A new implicit branching strategy for exact maximum clique

Pablo San Segundo, Cristóbal Tapia
Intelligent Control Group
CAR (UPM-CSIC), Spain
pablo.sansegundo@upm.es

Abstract

We present a new implicit branching strategy for maximum clique. The new strategy is based in Konj and Janečič's improvement over reference MCR algorithm. It uses a fixed initial non increasing degree vertex ordering at every step of the search, to obtain tighter bounds than MCR on average.

We show that the new branching strategy integrates nicely with a natural bit model for the domain. This allows for efficient bound computing using bit masking operations, so that overall improvement in performance is achieved. We present empirical validation over structured and random graphs.

Keywords: maximum clique, branch and bound, exact search, coloring.

1. Introduction

A complete graph, or clique, is a graph such that all its vertices are pairwise adjacent. For a given graph $G=(V,E)$, the *k-clique* problem determines the existence of a subgraph which forms a clique of size k and is well known to be NP-complete [1]. The corresponding optimization problem is the maximum clique problem (MCP) which consists in finding the largest possible clique hidden in G. MCP is known to be NP-hard so no efficient exact polynomial time algorithms are expected to be found.

Finding a maximum clique has been deeply studied in graph theory and is a very important NP-hard problem with applications in many fields: bioinformatics and computational biology [2][3], computer vision [4, robotics [5] etc. A slightly outdated but nevertheless good survey on maximum clique applications can be found in Chapter 7 of [6].

Many efforts have been made at implementing fast MCP algorithms in practice. Most successful attempt at exact MCP use *branch and bound* as metaheuristic. One of the earliest was [7], but there have followed [8], [9] [10], [11], to name but a few. These algorithms perform a systematic search pruning false solutions by computing upper bounds for the maximal clique achievable at each step. The tradeoff between computational cost and tight bounds is maximized using sequential vertex coloring heuristics to obtain the bounds.

In sequential vertex coloring, vertices in a graph are assigned a symbol (usually referred to as *color* by analogy with the famous map coloring problem) such that pairwise adjacent vertices are all colored differently. It is well known that the number of colors employed to color a graph G is an upper bound of the size of its maximum clique. The tightest possible upper bound is therefore the minimum number of colors needed to *paint* the graph, known as its *chromatic* number $\chi(G)$. Unfortunately determining $\chi(G)$ is also NP-hard [1] so, in practice, heuristics which produce approximate colorings are employed.

This paper is structured as follows: Section 2 and 3 deal with preliminary definitions and related word. Section 4 presents the new approximate coloring heuristic. Section 5 presents a comparison between the new algorithm and current leading reference algorithm. Finally Section 6 summarizes the paper's contribution.

2. Preliminaries and notation

A simple undirected graph $G=(V,E)$ consists of a finite set of vertices V and a finite set of edges $E \subset VxV$ made up of pairs of distinct vertices. Two vertices are said to be adjacent if they are connected by an edge. The complement of G is a graph $\overline{G}$ on the same vertices as G such that two vertices u, v in $\overline{G}$ are adjacent iff $(u,v) \notin E$. $N_G(v)$ (or just $N(v)$ when the graph is clear from the context) denotes the neighbor set of v in G, i.e. the set of all vertices in G which are adjacent to v. A set of pairwise non adjacent vertices is an *independent set*. The set of non adjacent vertices of any vertex $v \in V$ (i.e. those which are not its neighbors) will be referred to as $N_{\overline{G}}(v)$ (its neighbors in the complement graph).

For any set of vertices $U \subseteq V$, $G(U)=(U,E(U))$ refers to the induced subgraph over G by vertices in U (i.e. $E(U)=\{(u,v)/u \in U, v \in U, (u,v) \in E\}$). deg(v) is the degree of vertex v, the number of its neighbors. The degree of a graph, ΔG, is the maximum degree of any of its vertices. The density p of a graph is the probability of having an edge between any two pair of

vertices (for undirected graphs with n vertices and m edges, $p = 2m/n(n-1)$). $\omega(G)$ refers to the number of vertices in a maximum clique in G and $u(G)$ denotes any upper bound over $\omega(G)$ ($\omega(G) \le u(G)$). Unless otherwise specified, it will be assumed that vertices in a graph are ordered; v_i or $V[i]$ refer to the i-th vertex in the set.

3. Related work on MCP

Procedure *RMCP* (U, C, S, S_{max})

Initial values: U=G, $C = \{c_1, \cdots c_n\} / c_i = \min\{i, \Delta G\}$
$S = \phi$, $S_{max} = \phi$

Step 1: Select a vertex v with maximum color in U

Step 2: $U \leftarrow U \setminus \{v\}$

Step 3: **if** ($|S| + C(v) \le |S_{max}|$) return to previous level of recursion

Step 4: $S \leftarrow S + \{v\}$

Step 5: **if** ($U \cap N_U(v) = \phi$) **then**

if($|S| > |S_{max}|$) **then** $|S_{max}| \leftarrow |S|$

goto step 8

endif

Step 6: $C \leftarrow$ COLOR(G($U \cap N_U(v)$))

Step 7: MAX_CLIQUE ($U \cap N_U(v)$,C)

Step 8: $S \leftarrow S - \{v\}$

Step 9: Repeat all steps from 1 to 9 until $U = \phi$

Figure 1. The reference maximum clique algorithm

Our reference basic branch and bound procedure for finding a maximum clique (RMCP) is described in Figure 1. It is the general outline of the Tomita and Seki algorithm [10]. In *RMCP* search takes place in a graph space. It uses two global sets S and S_{max}, where S is the set of vertices of the currently growing clique and S_{max} is the largest maximal clique found so far. The algorithm starts with an empty set S and recursively adds (and removes) vertices from S until it verifies that it is no longer possible to unseat the current champion S_{max}. At any node, S always holds the vertices in the current path (i.e. $|S|$ corresponds with depth).

Candidate set U is initially set to G, the input graph. At each level of recursion a new vertex v is added to S from the set of vertices in U (step 4). At every new node a maximum color vertex $v \in U$ is selected and deleted from U (steps 1,2). The result of vertex coloring $C(v)$ (step 6) is an upper bound to the maximum clique in U in the descendant node.

The search space is pruned in step 3, when the sum of the size of the current clique and the upper bound estimate obtained in the previous level cannot improve the current best clique found at present. If this is not the case v is added to S and a new induced graph $G(U \cap N_U(v))$ is computed (and coloured) to become the new graph in the next level of recursion (step 7).

If a leaf node is reached ($U \cap N_U(v)$ is the empty set) and $|S| > |S_{max}|$ (i.e. the current clique is larger than the best clique found so far) the current champion is unseated and the new maximal is recorded as new best. On backtracking, the algorithm deletes v from S and picks a new vertex from candidate set U until there are no more candidates left to be examined.

3.1. Approximate colorings

The number of subproblems analyzed by RMPC diminishes with the number of hits in the pruning condition evaluated in step 3, i.e. with decreasing number of color assignments used by COLOR.

Finding the chromatic number of a graph is intractable, so in practice, some form of approximate coloring is used. An elaborate approximate coloring procedure can significantly reduce the search space but is also time-consuming; therefore an adequate trade-off is required in the coloring stage.

RMCP does not tell the whole story about the Tomita and Seki algorithm. Their COLOR function employed the well known GREEDY sequential coloring heuristic but seamlessly combined it with a reordering of vertices by decreasing color so that maximum color vertex selection (step 1) is then achieved in constant time. This general strategy is still the standard de facto for MCP branch and bound.

The GREEDY algorithm can be formalized as follows:

Definition: Let V be the set of vertices to be colored and let $K = \{v_1, \cdots, v_n\}$ be any strict ordering of V.

Procedure *GREEDY* (V, K)
For $v \leftarrow v_1$ to v_n
assign vertex v the smallest possible color
EndFor

In practice reordering by decreasing color is done so that vertex selection in step 1 is made in *reverse-order* (maximum color vertices are placed last at the output of *COLOR*). However, *GREEDY* selects vertices in the order inherited by the previous level. It is well known that the number of colors used by *GREEDY* is improved if vertices are placed initially by non increasing degree. In *RMCP*, vertices are dynamically ordered at each node by non increasing color prior to the COLOR call, which is suboptimal in this respect (but good for maximum color selection in step 1).

In [11], Konc and Janečič introduced an improvement over *RMCP*. They realized that it was necessary to reorder by color only those vertices in the input set U with color numbers high enough to be added to the current clique set S in a direct descendant. Any vertex $v \in U$ with color number $C(v)$ below a threshold $K_{min} := |S_{max}| - |S| + 1$ cannot possibly unseat the current champion (S_{max}) in the next level of recursion and is kept in the same relative order it was presented to *RMCP* initially. Their experiments show that this strategy prunes the search space better than the Tomita and Seki coloring algorithm, especially in the case of graphs with high density.

Konc and Janečič did some experiments with dynamical rearrangement of vertices at each step by non increasing degree in order to optimise results obtained by *COLOR*. They showed that, on averaged, this strategy pruned the search space best, but the overhead introduced made it impractical unless it was only used in very shallow levels of the search tree.

4. A new approximate coloring heuristic

Based on [11] we propose a further improvement over [10]: sort vertices dynamically at each step prior to the COLOR call, not by non-increasing degree, but by the relative order in which they were presented to *RMCP* initially.

Prior to the initial call to *RMCP*, vertices are conveniently ordered by non increasing degree so that in the first call to *COLOR*, there is no need for dynamic rearrangement. It is logical to assume that as the depth of nodes increase, the quality of initial relative order decreases compared to local non increasing degree. However, experiments show that it is better, on average, than the original dynamic color ordering as proposed by Tomita and Seki.

The additional (practical) point behind a fixed initial arrangement of nodes is that it is now possible to implement COLOR efficiently using bit masking operations. In particular we use bit strings to encode:

- Each row of the adjacency matrix
- Vertex sets induced at each step
- Color sets obtained during COLOR

When it is necessary to make the bit encoding explicit, we will add subindex *BB* (i.e. U_{BB} refers to vertex set U encoded as bit string).

Procedure *BB-COLOR* (U_{BB}, U_L, C, k_{min})
Initial values: $Q_{BB} := U_{BB}$; $k_{min} := |S_{max}| - |S| + 1$;
$k = 0$;
/* Vertices in the input candidate set U_{BB} must be in the same relative order as in the initial input graph presented to *RMCP* */

Step 1: if ($U_{BB} = \phi$) return

Step 2: $C_k \leftarrow \phi$

Step 3: if ($Q_{BB} = \phi$) goto *step 9*

Step 4: select the next vertex v in Q_{BB}

Step 5: $C_k \leftarrow C_k \cup \{v\}$

Step 6: $Q_{BB} \leftarrow Q_{BB} \setminus \{v\}$;

Step 7: $Q_{BB} \leftarrow Q_{BB} \cap N_{\overline{Q_{BB}}}(v)$

Step 8: goto *step 3*

Step 9: $U_{BB} \leftarrow U_{BB} - C_k$

Step 10: $Q_{BB} \leftarrow U_{BB}$

Step 11: if ($k \geq k_{min}$) then
store C_k in U_L in the same order
$C[v \in k] \leftarrow k$
endif

Step 12: $k \leftarrow k + 1$

Step 13: Repeat all steps from 1 to 13

Figure 2. The new approximate color procedure BB-COLOR. Highlighted rectangles in red mark computations which benefit from bit-parallelism

The new approximate color heuristic, *BB-COLOR*, is described in Figure 2. *BB-COLOR* obtains tighter bounds because it reorders the vertices in the input set U as they were presented initially to the *RMCP* procedure (conveniently sorted by non increasing degree). In practice, the input candidate set is bit encoded (U_{BB}) and therefore automatically rearranged

on generation at each level of recursion by *RMCP*, prior to the call to *BB-COLOR*.

On output, vertices need to be ordered by color so a new (conventional) data structure U_L is used to decouple the input and output sets. U_L is not bit encoded to avoid the overhead of bit scanning when selecting vertices from this set in *RMCP*. The decoupling has an additional benefit in that only vertices which may possibly be selected from U_{BB} by *RMCP* in the next level of recursion need to be explicitly stored. To select these vertices we use parameter k_{min} as in [11].

Let $C=\{C_1,C_2,\cdots,C_k\}$ be the k-coloring output of *BB-COLOR* (in our implementation color sets are numbered from 0 to n-1, but we will employ standard notation for clearness). At the start of the procedure the first color class C_1 (initially the empty set) is selected and the first vertex v_1 of the input candidate set U_{BB} is added to C_1 (step 4). As a result, the set U'_{BB} of remaining candidate vertices $v \in U_{BB}$ which can still be assigned color C_1 are non adjacent vertices to v_1 (i.e. $U'_{BB}=U_{BB}\cap N_{\overline{U_{BB}}}(v_1)$).

U'_{BB} is computed in step 7 (Q_{BB} is simply needed for auxiliary storage); the first vertex $v'_1 \in U'_{BB}$ is then added to C_1 and a new set $U''_{BB}=U'_{BB}\cap N_{\overline{U'_{BB}}}(v'_1)$ is computed. The process continues until the resulting induced graph is empty, in which case the assigned vertices are removed from candidate set U_{BB} and the process is repeated for the next empty color class C_2. *BB-COLOR* ends when all vertices in U_{BB} have been assigned a particular color.

Marked in red in Figure 2 are the operations that benefit form bit parallelism. In particular step 7 is critical and explains the improvement in overall performance obtained empirically over the reference algorithm. The proposed bit encoding allows $Q_{BB}\cap N_{\overline{Q_{BB}}}(v)$ to be computed by the following two bit masks:

$$Q_{BB}\cap N_{\overline{Q_{BB}}}(v) \equiv Q_{BB}\ AND\ NOT\ BB(A_i)$$

where $BB(A_i)$ refers to the i-th row of the adjacency matrix of the inital input graph. A similar analysis shows that step 9 can be computed with a combination of bit masks.

To sum up, *BB-COLOR* differs from [10][11] in the following points:

- It receives as input a (bit) set of vertices in the fixed initial order they were presented to *RMCP*.
- Outputs a (conventional) list of vertices sorted by color (as long as the color is equal or greater than parameter k_{min}). The rest of colours are not stored in U_L (step 11).
- Uses a *sequential by color* approximate coloring heuristic (w.r.t. a typical vetex coloring).

All three issues allow *BB-COLOR* to be computed by efficient bitwise operations.

5. Experiments

We have implemented in C language a number of algorithms for the experiments: *RMCP* (the reference MCP algorithm proposed in [10] (which includes the original approximate coloring procedure), *RKJ* (the approximate coloring variant in [11], which introduces selective rearrangement of vertices only in color classes with color greater than a certain threshold), and our new bit-parallel *BB-MCP* which uses *BB-COLOR* for approximate coloring.

Computational experiments were performed on a 2.4 GHz Intel Quad processor with a 64 bit Windows operating system (so as to exploit bit masking operations by a factor of 64), against a set of structured graphs from the well known DIMACS[1] benchmark, and a number of random graphs.

The time limit for all the experiments was set at 5h (18000 seconds) and instances not solved in the time slot were classified as *Fail*. Our user times for the DIMACS machine benchmark graphs r100.5-r500.5 are 0.000, 0.031, 0.234, 1.531 and 5.766 seconds respectively.

We note that while *BB-MCP* uses bit parallel enhancements, we have not been able to find similar efficient bit encodings for the *RMCP* and *RKJ* COLOR procedure, which requires a different sorting of vertices in each call. It is the fixed initial order of vertices that allows a natural integration of a typical graph bit model so that bit operators over sets of vertices do not lose the reference inside the bit strings.

In any case an important effort has been made in optimizing all three algorithms, so this might explain discrepancies between user times for *RMCP* and *RKJ* found elsewhere. The latter are, on average, at least no worse (and in many cases better).

Tables 1 and 2 record user times and number of steps taken by *RMCP* and our algorithm for a subset of DIMACS graphs. As expected, the number of steps is, on average, reduced by our *BB-COLOR* procedure. In some of the structured graphs the reduction in the number of subproblems is very significant (e.g. *brock*s or *phat*) whereas in other families it has no effect. In

[1] *<URL:ftp://dimacs.rutgers.edu/pub/ challenge /graph/ benchmarks/clique>*

the majority of cases *BB-MCP* performs better overall and this trend appears to be more acute with the more difficult, denser graphs.

Table 3 records user times for all 3 algorithms against random graphs. As can be seen, *BB-MCP* outperforms the other algorithms and, as in the case of structured graphs, the difference tends to be more acute as density rises while keeping the size fixed.

Table 1. Number of steps taken for tests on DIMACS benchmark graphs. RMCP is the reference algorithm. BB-MCP is the new bit parallel algorithm.

Graph (size, density)	ω	RMCP	BB-MCP
brock200_1 (200, 0.745)	21	485009	295754
brock200_2 (200, 0.496)	12	4440	4004
brock200_3 (200, 0.605)	15	15703	13534
brock200_4 (200, 0.658)	17	87789	64676
brock400_1 (400, 0.748)	27	302907107	168767856
brock400_2 (400, 0.749)	29	114949717	66381296
brock400_3 (400, 0.748)	31	224563898	118561387
phat300-1(300, 0.244)	8	2043	1982
phat300-2 (300, 0.489)	25	10734	6226
phat300-3 (300, 0.744)	36	2720585	590226
c-fat200-1 (200, 0.077)	12	216	216
c-fat200-2 (200, 0.163)	24	241	241
c-fat200-5 (200, 0.426)	58	308	308
c-fat500-1 (500, 0.036)	14	520	520
c-fat500-2 (500, 0.073)	26	544	544
hamming6-2 (64, 0.905)	32	63	63
hamming6-4 (64, 0.349)	4	164	164
hamming8-2 (256, 0.969)	128	255	255
johnson8-2-4 (28, 0.556)	4	51	51
johnson8-4-4 (70, 0.768)	14	295	289

Large sparse graphs are the worse scenario for BB-MCP, since our bit-parallel kernel does not use sparse encodings and many of the bit masking operations is probably a waste of time. Still BB-MCP fares no worse for the instances presented here. It is however possible that, in the current implementation, it may perform poorly with very sparse graphs with many thousands of nodes. This would need empirical validation.

6. Conclusions

In this paper we describe a new implicit branching strategy for exact maximum clique search, which allows for efficient bit masking operations to compute a standard approximate coloring procedure, and also obtains, on average, tighter bounds.

The algorithm employs well known initial non increasing order of vertices. After sorting vertex by color so as to choose each time a maximum colored vertex from the candidate set, it rearranges the remaining vertices following the fixed initial order.

Table 2. CPU user times (sec.) for DIMACS benchmark graphs (with a limit of 5h).

Graph (size, density)	ω	RMCP	BB-MCP
brock200_1 (200, 0.745)	21	1.563	0.500
brock200_2 (200, 0.496)	12	0.031	<0.001
brock200_3 (200, 0.605)	15	0.062	0.015
brock200_4 (200, 0.658)	17	0.219	0.078
brock400_1 (400, 0.75)	27	1486.125	502.796
brock400_2 (400, 0.75)	29	647.875	211.218
brock400_3 (400, 0.75)	31	1017.680	336.188
phat300-1(300, 0.244)	8	<0.001	<0.001
phat300-2 (300, 0.489)	25	0.063	0.038
phat300-3 (300, 0.744)	36	15.907	5.495
c-fat200-1 (200, 0.077)	12	<0.001	<0.001
c-fat200-2 (200, 0.163)	24	<0.001	<0.001
c-fat200-5 (200, 0.426)	58	0.016	<0.001
c-fat500-1 (500, 0.036)	14	<0.001	<0.001
c-fat500-2 (500, 0.073)	26	0.016	<0.001
hamming6-2 (64, 0.905)	32	<0.001	<0.001
hamming6-4 (64, 0.349)	4	<0.001	<0.001
hamming8-2 (256, 0.969)	128	0.016	0.031
johnson8-2-4 (28, 0.556)	4	<0.001	<0.001
johnson8-4-4 (70, 0.768)	14	0.016	<0.001

With the help of adequate bit data structures, this reordering can be done in constant time. Moreover the coloring procedure also benefits by the bit encoding so that overall performance improves, as shown by the experiments.

BB-MCP seems to outperform current reference algorithms in the dense *difficult* graphs but we expect it

to perform worse in very large sparse scenarios, because the bit kernel in the current implementation does not account for sparseness (we are currently working on this issue). In addition, further tests are required to validate the current version of the algorithm against large sparse random graphs.

7. Acknowledgments

This work is funded by the Spanish Ministry of Science and Technology (ARABOT: DPI 2010-21247-C02-01) and supervised by CACSA whose kindness we gratefully acknowledge.

Table 3. CPU user times (sec.) for a collection of random graphs (with a limit of 5h).

n	p	RMCP	RKJ	BB-MCP
100	0.60	0.009	0.006	0.006
100	0.70	0.009	0.009	0.003
100	0.80	0.028	0.022	0.010
100	0.90	0.056	0.053	0.016
100	0.95	0.035	0.034	0.016
150	0.50	0.006	0.003	0.006
150	0.60	0.022	0.022	0.006
150	0.70	0.078	0.075	0.028
150	0.80	0.581	0.497	0.178
150	0.90	7.381	4.200	0.984
150	0.95	2.928	1.175	0.212
200	0.40	<0.001	0.006	0.003
200	0.50	0.019	0.025	0.009
200	0.60	0.094	0.094	0.047
200	0.70	0.662	0.625	0.250
200	0.80	12.600	10.994	3.594
300	0.40	0.028	0.031	0.016
300	0.50	0.134	0.128	0.066
300	0.60	1.150	1.088	0.491
300	0.70	21.156	18.756	7.647
300	0.80	1020.331	821.769	254.706
500	0.30	0.047	0.050	0.031
500	0.40	0.319	0.312	0.206
500	0.50	3.153	3.087	1.906

8. References

[1] R.M. Karp. *Reducibility among Combinatorial Problems*. Editors: R.E. Miller, J. W. Thatcher, New York, Plenum, pp.85-103, 1972.

[2] Bahadur, D.K.C., Akutsu, T., Tomita, E., Seki, T., Fujijama, A.: Point matching under non-uniform distortions and protein side chain packing based on efficient maximum clique algorithms. Genome Inform. 13: 143-152, 2006.

[3] Butenko, S., Wilhelm, W.E.; Clique-detection models in computational biochemistry and genomics, European Journal of Operational Research 173: 1-17, 2006.

[4] Hotta, K., Tomita, E., Takahashi, H.: A view invariant hu man FACE detection method based on maximum cliques. Trans. IPSJ, 44, SIG14 (TOM9): 57-70, 2003.

[5] San Segundo, P., Rodríguez-Losada, D., Matía, F., Galán, R.; Fast exact feature based data correspondence search with an efficient bit-parallel MCP solver. Applied Intelligence, 2008.

[6] Bomze, I.M., Budinich, M., Pardalos, P.M., Pelillo, M.; HandBook of Combinatorial Optimization, Supplement A. Kluwer Academic Publishers, Dordrecht (1999) 1-74.

[7] Wood, D.R.; An algorithm for finding a maximum clique in a graph. Operations Research Letters 21 (1977) 211-217.

[8] Carraghan, R, Pardalos, P.M.; An exact algorithm for the maximum clique problem Operations Research Letters 9 (1990) 375-382.

[9] ÖOstergård, P.R.J.; A fast algorithm for the maximum clique problem. Discrete Applied Mathematics, 120 (1): 97-207, 2002.

[10] Tomita E., Seki, T.; An efficient branch and bound algorithm for finding a maximum clique. Proc. Discrete Mathematics and Theoretical Computer Science. LNCS 2731, (2003) 278-289.

[11] Konc, J., Janečič, D.; An improved branch and bound algorithm for the maximum clique problem. MATCH Commun. Math. Comput. Chem. 58 (2007) 569-590.