Proceedings of the 10th WSEAS International Conference on COMPUTERS, Vouliagmeni, Athens, Greece, July 13-15, 2006 (pp938-943)

Improving the Maximum-Weight Clique Algorithm for the Dense Graphs

DENISS KUMLANDER
Department of Informatics
Tallinn University of Technology
Raja St.15, 12617 Tallinn
ESTONIA

Abstract: In this paper we present a fast algorithm for the maximum-weight clique problem on arbitrary undirected graphs, which is improved for the dense graphs. The algorithm uses colour classes and backtracking techniques inside itself in parallel in a form of a branch and bound algorithm. The algorithm contains also several improvements for the most complex case, which are dense graphs. Computational experiments with random graphs show that the proposed algorithm works faster than earlier published algorithms - more than 10 times faster than the best known algorithms. This difference is very sufficient for the maximum-weight clique problem.

Key-Words: maximum-weight clique, backtracking, heuristic vertex-colouring, colour classes

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 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 with side constraints. More this problem is surveyed in [7].

In this paper a new algorithm for finding the maximum-weight clique is introduced. In the following section the algorithm is described in details and in the later section it is benchmarked by some algorithms that are reported to be the best at the moment. Random graphs are used for that, so that the same graphs are given to each algorithm and then the speed of finding the maximum-weight clique is compared. Unfortunately the DIMACS test graphs are not weighted and therefore cannot be applied for testing. The last section concluded the paper and describes open problems.

2 Description of the Algorithm

This section contains description of an algorithm proposed in this paper. An idea of using colour classes fro the pruning of the branch and bound algorithm is introduced in this first sub-chapter for the unweighted case. The unweighted case makes possible to show an essence of the idea by distancing from the weights that complicates this idea a bit. Later the colouring classes pruning idea is demonstrated for the weighted case. An important part of the new algorithm is presented after that, which describes the backtracking search basing on the colour classes. This is another pruning strategy, which is used in parallel with the previous one. Finally a formal description of the algorithm is presented.

2.1 Initial idea of using colour classes

The algorithm uses an elementary property of a clique: vertices that are unadjacent cannot be included into the same clique. The following explains how this property is used for finding the maximum clique (un-weighted case). Later the same principles will be applied to produce the maximum-weight clique finding algorithm.

Before starting the algorithm we find a vertex-colouring by using any heuristic algorithm, for example in a greedy manner. We determine colour classes one by one as long as uncoloured vertices exist. The vertices are resorted in the order they are added into colour classes. This order affects the algorithm's performance in finding the maximum clique and therefore is very important.

Definition 1: A colour class is a set of vertices, which were coloured by the same colour during applying a vertex-colouring algorithm.
Note: A similar definition has been proposed by West in 2001, who defined the colour class as the following: vertices receiving a particular label (colour) for a colour class.

Definition 2: A colour class is called existing on a subgraph G_p if any vertex from this colour class belongs to this subgraph G_p.

Definition 3: Degree of a subgraph G_p equals to the number of colour classes existing on that subgraph.

Crucial to the understanding of the algorithm is a notation of the depth and pruning formula. Basely, at the depth 1 we have all vertices, i.e. $G_1 \equiv G$. We are going to expand all vertices of a subgraph so that vertex is deleted from the subgraph after it is expanded. Another way is to have a cursor pointing to the vertex under analyses, so vertices in the front of that are excluded from the analyses / a subgraph of the current depth. Suppose we expand vertex v_1. At the depth 2, we consider all vertices adjacent to v_1 from the previous depth vertices, i.e. belonging to G_1. Those vertices form a subgraph G_2. At the depth 3, we consider all vertices (that are at the depth 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. In that case 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.

The pruning formula is the next: If $d - 1 + Degree(G_d) \leq CBC$, where CBC is a size of the current maximum clique then we prune, since the size of the largest possible clique (formed by expanding any vertex of G_d) would be less or equal to CBC. If we are at depth 1 and this inequality holds then we stop; we have found the maximum clique.

2.2 Colour classes and the maximum-weight clique

The previously described branch and bound algorithm is the base for the maximum-weight algorithm with the following changes. We cannot determine values of the function *Degree* as a number of existing colour classes on a subgraph since vertices have different weights. Therefore a degree of a subgraph will be calculated as a sum of maximum weights of each colour class existing on this subgraph: for each existing lass we have to find a vertex of the maximum-weight and then sum up weights of those vertices.

The order of vertices here becomes crucial here from the performance point of view. The difference in vertices weights increase complexity of the problem a lot. The colour classes approach could decrease the number of considered weights as vertices belonging to the same colour class cannot form a clique. Therefore it is important to sort vertices by weights before applying heuristic vertex colouring (in our case the greedy one) to increase probability that similar weights will group in colour classes decreasing the difference of weights we need to consider. Thereafter vertices should be resorted after applying the colouring by weights inside each colour class in decreasing order. This order affects the degree calculation and recalculation algorithms.

Definition 4: A degree of a subgraph equals to the sum of the largest vertex' weights of each colour class existing on the subgraph independent on which vertices of a colour class exists on this subgraph.

The new order reformulates the degree calculation rule (considering only existing vertices of each colour class) – the degree always equals to the sum of the first vertex of each existing colour class on the subgraph. Instead of calculating the degree each time on a subgraph, while the algorithm analyse and eliminate from the analyses vertices one by one, it should be calculated only once and later

adjusted by the following rule: if the next vertex on this depth to be expanded is from the same colour class as the previous one then degree should be decreased remains the same otherwise should be decreased on a weight of the previous vertex (there is no more vertices from the previous vertex' colour class and the previous vertex weight was the largest in that colour class by resorting).

The pruning formula for the maximum-weight algorithm is the following: we can prune the search if $W' + Degree() \leq W$, *where* W' *is* the total weight of the forming current clique and W is the current maximum-weight clique.

We can prove, that this pruning formula can be applied, by the following theorem:

Theorem 1: If a degree of a subgraph of G formed by vertices existing connected to a forming clique vertices and induced by E is smaller or equal to the size of the current maximum-weight clique minus the total weight of the forming clique then this subgraph cannot form a clique, which is larger than the already found.

Prove: The total weight of the forming clique represent a weight accumulated so far by vertices including into the forming clique. Vertices remaining on the subgraph are vertices that potentially can be included into the clique as those are connected to all vertices of the forming clique by the branch and bound algorithm logic. It will be possible to find a larger clique than the already found one if and only if this subgraph can contain a clique, which is larger than a size of the current maximum-weight clique minus the total weight of the forming clique. If such clique exists then the maximal clique of the graph G will be the clique of the subgraph plus vertices selected on previous depths, i.e. vertices of the forming clique. So, the only statement we need to prove is: the *Degree* function's value of the subgraph is never smaller than the maximum-weight clique's weight that can be found on the subgraph. The degree function examines what colour classes exist on a subgraph, selects the maximum weight vertex among existing vertices from each existing class and sum up those by definition. Each colour class is an independent set by definition, therefore no more than one vertex of each colour class can participate in the maximum-weight clique. Let's say that the maximum-weight clique of the subgraph is composed from the vertices having the maximum weight of each class. Then this clique equals to the degree function described above. Otherwise we will have to include into a clique vertices that are smaller by weight than the maximum one and the total weight will also be smaller. It shows that the degree function always shows the maximum possible weight that we can achieve, i.e. is never smaller than the maximum-weight clique existing on a subgraph and the theorem inequality holds. ■

2.3 Backtracking by colour classes

A backtracking process is widely known in different types of combinatorial algorithm. It can be illustrated for the branch and bound type algorithms by considering a description of P. Östergård [10] algorithm, which is claimed to be the fastest at the moment. In the algorithm values of 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 $\{v_i, v_{i+1}, \ldots, v_n\}$. Obviously *c(n)* = weight of v_n and *c(1)* is the weight of the maximum-weight clique. For each vertex starting from the last one and up to the first one a backtrack search is carried out to find *c(i)*. The backtracking search means that the algorithm considers first of all all maximum-weight cliques composed by the $\{v_n\}$ vertex. Thereafter it considers cliques composed by $\{v_{n-1}, v_n\}$ and so forth. Those *c(i)* values are used to prune the search of the maximum-weight clique. As we search for a clique with weight greater than W, if the total weight of the forming current clique vertices is W' and we consider v_i to be the next expanded vertex then we can prune the search if $W' + c(i) \leq W$. Please refer to P. Östergård original work [10] for prove of the backtracking search algorithm's correct work.

A new idea proposed in this paper is a backtracking search by colour classes. In other words, colour classes can be used instead of individual vertices to carry the backtracking search. The algorithm considers first of all all cliques composed by vertices of the last colour class $\{C_n\}$. Thereafter cliques composed by the last and the previous to the last colour classes and so forth. Notice the last colour class is the first found colour class by reordering. Values of the function *c(i)* is calculated (i is a colour class number) which denotes 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. Notice that the backtracking order base on the fixed ordering, so vertices colouring and reordering should be done before starting the backtracking order.

2.4 Algorithm

Notice that is advisable to use a special array to solve the order of vertices to avoid the work of changing the adjacency matrix during vertices reordering. 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 after the cursor are excluded from it.

W – 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. Vertex-colouring:
Reorder vertices by weights
Find a vertex colouring
Reorder vertices inside each colour class if the original ordering is broken.

Step 1. Back track 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. Control: If the current level can contain a larger clique than already found:
If $W(d) + Degree(G_d) \leq W$ 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 and go to step 7.
Note: Vertices are examined starting from the first one on the depth.
Step 5. Control: 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*$) \leq W$ 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 analyze on this depth;
Go to step 2.
End: Return the maximum-weight clique.

3 Computational Results and Discussion

Usually two types of test cases are used: randomly generated graphs and fixed instances like the DIMACS test graphs. Unfortunately for the later type such instances are lacking for the maximum-weight clique problem. The DIMACS graphs are not weighted and can therefore not used for our testing. That's why only random graphs are tested. For each vertices/density case 100 cases were generated and average time was measured.

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. Recently one more algorithm was published by P. Östergård [10]. He also has compared his algorithm with earlier published algorithms and has shown his algorithm works better. Besides, those two algorithms were used as a base for our new 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. The compared algorithms were programmed using the same programming language and the same programming technique (since the new and P. Östergård algorithms are just modifications of Carraghan and Pardalos algorithm). 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. Östergård algorithm [10] – 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.

Note that the density parameter is shown first of all and only then the number of vertices since the second parameter depends on the first one as we stated earlier – the number of vertices is chosen so, that the time spent on finding the maximum clique for a corresponding density is no less than 2-3 seconds and also is not too big (in our experiments no more than 1 hour). That's why the lower density is, the more vertices are in use.

Table 1. Benchmark results at random graphs

Edge density	Vertices	*PO*	*New*
0.1	1000	1.10	1.28
0.2	800	1.21	1.72
0.3	500	1.40	2.64
0.4	300	1.59	3.47
0.5	200	1.68	4.45
0.6	200	1.82	6.80
0.7	150	2.10	12.11
0.8	100	2.07	18.33
0.9	100	11.25	155.57

For example, 6.80 in the column marked *New* means that Carraghan and Pardalos [8] algorithm requires 6.80 times more time to find the maximum-weight clique than the new algorithm. Presented results show that the new algorithm performs very well on any density. It is faster than both algorithms we compare with. Especially great results are shown on the dense graphs, where the new algorithm is faster than the Carraghan and Pardalos algorithm [8] in 155 times and than P. Östergård algorithm [10] in 14 times.

4 Conclusion

In this paper the new fast algorithm for finding the maximum-weight clique is introduced. The algorithm is the branch and bound one and uses a heuristic vertex-colouring in the pruning rules. The backtracking search on colour classes' level in the main proposition of this paper and the performance of this improvement grows with density growing. The direct pruning by colour classes is also the algorithm feature, which is not widely used so far although was introduced by us in some later papers [12]. A superposition of those techniques produces a very fast result from the performance point of view. The algorithm is always better than other best known algorithms that were used in comparison test. Unlike unweighted cases, the weighted case is much harder to achieve a sufficient difference and therefore the dense graph case, where the difference is more than 10 times is especially remarkable result of this paper. Another advantage of the algorithm is it's simplicity to implement and study.

Notice also that there are two main approaches of using heuristic vertex colouring for finding the maximum clique. The first one finds colouring before algorithm start and re-use it permanently, while the other re-colours vertices on each depth. The first approach is the only possible to apply for the backtracking as the backtracking relies on a fixed ordering of vertices or colour classes.

References:

[1] J. MacWilliams, N.J.A Sloane, *The theory of error correcting codes*, North-Holland, Amsterdam, 1979

[2] K. Corradi, S. Szabo, A combinatorial approach for Keller's Conjecture, *Periodica Mathematica Hungarica*, Vol. 21, 1990, pp. 95-100

[3] P. Berman, A. Pelc, Distributed fault diagnosis for multiprocessor systems, *Proceedings of the 20th Annual International Symposium on Fault-Tolerant Computing*, Newcastle, UK, 1990, pp. 340-346

[4] R. Horaud, T. Skordas, Stereo correspondence through feature grouping and maximal cliques, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 11, 1989 pp. 1168-1180

[5] E. M. Mitchell, P.J. Artymiuk, D.W. Rice, P. Willet, Use of techniques derived from graph theory to compare secondary structure motifs in proteins, *Journal of Molecular Biology*, Vol. 212, 1989, pp. 151-166

[6] K. Jansen, P. Scheffler, G. Woeginger, The disjoint cliques problem, *Operations Research*, Vol. 31, 1997, pp. 45-66

[7] M. Bomze, M. Budinich, P.M. Pardalos, M. Pelillo, 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

[8] R. Carraghan, P.M. Pardalos, A parallel algorithm for the maximum weight clique problem, *Technical report CS-90-40*, Dept of Computer Science, Pennsylvania State University, 1990

[9] R. Carraghan, P.M. Pardalos, An exact algorithm for the maximum clique problem, *Op. Research Letters*, Vol. 9, 1990 pp. 375-382

[10] P.R.J. Östergård, A new algorithm for the maximum-weight clique problem, *Nordic Journal of Computing*, Vol. 8, 2001, pp. 424-436

[11] D.S. Johnson, M.A. Trick, Cliques, Colouring and Satisfiability: Second DIMACS Implementation Challenge, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, American Mathematical Society, Vol. 26, 1996

[12] D. Kumlander, An exact algorithm for the maximum-weight clique problem based on a heuristic vertex-coloring, *Modelling, Computation and Optimization in Information Systems and Management Sciences, Le Thi Hoai An and Pham Dinh Tao eds*, Hermes Science Publishing, 2004, pp. 202-208