be described in quite details in order to understand the improvement idea and will be called a base algorithm in the entire text of the paper. The base algorithm is a typical branch and bound algorithm and is a mix of the classical approach proposed by Carraghan and Pardalos in 1990s [8,9] and of a backtracking strategy proposed by Ostergard [10].

2.1 Branch and Bound Routine and Pruning Using Colour Classes

Crucial to the understanding of the branch and bound algorithms is a notation of the *depth* and *pruning formula*. Initially, at the depth 1 we have all vertices of a graph, i.e. $G_1 \equiv G$. Now the algorithm is going to pick up vertices one by one and form a set of vertices that are connected to it forming a new lower depth. This process is normally called *expanding a vertex* and is repeated for each depth. Notice that only vertices existing on the current depth can be promoted to the lower one. Repeating this routine we always will have a set of vertices on the lowest depth that are connected to vertices selected (expanded) on previous depths. Moreover all vertices expanded on different depth are also connected to each other by the expansion logic. That is a way the maximum clique is formed. The more formal illustration will be the following. Suppose we expand initially vertex v_{11}. At the next depth the algorithm considers all vertices adjacent to the vertex expanded on the previous level, i.e. v_{11} and belonging to G_1. Those vertices will form a subgraph G_2. At the depth 3, we consider all vertices (that are at the depth 2, i.e. from G_2) adjacent to the vertex expanded in depth 2 etc. Let v_{d1} be the vertex we are currently expanding at the depth d. That is:

Let's say that G_d is a subgraph of G on a depth d that contains the following vertices: $V_d = (v_{d1}, v_{d2}, \ldots, v_{dm})$. The v_{d1} is the vertex to be expanded. 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 $(v_{(d+1)i}, v_{d1}) \in E$.

As soon as a vertex is expanded and a subgraph, which is formed by this expansion, is analysed, this vertex is deleted from the depth and the next vertex of the depth become active, i.e. will be expanded. This should be repeated until

there are vertices that are not analysed and then the algorithm returns to the higher level. The algorithm should stop if all vertices are analysed on the first level.

The branch and bound algorithm by itself is nothing else than an exhaustive search and is very pure from the combinatorial point of view. Therefore it is always accomplished by a special analyses that identifies whether the current depth could produce a bigger clique that the already found one. Such analysis is normally done by so called pruning formula. If $W(d) + Degree(G_d) \leq CBCW$, where $CBCW$ is a size (weight) of the current maximum weight clique, $W(d)$ is a sum of weights of vertices expanded on previous to d depths and $Degree$ is function that defines how much larger the forming clique can become using vertices of the depth (i.e. vertices forming G_d). If this formula holds then the depth is pruned - it is not analysed further and the algorithm immediately returns to the previous level. The main art of different algorithm of this class is setting how the degree function works. The classical approach [8] will just sum up weight of remaining vertices of the depth. The modification made in the base algorithm [12] is applying colour classes. A vertex colouring is found before running the main algorithm and only the highest weight vertex of each colour is included into the degree function calculation during the main algorithm work applying the fact that no more than one vertex of each colour class (independent set) can be included into any clique. Please check the original work for any proves of the previously stated and for more details of the described approach.

2.2 Backtracking and Colour Classes

A backtracking process is widely known in different types of combinatorial algorithm including one proposed by P. Ostergard [10]. The algorithm starts to analyse vertices in the backward order by adding them one by one into analyses on the highest level instead of excluding as others do (although the lower levels work still the same was as the branch and bound one). The main idea of the algorithm is to introduce one more pruning formula - for each vertex starting from the last one and up to the first one a function $c(i)$ is calculated (i is a vertex number), which denotes the weight of the maximum-weight clique in the subgraph induced by the vertices $\{v_i, v_{i+1}, \ldots, v_n\}$. In other words $c(i)$ will be a maximum-weight clique that can be formed using only vertices with indexes are starting from i. So, the original backtracking search [10] algorithm will define that $c(n)$ equals to the weight of v_n and $c(1)$ is the weight of the maximum-weight clique for the entire graph. Obviously the following new pruning formula can be introduced in the backtracking search using the calculated function: if $W(d) + c(i) \leq CBCW$, where $CBCW$ is still a size (weight) of the current maximum weight clique and $W(d)$ is a sum of weights of vertices expanded on previous to d depths.

Colour classes also improve this idea as well it was demonstrated in the base work [12]. The idea is to calculate c function (actually an array) by colour classes

instead of individual vertices. Lets say that the graph colouring before the main algorithm has produced the set of colours $\{C_1, C_2, \ldots, C_n\}$ and vertices are reordered accordingly to their colours. Now, $c(n)$ will equal to the largest weight vertex of $\{C_n\}$, $c(1)$ is still the weight of the maximum-weight clique for the entire graph and $c(i)$ is the weight of the maximum-weight clique in the subgraph induced by the vertices $\{C_i, C_{i+1}, \ldots, C_n\}$. The pruning formula remains the same although the i indicates now the colour class index of the examined vertex instead of the vertex index. Notice that the backtracking order base on the fixed ordering, so vertices colouring and reordering should be done before starting the backtracking order.

3 New Algorithm Including Sorting Strategy

3.1 Sorting

Sorting always played quite a crucial role in many algorithms. Unfortunately the right ordering doesn't guarantee that the final solution could be obtained immediately in problems like finding the maximum-weight clique (at least in nowadays algorithms). The reason is simple - the answer should be proved by revising all other vertices and cases. So even if a solution is obtained during the first search iteration it still takes long to conclude that the already found clique is the maximum one. Despite of this the sorting is still important since could sufficiently affect the performance of an algorithm. Moreover some algorithms use sorting as a core element of their structure in the maximum clique finding routine. The base algorithm only recommends sorting vertices by weights inside each colour class in the decreasing order. This sorting lets just pick up the last vertex per each colour on whatever depth calculating the degree function since ensures that it will always be the maximum weight one among all vertices remaining on that depth in that colour class. This sorting by itself is a sufficient part of the algorithm, but this paper is about to extend this sorting strategy in order to improve the overall efficiency of the algorithm. The complexity produced by introducing into the maximum clique task weights lays first of all in the sufficient variation of weights among vertices. This variation produces situation when one vertex been included into the forming clique gives much more that a set of others. As it was mentioned earlier describing the base algorithm the degree function calculation is conducted basing on the highest weight vertex that appears in each class among remaining in the subgraph on the depth. Therefore a sufficient distribution of high weight vertices among different classes can sufficiently increase the degree calculation result. At the same time, if any algorithm will be able to propose how we could group high weight vertices into same colour classes then we would improve the degree function as one high weight vertex will cover other, similar high weight vertices - once again only the highest weight vertex is used in calculation by the algorithm logic per colour class. Notice that the task formulated earlier is not a pure sorting one, since it should improve the search

basing on colouring. So the heuristic colouring task is the main constraint here. The desired order should appear after the algorithm has:

1. Defined initial sorting
2. Coloured vertices

It is quite common that the colouring is the task that will sufficiently change the order. Therefore we cannot talk here about a precise ordering, but should say "a probabilistic one", i.e. such ordering that will keep the desired ordering after the heuristic colouring is applied with a certain probability. Notice the term heuristic in the previous sentence. Our analysis of the base algorithm source code, which is published in Internet [13], have shown that the initial proposed ordering by weights is dramatically broken by the colouring strategy during which a vertex to be coloured is always moved to the end of the uncoloured vertices line by swapping, so initial positions of the high weights' vertices are lost just after some colouring iterations. This paper proposes that the colouring should be done in such a way that the ordering is kept as long as possible.

The order direction - increasing or decreasing is another interesting topic. Notice that the key technique of the base algorithm is moving backward in the backtrack search. Generally the backtracking algorithm works better if the larger clique is found right in the beginning therefore the paper suggests to order vertices so that the last colour classes (from which the backtracking search will start) will include the higher weight vertices in average.

3.2 Colouring Algorithm for the Maximum Weigh Clique Algorithm

It is well known that the number of colour classes can be sufficiently larger than the size of the maximum clique. That is why most best known algorithms [10,12] are using a greedy colouring as a heuristic one - there is no points to spend time on more precise colouring since even the best colouring will not guarantee to give a number that will be close to the maximum clique size. At the same time the earlier stated wish to keep the initial ordering by weights force us to propose the following algorithm:

Algorithm for the ordering and colouring

Variables:
N - the number of vertices
a - an array with an initial ordering of vertices: a_i contains a vertex number been in the i-th position of that vertices ordering
b - the new ordering after colouring
C_i - a set of vertices coloured by the i-th color

Operations:
!= - a comparison operation called "not equal"
== - a comparison operation "equals"

Step 1. Initial sorting:
Sort vertices by weights in the increasing order producing an ordering array a

Step 2. Initialise:
$i := 0$
$m := N$

Step 3. Pick up a colour:
$i := i + 1$

Step 4. Colour:
For $k := N$ downto 1
If $a_k! = 0$ & there is no such $j : v_j \in C_i, (a_k, v_j) \in E$ then
$a_k := 0, b_m := a_k, m := m - 1, C_i := C_i$ U b_m

if $m == 0$ then go to the "Final sorting" step
Next
Go to step 3

Step 5. Final sorting:
Re-order vertices inside each colour class in the increasing order by weights.

End: Return the new order of vertices b and colouring C.

3.3 Maximum Weigh Clique Algorithm

Algorithm for the maximum - weight clique problem

$CBCW$ - weight of the current best (maximum-weight) clique
d - depth
G_d - subgraph of G formed by vertices existing on depth d and is induced by E
$W(d)$ - weight of vertices in the forming clique
$w(i)$ - weight of vertex i

Step 0. Sorting and colouring (See the above algorithm):
Sort vertices by weights in the increasing order.
Find a vertex colouring starting from the highest weight vertices. Keep the order of uncoloured vertices.
Re-order vertices inside each colour class in the increasing order by weights.

Step 1. Backtrack search runner:
For $n := NumberOfColourClasses$ downto 1
Goto step 2
$c(n) := W$
Next
Go to End

Step 2. Initialization: Form the depth 1 by selecting all vertices belonging to colour classes with an index greater or equal to n.
$d := 1$.

Step 3. Prune: If the current level can contain a larger clique than already found:
If $W(d) + Degree(Gd) \leq CBCW$ then go to step 7.

Step 4. Expand vertex: Select the next vertex to expand on a depth. If all vertices have been expanded or there is no vertices then control if the current clique is the largest one. If yes then save it (including its size as $CBCW$) and go to step 7.
Note: Vertices are examined starting from the first one on the depth.

Step 5. Prune: If the current level can contain a larger clique than already found:
If expanding vertex colour class index $<> n$
If $W(d) + c$(expanding vertex colour class index) $\leq CBCW$ then go to step 7.

Step 6. The next level: Form the new depth by selecting vertices that are connected to the expanding vertex from the current depth among remaining;
$W(d+1) := W(d) + w$(expanding vertex index)
$d := d + 1$;
Go to step 2.

Step 7. Step back:
$d := d - 1$;
if $d == 0$, then return to step 1
Delete the expanded vertex from the analysis on this depth;
Go to step 2.

End: Return the maximum-weight clique.

Note: It is advisable to use a special array to solve the order of vertices to avoid work by changing adjacency matrix during reordering vertices. Besides, instead of removing vertices from a depth, it is advisable to have a cursor that moves from the first vertex on a depth to the last one. All vertices that are in the front of the cursor are in the analyses, while vertices behind the cursor are excluded from it (already analysed).

4 Computational Results and Discussion

It is common to apply tests on two types of case: randomly generated and standard (like for example the DIMACS package for unweighted case of finding

maximum clique problem). Unfortunately there is no such widely adopted standard package for the weighted case although application of maximum clique with weights plays no less important role in industry and health care. Therefore tests to be conducted in this paper will be restricted to randomly generated graphs.

Several algorithms were published since 1975s. The easiest and effective one was presented in an unpublished paper by Carraghan and Pardalos [8]. This algorithm is nothing more that their earlier algorithm [9] for the unweighted case applied to weighted case. They have shown that their algorithm outperforms algorithm their have compared with. Another work, which is quite widely referenced in different sources as the best was published by P. Ostergard [10]. He also has compared his algorithm with earlier published algorithms and has shown his algorithm works better by the publishing time. The last algorithm to be used in the tests in the base one [12] that was described in details earlier. In order to produce comparison results a set of instances where generated and each instance was given to each algorithm and their spent time (on producing a solution) was measured. The table below demonstrates that tests were conducted from densities from 10% to 90% with a step 10%. For each vertices/density case 1000 instances of graphs were generated. Results are presented as ratios of algorithms spent times on finding the maximum clique. Although this presentation is slightly different from common it has one sufficient advantage from our point of view - it gives platform independency, so the same results can be reproduced on any computer and ratios should stay the same. The compared algorithms were programmed using the same programming language and the same programming technique (since all algorithms are quite similar). The greedy algorithm was used to find a vertex-colouring.

PO - time needed to find the maximum-weight clique by Carraghan and Pardalos algorithm [8] divided by time needed to find the maximum-weight clique by P. Ostergard algorithm [10] - an average ratio.

$VColor - BT - w$ - time needed to find the maximum-weight clique by Carraghan and Pardalos algorithm [8] divided by time needed to find the maximum-weight clique by the base [12] algorithm - an average ratio.

New - time needed to find the maximum-weight clique by Carraghan and Pardalos algorithm [8] divided by time needed to find the maximum-weight clique by the new algorithm - an average ratio.

The following table is constructed in such a way to guarantee that each algorithm execution will take at least one second and no more than one hour. That is why the vertices count locates in the second column of the table below - the number of vertices is a dependent parameter (on the density) and is chosen by the time constraint. As the result the number of used vertices the smaller the higher density is. At the same additional tests have shown no sufficient change of results on other number of vertices for each density and it proved from our point results independency from the number of vertices we actually use.

For example, 38.62 in the column marked New means that Carraghan and Pardalos [8] algorithm requires 38.62 times more time to find the maximum-weight clique than the new algorithm proposed in this paper. Presented results

Table 1. Benchmark results on random graphs

Edge density	Vertices	*PO*	*VColor-BT-u*	*New*
0.1	1000	1.01	1.26	1.40
0.2	800	1.25	2.11	2.93
0.3	500	1.58	2.64	3.93
0.4	300	1.71	3.02	4.61
0.5	200	1.78	3.41	5.87
0.6	200	2.07	6.53	10.42
0.7	150	2.37	10.16	18.25
0.8	100	2.98	17.36	38.62
0.9	100	4.51	79.80	293.64

show that the new algorithm performs very well on any density. It is faster than all algorithms we compare with. Especially good results are shown on the dense graphs, where the new algorithm is faster than the Carraghan and Pardalos algorithm [8] in 293 times and than the best known algorithm [12] circa 3 times.

5 Conclusion

In this paper a new fast algorithm for finding the maximum-weight clique is introduced. The algorithm is based on the best know algorithm, which is a branch and bound one, uses a heuristic vertex-colouring in the pruning rules and a backtracking search by colour classes. The algorithm is always better than other best known algorithms that were used in the comparison test. Notice that unlike the unweighted case, the weighted case is much harder to improve the performance and therefore achieved results, like for example one on the dense graphs, where the new algorithm is 3 times faster than the best known algorithm and 300 times faster than the standard benchmarking base one is a remarkable result from our point of view.

References

1. MacWilliams, J., Sloane, N.J.A.: The theory of error correcting codes. North-Holland, Amsterdam (1979)
2. Corradi, K., Szabo, S.: A combinatorial approach for Keller's Conjecture. Periodica Mathematica Hungarica 21, 95–100 (1990)
3. Berman, P., Pelc, A.: Distributed fault diagnosis for multiprocessor systems. In: 20th Annual International Symposium on Fault-Tolerant Computing, Newcastle, UK, pp. 340–346 (1990)
4. Horaud, R., Skordas, T.: Stereo correspondence through feature grouping and maximal cliques. IEEE Transactions on Pattern Analysis and Machine Intelligence 11, 1168–1180 (1989)
5. Mitchell, E.M., Artymiuk, P.J., Rice, D.W., Willet, P.: Use of techniques derived from graph theory to compare secondary structure motifs in proteins. Journal of Molecular Biology 212, 151–166 (1989)

6. Jansen, K., Scheffler, P., Woeginger, G.: The disjoint cliques problem. Operations Research 31, 45–66 (1997)
7. Bomze, M., Budinich, M., Pardalos, P.M., Pelillo, M.: The maximum clique problem. In: Handbook of Combinatorial Optimization 4. Kluwer Academic Publishers, Boston (1999)
8. Carraghan, R., Pardalos, P.M.: A parallel algorithm for the maximum weight clique problem. Technical report CS-90-40, Dept of Computer Science, Pennsylvania State University (1990)
9. Carraghan, R., Pardalos, P.M.: An exact algorithm for the maximum clique problem. Op. Research Letters 9, 375–382 (1990)
10. Ostergard, P.R.J.: A new algorithm for the maximum-weight clique problem. Nordic Journal of Computing 8, 424–436 (2001)
11. Johnson, D.S., Trick, M.A. (eds.): Cliques, Colouring and Satisfiability: Second DIMACS Implementation Challenge. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 26. American Mathematical Society (1996)
12. Kumlander, D.: Some practical algorithms to solve the maximum clique problem. Tallinn University of Technology Press, Tallinn (2005)
13. Kumlander, D.: Network resources for the maximum clique problem, `http://www.kumlander.eu/graph`