Proceedings of the 10th WSEAS International Conference on COMPUTERS, Vouliagmeni, Athens, Greece, July 13-15, 2006 (pp932-937)

Comparing the Best Maximum Clique Finding Algorithms, Which are Using Heuristic Vertex Colouring

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

Abstract: In this paper two best known at the moment algorithms for finding the maximum clique is compared. A key technique that both algorithms rely on is using a heuristic vertex colouring. The only difference between those algorithms is a way of using vertex-colouring and this difference in the very important. The algorithms' approaches are divided in this paper into two classes: re- applying vertex colouring on each depth and re-using colour classes obtained before the start of the main part. Historical attempts to use the colour classes for the maximum clique finding are also reviewed. Those two algorithms were never compared so far as is not widely announced also. The comparison is essential information for finding further development ways.

Key-Words: maximum clique, vertex colouring, graph theory

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 $\hat{G}=(V,\hat{E})$, where $\hat{E} = \{ (v_i , v_j) \mid v_i , v_j \in V, i \neq j, (v_i , v_j) \notin E \}$ – this is a slightly reformulated definition provided by Bomze et al 1999 [3]. A neighbourhood of a vertex v_i is defined as a set of vertices, which are connected to this vertex, i.e. $N(v_i) = \{v_1, \ldots, v_k \mid \forall j\colon v_j \in V, i \neq j, (v_i , v_j) \in E \}$ 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, which 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 $\hat{G}$. 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 [8] 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 [4], computer vision [2], and biochemistry [12]. Another wide area of applying the maximum clique is the coding theory [6, 13]. There are many others areas of the maximum clique application that makes this problem to be important.

2 Historical Review

Some historical attempts to use a heuristic vertex-colouring for the maximum clique finding are reviewed here before analyzing best known algorithms. Notice that this article doesn't contain algorithms that just use a vertex-colouring to derive once bounds for the maximum clique problem, but reviews algorithms containing vertex-colouring as an important part of the maximum clique finding, i.e. where those bounds are permanently found and evaluated.

2.1 Babel and Tinhofer's algorithm

The first algorithm to be reviewed is an algorithm proposed by Babel and Tinhofer in 1990 [1]. This algorithm is the branch and bound one and was positioned by authors as an algorithm, which is especially efficient for graphs with a large edge density. The branching is the first phase of the algorithm, which is done exactly in the same way as Carraghan and Pardalos one does [7]. The second phase, which is bounding uses mainly a well known fact that the chromatic number of a graph is always larger or equal to the size of this graph's maximum clique. The problem of finding a vertex-colouring is NP-complete therefore the algorithm uses DSATUR technique [5] to find it. The following sentence describes an essence of Babel and Tinhofer algorithm: "In each G_i we look for a clique and a colouring", where G_i is a subgraph that is produced during the branching part of the algorithm. The algorithm can be presented using the following pseudo-code:

```
function Main
  CBC :=0        // the maximum clique's size
  clique (V, 0)
  return
end function

function clique(V, depth)
  if'|V| = 0 then
    if depth > CBC then
      New record - save it. CBC := depth
    end if
    return
  end if

  while $V \neq \emptyset$ do
     if $depth + |V| \leq CBC$ then return     // prune
//use DSATUR to find the max. clique and colouring
     // $C = \{C_1,..,C_k\}$, Q – max. clique
     Q, C :=DSATUR(G(V))
     // $C = \{C_1,..,C_k\}$, Q – max. clique
     if $depth + |Q| > CBC$ then
       New max clique: Q + vertices of prev depths.
       $CBC = depth + |Q|$
     end if
     if $depth + |C| \leq CBC$ then return
     clique ($N(v_1)$, depth + 1)
     $V := V \setminus v_1$
  end while
  return
end function
```

Notice that the DSATUR algorithm provides both a heuristic colouring and a heuristic maximum clique. Different modifications of this base algorithm were proposed by Babel and Tinhofer in their work to decrease the time for computing the upper and lower bounds, but those modifications do not change the base algorithm dramatically and therefore can be omitted for this review. The algorithm was successfully used for some types of graphs, including chordal graphs [1], but the general performance of the algorithm was quite bad in comparison to the Carraghan and Pardalos algorithm [7] that was released simultaneously, mostly due to the fact that the algorithm consumes too much time to find bounds in combinatorial cycles.

2.2 Wood's algorithm

An algorithm invented by Wood in 1997 [15] is another attempt to employ a heuristic vertex colouring for the maximum clique finding basing on previous works – mostly using Babel and Tinhofer algorithm [1], Carraghan and Pardalos algorithm [7] and some their later works. The algorithm can be described using the following pseudo-code with a certain simplification of unimportant details, which will not change the main idea:

```
function Main
  CBC :=0                // the maximum clique's size
  clique (V, 0)
  return
end function

function clique(V, depth)
  Q :=greedy(V)
  if $depth + |Q| > CBC$ then
    New max clique: Q + vertices of prev depths.
    $CBC = depth + |Q|$
  end if

  //Find a vertex colouring of G(V) by DSATUR
  C :=DSATUR(G(V)) // $C = \{C_1,..,C_k\}$
  if $depth + |C| \leq CBC$ then return
  while $depth + |C| > CBC$ do   // pruning condition
    $k := |C|$
    $v :=\text{maxdegree}(v \in C_k)$
    $C_k := C_k \setminus v$
    if $C_k = \emptyset$ then $C := C \setminus C_k$
    clique ( $N(v_i)$, depth + 1)
  end while
  return
end function

function greedy(V)
  $S := V$
  $Q := \emptyset$
  while $S \neq \emptyset$ do
    $v_i = \text{maxdegree}(v \in S)$
```

```
        $Q := Q \cup \{v_i\}$
        $S := S \& \{v_i\}$
    end while
end function
```

It is easy to see that the described algorithm is also a branch and bound one. The author mainly concentrates on three issues that he thought are very important for such algorithms:

- How to find a good lower bound, i.e. a clique of large size?
- How to find a good upper bound on the size of the maximum clique?
- How to branch, i.e. break a problem into subproblems? [15]

The greedy approach is used find the lower bound, DSATUR [5] is used to find the upper bound. Those bounds are produced on each new depth, i.e. practically for each new branch. The original paper also employs a fractional colouring in addition to DSATUR to find a better upper bound, since the upper bound for pruning is set to be a minimum number of colours provide by those heuristic vertex-colouring algorithms – see this Wood paper for more details on fractional colouring and how it is used [15]. The biggest differences from the previously described Babel and Tinhofer algorithm [1] are:

- The algorithm looks for a heuristic clique and colouring only for a new branch, i.e. it is not done in the internal cycle of analysing the branch. Notice that Babel and Tinhofer tried to improve efficiency of their algorithm by speeding up the colouring sub-algorithm for each subgraph by reordering vertices of it [1], while Wood decided not to colour in the internal cycle;
- More than one heuristic vertex colouring algorithm is used to provide a better bound.

Although the algorithm meets the defined issues, it seems to be still impractical since it spends too much time on finding bounds and this greatly affects its performance characteristics. Therefore this algorithm is treated by many authors as another unsuccessful attempt to use vertex-colouring for the maximum clique finding, which proved that the vertex-colouring cannot be efficiently used for the maximum clique finding.

3 Different Levels of Using a Vertex Colouring in Nowadays Algorithms

Two algorithms, which are using vertex colouring for finding the maximum clique on different levels, will be reviewed here. A core idea of both algorithms is the following: any colour class (a set of vertices coloured by the same colour) is an independent set and therefore no more than one vertex from each colour class can participate in a clique. Colour classes are found using a heuristic vertex-colouring, for example the greedy one. The first algorithm described below proposes to re-apply a heuristic vertex colouring on each new depth of the branch and bound algorithm. The second one proposes to apply the colouring only once before the branch and bound routine starts and then use obtained colouring on the permanent base during branch and bound steps. Both algorithms are claimed to be the quickest at the moment; therefore a comparison of those algorithms is worth to do to identify how those different ways affect the performance in different cases (graph types) to be solved.

3.1 Depth notation

Crucial to the understanding of algorithms described below is a notation of a depth. Basely, at the depth 1 an algorithm has all vertices under analysis, i.e. $G_1 \equiv G$. The algorithm is going to expand all vertices of a subgraph during the analyse so that vertex is deleted from the subgraph after it is expanded. Suppose it expands vertex v_1. At the depth 2, it considers all vertices adjacent to v_1 from vertices from the previous depth, i.e. belonging to G_1. Those vertices form a subgraph G_2. At depth 3, it considers all vertices (that are in the depth 2) adjacent to the vertex expanded in the depth 2 etc. Let v_{d1} be the vertex the algorithm is currently expanding at the depth d. That is:

Let us 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 $(v_{d+1\ i}, v_d) \in E$ [7].

3.2 Re-applying a heuristic vertex colouring

Tomita and Seki algorithm [14] is an algorithm that belongs to a class of algorithms re-applying a heuristic vertex colouring. This algorithm can be seen as a successful development of the classical Carraghan and Pardalos algorithm [7]. The last one used the following rule for the pruning:

$$\text{if } d - m + n \le CBC \qquad (1)$$

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

Here the "$- m + n$" part represents a "degree" of the branch from the maximum clique point of view, i.e. how many vertices can be potentially included into the forming maximum clique. It is obvious that this estimation is very rough and a lot of attempts have been done to improve that. Tomita and Seki have replaced this measure with a number of colour classes existing on a branch. The pruning formula of the algorithm is the following:

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

where d is a depth, *Degree* is the depth (subgraph) degree, which is the number of colour classes existing on the subgraph and *CBC* is the current maximum clique size.

The number of existing colour classes is much more precise estimation than the number of remaining vertices, so the number of analysed subgraphs decreased dramatically. An idea of using a heuristic vertex colouring has been introduced so far, which is used by both algorithms classes described in this chapter. The main difference is in approaches to calculating the number of existing colour classes. The algorithm presented in this subchapter finds the heuristic vertex-colouring on each new depth. Besides, it reorders vertices after finding the colouring by colour index in decreasing order and considers vertices during the maximum clique search on the depth starting from the maximum colour index one. This allows using vertex colour indexes instead of re-counting colour classes to find the degree. The pruning formula is reformulated in Tomita and Seki algorithm and is the following:

$$\text{if } d - 1 + colour\_index(m) \leq CBC \qquad (3)$$

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

3.3 Re-using a heuristic vertex colouring

A class of algorithms obtaining a vertex colouring only once and thereafter re-using it is represented by an algorithm developed by Kumlander [11]. This algorithm was developed 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 colour index decreasing order. The algorithm uses the vertex colouring to apply a pruning rule similar to the described above and a backtracking search that will be described below. The backtracking search cannot be used if vertices are re-colouring on each depth as the backtracking relies on a fixed vertices ordering. Therefore it is a natural part of the colouring re-using algorithms.

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:

$$\text{if } d\text{-}1 + Degree \leq CBC \qquad (4)$$

where d is a depth, *Degree* is the depth (subgraph) degree, which is the number of which is the number of colour classes existing on the subgraph and *CBC* is the size of the current maximum clique. The algorithm calculates the degree by counting colour classes existing on the depth. The degree is calculated only ones when the depth is formed and later only adjusted by decreasing it on one when the next vertex to be analysed is from another colour class than the previous one. It is because the ordering since the described case means that there are no more vertices of the previous vertex colour class and the class ca be excluded from the count.

The backtracking search technique examines the graph in the opposite to the standard branch and bound algorithm's order. First of all it considers all cliques that could be built using only v_n. Then it considers all cliques that contain v_{n-1} and could contain v_n, and so forth. The general rule – it considers at the i-th step all cliques that contain v_i and could contain vertices $\{v_{i+1}, v_{i+2}, \dots, v_n\}$. So we move from the n-th step to the first step decreasing the step number. The algorithm uses to remember the maximum clique found for each vertex at the highest level into a special array b. So $b[i]$ is the maximum clique for the i-th vertex while searching backward. This numbers are used later by the following rule: if we search for a clique of size greater than s, then we can prune the search if we consider v_i to become the $(j + 1)$-the vertex and $j + b[i] \leq s$ [16]. Besides we can stop the backtracking iteration and go to the next one if a new maximum clique is found since the maximum clique size of a subgraph formed by $\{v_{i+1}, v_{i+2}, \dots, v_n\}$ is either equal to the maximum clique size of a subgraph formed by $\{v_{i+2}, v_{i+3}, \dots, v_n\}$ (the previous step) or is larger on 1.

A colour classes backtracking technique does the same, but operates on the colour classes' level. Initially vertices are sorted by colour classes, i.e. $V = \{C_n, C_{n-1}, \dots, C_1\}$, where C_i is the i-th colour (we call it the i-th colour class). The algorithm 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 it considers all cliques that could be built using vertices of C_1 and C_2, i.e. of the first and second colour classes, and so forth. The general rule – it considers during the i-th iteration all cliques that can be build using $\{C_i, C_{i-1}, \dots, C_1\}$ vertices. The algorithm also uses to remember the maximum clique found for each backtracking iteration into a special array b. So $b[i]$ is the maximum clique for a subgraph formed by $\{C_i, C_{i-1}, \dots, C_1\}$ vertices. The remembered clique sizes are used in the following pruning rule: if the algorithm searches for a clique of size greater than s, then it can prune the search if the next vertex on a depth to be analysed has index $j + 1$, belongs to the k-th colour class and $j+ b[k] \leq s$. The stopping condition of the backtracking search iteration remains since the maximum clique size of a subgraph formed by $\{C_i, C_{i-1}, ..., C_1\}$ is either equal to the maximum clique size of a subgraph formed by $\{C_{i-1}, ..., C_1\}$ or is greater on 1. It can be explained by the fact that each next iteration contains just one more colour class, which is an independent set, into addition to the analysed set of vertices, so all added vertices are pairways nonadjacent and therefore there are no two or more vertices which can be added into a new maximum clique.

4 Tests

Here the described algorithms (i.e. both ways of using classes) are analysed on DIMACS graphs, which are a special package of graphs used in the Second DIMACS Implementation Challenge [9, 10] to measure performance of algorithms. Those graphs represent different graph types

As it has been mentioned already, there is a very simple and effective algorithm for the maximum clique finding proposed by Carraghan and Pardalos [7]. This algorithm was used as a benchmark in the Second DIMACS Implementation Challenge [10]. Besides, using of this algorithm as a benchmark is advised in one of the DIMACS annual reports [9]. That's why it will be used in test below and is called the "base" algorithm. Test 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 [7]. The quicker a tested algorithm works the larger ratio is, so the ratio shows how much quicker the tested one is in compare to the base one. The compared algorithms were programmed using the same programming language and the same programming technique.

One more algorithm proposed by Östergård [16] was selected to participate in the comparison tests since this algorithm was reported as the quickest one several years ago. This will show the performance of the described algorithms in compare to other quick algorithms.

The greedy algorithm was used to find a vertex-colouring. A table below contains the following columns:

PO – time needed to find the maximum clique by the base algorithm divided by time needed to find the maximum clique by P. Östergård algorithm [16].

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 [14].

DK – 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 [11].

Table 1. Benchmark results on DIMACS graphs

Graph name	*PO*	*TS*	*DK*
brock200_2	2.3	2.3	*4.0*
brock200_3	1.2	*3.3*	3.2
hamming8-4	247.8	39.9	*7848.3*
johnson16-2-4	4.4	7.0	*20.9*
keller4	2.8	6.7	*11.8*
p_hat300-1	1.0	1.0	*1.3*
p_hat300-2	2.0	4.8	*6.6*
p_hat500_1	0.9	0.9	*1.5*
p_hat700_1	1.1	1.1	*1.9*
sanr400_0.7	1.7	1.4	*5.6*
2dc.256*	4.6	6.6	*14.5*

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

For example, 2.3 in the column marked *PO* means that Östergård [16] algorithm requires 2.3 times less time to find the maximum clique than the base algorithm. The quickest result of each row is highlighted by the italic font. Presented results show that the *DK* algorithm [11] outperforms others in many cases. Both reviewed in the paper algorithms are faster than the benchmarking algorithms.

5 Conclusion

In this paper two best known algorithms for finding the maximum clique has been reviewed. The first one is proposed by Tomita and Seki [14], the second one by Kumlander [11]. Both algorithms are branch and bound and both are using colour classes obtained from a heuristic vertex-colouring to find the maximum clique. Unlike other historical attempts to use vertex-colouring, those algorithms contain using colour classes as an essential part rather than just a bounding stopping condition. The difference between algorithms is identified in a "level" of using colour classes. One of them is re-applying continuously the heuristic vertex-colouring on each new depth. The other one re-use the colouring obtained during a pre-step through the algorithm work. The re-applying technique makes it possible to estimate the potential maximum clique size existing on a depth (subgraph) easily. The re-using technique makes it possible to use the backtracking search in addition since is not breaking the vertices order on each depth. The degree calculation routine is still a clear advantage over using the number of remaining vertices. The paper contains a description of both algorithms on a high level. Thereafter tests on several DIMACS graphs are conducted. The general result is that the re-using is the better technique is most cases, although the re-applying is reported to be better on random graphs. The re-using technique is winning mostly because of using the backtracking technique one in addition. Anyway both techniques are better than algorithms reported to be best several years ago.

References:

[1] L. Babel, G. Tinhofer, A branch and bound algorithm for the maximum clique problem, *Methods and Models of Operations Research*, Vol. 34, 1990, pp. 207-217

[2] D.H. Ballard, M. Brown, *Computer Vision*, Prentice-Hall, Englewood Cliffs, NJ, 1982

[3] 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

[4] R.E. Bonner, On some clustering techniques, *IBM J. of Research and Development*, Vol. 8, 1964, pp. 22-32

[5] D. Brelaz, New Methods to Color the Vertices of a Graph, *Communications of the ACM*, Vol. 22, 1979, pp. 251-256

[6] A.E. Brouwer, J.B. Shearer, N.J.A. Sloane, W.D. Smith, A new table of constant weight codes, *J.IEEE Trans. Information Theory*, Vol. 36, 1990, pp. 1334-1380

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

[8] M.R. Garey, D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, Freeman, New-York, 2003

[9] DIMACS, *Annual Report*, Center for Discrete Mathematics and Theoretical Computer Science, 1999

[10] 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

[11] D. Kumlander, Problems of optimization: an exact algorithm for finding a maximum clique optimized for dense graphs, *Proceedings of the Estonian Academy of Sciences*, Vol. 54, No. 2, 2005, pp. 79-86

[12] W. Miller, Building multiple alignments from pairwise alignments, *Computer Applications in the Biosciences*, Vol. 9, 1992, pp. 169-176

[13] N.J.A. Sloane, Unsolved problems in graph theory arising from the study of codes, *Graph Theory Notes of New York*, Vol. 18, 1989, pp. 11–20

[14] E. Tomita, T. Seki, An effcient branch-and-bound algorithm for finding a maximum clique, *Discrete Mathematics and Theoretical Computer Science, 4th International Conference*, *LNCS 2731 Springer*, Vol. 2003, pp. 278-289

[15] D.R. Wood, An algorithm for finding a maximum clique in a graph, *Operations Research Letters*, Vol. 21, 1997, pp. 211-217

[16] P.R.J. Östergård, A fast algorithm for the maximum clique problem, *Discrete Applied Mathematics*, Vol. 120, 2002, pp. 197-207