IADIS International Journal on Computer Science and Information Systems
Vol. 1, No. 2, pp. 32-49
ISSN: 1646-3692

A SIMPLE AND EFFICIENT ALGORITHM FOR THE MAXIMUM CLIQUE FINDING REUSING A HEURISTIC VERTEX COLOURING

Deniss Kumlander *Department of Informatics, Tallinn University of Technology, Raja St.15, 12617 Tallinn, Estonia*

ABSTRACT

In this paper a practical algorithm for finding the maximum clique is proposed. The maximum clique problem is well known to be NP-hard and is a core problem for a lot of applications in artificial intelligence systems, data mining and many others. The presented algorithm contains some additions to its earlier publications, which makes it much faster. It is based on colour classes and the backtracking technique. This paper includes a description of the algorithm, an example of its work and some analytical discussion topics. The algorithm is tested on DIMACS graphs to compare it with other well-known algorithms. It has shown very good performance and is more than 1000 times faster than others best known algorithms on some graph types. Moreover certain modifications of the heuristic colouring strategies described in the article produce even better algorithms for some graph types introducing a need for an artificial intelligence approach in the maximum clique finding algorithms' implementations. The described algorithm is fast and easy to implement, which makes it very practical to apply in a plenty of areas.

KEYWORDS

Graph theory, maximum clique, DIMACS graphs, heuristic vertex colouring.

1. INTRODUCTION

Let $G=(V, E)$ be an undirected graph, where V is the set of vertices and E is the set of vertices. 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 colours to its vertices so that no pair of adjacent vertices shares an identical colour. 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. Those problems are NP-hard on general graphs and no polynomial time algorithms are expected to be found.

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 [MacWilliam and Sloane 1979], geometric tiling [Corradi and Szabo 1990], fault diagnosis [Berman and Pelc 1990], pattern recognition [Horaud and Skordas 1989], molecular biology [Mitchell et al 1989], and scheduling [Jansen et al 1997]. Additional applications arise in more comprehensive problems that involve graph problems with side constraints. This problem is surveyed in [Bomze et al 1999].

The basic ideas of using colour classes for the maximum clique finding has been presented previously by us in several papers [Kumlander 2003, Kumlander 2004, Kumlander 2005]. We started from an algorithm that was quickest only on dense graphs [Kumlander 2005] and then proposed a general type algorithm [Kumlander 2004]. The algorithm has shown that it is the best one at the moment [Kumlander 2004] on random graphs. Here we are going to present an adjusted version of the algorithm, which contains some speeding techniques and therefore is slightly better than previous its versions. Besides this paper is the first one where we conduct a comparison test on DIMACS graphs with other algorithms to find out how this new algorithm works on different special graph types.

2. NEW ALGORITHM

In this subchapter we introduce an algorithm based on the idea of using independent sets or colour classes. It also uses a backtracking search [Östergård 2002]. The new algorithm is based on the branch and bound idea and is further development of the Carraghan and Pardalos algorithm [Carraghan and Pardalos 1990], which have shown its efficiency in many tests [Johnson and Trick 1996, Bomze et al 1999].

2.1 Description

2.1.1 Initial idea of using colour classes

The algorithm is based on an elementary property of a clique: vertices that are unadjacent cannot be included into the same clique. The following will explain how this property can be used for finding the maximum clique to speed up this search.

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.

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 becomes 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.

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 on a d-th depth and induced by E is smaller or equal to the size of the current maximum clique minus $(d - 1)$ then this subgraph cannot form a clique, which is larger than the already found.

Prove: It is clear to see that $(d - 1)$ equals to the number of vertices formed the d-th depth subgraph, i.e which where expanded on previous depths. Those $d - 1$ vertices are connected pairways and to each vertex of the subgraph of the d-th depth by the logic of the branch and bound algorithm. 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 clique minus $(d-1)$. If such clique exists then the maximal clique of the graph G will be the clique of the subgraph plus d-1 vertices selected on previous depths, which are connected to all vertices of the subgraphs by the branch and bound algorithms logic and this maximal clique will be larger than an already found, so it will be a new maximum one. So, the only statement we need to prove is: the *Degree* function's value of the subgraph is never smaller than the maximum clique size that can be found on the subgraph, because then we can use in the pruning formula the degree function to estimate the size of the clique instead of finding it. The degree function gives a number of colours (colour classes) by definitions above and each colour class is an independent set of vertices existing on the depth. No more than one vertex of

each colour class can participate in the maximum clique by the independent set's definition. Therefore the number of colours classes existing on the subgraph always equals or is bigger than a size of the maximum clique that can exist on the subgraph. ■

2.1.2 Colour classes and backtracking

Here we are going to demonstrate a backtracking technique, which is proposed by Östergård [Östergård 2002], and how previously described colour classes can dramatically increase performance. The initial backtracking idea of Östergård [Östergård 2002] is also based on the Carraghan and Pardalos algorithm [Carraghan and Pardalos 1990].

The original Carraghan and Pardalos algorithm considers, first of all, all cliques that contain the first vertex v_1 and could contain other graph vertices. Then it considers all cliques that contain v_2 and could contain all other vertices except v_1. Generally saying, it considers at the i-th step all cliques that contain v_i and could contain vertices $\{v_{i+1}, v_{i+2}, \ldots, v_n\}$. This technique is nothing else than a standard branch and bound way of drilling a graph for finding the solution.

The backtracking technique does the graph research in the opposite order, although the list of vertices on the i-th step is the same. 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 $\{v_{i+1}, v_{i+2}, \ldots, v_n\}$. So we move from the n-th step to the first step decreasing the step number. Initially it looks like a slower technique in compare to the original Carraghan and Pardalos algorithm [Carraghan and Pardalos 1990], but it makes possible to introduce a new backtracking pruning technique speeding up the algorithm's work. First of all, note that the backtracking vertices selection is used only on the "general" level – as soon as vertices are selected for the i-th backtracking step the same branch and bound technique is used. The branch and bound algorithm uses the same Carraghan and Pardalos pruning technique and the new backtracking pruning technique described below. 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 $j + b[i] \leq s$ [Östergård 2002]. 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 $\{v_{i+1}, v_{i+2}, \ldots, v_n\}$ is either equal to the maximum clique size of a subgraph formed by $\{v_{i+2}, v_{i+3}, \ldots, v_n\}$ (the previous step) or is larger on 1. Please refer to the original Östergård article [Östergård 2002] for proves that this technique always gives the exact solution.

Now we are going to introduce the "independent sets (or colour classes) backtracking technique. We do the same as described above, except we operate on the "independent sets" level of considering a graph. Initially we sort vertices by colour classes obtained by a heuristic vertex colouring algorithm, i.e. $V = \{C_n, C_{n-1}, \ldots, C_1\}$, where C_i is the i-th colour (or we call it the i-th colour class). During the algorithm work we consider first of all all cliques that could be built using only vertices of the C_1, i.e. of the first colour class. Then we consider 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 – we consider at the i-th step all cliques can that contain vertices of $\{C_i, C_{i-1}, \ldots, C_1\}$. Note that here we again move from the first step to the n-th since colour classes are in the backward order.

Besides the algorithm uses to remember the maximum clique found for each step on the high level into a special array *b*. So $b[i]$ is the maximum clique for a subgraph formed by $\{C_i, C_{i-1}, ..., C_1\}$ vertices 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 $(j + 1)$-th vertex and it belongs to the *k*-th colour class and $j + b[k] \leq s$. The stopping condition of the backtrack search iteration is also 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 larger on 1. It is so because each time we just add a colour class, i.e. an independent set into addition to the analysed set of vertices. The new maximum clique cannot differ more than on 1 vertex from the maximum clique on the previous iteration since all added vertices are pairways nonadjacent and therefore there are no two or more vertices which are adjacent and can be used / added to a new maximum clique.

Note, that it is important to sort vertices as we have shown it at the start of the description: $V = \{C_n, C_{n-1}, ..., C_1\}$, i.e. first of all in the new sorted order vertices of the *n*-th colour class should appear, then vertices of the $(n-1)$-th colour class and so forth. This will speed-up the *Degree* function calculation - instead of calculating the degree of a subgraph each time on a depth we will calculate it only once, when the depth is formed and later only adjust this value by the following rule: if the next vertex on the depth to be expanded is from the same colour class as the previous one then the degree remains the same otherwise the degree should be decreased by 1 (there are no more vertices from the previous vertex' colour class and it is eliminated).

Algorithm for the maximum clique problem – "*VColor-BT-u*"

CBC - current best (maximum) clique
d – depth
i – index of the currently processed colour class in the backtracking
b – array of the backtrack search results
$C(v_i)$ – a function that return a colour class to which the vertex v_i belongs
G_d – subgraph of *G* formed by vertices existing on the depth *d*

Step 0. Heuristic vertex-colouring: Find a vertex colouring and reorder vertices so that first vertices belong to the last found colour class then vertices of the previous to last colour class and so forth – vertices at the end should belong to the first colour class. *Note: It is advisable to use a special array to solve order of vertices to avoid a work by changing the adjacency matrix during reordering vertices.*

Step 1. Backtracking: For each colour class starting from the first one up to the last, i.e. $i = i+1$:

Step 1.1. Subgraph building. Form the first depth by selecting all vertices of the current colour class under the analysis and other colour classes, whose index is smaller than the index of the current colour class.

i = to the index of the current colour class.

Step 1.2. Run the subgraph research: Go to step 2

Step 2. Initialization: $d = 1$.

Step 3. Control: If the current depth can contain a larger clique than already found:

Step 3.1. If $d - 1 + Degree(G_d) \leq |CBC|$ then go to the step 6.

Step 3.2. if $C(v_{d,1}) > i$ then If $d - 1 + b[C(v_{d,1})] \leq |CBC|$ then go to the step 6.

Step 4. Expand vertex: Get the next vertex to expand. If all vertices have been expanded or there are no vertices then control if the current clique is the largest one. If yes then save it as the maximum clique and go to the step 1.3.

Step 5. The next depth: Form the new depth by selecting all remaining vertices that are connected to the expanding vertex from the current depth;

$d = d + 1$;

Go to the step 3.

Step 6. Step back:

$d = d - 1$;

Delete the expanded vertex from the analyse on this depth;

if $d = 0$, then go to the step 1.3, otherwise go to the step 3.

Step 1.3. Completing iteration: $b[i] = CBC$, go to the step 1.

End: Return the maximum clique.

Steps from 2 to 6 can be considered as a subprocedure that the backtracking runs iteratively in a cycle for each colour class.

2.1.3 Examples

In this chapter we are going to demonstrate some examples of the described algorithm work. The same graphs will be used as for the previous algorithm.

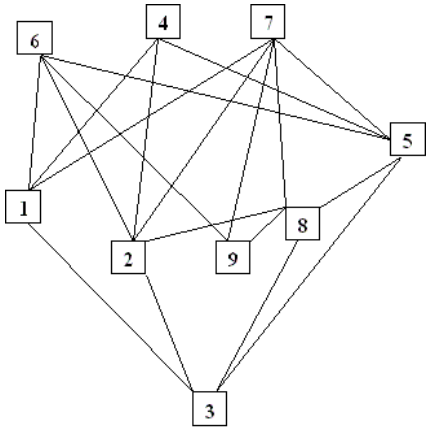

Figure 1. A graph for the example number 1

Description of the example graph

Consider graph shown in Figure 1. It is a graph that is built using the Moon-Moser type subgraph containing vertices 1, 2 and 5 for the first class and vertices 6, 4 and 7 for the second class. Vertices 3, 9 and 8 are used to make the graph's structure more complex and contain larger cliques that the Moon-Moser subgraph produces.

Algorithm's steps

We determine colour classes one by one as long as uncoloured vertices exist in a greedy manner. Vertices are also resorted in an order those are added into colour classes. So, vertex colouring gives the following result:

Colour class 1={1, 2, 5, 9}
Colour class 2={3, 4, 6,7}
Colour class 3={8};
The order of vertices is the following: {8, 7, 6, 4, 3, 9, 5, 2, 1}

Let's use the following notation in the example: *CBC* – the current best clique and |*CBC*| is its size. A grey vertex in the table below is a vertex under analysis and vertices in front of that are vertices that have been already analysed and cannot participate in the forming maximum clique any longer. So instead of deleting vertices we will just process them one by one in the example by moving a cursor, which always point to the grey vertex. A grey vertex in the table below is a vertex under analysis and vertices in front of that are vertices that have been already analysed and cannot participate in the forming maximum clique any longer.

Steps of the main algorithm's part (finding the maximum clique) are described in the next table.

Table 1. "*VColor-BT-u*" - Example / Steps of finding the maximum clique

Depth	Subgraph	Step's description
Depth 0:	8,7,6,4,3, *9,5,2,1*	To do: Start a backtrack search from the first class by selecting vertices of it into the depth 1 and run main steps. $i = 1$
Depth 1:	9,5,2,1	|*CBC*| is 0; *Degree* = 1, since only first colour class' vertices exist. *d*-1+*Degree*=1-1+1=1. Since 1>|*CBC*| we can continue. $C(v_{11})$=1 since v_{11} belongs to the colour class number 1. The backtracking pruning is skipped since $C(v_{11}) = i$. Go to the next depth: the grey vertex v_{di} (v_{11}) to be expanded.
Depth 2:	∅	The depth doesn't contain any vertices ⇒ Check if the formed clique is the largest one: The formed clique is {9} and |*CBC*| =0, so *CBC* becomes {9}, size=1. *b*[1] =1. Go to the next iteration of the backtrack search.
Depth 0:	8, *7,6,4,3,9,5,2,1*	To do: Start the next step of the backtrack search by selecting into the depth 1 vertices of colour classes 1 and 2, and run main steps. $i = 2$
Depth 1:	7,6,4,3,9,5,2,1	|*CBC*| is 1; *Degree* = 2, since existing vertices belong to colour classes 1 and 2. *d*-1+*Degree*=1-1+2=2. Since 2>|*CBC*| we can continue. $C(v_{11})$=2 since v_{11} belongs to the colour class number 2. The backtracking pruning is skipped since $C(v_{11}) = i$. Go to the next depth: the grey vertex v_{di} (v_{11}) to be expanded.

Depth 2:	9,5,2,1	|*CBC*| is 1; *Degree* = 1, since all vertices belong to the colour class number 1. *d*-1+*Degree*=2-1+1=2. Since 2>|*CBC*| we can continue. $C(v_{21})$=1 since v_{21} belongs to the colour class number 1 => We can check the backtrack pruning condition: d –1 + $b[C(v_{21})]$ = 2-1+1 = 2 >|*CBC*| we can continue. Go to the next depth: the grey vertex v_{di} (v_{21}) to be expanded.
Depth 3:	∅	The depth doesn't contain any vertices ⇒ Check if the formed clique is the largest one: The formed clique is {7, 9} and |*CBC*| =1, so *CBC* becomes {7, 9}, size=2. b[2] =2. Go to the next iteration of the backtrack search.
Depth 0:	*8,7,6,4,3,9,5,2,1*	To do: Start the next step of the backtrack search by selecting into the depth 1 vertices of colour classes 1, 2 and 3, and run main steps. i = 3
Depth 1:	8,7,6,4,3,9,5,2,1	|*CBC*| is 2; *Degree* = 3, since vertices belong to colour classes 1, 2 and 3. *d*-1+*Degree*=1-1+3=3. Since 3>|*CBC*| we can continue. $C(v_{11})$=3 since v_{11} belongs to the colour class number 3. The backtracking pruning is skipped since $C(v_{11}) = i$. Go to the next depth: the grey vertex v_{di} (v_{11}) to be expanded.
Depth 2:	7,3,9,5,2	|*CBC*| is 2; *Degree* = 2, since all vertices belong to colour classes 1 and 2. *d*-1+*Degree*=2-1+2=3. Since 3>|*CBC*| we can continue. $C(v_{21})$=2 since v_{21} belongs to the colour class number 2 => We can check the backtrack pruning condition: d –1 + $b[C(v_{21})]$ = 2-1+2 = 3 >|*CBC*| we can continue. Go to the next depth: the grey vertex v_{di} (v_{21}) to be expanded.
Depth 3:	9,5,2	|*CBC*| is 2; *Degree* = 1, since all vertices belong to the colour class number 1. *d*-1+*Degree*=3-1+1=3. Since 3>|*CBC*| we can continue. $C(v_{31})$=1 since v_{31} belongs to the colour class number 1 => We can check the backtrack pruning condition: d –1 + $b[C(v_{31})]$ = 3-1+1 = 3 >|*CBC*| we can continue. Go to the next depth: the grey vertex v_{di} (v_{31}) to be expanded.
Depth 4:	∅	The depth doesn't contain any vertices ⇒ Check if the formed clique is the largest one: The formed clique is {8, 7, 9} and |*CBC*| =2, so *CBC becomes* {8, 7, 9}, size=3. b[3] =3. Go to the next iteration of the backtrack search.
Depth 0:	*8,7,6,4,3,9,5,2,1*	Since all colour classes are analysed the algorithm stops.

The maximum clique is {8, 7, 9}, size = 3.

Analysis of this example

The algorithm needed just 17 steps to find the maximum clique from the graph of 8 vertices. This result is very good, since the maximum clique finding problem is NP-hard and a lot of algorithms just do an exhaustive search or need a sufficient number of steps to find a solution. This graph is not so "good" for applying with this type of algorithm – as you have probably marked the backtracking pruning formula never worked in this example. At the same time it is possible to learn a lot from this example as well. It demonstrated to us a power of using:

- The backtracking with independent sets – using of backtracking with independent sets has a set of advantages. First of all we do less iterations since select all vertices of a class. At the same moment the number of steps inside each iteration do not increase since colour class' vertices are "parallel", i.e. cannot be included into the same maximum clique and have equal $b[i]$ value, since are coloured into the same colour.
- The stopping condition of the backtracking iteration – We have skipped a lot of steps using a rule that if we have found a new maximum clique then we can go directly into the next backtracking iteration, since the current iteration's subgraph cannot produce any larger clique. This stop condition is a very important technique in addition to the backtracking pruning rule.

2.2. Preliminary algorithm verification on DIMACS graphs

In this section we are going to present results showing a general efficiency of the previously described algorithm to ensure that the algorithm is worth to implement, research and improve further.

A very simple and effective algorithm for the maximum algorithm problem proposed by Carraghan and Pardalos [Carraghan and Pardalos 1990] was used as a benchmark in the Second DIMACS Implementation Challenge [Johnson and Trick 1996]. That's why we are going to use it in the benchmarking. Besides, using of this algorithm as a benchmark in advised in one of the DIMACS annual reports. Therefore we used this algorithm to compare with the new algorithm. Besides we have chosen one more algorithm proposed by Östergård [Östergård 2002] to participate in the comparison test since this algorithm is reported to be the quickest at the moment and this algorithm is also another modification of Carraghan and Pardalos algorithm [Carraghan and Pardalos 1990]. Moreover our algorithm was also based on its ideas.

Results are presented as a ratio of algorithms spent times on finding the maximum 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 (it was possible since the new algorithm and Östergård [Östergård 2002] algorithm are just modifications of Carraghan and Pardalos algorithm). The greedy algorithm was used to find a vertex-colouring.

Preliminary tests were conducted on DIMACS graphs, which are a special package of graphs used in the Second DIMACS Implementation Challenge [Johnson and Trick 1996] to test different algorithms and find out what of them are the best one and on what types of graphs.

The following notation is used in the table 2 below:

PO – time needed to find the maximum clique by Carraghan and Pardalos [Carraghan and Pardalos 1990] algorithm divided by time needed to find the maximum clique by Östergård [Östergård 2002] algorithm.

VColor-BT-u – time needed to find the maximum clique by Carraghan and Pardalos [Carraghan and Pardalos 1990] algorithm divided by time needed to find the maximum clique by the invented algorithm.

Table 2. Benchmark results at DIMACS graphs – ratios of time spent on the maximum clique finding / the base algorithm's time divided by a corresponding algorithm's time

Graph name	Edge density	Vertices	Maximum clique size	*PO*	*VColor-BT-u*
brock200_1	75%	200	21	2.1	8.4
brock200_2	50%	200	12	2.3	4.0
brock200_3	61%	200	15	1.2	3.2
brock200_4	66%	200	17	2.0	6.0
c-fat200-5	43%	200	58	58.2	49.2
c-fat500-1	4%	500	14	0.7	1.0
c-fat500-2	7%	500	26	1.2	2.2
c-fat500-5	19%	500	64	72.1	85.4
hamming6-2	90%	64	32	493.0	493.0
hamming8-4	64%	256	16	247.8	7848.3
johnson8-4-4	77%	70	14	11.9	53.3
johnson16-2-4	76%	120	8	4.4	20.9
keller4	65%	171	11	2.8	11.8
MANN_a9	93%	45	16	12.5	42 400.0
p_hat300-1	24%	300	8	1.0	1.3
p_hat300-2	49%	300	25	2.0	6.6
p_hat500_1	25%	500	9	0.9	1.5
p_hat700_1	25%	700	11	1.1	1.9
sanr400_0.7	70%	400	21	1.7	5.6
2dc.256*	47%	256	7	4.6	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, 20.9 in the "*VColor-BT-u*" column means that the algorithm proposed in this paper is 20.9 times faster than the base algorithm, which is Carraghan and Pardalos one.

The following table provides a brief description of the used graphs:

Table 3. Description of DIMACS graphs

Graph type	Description
Bro	Instances from Mark Brockington and Joe Culberson's generator that attempts to "hide" cliques in a graph where the expected clique size is much smaller. For more instances, see their generator in graph/contributed/brockington. From Mark Brockington.
CFat	Problems based on fault diagnosis problems [Berman and Pelc 1990]. For more instances, see the generator in graph/contributed/pardalos. From Panos Pardalos.
Ham*	Another coding theory problem. A Hamming graph with parameters *n* and *d* has a node for each binary vector of length *n*. Two nodes are adjacent if and only if the

	corresponding bit vectors are hamming distance at least *d* apart. For more instances, see the generator in graph/contributed/pardalos. It has been noted by participants that *n*--2 graphs have a maximum clique of size 2^{n-1}. For a proof of this, see the note in graph/contributed/bourjolly/hamming. From Panos Pardalos.
Joh*	Problems based on problem in coding theory. A Johnson graph with parameters *n*, *w*, *d* has a node for every binary vector of length *n* with exactly *w* 1s. Two vertices are adjacent if and only if their hamming distance is at least *d*. A clique then represents a feasible set of vectors for a code. For more instances, see the generator in graph/contributed/Pardalos. From Panos Pardalos.
Kel*	Problems based on Keller's conjecture on tilings using hypercubes [Lagarias and Shor 1992]. For more instances (though they get very large very fast) see either the generator in graph/contributed/shor or the generator in graph/contributed/pardalos. From Peter Shor.
MANN* (Stein)	Clique formulation of the set covering formulation of the Steiner Triple Problem. Created using Mannino's code to convert set covering problems to clique problems. From Carlo Mannino.
PHat*	Random problems generated with the *p hat* generator which is a generalization of the classical uniform random graph generator. Uses 3 parameters: *n*, the number of nodes, and *a* and *b*, two density parameters verifying $0 \leq a \leq b \leq 1$. Generates problem instances having wider node degree spread and larger clique sizes [Gendreau et al 1993]. From Patrick Soriano and Michel Gendreau.
San*	Instances based on Sanchis paper [Sanchis 1992] From Laura Sanchis laura@cs.colgate.edu
SanR*	These are random instances with sizes similar to those in *San.* From Laura Sanchis.
2dc*	Graphs From Two-Deletion-Correcting Codes. From N. J. A. Sloane.

The first result to be highlighted in the previous results' table is that on most instances the "*VColor-BT-u*" algorithm is the quickest one. The only instance, where Östergård [Östergård 2002] algorithm is the best one, is the "c-fat200-5". This occurs because the backtracking pruning technique is decreasing the performance of the applying colour classes on those instances, but the "*VColor-BT-u*" is still close. Another interesting result is an extremely good performance of new algorithms for "MANN" and "hamming" type graphs. Those graphs are again graphs of the high density.

2.3 Algorithm's analysis

The previous section that contains preliminary tests for the invented algorithm has demonstrated its efficiency and enables further researches of the algorithm.

2.3.1 Colour classes based estimation

The first element to be analysed is a concept of using colour classes instead of vertices in the pruning formulas described so far. The theorem proved earlier ensures that this replacement is exactly correct and this allows estimating a potential size of the maximum clique of a graph's branch under analyses much better than the number of remaining vertices. The question that will be asked below: is it enough and how close the algorithm could be in its estimations using the colour classes concept. An idea of using a chromatic graph number as an upper bound during the maximum clique search is widely known [Babel and Tinhofer 1990, Wood 1997] and looks to be a natural way for such estimations because the maximum clique by its definition requires a special colour for each vertex, i.e. the maximum clique size cannot be

larger than a chromatic number. There is a "sandwich" theorem [Knuth 1994] that is focused on a *Lovasz number*: $\theta(\hat{G})$, which is said to be a sandwich between the minimum number of colours required (the chromatic number - $\chi(G)$) and a size of the maximum clique - $w(G)$:

$$w(G) \leq \theta(\hat{G}) \leq \chi(G),$$

where $\hat{G}$ is the complement to G graph. The *Lovasz number* could be calculated in a polynomial time [Bomze et al. 1999]. First of all this theorem shows that the chromatic number is always larger that the maximum clique size as it was stated before. Besides it demonstrates that there could be some distance between those numbers. A size of this distance is a core element defining efficiency of the proposed algorithms. The smaller this distance the faster algorithms are. Fortunately, in compare with other (best) well-known algorithms, independently of this distance, the pruning formulas based on the vertex-colouring is able to produce a faster solution, especially on dense graphs, where it is practically the only pruning technique that keeps working. The smaller density of a graph the more depending on this distance becomes the algorithm in comparing with others. This dependency is explained by considering a question: if the maximum clique is already found then how fast the algorithm could prove in someway that it is the maximum one. So the main thing that makes our problem so hard in many cases is not the problem of finding of the maximum clique but the problem of proving that it is the maximum. The closer a size of the maximum clique to a chromatic number, the more efficiently the algorithms prune and the faster it is possible to prove that the found clique is the maximum one.

Graphs "easy" to solve

As it was shown before, the "best" graphs to be solved by the new algorithm are graphs where the chromatic number is close to the maximum clique size. There are a sufficient number of graphs' classes where it is true. The most interesting example of such graphs, if we consider the hardness to solve it by other algorithms, is a graph with a lot of semi-parallel structures like Moon-Moser graphs. The more such structures are there the easier this graph is to solve by the introduced algorithm in compare with other algorithms since the complexity of such structures are eliminated by using the vertex-colouring strategy since the vertex-colouring degree function could produce a closer estimation for a potential clique size in such (sub)graphs than other techniques.

Graphs "hard" to solve

Unfortunately there are much more graphs / graph classes where the chromatic number sufficiently differs from the maximum clique size [West 2001]. It is even possible to construct a graph with the chromatic number as large as you would like while the maximum clique size remains the same. See for example the "Mycielski's construction" [West 2001] that does it. Such graphs should be "hard" to solve since the invented pruning technique will not help to avoid producing the combinatorial branch and bound search although the situation should be still better than the pruning strategy invented by Carraghan and Pardalos [Carraghan and Pardalos 1990a]. Unfortunately this difference will not be enough for a sufficient change in a time needed to find the maximum clique. Another property of a graph "hard" to solve in addition to the previous one is to have as less parallel structure as possible. Otherwise vertices

producing a large chromatic number would be eliminated during first steps and a graph will degenerate to the "easy" to solve case.

2.3.2 Overall algorithm's structure and backtracking

The second element to be analysed is the implemented transition of the backtracking search from the vertices level to the colour classes' one regarding the colour classes ordering strategy. At first glance it looks like such transition moves the precision grain from a better estimated one to a much rougher. The vertices level estimation (the backtracking pruning rule) of course is not exact as the algorithm cuts branches basing on the measure showing how large clique can be produced using a vertex and including remaining upto the last vertex, but some vertices among remaining are filtered out on different depths by the forming maximum clique vertices (see the maximum clique definition). Therefore the actual maximum clique that can be formed using depth's vertices is usually much smaller. Despite that the colour classes' level backtracking estimation still looks to be weaker since more vertices are included to the measuring process and the probability that remaining classes' vertices are connected (i.e. form a large clique) is lower. Fortunately it is not the case. Notice that the maximum clique size during the backtracking process is never decreased, but is monotonously increasing. The colour classes' definition says that no more than one vertex of the same colour can be used to form a clique and therefore the size of the backtracking process' clique cannot increase if those vertices are analysed one by one. At the same time the size of that backtracking maximum clique cannot decrease by the backtracking search definition. Therefore treating those colour class vertices at the same time (in one backtracking iteration) gives us the same estimation as treating those one by one, so using colour classes instead of individual vertices generates the same estimation grain.

Notice also the colour classes sorting strategy. Those are sorted in the colour number decreasing order, i.e. guarantees that each previous colour class' vertex is connected at least to one vertex of each following colour class, because the vertex wasn't included into any of those following colour classes having a smaller number than the vertex colour class by the ordering strategy. It means that at the moment of colouring a vertex to be considered those following colour classes already contained some vertices preventing including this vertex into those. Therefore during the backtracking strategy the algorithm considers all following vertices (after the active one) at the first level, so the pruning estimation described so far holds at least on that level and is starting to loose actuality with moving to other depths.

Another interesting fact to notice is the overall structure of the algorithm - the heuristic colouring is just reused during the algorithm work time, i.e. is formed only once and thereafter is concurrently used instead of re-colouring on each depth like some other algorithms do [Tomita and Seki 2003]. This strategy has pros and contras and the advantage element will be used in the following subchapter called "further improvements".

2.4 Further improvements for certain graph types

A vertex colouring problem is a NP-hard problem therefore the maximum clique problem is using a heuristic algorithms that doesn't guarantee producing an optimal solution (the minimum number of colours). Notice that there are different heuristic algorithms for solving the colouring problem. Generally saying the more time is spent on searching a solution the better result is found. So far the greedy colouring was used to obtain a heuristic solution in

most papers regarding the maximum clique finding problem [Kumlander 2005, Tomita and Seki 2003, Östergård 2002]. A complexity of this algorithm is approximately $O(V^2)$ depending on the algorithm's structure (memory use). The primary reason of using this algorithm as one producing a good enough solution is minimising the time spent on finding the maximum clique (as the colouring subtask time is included into the overall timing) knowing that there could be a sufficient gap between a colours classes' number and the maximum clique size as it was described in the previous subchapter. Moreover sometimes the vertex colouring subtask is rapidly re-run – see for example Tomita and Seki algorithm [Tomita and Seki 2003] re-obtaining colours for each depth and that is why sometimes the complexity of the used heuristic colouring algorithm affects the overall complexity greatly.

The algorithm proposed in this paper obtains colours only once and this fact opens a possibility to use more complex colouring algorithms if those will provide less colours, as the time spend on finding colouring will be included into the overall time only once, but the effect would be rapid in the combinatorial maximum clique search. Notice again that certain heuristic algorithms do produce a better solution only on certain types of graphs. Therefore an artificial intelligence principles similar to described in [Kumlander 2006] could be used to decide which colouring strategy suits best for a particular graph type.

Returning to the algorithm modification to be proposed the following heuristic algorithm and its implementation are to be suggested: instead of using the greedy algorithm it can be a good idea to use *DSatur* colouring strategy [Brelaz 1979] that is known to produce less colours in compare to the greedy one. *DSatur* (*degree of saturation largest first*) is a sequential colouring algorithm where the saturation degree defined as a number of colours a vertex is adjusted to. The algorithm identifies a vertex with the maximum saturation degree among uncoloured at each step and colours it with the least possible colour for that vertex (in its coloured neighbourhood). If a saturation degree will be equal for several vertices then the number of uncoloured neighbours is advised to be the next measure to use for the choice.

The saturation degree core idea is to try minimizing probability of setting an incorrect colour (i.e. a colour that will increase the number of colours required to colour a graph) by setting colours to a vertex with a maximum number of identified restrictions, which are colours of already coloured neighbours. The algorithm can be described in pseudo-code as the following:

Let's say that we have n vertices, W will be uncoloured vertices and *Colour*(v) is a function that returns a colour already assigned to a vertex v.

While $W \neq \emptyset$ (n steps)
    Find a vertex $v \in W$ with a maximum saturation degree
    Find a minimum colour that is not used in neighbourhood of v:
        $k := \min (i \mid \text{there is no } s : \text{Colour}(s) = i , (s,v) \in E)$
    Colour v with the k colour
    $W = W \setminus v$

The algorithm is known to have $O(V^3)$ complexity, but one more improvement could be done knowing that *DSatur* is used together and before the maximum clique finding algorithm like the described so far one. "*VColor-BT-u*" algorithm uses intensively memory to select vertices to different depths. An array is usually allocated to be used as a data structure to handle depths. Its size is N x N, where N is a number of vertices and this is a structure

(memory) that can be used during the colouring process to decrease *DSatur* complexity from $O(V^3)$ to $O(V^2)$ in the following way:

The main task is to make the "Find a vertex … with a maximum saturation degree" and the "Find a minimum colour" tasks to have $O(V)$ complexity, i.e. to be just a simple cycle. It is possible if information on neighbour colours is saved and reused later for each vertex. A "neighbour colours for each vertex" can be defined as colours that are already used for vertices that are adjacent to this vertex. Let's denote early described array as S. The modified *DSatur* algorithm is:

$W := V$
$N := |V|$
While $W \neq Ø$ (n steps)
    $v_{max} := \max(v \mid v \in W, \forall t: t <> v, S(v, N) >= S(t, N))$
    $k := \min (i \mid$ there is no $j : j < i, S(v_{max}, j) = 0)$
    Colour v_{max} with the k colour
    $W = W \setminus v$

    For all $v \in W$ and $(v_{max}, v) \in E$
        If $S(v, k) = 0$ then $S(v, N) := S(v, N) + 1$
        $S(v, k) := 1$

So, $S(v, N)$ element contains a number of neighbour colours for a vertex v and $S(v, k)$ equals to 1 if any vertex connected to v is coloured using the k-th colour and equals to 0 otherwise.

Notice that the proposed maximum clique algorithm requires re-ordering of coloured vertices by colour classes also and this task should be done either before or during the process of obtaining colour classes. Such resorting is never more complex than $O(V^2)$.

3. TESTS

In this section final testing results are presented showing efficiency of the described algorithms. Results are presented again as a ratio of algorithms spent times on finding the maximum clique – so the same results can be reproduced on any platforms. Tests are conducted on DIMACS graphs, which are a special package of graphs used in the Second DIMACS Implementation Challenge [Johnson and Trick 1996] to test different algorithms and find out what of them are the best one and on what types of graphs.

The following notation is used in the table 4 below:

PO – time needed to find the maximum clique by Carraghan and Pardalos [Carraghan and Pardalos 1990] algorithm divided by time needed to find the maximum clique by Östergård [Östergård 2002] algorithm.

VColor-BT-u – time needed to find the maximum clique by Carraghan and Pardalos [Carraghan and Pardalos 1990] algorithm divided by time needed to find the maximum clique by the invented algorithm.

DSColor-BT-u – time needed to find the maximum clique by Carraghan and Pardalos [Carraghan and Pardalos 1990] algorithm divided by time needed to find the maximum clique by the invented algorithm.

Greedy colors – number of colors (color classes) produced by the greedy coloring.

DSatur colors – number of colors (color classes) produced by *DSatur* coloring.

Table 4. Benchmark results at DIMACS graphs –colour classes and ratios of time spent on the maximum clique finding / the base algorithm's time divided by a corresponding algorithm's time.

Graph name	Edge density	Vertices	Maximum clique size	*Greedy colors*	*PO*	*VColor-BT-u*	*DSatur colors*	*DSColor-BT-u*
brock200_1	75%	200	21	*54*	2.1	8.4	*51*	4.6
brock200_2	50%	200	12	*36*	2.3	4.0	*31*	3.0
brock200_3	61%	200	15	*44*	1.2	3.2	*39*	2.7
brock200_4	66%	200	17	*49*	2.0	6.0	*44*	3.7
c-fat200-5	43%	200	58	*68*	58.2	49.2	*84*	2.0
c-fat500-1	4%	500	14	*14*	0.7	1.0	*14*	1.0
c-fat500-2	7%	500	26	*26*	1.2	2.2	*26*	2.2
c-fat500-5	19%	500	64	*64*	72.1	85.4	*64*	90.1
hamming6-2	90%	64	32	*32*	493.0	493.0	*32*	493.0
hamming8-4	64%	256	16	*32*	247.8	7848.3	*22*	5674.1
johnson8-4-4	77%	70	14	*19*	11.9	53.3	*17*	60.2
johnson16-2-4	76%	120	8	*14*	4.4	20.9	*17*	11.6
keller4	65%	171	11	*26*	2.8	11.8	*23*	8.8
MANN_a9	93%	45	16	*21*	12.5	42 400.0	*18*	81 354.0
p_hat300-1	24%	300	8	*28*	1.0	1.3	*22*	1.5
p_hat300-2	49%	300	25	*55*	2.0	6.6	*42*	26.2
p_hat500_1	25%	500	9	*43*	0.9	1.5	*33*	1.2
p_hat700_1	25%	700	11	*55*	1.1	1.9	*41*	2.2
sanr400_0.7	70%	400	21	*89*	1.7	5.6	*83*	5.8
2dc.256*	47%	256	7	*13*	4.6	14.5	*9*	213.1

* - 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, 20.9 in the "*VColor-BT-u*" column means that the algorithm proposed in this paper is 20.9 times faster than the base algorithm, which is Carraghan and Pardalos one. Graphs in the previous table as the same as described in the table 3.

The modified algorithm is not better than the core algorithm in many cases although the number of produced colours is always slightly smaller and this allows concluding that there are a lot of graphs where a small decrease of colours doesn't produce any direct advantages. At the same time the modified algorithm produces a huge difference for some graph types – see for example "2dc.256" graph. Therefore there is a certain need for artificial intelligence type rules for selecting one or another colouring strategy or even a maximum clique finding algorithm to be applied [Kumlander 2006].

4. CONCLUSION

In this paper a new adjusted algorithm for the maximum clique finding is presented. It is based on the heuristic vertex-colouring (i.e. uses colour classes) and the backtracking search. In the

first part of the paper a general idea of using colour classes during the maximum clique finding is explained. Thereafter an idea of backtracking on colour classes' level is proposed. The actual algorithm is described in the meta-programming language. Besides a set of comparison tests is conducted on DIMACS graphs using Carraghan and Pardalos algorithm [Carraghan and Pardalos 1990] as it was advised by the Second DIMACS Implementation Challenge [Johnson and Trick 1996] and Östergård algorithm [Östergård 2002], which is reported to be one of the best at the moment. The new algorithm was the quickest on the major graph types – there were only once instance "c-fat200-5" where it was slightly slower than Östergård algorithm. For the "MANN" instance it was around 4 000 time faster than others.

The proposed algorithm uses a heuristic vertex colouring only once before the core algorithm starts and this introduces a possibility to use more advanced techniques for finding the colouring rather than the greedy one. Moreover the algorithm uses intensively memory and those allocated spaces can be used by a colouring algorithm as well. The paper contains a modification of the algorithm by using *DSatur* colouring strategy, which is no more complex than $O(V^2)$ including this colouring sub-algorithm description. The modified algorithm is not better than the core one in many cases as it was expected in the discussion subchapter of the analytical section of this paper, but is worth to apply on some graphs producing a solution up to 15 times faster. Therefore artificial intelligence principles are needed to select a right modification of the core algorithm to be applied.

There are one additional advantage of the proposed algorithm in compare to many others for finding the maximum clique – it is very simple from the implementation point of view. This makes it possible to grant the algorithm the title of "practical" and effectively apply as a subroutine in many real live problems as data mining, pattern recognition, artificial intelligence and many others.

REFERENCES

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

Berman, P., Pelc, A., 1990. Distributed fault diagnosis for multiprocessor systems. *In Proc. of the 20th Annual Intern. Symp. on Fault-Tolerant Computing*, Newcastle, UK, pp 340–346.

Bomze, M., Budinich, M., Pardalos, P.M., Pelillo, M., 1999. The maximum clique problem. *Handbook of Combinatorial Optimization*, Vol. 4, In D.-Z. Du and P. M. Pardalos, eds. Kluwer Academic Publishers, Boston, MA.

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

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

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

Gendreau, A., Salvail, L., Soriano, P., 1993. Solving the maximum clique problem using a tabu search approach. *Ann. Oper. Res.*, Vol. 41, pp 385-403.

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

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

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

Knuth, D.E., 1994. The sandwich theorem, *Electr. J. Comb*., Vol. 1(A1), pp 1-48.

Kumlander, D., 2003. Problems of optimisation: an exact algorithm for finding a maximum clique optimized for the dense graphs. *ISMP2003: Program and Abstracts book*, Copenhagen, pp 114.

Kumlander, D., 2004. A new exact algorithm for the maximum-weight clique problem based on a heuristic vertex-coloring and a backtrack search. *Proceedings of The Forth International Conference on Engineering Computational Technology*, Civil-Comp Press, pp 137.

Kumlander, D., 2005. 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, pp 79-86.

Kumlander, D., 2006. Improving the maximum clique finding applications by using artificial intelligence principles, *WSEAS Transactions on Computers*, Vol. 5, No. 8, pp 1726-1732.

Lagarias, J.C., Shor, P.W., 1992. Keller's Cube–Tiling Conjecture is False in High Dimensions. *Bulletin of the AMS*, Vol. 27, pp 279–283.

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

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

Tomita, E., Seki, T., 2003. An efficient branch-and-bound algorithm for finding a maximum clique. *Discrete Mathematics and Theoretical Computer Science*, Springer, Vol. 2731, pp. 278-289.

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

West, D.B., 2001. Introduction to Graph Theory (2nd edition), *Prentice Hall.*

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