International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:6, No:12, 2012

Some Improvements on Kumlander's Maximum Weight Clique Extraction Algorithm

Satoshi Shimizu, Kazuaki Yamaguchi, Toshiki Saitoh and Sumio Masuda

Abstract—Some fast exact algorithms for the maximum weight clique problem have been proposed. Östergård's algorithm is one of them. Kumlander says his algorithm is faster than it. But we confirmed that the straightforwardly implemented Kumlander's algorithm is slower than Östergård's algorithm. We propose some improvements on Kumlander's algorithm.

Keywords—Maximum weight clique, exact algorithm, branch-and-bound, NP-hard.

I. Introduction

FOR an undirected graph $G = (V, E)$ and a set $V' \subseteq V$, $G(V')$ denotes the subgraph G induced by V'. If any two vertices in $G(V')$ are adjacent to each other, V' is called a *clique* in G. Given an undirected graph G, the maximum clique problem is to find a clique of the maximum size. It is a well-known and important NP-hard problem [4], [7]. Given an undirected graph G and vertex-weight $w(\cdot)$, the maximum weight clique problem is to find a clique of the maximum weight. The maximum weight clique problem is a generalization of the maximum clique problem.

A fast exact algorithm for the maximum weight clique problem is proposed by Östergård [5]. It is based on the branch-and-bound technique. Before the main routine, Östergård's algorithm decides the branching order by the weight of each vertex. Then the optimum solution is searched by the branch-and-bound method with the upper bounds which can be immediately obtained. Östergård's algorithm makes many branches than other algorithms based on the branch-and-bound technique, but the time consumed on each subproblem is very short. A program named Cliquer, based on the algorithm in [5], is released in [6].

Kumlander [1] extended Östergård's algorithm, which tries to prune many subproblems by calculating the heuristic vertex coloring . Kumlander insists that his algorithm is always faster than Östergård's algorithm for any instances.

In this paper, we verify the validity of the experiments in [1]. Kumlander insists algorithms were compared fairly. However, in our experiments, we compare his algorithm with well-tuned programs of other algorithms, and we find out that Kumlander's algorithm is slower than Östergård's Cliquer for the sparse graphs, and that it is slower than YM algorithm [3] for the dense graph.

We propose a new initial ordering and a new implementation for Kumlander's algorithm. By computational experiments, we show our new algorithm reduces number of branches and computation time and it is faster than other algorithms in many cases.

The section II describes two previous algorithms, Östergård's algorithm and Kumlander's algorithm. We also show experimental results to compare previous algorithms. In the section III, we propose some improvements for Kumlander's algorithm. The experimental results of our new algorithm is shown in the section IV. The conclusion is in the section V.

Satoshi Shimizu is with the Faculty of Engineering, Kobe University, 1-1 Rokkodai, Nada, Kobe 657-8501 JAPAN.(e-mail:124t220t@stu.kobe-u.ac.jp)
Kazuaki Yamaguchi is with the Faculty of Engineering, Kobe University. (e-mail:ky@kobe-u.ac.jp)
Toshiki Saitoh is with the Faculty of Engineering, Kobe University.(e-mail:saitoh@eedept.kobe-u.ac.jp)
Sumio Masuda is with the Faculty of Engineering, Kobe University.(e-mail:masuda@kobe-u.ac.jp)

II. Previous algorithms

This section describes Östergård's algorithm and Kumlander's algorithm. The first algorithm determines the branching order first and then performs a branch-and-bound method according to the order. The second one is the extended version of the first one.

A. Östergård's algorithm

Given an undirected graph $G = (V, E)$ and weight $w(v)$ of each $v \in V$, Östergård's algorithm first sorts vertices in the descending order of weights. Let the sequences $v_1, v_2, \cdots, v_n$, where n is the number of vertices in V. For $i = 1, 2, 3, \cdots, n$, V_i denotes $\{v_i, v_{i+1}, \cdots, v_n\}$ (clearly, $V_1 = V$).

Östergård's algorithm searches the maximum weight clique of $G(V_i)$ for $i = n, n-1, \cdots, 2, 1$, and stores its weight in $c[i]$. The branch-and-bound technique is used in searching process for each i. During the process, for any subset $S \subseteq V$, $c[\min\{j \mid v_j \in S\}]$ is an upper bound of the maximum weight of cliques in $G(S)$. This upper bound is calculated in $O(1)$ time by finding the minimum index of $v \in S$. An obvious upper bound $\sum_{v \in S} w[v]$, the sum of weights of vertices in S, is also used for bounding steps. This is calculated in $O(|S|)$ time. The process described above is called "backtrack search".

The performance of backtrack search greatly depends on the ordering of V. In Cliquer, vertices are sorted in descending order of weights. And vertices with same weights are sorted by descending order of the sum of weights of adjacent vertices.

B. Kumlander's algorithm

The description of Kumlander's algorithm is as follows. First, it divides V into independent sets $C_1, C_2, \cdots, C_k$, where

k is the number of independent sets. This process is called vertex coloring. And each C_i is called a color class. In this step, vertices in each color class are sorted in ascending order of weights. Let $V_i = \{C_i, C_{i+1}, \cdots, C_k\}$. Then, for $i = k, k-1, \cdots, 1$, search maximum weight clique in $G(\cup_{j=i}^{k} C_j)$. As in Östergård's algorithm, store its weight in $c[i]$ and use it as an upper bound.

Kumlander's algorithm seems to be similar to Östergård's algorithm. The difference is that Kumlander's algorithm uses not only $c[\cdot]$ as an upper bound but also $\sum_{i=1}^{k} \max\{w[u] \mid u \in C_i \cap S\}$, "the coloring upper bound", for pruning subproblems S. Due to the sorting process, the coloring upper bound is equal to the sum of last vertices in each color class. Therefore it can be calculated in $O(k)$ time. By the effect of the coloring upper bound, Kumlander's algorithm prunes subproblem more frequently than Östergård's algorithm. The coloring upper bound is an obvious upper bound of the weight of the maximum clique, because at most one vertex in each color class can be included in a clique.

The detail of Kumlander's algorithm is shown in Fig.1. Let $N(v)$ the set of vertices adjacent to v.

The performance of Kumlander's algorithm greatly depends on coloring strategy. In [1], vertices are assigned to the color class with as small index as possible in descending order of weights.

C. Implementation of algorithms

In the following, we compare some previous algorithms by computer experiments. Before comparing algorithms, we briefly show how we get the program of each algorithm. We use the C source code of Cliquer published in [6]. And we use the C++ program of YM algorithm that we implemented when we contributed [3]. We implemented Kumlander's algorithm in C because the source code released in [2] is written in Visual Basic, which is much slower than C.

For the implementation, vertex sets $S \subseteq V$ can be implemented with a list and a bit vector. We implement both the list and the bit vector as follows :

List
: The elements in S are divided into color classes, and vertices in each color class are stored in an array. The last vertex in an array has the maximum weight in that color class. So the upper bound of coloring is immediately obtained by calculating the sum of the weights of the last vertices of arrays.

Bit vector
: For each color class, an array of integer variables is used. Each bit of the integer variable is corresponding to the element in the color class. During calculating the coloring upper bound, the least significant bit(LSB) of the last non-zero integer variable in each array is searched. And the sum of weights of LSBs is the coloring upper bound.

```
Inputs: an undirected graph G and vertex weight w[·]
Output: the maximum weight clique
1: function main
2:     get coloring C_1, C_2, ..., C_k, and order vertices
3:     record ← 0
4:     for i from k to 1 do
5:         expand(V_i(∪_{j=i}^{k} C_j), 0)
6:         c[i] ← record
7:     end for
8:     output: the maximum weight clique
9:     return

10: function expand(S, weight)
11:     if |S| = 0 then
12:         if weight > record then
13:             record ← weight
14:         end if
15:         return
16:     end if
17:     while |S| > 0 do
18:         i ← min{k | v_k ∈ S}
19:         if weight + coloring upper bound ≤ record then
20:             return
21:         end if
22:         if weight + c[i] ≤ record then
23:             return
24:         end if
25:         S ← S \ v_i
26:         expand(S ⋂ N(v_i), weight + w[v_i])
27:         end while
30:     return
```

Fig. 1. Kumlander's algorithm

D. Computational Experiments

Table.I shows experimental results with 4 programs mentioned in the previous section. In the table, DK denotes Kumlander's algorithm. In each row, the average CPU time of each program for 10 random graphs are shown, where the weights of vertices are from 1 to 10. The CPU of the computer used in the experiments was Intel(R) Core(TM) i7-2600 3.40GHz. The OS was GNU/Linux. The memory was 8GB. The compiler was gcc 4.4.6 with an optimization option -O2. The parameter d denotes the edge density of graphs.

With respect to Kumlander's algorithm, the bit vector implementation is faster than the list implementation in most cases. However Cliquer is the fastest when the edge density is less than or equal to 0.7, and YM algorithm is fastest when the edge density is from 0.8 to 0.95. Kumlander's algorithm is fastest only when the density is 0.98.

In order to investigate the difference between our results and the results in [1], we read the Visual Basic program of Östergård's algorithm implemented by Kumlander [1], Ö-K for short. Ö-K figured out that the vertex ordering by Kumlander is quite different from the ordering in Cliquer. Specifically, in Ö-K, the ordering is done by greedy coloring, whereas in

TABLE I
COMPUTATION TIME OF ALGORITHMS [S]

n	d	Cliquer	DK (bit)	DK (list)	YM	Ö-K (in C)
4000	0.1	0.36	0.42	0.92	1.15	0.48
3000	0.1	0.18	0.18	0.35	0.45	0.19
3000	0.2	1.35	2.73	6.11	6.09	4.51
2500	0.2	0.69	1.19	2.64	2.45	1.94
2500	0.3	10.12	24.5	44.21	37.99	58.29
2000	0.3	3.4	7.73	14.19	12.16	17.59
1500	0.4	14.74	35.12	55.16	41.27	100.16
1000	0.4	1.61	3.34	5.59	3.63	9.31
1000	0.5	30.08	66.34	96.75	62.6	274.81
900	0.5	14.6	33.94	49.01	31.48	133.66
700	0.6	67.08	172.38	230.89	106.21	899.52
500	0.6	5.69	14.1	19.23	7.53	59.29
500	0.7	186.96	500.03	639.6	216.49	3934.02
300	0.7	3.28	6.63	8.52	2.34	32.74
300	0.8	194	347.09	421.06	83.52	3868.94
200	0.8	4.74	5.87	7.64	1.36	52.55
200	0.9	1172.94	631.47	724.49	99.16	
150	0.9	24.83	9.2	9.7	1.75	
150	0.95	1296.95	62.26	56.59	16.97	
100	0.95	1.03	0.09	0.09	0.03	
150	0.98	13847.82	16.97	13.1	41.65	
100	0.98	0.99	0.02	0.01	0.01	

TABLE II
RESULT OF NEW ORDERING[S]

n	d	Cliquer	DK (new)	DK (bit)	DK (list)	YM
4000	0.1	0.36	0.29	0.42	0.92	1.15
3000	0.1	0.18	0.13	0.18	0.35	0.45
3000	0.2	1.35	2.25	2.73	6.11	6.09
2500	0.2	0.69	0.99	1.19	2.64	2.45
2500	0.3	10.12	20.39	24.5	44.21	37.99
2000	0.3	3.4	6.07	7.73	14.19	12.16
1500	0.4	14.74	27.49	35.12	55.16	41.27
1000	0.4	1.61	2.7	3.34	5.59	3.63
1000	0.5	30.08	50.92	66.34	96.75	62.6
900	0.5	14.6	24.63	33.94	49.01	31.48
700	0.6	67.08	136.02	172.38	230.89	106.21
500	0.6	5.69	9.77	14.1	19.23	7.53
500	0.7	186.96	330.77	500.03	639.6	216.49
300	0.7	3.28	4.55	6.63	8.52	2.34
300	0.8	194	201.77	347.09	421.06	83.52
200	0.8	4.74	3.81	5.87	7.64	1.36
200	0.9	1172.94	333.96	631.47	724.49	99.16
150	0.9	24.83	5.35	9.2	9.7	1.75
150	0.95	1296.95	28.35	62.26	56.59	16.97
100	0.95	1.03	0.03	0.09	0.09	0.03
150	0.98	13847.82	4.67	16.97	13.1	41.65
100	0.98	0.99	0.02	0.02	0.01	0.01

Cliquer vertices are ordered by weights and the sum of weights of adjacent vertices.

To show the effect of the ordering to the performance, we implemented Ö-K in C and compare to the original Cliquer. The results are in Table.I. In this comparison, Östergård's algorithm is greatly slower than Kumlander's algorithm. This is same as [1]. From these results, we can get the following conclusion.

- The overall performance of Östergård's algorithm greatly depends on the initial ordering.
- The ordering for Östergård's algorithm used in [1] was not efficient.

III. Our algorithm

In this section, first, we propose new initial ordering for Kumlander's algorithm. Second, we show positive effect of "limitation on color class size". Then we propose a new implementation technique for calculating the coloring upper bound.

A. Initial ordering

From the results in the previous section, vertices should be sorted in descending order of weights because the initial ordering in Cliquer is in such a way. But Kumlander's algorithm cannot adopt the same ordering because vertices in a color class, which may contain vertices of different weights, must be consecutive in the initial ordering. Therefore, we devise a new ordering algorithm, which produces similar order to the descending order of weights.

1) Coloring ordering: First we show a new initial sorting for greedy coloring. In our algorithm, vertices are assigned to a color class in weight descending order like Kumlander's algorithm, and moreover, we consider the sum of the weights of adjacent vertices (denoted by $adjw[\cdot]$).

Vertices are assigned to color classes in descending order of weights. If vertices u, v have the same weight and $adjw[u] \leq adjw[v]$, u is colored first. We briefly show the reason in the following.

Suppose that a vertex v has the maximum weight in V and is adjacent to more vertices with weight $w[v]$ than other vertices. If v is stored in the first color class, many vertices of weight $w[v]$ cannot get into the first color class. The resulting ordering may be quite different from the descending order. If v is colored after any adjacent vertex with same weight, other vertices with same weight may get into the first color class, and the ordering may be closer to the weight descending order.

2) Sorting in color classes: In Kumlander's algorithm, vertices in each color class are sorted in ascending order of weights. We change it to the descending order of weights. This change makes the ordering closer to the descending order of weights than Kumlander's.

We add another modification. Although branching by color classes is proposed as a new technique in [1], the average performance got worse. So we just omit the technique, and make the backtrack search for each vertex, same as Östergård's algorithm.

3) Computational Experiments: To investigate the performance of our new ordering, we compared our ordering with Kumlander's ordering by experiments. Results are shown in Table.II. In each row, the average CPU time of each program for 10 random graphs are shown, where the weights of vertices are from 1 to 10.

For all instances, our new ordering is faster than the original Kumlander's ordering. But still the improved Kumlander's algorithm is not fastest.

B. Limitation on color class size

If the length of a bit vector of a color class is longer than the one word length, iterations on each word arise for calculation

of AND operation (intersection of sets), finding MSB and LSB (to find branching variable and upper bound), and so on. In order to reduce the iteration, we limit the maximum size of a color class beforehand. If the size of a color class reaches the size limit during greedy coloring, we stop adding vertices to that color class and create a new color class. If the limit is smaller than the length of one word, the size of each color class must be smaller than the length of one word, and iterations in processing color classes can be removed.

Although this limit may increase the number of color classes and seems to increase the coloring upper bound, this limit actually works in a good way as follows :

- The new color classes made by this limit usually has vertices with small weights. So the coloring upper bound does not increase seriously.
- This limit makes the ordering closer to the descending order of weights. Because, vertices not added to the color class because of the limit probably have small weights and are assigned to a color class later.
- After all, the number of branches are reduced.

If the limit size is too small, a lot of small color classes are created and the performance becomes worse. We experimentally investigated the optimal limit for several cases.The fastest limit size is 4 if the edge density is greater than 0.6, 8 if the density is between 0.4 and 0.6, 12 if the density is 0.3, and 20 if the density is less than 0.3.

C. upper bound table

With an ordinary coding, the coloring upper bound is calculated by the following steps :

1) Find the last vertex by finding LSB from the bit vector.
2) Get the weight of the last vertex by referring to the array of weights.
3) Do step 1. and 2. for all colors, and calculate the sum of them.

This computation time can be reduced in the following way. Let k be the size of a color class, the number of all combinations of the bit vector is 2^k. We calculate the coloring upper bound for all patterns and store them to a table (an array in C). With this upper table, the process to get the maximum weight of a color class can be replaced with only one operation to refer the upper table.

Examples of upper tables are shown in the following. Let the size of a block is 8.
Given a subset $\{01100011\ 11010110\ 01010101\}$,
the upper bound of block 3 is
$upperTable[3][01100011_b] = upperTable[3][99]$,
the upper bound of block 2 is
$upperTable[2][11010110_b] = upperTable[2][214]$, and
the upper bound of block 1 is
$upperTable[1][01010101_b] = upperTable[1][85]$.
And the sum of these value is an upper bound of the subset. As written in III-B, the optimal size of one block is not too big. Therefore, the amount of memory space is several mega bytes per block.

When the graph is dense and the limit of color class size is small, more than one color class can be packed in a word. This is especially efficient in dense graphs, because sizes of color classes are small.

TABLE III
CPU TIMES ON RANDOM GRAPHS[S]

n	d	Cliquer	ours	DK(bit)	DK(list)	YM
4000	0.1	0.36	0.86	0.42	0.92	1.15
3000	0.1	0.18	0.51	0.18	0.35	0.45
3000	0.2	1.35	2.71	2.73	6.11	6.09
2500	0.2	0.69	1.25	1.19	2.64	2.45
2500	0.3	10.12	16.48	24.5	44.21	37.99
2000	0.3	3.4	8.03	7.73	14.19	12.16
1500	0.4	14.74	14.62	35.12	55.16	41.27
1000	0.4	1.61	1.23	3.34	5.59	3.63
1000	0.5	30.08	18.69	66.34	96.75	62.6
900	0.5	14.6	10.26	33.94	49.01	31.48
700	0.6	67.08	37.28	172.38	230.89	106.21
500	0.6	5.69	2.68	14.1	19.23	7.53
500	0.7	186.96	65.97	500.03	639.6	216.49
300	0.7	3.28	1.12	6.63	8.52	2.34
300	0.8	194	39.68	347.09	421.06	83.52
200	0.8	4.74	0.94	5.87	7.64	1.36
200	0.9	1172.94	66.46	631.47	724.49	99.16
150	0.9	24.83	1.25	9.2	9.7	1.75
150	0.95	1296.95	6.27	62.26	56.59	16.97
100	0.95	1.03	0.06	0.09	0.09	0.03
150	0.98	13847.82	1.06	16.97	13.1	41.65
100	0.98	0.99	0.07	0.02	0.01	0.01

IV. Computer Experiments

We implement our new algorithm including all of our new ideas and compare it to other algorithms. Table.III shows experimental results.

In almost all conditions, our new algorithm is faster than Kumlander's algorithm of bit vector and list, especially in dense graphs. When the edge density is more than or equal to 0.4, our algorithm is greatly faster than other algorithms. In cases $n = 100$ and the density is 0.95 and 0.98, our algorithm is not faster than others. This is because creating the upper table needs $O(2^k)$ time before the branch-and-bound steps. The preprocessing time could have been reduced if k were smaller, therefore it does not matter. When the edge density is less than or equal to 0.3, Cliquer is fastest.

V. Conclusion

We proposed some improvements on Kumlander's algorithm. The original Kumlander's algorithm is not as fast as mentioned in [1], actually much slower than other algorithms. So we tried to reduce the computation time only by improving the initial ordering, but it was not succeeded. Then we propose two new ideas, limit of color class size and upper table. Then the program gets faster than others when the graph is dense.

The idea of upper table might be applied to other situations. We try to apply the upper table of branch-and-bound techniques of other NP-hard problems.

References

[1] D. Kumlander, "On importance of a special sorting in the maximum-weight clique algorithm based on colour classes," Proc. of the second international conference on Modelling, Computation and Optimization in Information Systems and Management Sciences Communications in Computer and Information Science Volume 14, 2008, pp 165-174.

[2] D. Kumlander, http://www.kumlander.eu/graph/
[3] K. Yamaguchi, S. Masuda, “A new exact algorithm for the maximum weight clique problem,” Proc. of the 23rd International Technical Conference on Circuits/Systems, Computers and Communications 2008, pp. 317 - 320.
[4] M.R. Garey and D.S. Johonson,, Computers and Intractability - A Guide to the Theory of NP-Completeness, Freeman, New York, 1979.
[5] P.R.J. Östergård, “A new algorithm for the maximum-weight clique problem,” Nordic Journal of Computing, vol. 8, pp.424-436, 2001.
[6] P.R.J. Östergård, http://users.tkk.fi/~pat/cliquer.html
[7] E. Tomita, Y. Sutani, T. Higashi, S. Takahashi, M. Wakatsuki, “A simple and faster branch-and-bound algorithm for finding a maximum clique,” Proc. of the 4th International Workshop, WALCOM Algorithms and Computation Lecture Notes in Computer Science Volume 5942, 2010, pp 191-203