Estimating Clique Size via Discarding Subgraphs

Sándor Szabó
University of Pécs, Hungary
E-mail: sszabo7@hotmail.com

Bogdán Zaválnij
Rényi Institute of Mathematics
E-mail: bogdan@renyi.hu

Keywords: combinatorial optimization, maximum clique problem, greedy coloring, Lovász' theta function, practical solutions of NP-hard problems

Received: April 1, 2020

The paper will present a method to establish an upper bound on the clique number of a given finite simple graph. In order to evaluate the performance of the proposed algorithm in practice we carry out a large scale numerical experiment on carefully selected benchmark instances.

Povzetek: Razvita in opisana je nova metoda za določanje zgornje meje števila klik v grafu.

1 Introduction

In this paper we will work with finite simple graphs. A graph is finite if it has finitely many nodes and finitely many edges. A graph is simple if it does not have double edges and if it does not have loops. If V is the set of nodes and E is the set of edges of a finite simple graph G, then the ordered pair (V, E) completely describes the graph G.

A set of nodes I of the finite simple graph $G = (V, E)$ is called an independent set if two distinct nodes in I are never adjacent. A set of nodes U of G is called a clique if two distinct nodes in U are always adjacent. Sometimes the subgraph Δ induced by U in G is called a clique of G. If the clique Δ has k nodes, then we say that Δ is a k-clique in G. The number k is sometimes referred as the size of the clique. A k-clique Δ in G is called a maximal clique if G does not have any $(k+1)$-clique that contains Δ as a subgraph. A k-clique Δ in G is called a maximum clique if G does not contain any clique with size larger than k. This well defined integer k is called the clique number of the graph G and it is denoted by $\omega(G)$. Plainly each maximum clique in G has the same size which is equal to $\omega(G)$. Finding both maximal and maximum cliques in a given graph has important and interesting theoretical and practical applications. (For further details consult with [2, 5, 12, 23, 24].)

It is a well known result of the complexity theory of computations that the problem of determining the clique number of a given finite simple graph belongs to the NP-hard complexity class. (For a proof see [9, 17].) This can be interpreted such that computing the clique number of a given graph is computationally demanding. From this reason instead of determining the clique number exactly sometimes we settle for finding a large but not necessarily maximum clique. In this sense we distinguish exact and non-exact clique search algorithms. The non-exact algorithms are further categorized as local search or stochastic or heuristic algorithm depending on the nature of the search technique involved. An example of the exact algorithm can be found in [22] and an example of the non-exact method can be seen in [13]. A strong upper bound can be used as a quality certificate to prove that a heuristically found clique is close enough to a maximum clique.

The main result of this paper is a procedure to establish an upper bound for the clique number of a given graph. The procedure employs legal coloring of the nodes of tactically chosen subgraphs of the given graph. Although the coloring schemes we use are complex the approach is essentially combinatorial. As a consequence it does not involve floating point arithmetic and free of rounding errors. It is known that the Lovász' theta function can be used to compute upper bounds for the clique number of a given graph. This estimate in practice boils down to solve semidefinite programs which is inherently a floating point computation. We were interested in how the elementary combinatorial and the less elementary techniques compares. In the lack of adequate theoretical tools we carried out a large scale numerical experiment on carefully selected problems.

2 Bounding the clique number

Let G be a finite simple graph and let b a fixed positive integer. We color the nodes of the graph G such that the following two conditions hold.

1. Each node of G receives exactly b distinct colors.

2. Two adjacent nodes never receive the same color.

A coloring of the nodes of G satisfying these conditions is called a b-fold legal coloring of the nodes of G. For each finite simple graph G there is an integer k such that the nodes of G can be legally colored with k colors in a b-fold manner but the nodes of G do not admit a legal b-fold coloring using $(k-1)$ colors. This well defined number k is referred as the b-fold chromatic number of G and it is denoted by $\chi_b(G)$. In the $b = 1$ particular case we drop the subscript b and use the notation $\chi(G)$. We refer to this number as the chromatic number of G. Coloring the nodes of a graph is an important problem with many applications and with a venerable history. (See [10].)

At a legal coloring of the nodes of G the nodes of a k-clique in G must receive k distinct colors and consequently $\omega(G) \leq \chi(G)$. In other words, legal coloring of the nodes can be used to establish upper bound for the clique number.

It is known from the complexity theory of computations that determining the chromatic number of a given graph is an NP-hard problem. A number of greedy coloring algorithms is available to construct a legal coloring of the nodes but not necessarily with an optimal number of colors. (See for instance [6, 14, 7].) Many clique search algorithms used in practice employs these greedy algorithms for upper bounding the clique size.

In this paper we will show that with some extra effort one may improve on the above upper estimates. We have carried out a number of computations in which approximate node coloring procedures were used. But a little contemplation can convince the reader that the computational scheme we use in fact can be combined with less elementary clique size estimates without major difficulty.

Let $I(G)$ denote the set of all independent sets of nodes of a graph G, and let $I(G,u)$ denote the independent sets of G that contain the node u. For each independent set I, we define a nonnegative real variable x_I. The fractional chromatic number of G, which is denoted by $\chi_f(G)$ is the minimum value of $\sum_{I \in I(G)} x_I$, subject to $\sum_{I \in I(G,u)} x_I \geq 1$ for each node u. This value provides an upper estimate of the clique number of G. (See [1].)

A connection between the fractional and the b-fold chromatic numbers is the following $\chi_f(G) = \lim_{b\to\infty} \frac{\chi_b(G)}{b}$. Therefore in practice one can use b-fold coloring [19] as an approximation.

Lemma 1. $\omega(G) \leq \frac{\chi_b(G)}{b} \leq \chi(G)$.

Proof. At any b-fold coloring of the nodes of G the nodes of a k-clique in G must receive $b \cdot k$ distinct colors. Thus $\chi_b(G)$ must be at least $b \cdot k$.

For any positive integer b and a legal node coloring of the graph with $\chi(G)$ colors one can construct a legal b-fold coloring with $b \cdot \chi(G)$ colors. For this replace each color with a list of b different colors on each node. From the conditions of legal node coloring and b-fold coloring this will yield to a proper b-fold coloring. Thus $\chi_b(G) \leq b \cdot \chi(G)$. □

Computing the b-fold chromatic number is an NP-hard problem, as for $b = 1$ case it reduces to the problem of computing the chromatic number. So one looks for heuristic algorithms in case of large graphs. The b-fold coloring of a graph can be reduced to a legal coloring of a suitable auxiliary graph as described in [19, 21], so any heuristic algorithm for legal coloring of the nodes can be easily adopted for finding a b-fold coloring as well.

We may use other upper bounding techniques. For each finite simple graph G there is a well defined graph parameter $\vartheta(G)$, which is called the Lovász' theta number [15]. The Lovász' theta number of the complement of a graph G also bounds the clique number of G from above. The values of the ϑ function can be computed in polynomial time [3, 4], but the degree of the polynomial bound of the running time is high. The bounds on the clique number we have listed so far are the following [8]:

$$\omega(G) \leq \vartheta(\overline{G}) \leq \chi_f(G) \leq \frac{\chi_b(G)}{b} \leq \chi(G).$$

3 The description of the algorithm

Let $v_1, \ldots, v_n$ be all the nodes of $G = (V, E)$. For each node v_i of G we define a graph K_i of G. Namely, let K_i be the subgraph of G induced by the set of nodes $N(v_i)$. Here $N(v_i)$ is the set of neighbors of v_i in G, that is,

$$N(v_i) = \{v : v \in V, \{v_i, v\} \in E\}.$$

For each i, $1 \leq i \leq n$ we compute an upper bound α_i for $\omega(K_i)$. For example using a greedy coloring algorithm we color the nodes of K_i legally and set α_i to be the number of colors the algorithm provided. Obviously one can use another upper bound as well. Using the given graph G we define a sequence of numbers $\mu_1, \ldots, \mu_n$. We pick a graph K_i with a minimum α_i. We set $\mu_n = \alpha_i$. Next we delete the node v_i from G. We repeat the whole procedure with this smaller graph and get a new quantity μ_{n-1}. When all nodes of G are deleted the computation terminates. At this stage we have a sequence $\mu_1, \ldots, \mu_n$.

Theorem 1. *Using the notation introduced above the inequality*

$$\omega(G) \leq 1 + \max\{\mu_1, \ldots, \mu_n\}$$

holds.

In the remaining part of this section we justify this claim.

Let $u_1, \ldots, u_n$ be a fixed rearrangement of the nodes $v_1, \ldots, v_n$. We consider the subgraph L_i of G induced by the set of nodes $N(u_i) \cap \{u_1, \ldots, u_i\}$ for each i, $1 \leq i \leq n$. Note that $u_i \notin N(u_i)$ and so

$$N(u_i) \cap \{u_1, \ldots, u_i\} = N(u_i) \cap \{u_1, \ldots, u_{i-1}\}.$$

In the $i = 1$ special case we should identify $\{u_1, \ldots, u_{i-1}\}$ with the empty set. Thus L_1 is a graph without any node. In this situation we set $\omega(L_1)$ to be 0.

We will need the following result.

Lemma 2. *Using the notations introduced above the inequality*

$$\omega(G) \leq 1 + \max\{\omega(L_1), \ldots, \omega(L_n)\}$$

holds.

Proof. Set $k = \omega(G)$. Clearly, the graph G contains a k-clique Δ. On the fixed list $u_1, \ldots, u_n$ of the nodes $v_1, \ldots, v_n$ there is a unique u_i such that u_i is a node of Δ but u_j is not a node of Δ for each j, $i < j \leq n$. The set $N(u_i) \cap \{u_1, \ldots, u_{i-1}\}$ contains $k-1$ nodes of Δ and so $k - 1 \leq \omega(L_i)$ holds. From $k \leq 1 + \omega(L_i)$ it follows that

$$k \leq 1 + \max\{\omega(L_1), \ldots, \omega(L_n)\}.$$

□

The alart reader will recognize that in the statement of Lemma 2 the inequality sign can be replaced by an equation sign. But we are content with this weaker result.

Computing $\omega(L_i)$ is computationally demanding. Therefore we use an easily computable upper bound μ_i for $\omega(L_i)$ instead. Lemma 2 shows that we can freely rearrange the nodes for the estimate of $\omega(G)$. This rearrangement influences the final result. Preliminary tests showed that different rearrangements can result in quite different upper bound values. The rearrangement we propose results a rather good upper estimate.

Let us turn back to our proposed algorithm. First we locate a subgraph K_i of G for which the corresponding α_i is a minimum among $\alpha_1, \ldots, \alpha_n$. We set $u_n = v_i$. This will be the first node fixed in our rearrangement. Putting it in another way we may say that we have identified the last element u_n of the fixed list of nodes $u_1, \ldots, u_n$. We have also identified the subgraph L_n. Namely, $L_n = K_i$.

We delete the node $u_n = v_i$ from the graph G. It means we are working with a smaller new graph induced by the set of nodes $V \setminus \{u_n\}$. Next we identify the node u_{n-1} and the subgraph L_{n-1} by performing the previous step again. Continuing in this way we get the subgraphs $L_1, \ldots, L_n$. Our procedure gives that $\omega(L_i) \leq \mu_i$, for each i, $1 \leq i \leq n$ and so

$$\max\{\omega(L_1), \ldots, \omega(L_n)\} \leq \max\{\mu_1, \ldots, \mu_n\}.$$

Combining this result with Lemma 2 we get $\omega(G) \leq 1 + \max\{\mu_1, \ldots, \mu_n\}$ as stated in Theorem 1. We we call this algorithm DISCARDING.

We make some remarks about streamlining the DISCARDING procedure. Remember that first employing a greedy coloring algorithm to the graphs $K_1, \ldots, K_n$ we compute the numbers $\alpha_1, \ldots, \alpha_n$. Since we are looking for the minimum value occurring among $\alpha_1, \ldots, \alpha_n$ in certain cases we need not to complete the coloring procedure. We may abort coloring a subgraph K_i if we have already used more colors than the minimum number of colors needed so far.

Next, when we delete the node v_i from G for which α_i is minimum we may delete each v_j for which α_j is minimum. In this way we end up with a shorter list $\mu_1, \ldots, \mu_s$ instead of the longer list $\mu_1, \ldots, \mu_n$. Note that the upper estimate $\omega(G) \leq 1 + \max\{\mu_1, \ldots, \mu_s\}$ is not weaker than the upper estimate $\omega(G) \leq 1 + \max\{\mu_1, \ldots, \mu_n\}$. We can also delete all the v_i nodes during the procedure, where α_i less than or equal to the maximum of the previous μ values.

Note that when the node v_j is not adjacent to the deleted node v_i, then the upper bound α_j of $\omega(K_j)$ need not to be recalculated. The reason is that in this situation deleting the node v_i leaves the subgraph K_j unchanged.

We would like to point out, that calculating the α_i values can be carried out independently of each other, thus we can perform this calculation in parallel fashion. We implemented the algorithm DISCARDING in both sequential and parallel manner using OpenMP for shared memory computers. Both programs resulted the same upper bounds. It is clear, that the program could be implemented using MPI for distributed computers as well thus achieving greater scalability.

4 A simpler estimating procedure

In this section we consider a less sophisticated version of the procedure DISCARD. We will call this new procedure SEQUENTIAL. We fix an ordering $w_1, w_2, \ldots, w_n$ of the nodes $v_1, v_2, \ldots, v_n$ of the given finite simple graph $G = (V, E)$. We consider the graph L_i induced by $N(w_i) \cap \{w_1, \ldots, w_i\}$ for each i, $1 \leq i \leq n$ and compute a μ_i such that $\omega(L_i) \leq \mu_i$ holds for each i, $1 \leq i \leq n$. Now the inequality $\omega(G) \leq 1 + \max\{\mu_1, \ldots, \mu_n\}$ holds.

Let P be an auxiliary procedure for computing an upper bound $\mu(G, P)$ for the clique number $\omega(G)$ of a given finite simple graph G. We say that P has the monotonicity property if $\mu(H, P) \leq \mu(G, P)$ holds whenever H is a subgraph of G.

For example the auxiliary procedure of computing the chromatic number of a graph has the monotonicity property as the inequality $\chi(H) \leq \chi(G)$ holds for each subgraph H of G. On the other hand the auxiliary procedure of using the not necessarily optimal number of colors of a legal coloring of the nodes does not have the monotonicity property. It can happen that we use more colors to color the nodes of a subgraph H than we use for coloring the nodes of the whole graph G.

The next result reveals a certain optimality property of the DISCARDING procedure.

Theorem 2. *Let $G = (V, E)$ be a finite simple graph and suppose that the auxiliary procedure used to establish the upper bound $\omega(K_i) \leq \alpha_i$ has the monotonicity property. Then the SEQUENTIAL procedure does not provide a better estimate for the clique number of G than the DISCARDING procedure.*

Proof. Suppose that the DISCARDING algorithm gives rise to the ordering $u_1, u_2, \ldots, u_n$ of the nodes $v_1, v_2, \ldots, v_n$ of G. Assume on the contrary that the

SEQUENTIAL procedure applied to some ordering $w_1, w_2, \ldots, w_n$ of the nodes $v_1, v_2, \ldots, v_n$ of G leads to a strictly smaller upper estimate of $\omega(G)$.

We show that such a sequence leads to contradiction. Let us assign colors red, green, yellow to the nodes $v_1, v_2, \ldots, v_n$. This coloring of the nodes is based on the sequence $u_1, u_2, \ldots, u_n$, and it is not connected to the concept of legal coloring. There is a special node in this sequence, called the pivot node, for which the upper estimate attains its maximum. If the maximum is attained at several nodes we choose the last one. The pivot node of the graph will receive color green. We color the nodes in the sequence preceeding the pivot node by red, and the nodes after the pivot node by yellow.

Of course the members of the sequence $u_1, u_2, \ldots, u_n$ and the sequence $w_1, w_2, \ldots, w_n$ are colored with red, green and yellow, as they are just reordering of $v_1, v_2, \ldots, v_n$.

The following observation will play a critical role. When in the course of the algorithm we deleted all yellow nodes from the graph G (let us denote the nodes of this new graph by V_y) we made an upper estimate for graphs using all remaining nodes, and the upper estimate for the graph using the green node was minimal. That means that the upper estimate for the graphs using the red nodes is at least as this figure after deleting the all yellow nodes of the graph.

Now we look at the other sequence $w_1, w_2, \ldots, w_n$ assumed to give a better upper estimate. We locate two nodes in it, the pivot node (the green one), and the last node among the nodes that colored red. We shall denote them w_g and w_r, respectively. Note, that all nodes w_i, $i > r$ are colored yellow or green. We distinguish two cases:

Case $(g > r)$**.** In this situation all nodes $w_i, i > g$ are colored yellow, as the last red node appears *before* the green node. Let us denote the nodes of the graph we get after deleting all nodes w_i, $i > g$ from G by V_g. As the set of deleted nodes is a subset of the set of all yellow nodes, the graph induced by V_y is a subgraph of the graph induced by V_g. But the upper estimate for the graph induced by $N(w_g) \cap V_g$ is assumed to be less than the upper estimate for the graph induced by $N(w_g) \cap V_y$. This contradicts the monotonicity property of the auxiliary procedure.

Case $(r > g)$**.** In this situation all nodes $w_i, i > r$ are colored yellow, as w_r is the last red node and the only green one precedes it. Let us denote the nodes of the graph we get after deleting all nodes w_i, $i > r$ from G by V_r. As the set of deleted nodes is a subset of the set of all yellow nodes, the graph induced by V_y is a subgraph of the graph induced by V_r. But the upper estimate for the graph induced by $N(w_r) \cap V_r$ is assumed to be less than the upper estimate for the graph induced by$N(w_r) \cap V_y$. This contradicts the monotonicity property of the auxiliary procedure.

□

5 Numerical experiments

We selected altogether 43 graphs for carrying out our extended measurements. As we were aiming at problems where it is hard to calculate the $\omega(G)$, we used sources of graphs according to this, and decided to use graphs having at least 500 nodes. We used those graphs from these sources for which either the Lovász' theta program or our algorithm could finish the calculation of the upper bound in 100 000 seconds using the available 48GB of memory. The first 29 graphs[1] come from the problems of the 2nd DIMACS Challenge [11]. The second 11 graphs[2] come from various error correcting code problems [18]. (Note, that we used complement graphs of those from the webpage, as the original problem asks for the maximum independent set.) The last 3 graphs[3] are reformulated problems of monotonic matrices [20, 16]. We choose these graphs so that they would represent extremely hard clique search problems and for some we even do not know the value of the size of the maximum clique as the available clique search programs are not able to find the exact $\omega(G)$ value. There are a few ones where the exact value of the clique number is known but the existing clique solvers are not able to compute them. The so-called Johnson codes give rise to such clique problems.

As the proposed algorithm instructs us to use an arbitrary upper bound for calculating the α values we need to choose one method as a starting point. We have chosen the b-fold coloring with Culberson's iterated recoloring scheme because it gives the best result in reasonable time of couple of seconds. This way we were using several hours of computational time and hoped to achieve improvements over the original upper bound of b-fold coloring. A b-fold legal coloring of the nodes of a given graph can be reduced to legally coloring the nodes of an auxiliary graph described in [21]. In the course of our numerical experiments we performed several measurements to establish and compare different upper bounds. We collected the results in the Table 1. Namely, we did the following measurements:

1. perform a legal coloring of the graph (column: "legal coloring");
2. find the Lovász' theta value of the complement graph (column: "$\vartheta(\overline{G})$");
3. perform a b-fold coloring of the graph (column: "b-fold coloring");
4. use b-fold coloring as a base and perform the proposed algorithm (column: "DISC").

All coloring programs started with a DSatur algorithm due to D. Brélaz [6]. Then we recolored the nodes several

[1] http://iridia.ulb.ac.be/~fmascia/maximum_clique/DIMACS-benchmark

[2] https://oeis.org/A265032/a265032.html

[3] http://mathworld.wolfram.com/MonotonicMatrix.html

times using a method due to J. C. Culberson [7]. The stopping criteria was if the number of colors did not changed after a specified number of iterations. For smaller and medium graphs we used 1000 iterations; for bigger graphs (over 3000 nodes) we used 500 iterations. We used 7-fold coloring for smaller graphs (up to 2000 nodes); 5-fold coloring for medium graphs (over 2000 nodes); and 3-fold coloring for the biggest graphs (over 3000 nodes). The Lovász' theta function was computed with the program of B. Borchers [3, 4]. The measurements were performed on a computer with two Intel Xeon X5680 3.33 GHz processors – all together 12 cores – and 48GB of RAM. (This computer happens to be a node of a supercomputer, but this is of no importance for the present paper.) The legal coloring and b-fold coloring of the graphs were performed in sequential manner. The Lovász' theta calculations used all 12 cores due to the underlying BLAS implementation. Our DISCARDING algorithm used also 12 cores as it performed the calculation of α_i values independently. Note, that for BLAS the 12 core is a limit as it needs to be run in shared memory environment, while our algorithm could have used more cores in distributed parallelization on a supercomputer but for the sake of comparability we limited it to the same 12 cores.

The running times for legal coloring were smaller than a second or in the range of a few seconds. The time for performing the b-fold coloring depends on the size of the graph and the value of b. The running times for b-fold coloring algorithm was in the range of several seconds up to a few minutes. The running times of the Borchers' program on 12 cores for calculating the Lovász' theta of the complement graph is indicated in the column "time (sec) $\vartheta(\overline{G})$ (12c)". The running times of our parallel algorithm using the same 12 cores is indicated in the column "time (sec) DISC (12c)". Those values that cannot be calculated on the used test hardware we indicated in the table with "$>$ 24h", as the 100 000 seconds limit is roughly 24 hours.

6 Evaluation of the proposed algorithm

From the results in the Table 1 we may conclude that the proposed algorithm is feasible, that is, it can be computed on a nowadays computer in reasonable time. But in this section we are making also a comparison between our algorithm based on the b-fold coloring and the Lovász' theta upper bound calculation. We should point out, that this comparison is far from trivial:

1. We needed to parallelize our program and use the same number of cores by both programs. Note, that Borchers' program uses shared memory, so we could not use more cores than cores in one node of a supercomputer. On the other hand our program could be written to use a distributed supercomputer with hundreds of cores.
2. As our program, which uses b-fold coloring, using only integer and bit calculation the different architectures are not affecting the running times much. Opposed to this Borchers' program uses floating point calculations and BLAS. Thus different architectures (Intel or AMD), different compilers (icc or gcc) and different implementations of BLAS (Reference BLAS, ATLAS, OpenBLAS or Intel MKL library) all have huge effect on the running times. A modern AMD desktop PC with 8 core Ryzen processor using gcc as a compiler and OpenBLAS resulted in up to 50 times(!) slower running times compared to the times presented in Table 1 for the Lovász' theta calculations on Intel architecture, icc compiler and the MKL library. The architecture, compiler and BLAS implementation used for the results of the present paper all strongly favor the Lovász' theta calculations. A reproduction of the results on a desktop PC may result much longer running times for the Lovász' theta calculation program due to Borchers while giving similar times for our algorithm.

The reader can see that the computation of the Lovász' theta number for 22 of 43 instances could not be completed. In each of these 22 cases the program terminated with the error message: "not enough memory". As it turns out the length of the calculation of the Lovász' theta function depends on the number of edges (m) of the complement graph (or the non-edges, the missing edges of the graph). The running time and memory requirements are $O\left(m^3\right)$ and $O\left(m^2\right)$, respectively. (See [3, 4].) Using these asymptotic results we estimated the running time as $7.2 \cdot 10^{-11} m^3$ seconds and the memory requirement as $9.2 \cdot 10^{-9} m^2$ GigaBytes. This estimate was consistent with the experimental results. Clearly 48GB of memory is not enough in the missing cases. Actually, some instances, as for example the `keller6` graph, would need 1 TB(!) of memory and would run for nearly a thousand of years. If one would use a PC with nowadays usual 8-16GB of memory 6 other instances would be out of reach, thus making 28 out of 43 instances incalculable.

To sum up the results, we can say, that our algorithm gives better upper bounds than the Lovász' theta calculation for 24 out of 43 instances. In addition it runs faster for 24 out of 43 instances. It holds for the cases when the Lovász' theta computation cannot be completed. Further there are instances when our algorithm actually gave lower bound than the Lovász' theta approach. These are the graphs `MANN_a45`, `MANN_a81`, `keller4` and `p_hat300-1` from which the former two are in the table, and the later two are not as they have less than 500 nodes. Obviously, there can be other cases as well among those where the Lovász' theta could not be calculated. The graphs `johnson10-4-4` and `p_hat300-2` gave the same bound – these two also was excluded from the table because of being too small. Note, that in several cases our algorithm gave sharp upper bound which is equal to the clique number: `p_hat300-1`,

latin10, c-fat500-1, queen40, queen50 and san1000 graphs.

Let us return to the question if the proposed algorithm can lower the upper bound provided by the base algorithm. One can see from the results that the proposed algorithm with few exceptions improves the upper bound of b-fold coloring, in some cases radically. We may conclude that there is computational evidence that the proposed algorithm lower the upper bound if we use more computational resources. Let us turn to the question how the proposed algorithm compares to the Lovász' theta based methodology. We are getting better result only a few times. However, we were able to compute our new bound for all but two graphs that cannot be said about the Lovász' theta number. Practically, our algorithm is so versatile that even for those cases where the upper bound could not be calculated we are able to tune it using smaller b and using smaller number of recoloring steps. Thus reducing the running time we can turn the instance calculable. Note, that one can replace the upper bounds computed by coloring the nodes of the tactically chosen subgraphs by upper bounds computed by the Lovász' theta function. The calculations involved in this situation would use much more time and need to be performed several thousand times. Thus we would expect it to be reasonable only using distributed computing on a supercomputer, as the calculations of the α values can be done independently.

Acknowledgements

The project has been supported by National Research, Development and Innovation Office – NKFIH Fund No. SNN-135643.

References

[1] E. Balas and J. Xue, *Weighted and unweighted maximum clique algorithms with upper bounds from fractional coloring,* Algorithmica 15 (1996), pp. 397–412. https://doi.org/10.21236/ada275328

[2] I. M. Bomze, M. Budinich, P. M. Pardalos and M. Pelillo, *The Maximum Clique Problem, Handbook of Combinatorial Optimization Vol. 4,* Kluwer Academic Publisher, 1999. https://doi.org/10.1007/978-1-4757-3023-4_1

[3] B. Borchers, *CSDP, A C library for semidefinite programming,* Optimization Methods and Software, 11(1) (1999), pp. 613–623. https://doi.org/10.1080/10556789908805765

[4] B. Borchers and J. G. Young, *Implementation of a primal-dual method for SDP on a shared memory parallel architecture,* Computational Optimization and Applications 37(3), (2007) pp. 355–369. https://doi.org/10.1007/s10589-007-9030-3

[5] A. Bóta and M. Krész, *A high resolution clique-based overlapping community detection algorithm for small-world networks,* Informatica, 39(2). (2015).

[6] D. Brélaz, *New methods to color the vertices of a graph,* Communications of the ACM, 22 (1979), pp. 251–256. https://doi.org/10.1145/359094.359101

[7] J. C. Culberson, *Iterated Greedy Graph Coloring and the Difficulty Landscape,* Technical Report, University of Alberta, 1992.

[8] C. Elphick and P. Wocjan, *An inertial lower bound for the chromatic number of a graph,* The Electronic Journal of Combinatorics, Volume 24, Issue 1 (2017) https://arxiv.org/abs/1605.01978 https://doi.org/10.37236/6404

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

[10] F. Harary, *Graph Theory,* Addison-Wesley, 1969.

[11] J. Hasselberg, P. M. Pardalos, and G. Vairaktarakis, *Test case generators and computational results for the maximum clique problem,* Journal of Global Optimization 3 (1993), pp. 463–482. http://www.springerlink.com/content/p2m65n57u657605n https://doi.org/10.1007/bf01096415

[12] D. Kumlander, *Some Practical Algorithms to Solve the Maximal Clique Problem,* PhD. Thesis, Tallin University of Technology, 2005.

[13] S. Lamm, P. Sanders, C. Schulz, D. Strash, R. F. Werneck, Finding near-optimial independent sets at scale, *Proceedings of the 16th Meeting on Algorithm Engineering and Experimentation (ALENEX'16).* 2016. https://doi.org/10.1137/1.9781611974317.12

[14] F. T. Leighton, *A graph coloring algorithm for large scheduling problems,* Journal of Research of National Bureau of Standards 84 (1979), pp. 489–506.

[15] L. Lovász, *On the Shannon capacity of a graph,* IEEE Transactions on Information Theory, Volume 25 Issue 1, January 1979 pp. 1–7. https://doi.org/10.1109/TIT.1979.1055985

[16] P. R. J. Östergård and A. Pöllänen, *New results on tripod packings.* Discrete Comput. Geom. 61 (2019), pp. 271–284. https://doi.org/10.1007/s00454-018-0012-2

[17] C. H. Papadimitriou, *Computational Complexity,* Addison-Wesley Publishing Company, Inc., Reading, MA 1994.

[18] N. J. A. Sloane, *Challenge Problems: Independent Sets in Graphs,* http://neilsloane.com/doc/graphs.html

[19] S. Stahl, *n-Tuple colorings and associated graphs,* Journal of Combinatorial Theory, Series B Volume 20, Issue 2, April 1976, pp. 185–203. https://doi.org/10.1016/0095-8956(76)90010-1

[20] S. Szabó, *Monotonic matrices and clique search in graphs,* Annales Univ. Sci. Budapest., Sect. Comp. 41 (2013), pp. 307–322.

[21] S. Szabó and B. Zaválnij, *Reducing graph coloring to clique search,* Asia Pacific Journal of Mathematics, 1 (2016), pp. 64–85.

[22] S. Szabó and B. Zaválnij, A different approach to maximum clique search. *20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC2018)* pp. 310–316. https://doi.org/10.1109/SYNASC.2018.00055

[23] S. Szabó and B. Zaválnij, *Reducing hypergraph coloring to clique search.* Discrete Applied Mathematics, 264 (2019), pp. 196–207. https://doi.org/10.1016/j.dam.2018.09.010

[24] Q. Wu and J-K. Hao, *A review on algorithms for maximum clique problems,* European Journal of Operational Research. Volume 242, Issue 3, 1 May 2015, pp. 693–709. https://doi.org/10.1016/j.ejor.2014.09.064

Table 1: The test graphs and their upper bounds for $\omega(G)$ from different legal coloring methods and Lovász' theta along with the new bound from the proposed algorithm. We indicated by an * those cases where the program could not reach the result within 100,000 seconds, that is roughly in 24 hours.

	$\|V\|$	density %	$\omega(G)$	legal coloring	$\vartheta(\bar{G})$	time (sec) $\vartheta(\bar{G})$ (12c)	b-fold	DISC	time (sec) DISC (12c)
brock800_1	800	64.93	23	117	*	$>$ 24h	103	**68**	**12717s**
brock800_2	800	65.13	24	118	*	$>$ 24h	103	**68**	**11670s**
brock800_3	800	64.87	25	117	*	$>$ 24h	103	**67**	**15414s**
brock800_4	800	64.97	26	118	*	$>$ 24h	103	**68**	**9849s**
C500.9	500	90.05	$\geq$ 57	139	**84.20**	**109s**	130	115	10235s
C1000.9	1000	90.11	$\geq$ 68	251	**123.49**	**6029s**	232	207	94916s
C2000.5	2000	50.02	16	194	*	$>$ 24h	174	**90**	**68024s**
c-fat500-1	500	3.57	14	14	*	$>$ 24h	14	**14**	**9s**
DSJC500_5	500	50.20	13	58	**22.74**	9960s	51	28	**850s**
DSJC1000_5	1000	50.02	15	107	*	$>$ 24h	92	**49**	**10913s**
hamming10-4	1024	82.89	40	74	*	$>$ 24h	55	**50**	**6227s**
johnson-12-5-4	792	95.58	80	140	**99.00**	**130s**	106	105	4318s
johnson-13-4-4	715	94.96	65	115	**71.50**	**107s**	76	76	1877s
johnson-13-5-4	1287	96.89	$\geq$ 123	212	**143.00**	**792s**	155	154	19485s
keller5	776	75.15	27	31	*	$>$ 24h	31	**31**	**598s**
keller6	3361	81.82	59	64	*	$>$ 24h	64	**63**	**5515s**
MANN_a45	1035	99.63	345	360	356.05	**23s**	360	**353**	29869s
MANN_a81	3321	99.88	1100	1134	1126.62	**589s**	1134	**1121**	91875s
p_hat1000-1	1000	24.48	10	52	*	$>$ 24h	46	**17**	**4041s**
p_hat700-1	700	24.93	11	40	*	$>$ 24h	35	**13**	**977s**
p_hat700-2	700	49.76	44	86	*	$>$ 24h	76	**52**	**31637s**
p_hat700-3	700	74.80	62	129	**72.00**	**11781s**	120	87	73518s
p_hat500-1	500	25.31	9	31	*	$>$ 24h	27	**11**	**376s**
p_hat500-2	500	50.46	36	63	**38.97**	13677s	57	40	**8540s**
p_hat500-3	500	75.19	50	101	**58.57**	**1391s**	92	67	14654s
latin10	900	75.97	90	110	*	$>$ 24h	93	**90**	**5529s**
queen40	1600	91.91	40	40	*	$>$ 24h	40	**40**	**5397s**
queen50	2500	93.49	50	50	*	$>$ 24h	50	**50**	**24986s**
san1000	1000	50.15	15	15	*	$>$ 24h	15	**15**	**2173s**
1dc.512-c	512	92.56	52	73	**53.03**	**76s**	55	54	2007s
1dc.1024-c	1024	95.41	94	137	**95.98**	**1004s**	101	99	9171s
1dc.2048-c	2048	97.22	$\geq$ 172	262	**174.73**	**14395s**	189	186	78627s
1et.1024-c	1024	98.17	171	215	**184.23**	**93s**	191	189	94866s
1et.2048-c	2048	98.93	316	401	**342.03**	**1026s**	356	*	$>$ 24h
1tc.1024-c	1024	98.48	196	227	**206.30**	**64s**	216	212	23475s
1tc.2048-c	2048	99.10	352	420	**374.64**	**753s**	387	*	$>$ 24h
1zc.512-c	512	94.72	62	93	**68.75**	**23s**	72	71	1110s
1zc.1024-c	1024	96.82	$\geq$ 112	180	**128.67**	**295s**	136	134	13793s
2dc.1024-c	1024	67.70	16	31	*	$>$ 24h	21	**18**	**4024s**
2dc.2048-c	2048	75.93	24	54	*	$>$ 24h	37	**32**	**53711s**
monoton-9	729	83.52	28	45	**34.41**	**5465s**	41	37	13670s
monoton-10	1000	85.14	32	61	*	$>$ 24h	51	**46**	**39031s**
monoton-11	1331	86.47	38	69	*	$>$ 24h	63	**57**	**52911s**