Matemática Contemporânea, Vol. 48, 240–252
http://doi.org/10.21711/231766362021/rmc4824

Lexicographic Breadth First-Search and Branch and Bound Algorithms for the Maximum Clique Problem

Matheus Correa, Renato Carmo, Alexandre Prusch Züge

Abstract

We perform an experimental analysis on five branch and bound based algorithms for the Maximum Clique problem. The goal is to evaluate the effectiveness of a certain heuristic for the bounding function. The analysis is done through the usual statistical hypothesis testing framework. The method shows itself powerful in discriminating the cases where the heuristic is effective and those where it is not.

1 Introduction

The Maximum Clique problem (MC) is the problem of finding a clique of maximum size on a given graph. Besides being a fundamental $\mathcal{NP}$–hard problem, it is also used to model important applications in several domains [8, 10, 22].

2000 AMS Subject Classification: 05C69, 68R10, 68W40, 62K15.
Keywords and Phrases: Lexicographic Breadth-First Search, Maximum Clique, Experimental Analysis of Algorithms.
This research was supported by CNPq Proc. 428941/2016-8.

Among the algorithms for the exact solution of MC, branch and bound based schemes stand out as one of the best approaches in practice [4]. Moreover, those which use (an upper bound on) the chromatic number of the graph as the bounding function are reported to have better performance [7, 16]. In these cases, the bounding function is usually computed through a greedy coloring of the graph.

The quality of the solution provided by a greedy coloring algorithm is sensitive to the order in which the vertices of the graph are colored. We evaluate the impact of using the order induced by a Lexicographic Breadth First-Search (LexBFS) [5] as the order in which vertices are colored. The motivation for this stems from the well known fact that the order induced by a LexBFS leads a greedy coloring algorithm to compute an optimal coloring for some classes of graphs [2, 13].

More precisely, we consider five exact branch and bound algorithms for MC which use greedy coloring of the graph as their bounding strategy and perform an experimental analysis [14] aimed at comparing each of them with a modified version where, before coloring, the vertices are ordered by a LexBFS. The comparison is done using statistical hypothesis testing [6]. We use the search tree size to compare the performance of the different algorithms. This allows for a comparison that is independent of the computational environment and more easily reproducible. We note that our implementation is not particularly geared towards running time efficiency.

The text is organized as follows. Section 1.1 establishes some definitions and notation. Section 2 discusses branch and bound based algorithms for MC and the role of vertex coloring in them so as to make clear the goal of our analysis. Section 3 describes in detail the methodology and the experimental design chosen for the analysis. Section 4 presents and discusses the results. In Section 5 we make some closing remarks.

1.1 Definitions and Notation

We employ the usual definitions and notation for graphs and related concepts. In particular, given a graph G, the set of vertices adjacent to a

vertex v, called the *neighborhood* of v, is denoted $N_G(v)$; a *clique* in G is a set of vertices inducing a complete graph; the size of a maximum clique is denoted $\omega(G)$; a *coloring* of G is a partition of $V(G)$ into independent sets (called *colors*); a coloring is optimal if the number of its colors is the minimum possible.

A *greedy coloring of* G *with respect to an ordering* $(v_1, \ldots, v_n)$ of $V(G)$ is the coloring $\{C_1, \ldots, C_k\}$ obtained by taking each vertex in the ordering and placing it in the color of minimum possible index.

2 Branch and Bound Algorithms for MC

Branch and bound based algorithms [12] stand out in the literature as one of the most effective approaches for MC. The average case analysis done in [24] gives an explanation for the somewhat surprising performance of such algorithms in practice. It is possible to describe these algorithms under a unified framework [4] as follows.

MaxClique(G, Q, K, C)

Input : A graph G, cliques Q and C and a set $K \subseteq V(G) - Q$
Output : The maximum clique between C and a maximum clique M satisfying $Q \subseteq M \subseteq Q \cup K$

```
1 if K = ∅
2 |   if |Q| > |C|
3 |   |   C ← Q
4 else
5 |   if |Q| + bound(G, K) > |C|
6 |   |   remove a vertex v from K
7 |   |   C ← MaxClique(G, Q ∪ {v}, K ∩ N_G(v), C)
8 |   |   C ← MaxClique(G, Q, K, C)
9 return C
```

Informally speaking, the set C in MaxClique() is the "largest clique up to this point" and the set Q is "a growing clique"; function $\mathsf{bound}(G, K)$ returns an integer b such that $\omega(G[K]) \leq b$. This is called the "bounding step" of the algorithm. The choice of vertex v in Line 6 is called

the "branching step" and the chosen vertex is called the "pivot" of the branching; each specific choice for the branching and bounding steps result in a different algorithm. We refer to these algorithms collectively as MCBB algorithms (see [4] for a more thorough discussion of this general scheme). Note that, in any case, MaxClique$(G, \emptyset, V(G), \emptyset)$ returns a maximum clique of graph G.

Many of the best performing algorithms for MC are MCBB algorithms which implement bound(G, K) by computing a greedy coloring of $G[K]$ and return its number of colors. We refer to these algorithms as MCBBχ algorithms. As the number of colors is sensitive to the ordering of the vertices with respect to which the coloring is computed, several heuristics for this ordering have been proposed.

For every graph, there is an ordering of its vertices such that the greedy algorithm with respect to this ordering returns an optimal coloring of the graph. Such an ordering can be computed in linear time for some classes of graphs using LexBFS [13, 15]. Our goal is to determine whether or not using the order induced by a LexBFS improves the efficiency of certain MCBBχ algorithms.

This is done by performing an experimental analysis [14] of the following MCBBχ algorithms: MCLIQ [19], MCQ [20], MCR [11], DYN [18], and MCS [21]. For each of them, we introduce a modified version, namely MCLIQ$_L$, MCQ$_L$, MCR$_L$, DYN$_L$, and MCS$_L$, respectively. In each case, the modified version consists in substituting the heuristic ordering on K in the original algorithm by the ordering induced by a LexBFS in $G[K]$.

3 Comparative Experimental Analysis

Let A be an MCBB algorithm and let G be a graph. The recursive calls starting from $A(G) := A(G, Q = \emptyset, K = V(G), C = \emptyset)$ can be represented by a binary tree whose nodes are labeled by the pair (Q, K) in each call. At each node of this tree, the subtrees corresponding to the calls in Lines 7

Maximum Clique Branch and Bound based.

and 8 are called its *left and right children*, respectively. This tree will be called the *search tree of A for G*, denoted $T_A(G)$, and its *size* (number of nodes) will be denoted $|T_A(G)|$.

We compare the algorithms with respect to two performance indicators: the size of the search tree and the running time. The former is a particularly interesting performance indicator, as it is independent of the computational environment.

The experimental design is as follows. For each level $n \in N := \{300, 350, \ldots, 1000\}$, a sample S_n of 10 uniformly distributed random graphs (i.e. $\mathcal{G}_{n,1/2}$ graphs [1]) is generated. For each algorithm $A \in \mathcal{A} := \{\mathsf{MCLIQ}, \mathsf{MCQ}, \mathsf{DYN}, \mathsf{MCR}, \mathsf{MCS}\}$ and each $G \in S_n \colon n \in N$ we run $A(G)$ and its respective variation $A_L(G)$ and compute $|T_A(G)|$ and $|T_{A_L}(G)|$, collecting $15 \times 10 \times 5 \times 2 = 1500$ design points in total.

Each $|T_{A,n}|$ and $|T_{A_L,n}| \colon n \in N, A \in \mathcal{A}$, is a random variable whose range is S_n. We define $D_{A,n} := |T_{A,n}| - |T_{A_L,n}|$ as the random variables of interest. For each of them, our *null hypothesis* is $H_{0,A,n} \colon \mu_{D_{A,n}} = 0$ meaning "the LexBFS ordering in Algorithm A makes no difference with respect to the size of the search tree for the sample S_n"; our *alternative hypothesis* is then $H_{1,A,n} \colon \mu_{D_{A,n}} \neq 0$ meaning the opposite statement. The same procedure is followed with respect to the running time as performance indicator.

To ensure applicability of Student's t-test, we perform normality tests for $D_{A,n}$ and $D_{A_L,n}$. More precisely, we apply the Shapiro–Wilk, the Kolmogorov–Smirnov, the Cramer–von Mises and the Anderson–Darling tests. When at least one of these confirms that the sample is normally distributed with confidence level of 95%, we compute the p-value for Student's t-test and the corresponding 95% confidence interval for $\mu_{D_{A,n}}$.

The algorithms are implemented in C++ version 11 [9]. The implementation follows the scheme described in [4]. Execution of the algorithms is

For concepts and terminology of experimental analysis of algorithms we follow [14].
We use the routines provided by the nortest package from the R project [17].
Available online at [23].

carried out on a Linux Mint 19 server, running kernel version 4.19.16, with 196.79 GiB of RAM and an Intel Xeon E5-2690v2 3.00 GHz CPU.

4 Experimental Results and Discussion

For each performance indicator, Tables 1 and 2 show the p-value and the values of $\mu_{D_{A,n}}$ and corresponding confidence interval for each algorithm $A \in \mathcal{A}$ and each level $n \in \{300, 400, \ldots, 1000\}$ (space constraints prevent us from presenting the full results).

We note that when reporting running time, we refer only to the time inside the execution of the MaxClique() algorithm. In particular, time spent in Input/Output routines is not part of what is reported.

Made available by Centro de Computação Científica e Software Livre [3].

Table 1: Experimental Results for MCLIQ, MCQ and DYN

		search tree size			running time		
$\mathcal{A}$ vs $\mathcal{A}_L$	n	p-value	$\mu_{D_{A,n}}$	confidence interval	p-value	$\mu_{D_{A,n}}$	confidence interval
5*MCLIQ vs MCLIQ$_L$	300	2.93×10^{-1}	5315.40	-5.5×10^{3} to 1.6×10^{4}	1.44×10^{-6}	−1.09	−1.31 to −0.87
	400	4.85×10^{-2}	50 861.60	4.1×10^{2} to 1.0×10^{5}	4.87×10^{-8}	−5.69	−6.47 to −4.91
	500	3.53×10^{-5}	268 637.20	1.9×10^{5} to 3.5×10^{5}	2.5×10^{-6}	−25.13	−30.58 to −19.69
	600	6.34×10^{-4}	833 043.20	4.6×10^{5} to 1.2×10^{6}	1.12×10^{-7}	−80.25	−92.35 to −68.15
	700	1.51×10^{-2}	2 040 281.80	5.0×10^{5} to 3.6×10^{6}	8.65×10^{-8}	−205.18	−235.19 to −175.17
	800	3.86×10^{-5}	4 628 911.80	3.2×10^{6} to 6.0×10^{6}	2.3×10^{-7}	−530.61	−617.50 to −443.71
	900	5.08×10^{-2}	7 935 305.80	-3.3×10^{4} to 1.6×10^{7}	6.24×10^{-8}	−1094.76	−1249.01 to −940.52
	1000	4.09×10^{-4}	21 423 725.00	1.3×10^{7} to 3.0×10^{7}	2.93×10^{-11}	−2359.02	−2499.05 to −2218.99
5*MCQ vs MCQ$_L$	300	1.98×10^{-6}	11 137.60	8.8×10^{3} to 1.3×10^{4}	8.54×10^{-7}	−1.11	−1.33 to −0.90
	400	3.45×10^{-5}	51 570.00	3.6×10^{4} to 6.7×10^{4}	6.66×10^{-8}	−5.36	−6.12 to −4.60
	500	3.22×10^{-7}	231 421.00	1.9×10^{5} to 2.7×10^{5}	7.89×10^{-9}	−26.24	−29.16 to −23.32
	600	4.23×10^{-6}	690 943.40	5.3×10^{5} to 8.5×10^{5}	2.62×10^{-7}	−83.53	−97.42 to −69.64
	700	3.04×10^{-7}	1 614 023.20	1.3×10^{6} to 1.9×10^{6}	5.14×10^{-7}	−200.05	−236.02 to −164.08
	800	9.78×10^{-9}	5 310 768.40	4.7×10^{6} to 5.9×10^{6}	3.71×10^{-7}	−544.93	−639.26 to −450.59
	900	5.84×10^{-7}	9 363 241.60	7.7×10^{6} to 1.1×10^{7}	7.6×10^{-9}	−1095.41	−1216.83 to −973.98
	1000	1.08×10^{-10}	19 517 230.40	1.8×10^{7} to 2.1×10^{7}	3.87×10^{-10}	−2292.65	−2474.34 to −2110.96
5*DYN vs DYN$_L$	300	4.7×10^{-6}	−6399.60	-7.9×10^{3} to -4.9×10^{3}	1.03×10^{-5}	−0.45	−0.57 to −0.34
	400	9.13×10^{-8}	−30 202.60	-3.5×10^{4} to -2.6×10^{4}	6.38×10^{-6}	−2.95	−3.66 to −2.23
	500	3.86×10^{-5}	−74 233.40	-9.7×10^{4} to -5.2×10^{4}	3.85×10^{-6}	−14.08	−17.29 to −10.86
	600	9.1×10^{-4}	−158 692.40	-2.3×10^{5} to -8.5×10^{4}	4.01×10^{-6}	−47.18	−58.00 to −36.36
	700	8.15×10^{-7}	−482 946.80	-5.7×10^{5} to -3.9×10^{5}	4.32×10^{-7}	−112.91	−132.80 to −93.02
	800	6.74×10^{-8}	−921 146.60	-1.1×10^{6} to -7.9×10^{5}	2.98×10^{-6}	−329.49	−402.40 to −256.58
	900	4.76×10^{-9}	−2 435 116.40	-2.7×10^{6} to -2.2×10^{6}	4.46×10^{-8}	−710.40	−806.72 to −614.08
	1000	1.98×10^{-8}	−4 902 904.00	-5.5×10^{6} to -4.3×10^{6}	2.78×10^{-10}	−1608.93	−1731.79 to −1486.06

Table 2: Experimental Results for MCR and MCS

		search tree size			running time		
$\mathcal{A}$ vs $\mathcal{A}_L$	n	p-value	$\mu_{D_{A,n}}$	confidence interval	p-value	$\mu_{D_{A,n}}$	confidence interval
	300	2.49×10^{-6}	8908.80	7.0×10^{3} to 1.1×10^{4}	1.23×10^{-7}	−1.02	−1.18 to −0.86
	400	3×10^{-5}	42 933.60	3.0×10^{4} to 5.6×10^{4}	8.28×10^{-7}	−5.08	−6.05 to −4.11
5*MCR vs MCR$_L$	500	3.1×10^{-7}	192 294.80	1.6×10^{5} to 2.2×10^{5}	3.55×10^{-6}	−23.00	−28.19 to −17.80
	600	5.27×10^{-6}	627 034.20	4.8×10^{5} to 7.8×10^{5}	9.63×10^{-7}	−78.37	−93.54 to −63.20
	700	1.59×10^{-8}	1 534 253.20	1.3×10^{6} to 1.7×10^{6}	1.89×10^{-8}	−212.47	−238.59 to −186.35
	800	2.09×10^{-8}	4 994 458.40	4.4×10^{6} to 5.6×10^{6}	4.24×10^{-7}	−514.60	−605.07 to −424.13
	900	7.3×10^{-7}	9 277 507.20	7.5×10^{6} to 1.1×10^{7}	7.64×10^{-9}	−1112.10	−1235.45 to −988.75
	1000	1.31×10^{-11}	18 230 854.60	1.7×10^{7} to 1.9×10^{7}	1.56×10^{-9}	−2273.07	−2483.77 to −2062.38
	300	1.45×10^{-8}	−3070.00	-3.4×10^{3} to -2.7×10^{3}	6.53×10^{-7}	−0.83	−0.98 to −0.67
	400	3.27×10^{-8}	−15 964.80	-1.8×10^{4} to -1.4×10^{4}	1.25×10^{-8}	−4.85	−5.42 to −4.28
5*MCS vs MCS$_L$	500	4.92×10^{-8}	−67 911.80	-7.7×10^{4} to -5.9×10^{4}	3.97×10^{-8}	−20.31	−23.02 to −17.59
	600	2.94×10^{-8}	−198 271.20	-2.2×10^{5} to -1.7×10^{5}	5.55×10^{-8}	−64.00	−72.90 to −55.10
	700	2.57×10^{-10}	−505 330.40	-5.4×10^{5} to -4.7×10^{5}	1.95×10^{-9}	−169.67	−185.80 to −153.54
	800	1.74×10^{-11}	−908 165.20	-9.6×10^{5} to -8.6×10^{5}	5.36×10^{-8}	−499.15	−568.27 to −430.04
	900	6.08×10^{-9}	−1 955 873.00	-2.2×10^{6} to -1.7×10^{6}	3.22×10^{-8}	−895.88	−1012.89 to −778.87
	1000	2.05×10^{-10}	−4 287 166.00	-4.6×10^{6} to -4.0×10^{6}	1.52×10^{-9}	−1857.87	−2029.58 to −1686.15

Note that the actual values of the performace indicators are not shown, only information concerning statistical testing. Recall that p-value ≤ 0.05 means that there is statistically significant difference for the respective performance indicator and the null hypothesis must be rejected. This difference is an improvement when the value of $\mu_{D_{A,n}}$ is positive.

Only in three cases we have not obtained p-value ≤ 0.05. All of them are respective to search tree size as performance indicator and algorithm MCLIQ for $n \in \{300, 850, 900\}$. In all other cases we have p-value ≤ 0.05.

For the search tree size we have $\mu_{D_{A,n}} > 0$ for $A \in \{\mathsf{MCLIQ}, \mathsf{MCQ}, \mathsf{MCR}\}$ and all $n \in N$, meaning that the LexBFS ordering significantly improves the efficiency of these algorithms by reducing search space.

It seems unexpected that there are cases where the search tree size grows significantly after the LexBFS ordering. The fact is, however, that the ordering of the vertices in these algorithms serves not only as an heuristic handle to the coloring in the bounding step but also as a guide to the choice of the pivot in the branching step. The conclusion, in these

cases, is that the reordering done by the LexBFS spoils this guidance.

For the running time we have $\mu_{D_{A,n}} < 0$ for every design point. That is, the running time of the modified algorithm is greater than the unmodified one. This was to be expected as the modification implies extra processing in the execution of the algorithmand, as already mentioned, our implementation was not particularly geared towards running time efficiency.

5 Concluding Remarks

We perform an experimental analysis aimed at evaluating the impact of an "improvement candidate" for some algorithms for the Maximum Clique problem. The analysis is carried out with respect to two performance indicators, the search tree size and the running time, and is based on statistical hypothesis testing.

The experimental results show that the proposed modification to the original algorithms is an improvement to three of them, with respect to the search tree size, while for the other two algorithms it actually degraded the same performance indicator.

The results also show that the modification increases the running time of the algorithm. As noted above, this was to be expected as the modification implies in extra processing along the execution of the algorithm and our implementation was not particularly geared towards running time efficiency. A more careful implementation may succeed in reducing the running times for the cases where the search tree size is reduced.

This is why we consider the analysis with respect to the search tree size to be more meaningful. On the one hand, it is less sensitive to implementation details and the computational environment. On the other hand, it was capable of separating the studied algorithms into two classes: those for which the modification is an improvement and those for which it is not. For one thing, this points the way to go for further improvement, as there would be no point in reducing the running time of the modification in those algorithms for which it does not reduce the search tree size.

References

[1] B. Bollobás. *Random Graphs*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, Cambridge, 2001.

[2] A. Brandstädt, V. B. Le, and J. P. Spinrad. *Graph classes: a survey*. SIAM, 1999.

[3] C3SL. Centro de Computação Científica e Software Livre. `https://www.c3sl.ufpr.br/`.

[4] Renato Carmo and Alexandre Züge. Branch and bound algorithms for the maximum clique problem under a unified framework. *Journal of the Brazilian Computer Society*, 18(2):137–151, December 2012.

[5] Derek G. Corneil. Lexicographic breadth first search–a survey. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 1–19. Springer, 2004.

[6] Jay L. Devore. *Probability and Statistics for Engineering and the Sciences*. Cengage Learning, 2015.

[7] Cleverson Sebastião dos Anjos, Alexandre Prusch Züge, and Renato Carmo. An experimental analysis of exact algorithms for the maximum clique problem. *Matemática Contemporânea*, 44:1–20, 2016.

[8] Elias P. Duarte Jr., Thiago Garrett, Luis C. E. Bona, Renato Carmo, and Alexandre P. Züge. Finding stable cliques of planetlab nodes. In *2010 IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 317–322. IEEE, June 2010.

[9] ISO. *ISO/IEC 14882:2011 Information technology — Programming languages — C++*. International Organization for Standardization, Geneva, Switzerland, February 2012.

[10] Marko Jukič, Janez Konc, Dušanka Janežič, and Urban Bren. ProBiS H2O MD approach for identification of conserved water sites in

protein structures for drug design. *ACS Medicinal Chemistry Letters*, 11(5):877–882, 2020.

[11] Janez Konc and Dušanka Janežič. An improved branch and bound algorithm for the maximum clique problem. *MATCH Communications in Mathematical and in Computer Chemistry*, June 2007.

[12] Donald L. Kreher and Douglas R. Stinson. *Combinatorial algorithms: generation, enumeration, and search.* Discrete Mathematics and Its Applications. CRC Press, Boca Raton, Florida, 1999.

[13] Frédéric Maffray. On the coloration of perfect graphs. In *Recent Advances in Algorithms and Combinatorics*, pages 65–84. Springer, 2003.

[14] Catherine C. McGeoch. *A guide to experimental algorithmics.* Cambridge University Press, 2012.

[15] Donald J. Rose, Robert E. Tarjan, and George S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on Computing*, 5(2):266–283, 1976.

[16] Pablo San Segundo, Alvaro Lopez, Mikhail Batsyn, Alexey Nikolaev, and Panos M. Pardalos. Improved initial vertex ordering for exact maximum clique search. *Applied Intelligence*, 45(3):868–880, 2016.

[17] The R Foundation. The R Project for Statistical Computing. `https://www.r-project.org/`.

[18] Etsuji Tomita and Toshikatsu Kameda. An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. *Journal of Global Optimization*, 37(1):95–111, January 2007.

[19] Etsuji Tomita, Yasuhiro Kohata, and Haruhisa Takahashi. A simple algorithm for finding a maximum clique. Technical Report 1, University of Electro-Communications, Tokyo, 1988.

[20] Etsuji Tomita and Tomokazu Seki. An efficient branch-and-bound algorithm for finding a maximum clique. In Cristian S. Calude, Michael J. Dinneen, and Vincent Vajnovszki, editors, *Proceedings of the 4th International Conference on Discrete Mathematics and Theoretical Computer Science (DMTCS)*, volume 2731 of *Lecture Notes in Computer Science*, pages 278–289. Springer, Berlin, Heidelberg, 2003.

[21] Etsuji Tomita, Yoichi Sutani, Takanori Higashi, Shinya Takahashi, and Mitsuo Wakatsuki. A simple and faster branch-and-bound algorithm for finding a maximum clique. In Md Rahman and Satoshi Fujita, editors, *Proceedings of the 4th International Workshop Algorithms and Computation (WALCOM)*, volume 5942 of *Lecture Notes in Computer Science*, pages 191–203. Springer, Berlin, Heidelberg, 2010.

[22] Qinghua Wu and Jin-Kao Hao. A review on algorithms for maximum clique problems. *European Journal of Operational Research*, 242(3):693–709, 2015.

[23] Alexandre Prusch Züge. MaxCliqueBB – GitLab repository. `https://gitlab.c3sl.ufpr.br/apzuge/maxcliquebb`.

[24] Alexandre Prusch Züge and Renato Carmo. On comparing algorithms for the maximum clique problem. *Discrete Applied Mathematics*, 247:1–13, 2018.

Matheus Correa
Universidade Federal do Paraná
Departamento de Informática
matheusviniciuscorrea@gmail.com

Renato Carmo
Universidade Federal do Paraná
Departamento de Informática
renato@inf.ufpr.br

Alexandre Prusch Züge
Universidade Federal do Paraná
Campus Jandaia do Sul
alexandrezuge@ufpr.br