A new exact algorithm for the maximum-weight clique problem based on a heuristic vertex-coloring and a backtrack search.

Deniss Kumlander

Informatics Department, Tallinn Technical University, Ehitajate tee 5, 19086 Tallinn, Estonia;
kalkin@solo.ee

Abstract: In this paper we present an exact algorithm for the maximum-weight clique problem on arbitrary undirected graphs. The algorithm based on a fact that vertices from the same independent set couldn't be included into the same maximum clique. Those independent sets are obtained from a heuristic vertex coloring where each of them is a color class. Color classes and a backtrack search are used for pruning branches of the maximum-weight clique search tree. Those pruning strategies together result in a very effective algorithm for the maximum-weight clique finding. Computational experiments with random graphs show that the new algorithm works faster than earlier published algorithms; especially on dense graphs.

Key words: maximum-weight clique, branch and bound algorithm, maximum-weight independent set, vertex-coloring

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 coloring problem is defined to be an assignment of color to its vertices so that no pair of adjacent vertices shares identical colors. All those problems are computationally equivalent, in other words, each one of them can be transformed to any other.

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 is divided into two parts – in the first part we look at the recently developed algorithm for the unweighted case while the next part contains modification of this algorithm for the weighted case.

2.1. The unweighted case

Before starting the algorithm we find a vertex-coloring by using any heuristic algorithm, for example in a greedy manner. We determine color classes one by one as long as uncolored vertices exist. Then vertices are resorted in an order they are added into color classes. This order affects the algorithm's performance in finding a maximum clique.

Definition 1: A color class is called existing on a subgraph G_p if any vertex from this color class belongs to the subgraph G_p.

Definition 2: Degree of a subgraph G_p equals to the number of color classes existing on that subgraph.

Crucial to the understanding of the algorithm is a notation of the depth and pruning formula. Basely, at 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. Suppose we expand vertex v_1. At depth 2, we consider all vertices adjacent to v_1 from vertices from previous depth, i.e. belonging to G_1. Those vertices form a subgraph G_2. At depth 3, we consider all vertices (that are in depth 2) adjacent to the vertex expanded in depth 2 etc. Let v_{d1} be the vertex we are currently expanding at depth d. That is:

Let's say that G_d is a subgraph of G on depth d that contains the following vertices: $V_d = (v_{d1}, \ldots, v_{dm})$.

Then a subgraph on 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 $(v_{d+1\,i}, v_d) \in E$.

The pruning formula is the next: If $d-1 + Degree(G_d) \le CBC$, where CBC is a size of the current best 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. The weighted case

The previously described algorithm is the base for the maximum-weight algorithm with the following changes. We cannot any longer determine values of the function *Degree* as a number of existing color classes on a subgraph since vertices have different weights and a color class' maximum weight can differ from 1. Therefore a degree of a subgraph will be calculated as a sum of maximum weights of each color 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 even more important. Vertices should be resorted first of all by color classes and then by weights inside each color class. So, a vertex of the maximum weight in any color class always will be the last inside this color class. Therefore a degree of a subgraph equals to the sum of the last vertex' weights of each color class existing on the subgraph independent on which vertices of a color class exists on this subgraph. Moreover, instead of calculating degree of a subgraph each time we will calculate it only first time on a depth and later only adjust by the following rule: if the next vertex on this depth to be expanded is from the same color class as the previous one then degree remains the same otherwise should be decreased on a weight of the previous vertex (there is no more vertices from the previous vertex' color class and the previous vertex weight was the largest in that color class by resorting).

Besides one more adjustment to the base algorithm will be done. We will use ideas of a backtrack search described by P. Östergård [10]. 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 $c(i)$. Those 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 $W' + c(i) \le W$. P. Östergård is also advised to use a vertex reordering by a vertex-coloring's color classes [10], therefore the ordering for the first pruning strategy will not slow down this backtrack search.

Other steps of the algorithm remain unchanged.

Note: It is advisable to use a special array to solve order of vertices to avoid work by changing adjacency matrix during reordering vertices.

Algorithm for the maximum – weight clique problem

N – number of vertices in the graph
W – weight of the current best (maximum-weight) clique
d – depth
G_d – subgraph of G formed by vertices existing on depth d
$W(d)$ – weight of all vertices in the forming clique
$w(i)$ – weight of vertex i

Step 0. Heuristic vertex-coloring: Find a vertex coloring, reorder vertices and apply new vertices indexes (renumber vertices from 1 to N for using in the backtrack search).

Step 1. Back track search runner:
For $n = N$ downto 1
Goto step 2
$c(i)=W$
Next
Go to End

Step 2. Initialization: Form the depth 1 by selecting all vertices with an index less than n and connected to the vertex n. d=1. $W(1)= w(n)$
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: Get the next vertex to expand. 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.
Step 5. Control: If the current level can contain a larger clique than already found:
If $W(d) + c(expanding\ vertex\ index) \leq W$ then go to step 7.
Step 6. The next level: Form the new depth by selecting all remaining vertices that are connected to the expanding vertex from the current depth;
$W(d+1)=W(d) + w(expanding\ vertex\ index)$
$d = d + 1$;
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.

The algorithm program code can be obtained from
http://www.simpleconcepts.ee/dk/index.html.

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. It gives possibility to conclude that worse cases of the new algorithm is practically the same as worse cases of those base algorithms and comparing them on worse cases cannot give another result than comparing on random graphs.

Results are presented as a ratio of algorithms spent times on finding the maximum-weight clique – so the same results can be reproduced on any platforms. 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-coloring.

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].

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.

Table 1.
Benchmark results at random graphs

Vertices	Edge density	*PO*	*New*
1000	0.1	1.12	1.28
800	0.2	1.23	1.93
500	0.3	1.42	2.78
300	0.4	1.63	2.81
200	0.5	1.73	2.81
200	0.6	1.90	4.90
150	0.7	2.12	5.54
100	0.8	2.27	6.83
100	0.9	11.25	69.85

For example, 6.83 in the column marked *New* means that Carraghan and Pardalos [8] algorithm requires 6.83 times more time to find the maximum 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 69 times and than P. Östergård algorithm [10] in 6 times.

Accordingly to the vertex-coloring step of the algorithm we should mark that the problem of finding an efficient vertex coloring can be treated as a separate problem. This problem is an NP-hard task; therefore we had to use a heuristic. Heuristic algorithm is an algorithm that

1. Doesn't guarantee the best result, but finds a result that is close enough to the best one.
2. Is quicker than an exact algorithm. In our case we use a polynomial heuristic – a result is found in a polynomial time.

The vertex-coloring step affects the overall result in the following way:

1. The closer number of color classes to the size of the maximum clique the quicker the maximum clique will be found because of more effective pruning;
2. The more time we spent on vertex coloring the slower our algorithm works in general (since the vertex coloring subroutine is included into the main algorithm and it's time should be get into account).

Moreover, the algorithm can evaluate without changing core steps by inventing a new and more effective heuristic algorithm for the vertex coloring.

Table 2.
Number of color classes by a greedy vertex coloring

Vertices	Density	Average size of the maximum clique	Number of color classes	Number of color classes containing only 1 vertex
100	0.1	3.88	7.16	0.40
100	0.2	5.08	10.36	0.48
100	0.3	6.52	13.88	0.64
100	0.4	8.24	17.20	0.92
100	0.5	10.44	20.76	1.12
100	0.6	13.60	24.80	1.56
100	0.7	18.00	30.00	1.76
100	0.8	24.04	37.24	3.16
100	0.9	34.36	46.08	4.80
100	0.99	69.56	71.20	42.48

It is easy to see that quite effective results of the new algorithm were reached not on the best splitting vertices into color classes - number of color classes approximately larger on 50%-25% than size of the maximum clique. This difference

is understandable if we will get into account that we have used a quite easy/rough heuristic on vertex coloring.

4. Conclusion

In this paper we have introduced the new algorithm for finding the maximum-weight clique based on a heuristic vertex-coloring and a backtrack search. Color classes found in the heuristic vertex-coloring are used to prune branches of the maximum-weight clique search tree – since vertices of the same color class cannot participate in the same maximum clique we can more effectively define if a current subgraph can contain a larger clique than already found or not using color classes than by just looking at the amount of vertices of this subgraph. A backtrack search is also proved to be effective information capture and branches pruning strategy (originally it was shown by P. Östergård [10]). Those two pruning strategies provide us with a very fast algorithm for the maximum-weight clique finding. Besides algorithm is easy to implement. Probably the triviality of ideas makes it so fast. The program listing can be obtained from http://www.simpleconcepts.ee/dk/index.html.

Another advantage is a way how a heuristic result is used for finding the exact result: usually a heuristic result just sets upper or lower bounds and them quite quickly lose actuality since usually are replaced by a current best result. In the new algorithm a heuristic vertex-coloring is employed on the permanent base through the whole algorithm's work. Moreover, the algorithm can evaluate without changes in algorithm's core steps by just inventing a new and more effective heuristic algorithm for the vertex coloring. The less color-classes we have found the more effectively we prune branches of a search tree.

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, Vol. 21. (1990) 95-100.
3. Berman, P., Pelc, A.: Distributed fault diagnosis for multiprocessor systems. Proceedings of the 20th Annual International Symposium on Fault-Tolerant Computing, Newcastle, UK. (1990) 340-346
4. Horaud, R., Skordas, T.: Stereo correspondence through feature grouping and maximal cliques. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 11. (1989) 1168-1180
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, Vol. 212. (1989) 151-166.
6. Jansen, K., Scheffler, P., Woeginger, G.: The disjoint cliques problem. Operations Research, Vol. 31. (1997) 45-66.

7. 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, editors,. Kluwer Academic Publishers, Boston, MA, (1999)
8. Carraghan, R. and 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. and Pardalos, P. M. An exact algorithm for the maximum clique problem. Op. Researc Letters, Vol. 9. (1990) 375-382
10. Östergård, P.R.J., A new algorithm for the maximum-weight clique problem. Nordic Journal of Computing, Vol. 8. (2001) 424-436
11. Johnson, D.S., Trick, M.A., editors. Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge, Vol. 26 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society (1996).