Initial Sorting of Vertices in the Maximum Clique Problem Reviewed

Pablo San Segundo[1(✉)], Alvaro Lopez[1], and Mikhail Batsyn[2]

[1] Centre of Automatics and Robotics (UPM-CSIC),
Jose Gutiérrez Abascal, 2, 28006 Madrid, Spain
pablo.sansegundo@upm.es
[2] Laboratory of Algorithms and Technologies for Networks Analysis,
National Research University Higher School of Economics,
136 Rodionova, Nizhny Novgorod, Russian Federation
mbatsyn@hse.ru

Abstract. In recent years there have been a number of important improvements in exact color-based maximum clique solvers, which have considerably enhanced their performance. Initial vertex ordering is one strategy known to have a significant impact on the size of the search tree. Typically, a degenerate sorting by minimum degree is used; literature also reports different tiebreaking strategies. A systematic study of the impact of initial sorting in the light of new cutting-edge ideas (e.g. recoloring [8], selective coloring [13], ILS initial lower bound computation [15, 16] or MaxSAT-based pruning [14]) is, however, lacking. This paper presents a new initial sorting procedure and relates performance to the new mentioned variants implemented in leading solver BBMC [9, 10].

Keywords: Search · Branch and bound · Algorithm · Experimental

1 Introduction

A clique is a complete subgraph whose vertices are all pairwise adjacent. For a given graph, the maximum clique problem (MCP) is an NP-hard problem which consists in finding a clique with the maximum number of vertices. MCP has found many applications in a wide scope of fields such as matching related problems which appear in computational biology [1], robotics [2, 3] or computer vision [4]. A good survey on applications may be found in [5].

Many improvements have appeared in exact MCP search since the Bron and Kerbosch algorithm [6] and the primitive branch-and-bound (BnB) algorithm of Carraghan and Pardalos [7]. Specifically in the last decade, there has been an outburst of ideas related to greedy coloring bounds from which MCS [8] and bit optimized BBMC [9, 10] standout. In the comparison survey [11], BBMC was reported the fastest.

Both MCS and BBMC implement clique enumeration recursively, branching on a candidate vertex at each step to enlarge a growing clique. The leaf nodes of the recursion tree construct maximal cliques, and the largest clique found so far during search is always stored in memory. An important theoretical result by Balas and Yu is that the number of colors in any vertex coloring of a graph is an upper bound on its

P.M. Pardalos et al. (Eds.): LION 2014, LNCS 8426, pp. 111–120, 2014.
DOI: 10.1007/978-3-319-09584-4_12

clique number [12]. Based on this property, many recent BnB exact solvers implement bounding using greedy coloring sequential heuristic SEQ at each step. Pruning occurs at nodes when the size of the current growing clique added to the color upper bound is not greater than the size of the best maximal clique stored at that moment.

MCS further introduced *recoloring*, a repair mechanism which attempts to reassign lower color numbers to a subset of vertices outputted by SEQ at the cost of linear complexity. In [10], it was reported to improve performance significantly but only in more difficult dense graphs.

Selective coloring is another new very recent idea, which has been implemented in the BBMC solver. Instead of computing a full vertex coloring, selective coloring relaxes SEQ to the minimum partial coloring such that every vertex will be pruned in the derived child node [13].

In [14], Li and Quan describe a stronger repair mechanism than recoloring. At every node, the MCP on the SEQ colored graph is reduced to an equivalent MaxSAT problem. It turns out that the basic inference mechanisms employed by current MaxSAT solvers, can also be used to produce tighter bounds than SEQ. Moreover, they can even be tighter then the chromatic number of the graph. We will refer to this idea as *logical pruning*.

Also recently, a new local search procedure for the maximum independent set problem was described in [15]. In combination with iterated local search (ILS) meta-heuristic it shows excellent results in a number of typical benchmarks. In [16], the authors propose to precompute ILS for the complement graph and use the output as initial solution to an exact clique procedure. This line of research (ILS0) is very much open at present, and results are very encouraging.

Besides these four cutting-edge ideas, two decision heuristics standout as critical for overall performance in the general scheme: (I) vertex selection by decreasing color number (first described in [17]) and (II) fixing the order of vertices, at the beginning of the search, as input to sequential coloring [18]. Note that this implies that SEQ will assign color numbers to vertices in the same relative order at every step of the search.

Independent of previous ideas, initial vertex sorting has long been known to have significant impact on overall performance. The general strategy is to pick vertices at the root node by increasing degree, in order to reduce the average branching factor in the shallower levels of the search tree. A number of variants have been described in literature [7, 11, 16], but a precise comparison survey is somewhat lacking.

This paper presents a new initial sorting for exact maximum clique search and reports improved performance w.r.t. a typical sorting procedure over a set of structured graphs taken from well known public benchmarks. Moreover, the paper also addresses the impact of initial sorting in recoloring, selective coloring, logical pruning and ILS0.

The paper is structured in 5 sections. Section 2 includes useful definitions and notation. Section 3 describes the new initial ordering; Sect. 4 presents empirical validation, and Sect. 5 conclusions and future lines of research.

2 Preliminaries

A simple undirected graph $G = (V, E)$ consists of a finite set of vertices $V = \{v_1, v_2, \ldots, v_n\}$ and edges $E \subseteq VxV$ which pair distinct vertices. Two vertices are said to

be adjacent (alias neighbors) if they are connected by an edge. $N(v)$ refers to the neighbor set of v. Standard notation used in the paper includes deg(v) for vertex degree, Δ for graph degree, $\omega(v)$ for the clique number and G_v for the induced graph of v in G.

Useful definitions and notation related to vertex orderings are:

- O(V): Any strict ordering of vertices in a simple graph
- *Width of a vertex* for a given O: the number of outgoing edges from that vertex to previous vertices in O.
- *Width of an ordering*: The maximum width of any of its vertices
- *Degeneracy ordering*: A sorting procedure which achieves a vertex ordering of minimum width. It does so by iteratively selecting and *removing* vertices with minimum degree [19]. In general, let $O(V) = v_1, v_2, \ldots, v_n$ be a degeneracy ordering of the vertices. Then v_n is a vertex of minimum degree in G, v_{n-1} will have minimum degree in $G - \{v_n\}$, v_{n-2} in $G - \{v_n, v_{n-1}\}$ and so on. How to break ties is not determined.
- *Vertex support* $\sigma(v)$: the sum of the degrees of vertices in $N(v)$ (notation is taken from [16]).

Literature reports degeneracy ordering as successful for exact MCP already in [7]. In MCS and others, ties are broken using minimum support criteria. For vertices having the same σ, ties are usually broken *first-found* or randomly. We denote by MWS (Min Width and min Support tiebreak) the latter sorting procedure, which will be considered as reference. MW stands for MWS without support tiebreak. Pseudocode for MWS is available in Algorithm 3 of [16]. Worth noting is that degeneracy ordering is defined in a last-to-first basis. This is consistent with both BBMC and MCS implementations, which pick vertices in *reverse order* at the root node (i.e. vertices with smallest degree first).

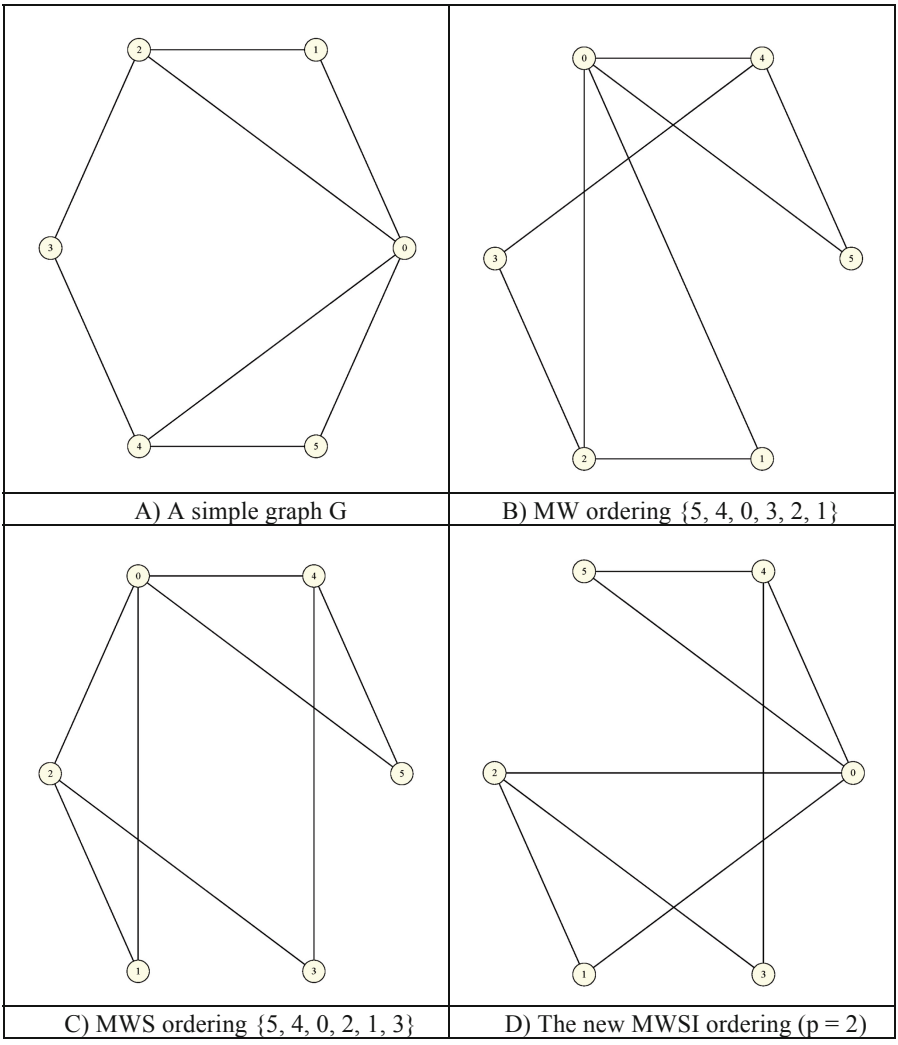
3 New Initial Sorting

SEQ is reported to produce tighter colorings if vertices with higher degrees are selected first. BBMC and MCS both keep an initial MWS coloring (actually BBMC uses simple MW) fixed throughout the search. Vertices are taken in reverse order at the root node, and in direct order by SEQ at every step. Moreover, while MWS achieves minimum vertex width looking from back to front, it does not preserve maximum degree at the other end. The distortion grows with size.

In the light of the above considerations, we propose a new initial sorting MWSI which can be seen as a repair mechanism to MWS w.r.t. to maximum degree at the head of the ordering. MWSI takes as input the ordering produced by MWS and sorts, according to non decreasing degree, a subset of k vertices $v_1, v_2, \ldots, v_k$ (ties are broken *first-found*). This second ordering is absolute (not degenerate) since it is directed to improve SEQ. The remaining n - k vertices are not modified and remain sorted by minimum width.

Parameter k (the number of vertices reordered by non increasing degree) should be neither too small (a low impact), nor too big (the first minimum width ordering would be lost). Instead of using k for tuning, we consider a new parameter p related to the total number of vertices, such that:

Table 1. An example of the new MWSI reordering



A) A simple graph G	B) MW ordering {5, 4, 0, 3, 2, 1}
C) MWS ordering {5, 4, 0, 2, 1, 3}	D) The new MWSI ordering (p = 2)

$$p = \left\lfloor \frac{|V|}{k} \right\rfloor, p = \{2, 3, \ldots\}$$

In practice, MWSI performs best when p ranges between 2 (50 % of the vertices) and 4 (25 % of the vertices). We present an example of MWSI ordering in Table 1. Table 1A is the simple graph G to be ordered. The number of every vertex uniquely identifies it in all the figures, but in the case of Table 1A it also indicates the actual ordering. In the remaining cases, the ordering is the same spatially (i.e. the starting point is the middle vertex to the right and the rest follow in anticlockwise direction). Vertices are always picked from G from first to last and ties broken on a first-found basis when necessary.

Table 1B presents minimum width ordering MW and Table 1C presents reference ordering MWS. The difference between them lies in the difference in support of vertices {1} and {3} which have lowest degree (2). In the case of MW, ties are broken *first-found*, so vertex {1} is placed last in the ordering. In the case of MSW, $\sigma(1) = 7$ whereas $\sigma(3) = 6$ and $\sigma(5) = 6$ so vertex {3} is the one placed at the end. After removing {3}, two triangles appear: {0, 1, 2} and {0, 4, 5}; vertices {1, 2, 4, 5} all have minimum degree and support so vertex {1} is picked in second place and so on.

Table 1D presents the new ordering with p = 2 (i.e. the first 3 vertices {5, 4, 0} are considered for reordering by non increasing degree). Clearly vertex {0} has the highest degree (deg(0) = 4), then comes {4} (deg(4) = 3) and finally {5} (deg(5) = 2). As a result vertices {5} and {0} are swapped.

4 A Comparison Survey

In this section, MWSI is validated against a subset of instances from the well known DIMACS benchmark. Moreover, the report also exposes the impact of initial sorting in the new variants considered, which is another contribution.

We have used leading BBMC (in particular optimized BBMCI) as starting point for all new variants. The algorithms considered are:

- BBMCI: The reference leading algorithm [10].
- BBMCL: BBMCI with selective coloring [13].
- BBMCR: BBMCI with recoloring, similar to MCS. It is described in [10].
- BBMCXR: BBMCI with logical pruning and recoloring. The 'X' in the name refers to MaxSAT and the 'R' to recoloring. BBMCXR is a new algorithm which adapts MaxSAT pruning to BBMCI without having to explicitly encode the graphs to MaxSAT as described in the original paper by Li & Quan. It has been specifically implemented as an improvement on BBMC and a full description has now been submitted for publication [20].

The orderings reported are (reference) MWS and new MWSI (with parameter p set to 4). Reordering the first 25 % of the vertices at the head by non increasing degree produced best results for the instances considered.

All algorithms have been implemented in C++ (VC 2010 compiler) and optimized using a native code profiler. The machine employed for the experiments was an Intel i7-2660@3.40 GHz with a 64-bit Win7 O.S. and 8 GB of RAM. In all experiments the time limit was set to 900 s and only user time for searching the graph is recorded. The instances used for the tests are taken from DIMACS[1] as well as BHOSHLIB[2]. Most graphs which are either too easy or too hard (for the chosen time limit) have not been reported. The subset under consideration for the tests, grouped by families, is as follows:

- C: 125.9
- Mann: a9 and a27

[1] http://cs.hbg.psu.edu/txn131/clique.html

[2] http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm

- brock: 200_1, 400_1, 400_2, 400_3 and 400_4
- dsjc: 500.5
- frb: 30-15-1, 30-15-2, 30-15-3, 30-15-5
- gen: 200_p0.9_44 and 200_p0.9_55
- phat: 300-3, 500-2, 500-3, 700-2, 1000-1, 1500-1
- san: 200_0.9-1, 200_0.9_2, 200_0.9_3, 400_0.7_1, 400_0.7_2, 400_0.7_3
- sanr: 200-0.7, 200_0.9, 400-0.5, 400-0.7

In the case of the *frb* family, *frb30-15-5* was in some cases not solved within the chosen time limit. It has, therefore, not been included in the tables but is explicitly mentioned in the related sections. Also *frb30-15-4* failed in all cases.

Two setups are considered for the tests:

1. ILS0: A (strong) initial solution is precomputed using ILS heuristic, as in [16] and used as starting solution in all algorithms
2. An initial solution is precomputed greedily and used as starting solution in all algorithms. It is constructed by selecting vertices in ascending order (starting from the first) until a maximal clique is obtained. This allows for a better comparison between algorithms avoiding noise from divergent initial branches of the search. The time taken for this initial solution is never greater than 1 ms.

4.1 Experiments Without ILS0

Table 2 reports the number of steps (scaled in millions) taken by the different algorithms considering reference MWS and new MWSI orderings. Each step is a call to a recursive algorithm. Each row reports the total number of steps for each family. The best result for each algorithm is shown in cursive (ties broken first-found). Note that steps between algorithms are not comparable, because the pruning effort could have an

Table 2. Cumulative steps ($\times 10^{-6}$) for different algorithms and orderings. In italics – best value for each algorithm.

	BBMCI [10]		BBMCL [13]		BBMCR [10]		BBMCXR [20]	
	MWS	MWSI	MWS	MWSI	MWS	MWSI	MWS	MWSI
C	0.01	0.01	0.01	0.01	<0.01	<0.01	<0.01	<0.01
Mann	0.02	0.02	0.02	0.02	<0.01	<0.01	<0.01	<0.01
brock	123.09	*122.33*	137.21	*134.11*	66.54	*66.47*	40.35	*39.86*
dsjc	0.26	0.26	0.28	0.28	0.17	0.17	0.11	0.11
frb	453.10	*379.05*	561.92	*466.55*	232.69	*192.93*	144.01	*122.10*
gen	0.22	*0.19*	*0.26*	0.27	0.10	*0.06*	0.05	*0.04*
phat	*5.61*	6.00	*6.14*	6.57	*2.85*	3.16	*1.70*	1.89
san	0.09	*0.08*	*0.28*	0.33	0.05	*0.04*	0.04	0.04
sanr	19.29	*18.40*	21.71	*20.33*	10.21	*9.92*	6.22	*5.94*
Total	601.68	*526.35*	727.83	*628.48*	312.63	*272.76*	192.48	*169.99*

Table 3. Cumulative time (seconds) for different algorithms and orderings. In italics – best value for each algorithm. In bold – best value for the row.

	BBMCI [10]		BBMCL [13]		BBMCR [10]		BBMCXR [20]	
	MWS	MWSI	MWS	MWSI	MWS	MWSI	MWS	MWSI
C	0.031	0.031	*0.016*	0.032	0.031	0.031	***0.015***	0.032
Mann	*0.156*	0.171	0.156	0.156	0.110	***0.109***	0.172	*0.156*
brock	411	*405*	411	*407*	*458*	462	346	***335***
dsjc	*0.670*	0.702	0.686	***0.655***	0.951	*0.827*	0.718	*0.671*
frb	3815	*3245*	4016	*3479*	3476	*3054*	2766	***2436***
gen	0.624	*0.516*	*0.687*	0.702	0.670	*0.422*	0.468	***0.390***
phat	*41.0*	43.9	*40.5*	43.2	*45.4*	49.7	***33.6***	36.3
san	0.655	*0.594*	*1.15*	1.23	***0.592***	0.593	0.703	*0.656*
sanr	58.5	*50.3*	59.5	*50.3*	107	*90.5*	45.7	***42.8***
Total	4328	*3746*	4530	*3982*	4090	*3658*	3193	***2851***

even bigger overhead. Table 3 reports total time in seconds taken by each of the families considered. In contrast to steps, time *is* comparable between algorithms. In bold face the best total time and best time for each family.

MWSI is the fastest in 6 out of the 9 families. Moreover it improves performance significantly in all algorithms considered, as shown by the row of totals. The overall fastest algorithm, BBMCXR, is improved by more than 10 %. Regarding steps, *frb* family is where the impact of MWSI is more significant. Performance is similar in *C*, *Mann* and *dsjc*, and only *p_hat* family becomes more difficult with MWSI.

Worth noting is that *frb30-15-5* failed in BBMCI and BBMCL with the reference sorting but was solved with MWSI (BBMCI took 633.4 s and BBMCL 713.2 s). The other two algorithms solved the problem under the 900 s limit with both orderings. As mentioned at the beginning of the section, this instance is not computed in the tables.

4.2 Experiments with ILS0

This section covers experiments in which the algorithms benefit from a good initial solution computed by ILS heuristic. Tables 4 and 5 report results for the same instances and in the same format as in the previous section.

Regarding MWSI, the trends w.r.t. MWS are similar to those described when ILS was not employed. It improves performance of all algorithms on average and only *phat* shows a bad behaviour towards new MWSI. This validates MWSI also for ILS0. Best overall performance is achieved by BBMXCR, as in the previous section. Worth noting is that *frb30-15-5* is now solved under the time limit in all cases.

Another interesting comparison is how ILS influences the impact of MWSI. Table 6 reports the percentage of improvement in performance (time) for all algorithms with and without ILS. With the exception of reference BBMCI, the rest of the algorithms benefit more of MWSI when fed with a good initial solution. In particular, selective coloring improves the most (over 15 %).

Table 4. Cumulative steps ($\times 10^{-6}$) for different algorithms, orderings and ILS0. In italics – best value for each algorithm.

	BBMCI [10]		BBMCL [13]		BBMCR [10]		BBMCXR [20]	
	MWS	MWSI	MWS	MWSI	MWS	MWSI	MWS	MWSI
C	0.01	0.01	0.01	0.01	<0.01	<0.01	<0.01	<0.01
Mann	0.02	0.02	0.02	0.02	<0.01	<0.01	<0.01	<0.01
brock	59.56	*54.32*	65.27	*59.62*	31.80	*28.94*	18.40	*16.70*
dsjc	0.24	0.24	0.26	0.26	0.15	0.15	0.10	0.10
frb	253.05	*214.12*	316.01	*264.26*	127.60	*108.02*	77.77	*66.87*
gen	0.03	0.03	0.04	*0.03*	0.01	0.01	<0.01	<0.01
phat	*2.78*	2.96	*3.02*	3.21	*1.39*	1.53	*0.81*	0.88
san	<0.01	<0.01	<0.01	<0.01	<0.01	<0.01	<0.01	<0.01
sanr	17.50	*16.88*	19.45	*18.60*	9.50	*9.29*	5.77	*5.64*
Total	333.18	*288.57*	404.07	*346.01*	170.46	*147.96*	102.85	*90.20*

Table 5. Cumulative time (seconds) for different algorithms, orderings and ILS0. In italics – best value for each algorithm. In bold – best value for the row.

	BBMCI [10]		BBMCL [13]		BBMCR [10]		BBMCXR [20]	
	MWS	MWSI	MWS	MWSI	MWS	MWSI	MWS	MWSI
C	***<0.001***	0.016	0.016	*0.015*	0.016	*0.015*	***<0.001***	0.015
Mann	*0.140*	0.156	0.140	0.140	**0.109**	**0.109**	0.156	*0.140*
brock	231	*216*	233	*225*	266	*249*	185	***177***
dsjc	0.640	***0.624***	0.640	*0.634*	0.764	*0.757*	0.650	0.650
frb	2809	*2418*	3138	*2603*	2484	*2174*	1794	***1553***
gen	0.109	*0.094*	0.124	*0.109*	0.063	*0.047*	0.047	***0.032***
phat	*22.4*	24.2	*22.2*	23.6	*24.2*	26.3	***17.6***	18.9
san	0.015	***<0.001***	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**<0.001**	**<0.001**
sanr	48.5	*41.8*	48.5	*46.5*	*53.8*	54.0	42.3	***40.6***
Total	3112	*2701*	3443	*2900*	2828	*2504*	2040	***1791***

Table 6. Improvement in performance (%) caused by the new ordering MWSI

	BBMCI [10]	BBMCL [13]	BBMCR [10]	BBMCXR [20]
ILS0	13.2 %	15.8 %	11.5 %	12.2 %
No ILS0	13.4 %	12.1 %	10.5 %	10.7 %

5 Conclusions and Future Work

A new initial sorting procedure has been described and empirically shown to improve performance of a leading exact maximum clique solver. The considered cutting-edge variants have also been improved. Moreover, the paper also compares these variants

when a good initial solution (computed by recent ILS heuristic) is known a priori. The latter is an open line of research, together with the majority of the algorithmic variants considered.

Acknowledgments. This work is funded by the Spanish Ministry of Economy and Competitiveness (ARABOT: DPI 2010-21247-C02-01) and supervised by CACSA whose kindness we gratefully acknowledge. Mikhail Batsyn is supported by LATNA Laboratory, National Research University Higher School of Economics (NRU HSE), Russian Federation government grant, ag. 11.G34.31.0057.

References

1. Butenko, S., Wilhelm, W.E.: Clique-detection models in computational biochemistry and genomics. Eur. J. Oper. Res. **173**, 1–17 (2006)
2. Hotta, K., Tomita, E., Takahashi, H.: A view invariant human FACE detection method based on maximum cliques. Trans. IPSJ **44**(SIG14(TOM9)), 57–70 (2003)
3. San Segundo, P., Rodriguez-Losada, D., Matia, F., Galan, R.: Fast exact feature based data correspondence search with an efficient bit-parallel MCP solver. Appl. Intell. **32**(3), 311–329 (2010)
4. San Segundo, P., Rodriguez-Losada, D.: Robust global feature based data association with a sparse bit optimized maximum clique algorithm. IEEE Trans. Rob. **29**(5), 1332–1339 (2013)
5. Du, D., Pardalos, P.M.: Handbook of Combinatorial Optimization, Supplement, vol. A. Springer, New York (1999)
6. Bron, C., Kerbosch, J.: Algorithm 457: finding all cliques of an undirected graph. Commun. ACM **16**(9), 575–577 (1973)
7. Carraghan, R., Pardalos, P.: An exact algorithm for the maximum clique problem. Oper. Res. Lett. **9**(6), 375–382 (1990)
8. Tomita, E., Sutani, Y., Higashi, T., Takahashi, S., Wakatsuki, M.: A simple and faster branch-and-bound algorithm for finding a maximum clique. In: Rahman, M., Fujita, S. (eds.) WALCOM 2010. LNCS, vol. 5942, pp. 191–203. Springer, Heidelberg (2010)
9. San Segundo, P., Rodriguez-Losada, D., Jimenez, A.: An exact bit-parallel algorithm for the maximum clique problem. Comput. Oper. Res. **38**(2), 571–581 (2011)
10. San Segundo, P., Matia, F., Rodriguez-Losada, D., Hernando, M.: An improved bit parallel exact maximum clique algorithm. Optim. Lett. **7**(3), 467–479 (2011)
11. Prosser, P.: Exact algorithms for maximum clique: a computational study. Algorithms **5**(4), 545–587 (2012)
12. Balas, E., Yu, C.S.: Finding a maximum clique in an arbitrary graph. SIAM J. Comput. **15** (4), 1054–1068 (1986)
13. San Segundo, P., Tapia, C.: Relaxed approximate coloring in exact maximum clique search. Comput. Oper. Res. **44**, 185–192 (2014)
14. Li, C.M., Quan, Z.: An efficient branch-and-bound algorithm based on MaxSAT for the maximum clique problem. In: Proceedings of AAAI-10, pp. 128–133
15. Andrade, D.V., Resende, M.G.C., Werneck, R.F.: Fast local search for the maximum independent set problem. J. Heuristics **18**(4), 525–547 (2012)
16. Batsyn, M., Goldengorin, B., Maslov, E., Pardalos, P.: Improvements to MCS algorithm for the maximum clique problem. J. Comb. Optim. **27**(2), 397–416 (2014)

17. Tomita, E., Seki, T.: An efficient branch and bound algorithm for finding a maximum clique. In: Calude, C.S., Dinneen, M.J., Vajnovszki, V. (eds.) DMTCS 2003. LNCS, vol. 2731, pp. 278–289. Springer, Heidelberg (2003)
18. San Segundo, P., Tapia, C.: A new implicit branching strategy for exact maximum clique. In: ICTAI, ICTAI Press, vol. 1, pp. 352–357 (2010)
19. Matula, D.W., Beck, L.L.: Smallest-last ordering and clustering and graph coloring algorithms. J. Assoc. Comput. Mach. **30**(3), 417–427 (1983)
20. San Segundo, P., Nikolaaev, A., Batsyn, A.: Infra-chromatic bound for exact maximum clique search (2014). (Manuscript submitted for publication)