University of Szeged

Szegedi Tudományegyetem

Természettudományi és Informatikai Kar

SZTE Informatika Doktori Iskola

Doctoral Theses

The k-Clique Problem

Usage, Modeling Expressivity, Serial and Massively Parallel Algorithms

Zaválnij, Bogdán

Szeged, 2020.

Advisors:
Dr. Krész, Miklós
Dr. Szabó, Sándor

Our thesis work is focused on discrete optimization problems, and specifically on problems represented by graphs. These problems emerge in various applications, and form an interesting subclass of Mathematical Programming. Our thesis concentrates on a special problem of this class, the k-clique problem. We shall in some cases also mention the maximum clique problem as well. As the problems in question belong to NP-complete and NP-hard problem class, these problems are considered to be hard even for medium sized problems. Thus in order to solve them we may want to use more efficient algorithms and more computational power, for example supercomputers. This will lead us to other problems, as dividing the problem to subproblems, scheduling these subproblems, and gathering the results.

There are numerous ways of modeling, and these usually can be freely interchanged. So the decision of choosing the model is rather backed up by the software tool at hand. The most widely used approach is to use a Linear Programming (LP) toolkit, or its variants according the specialty of the problem like Mixed Integer Linear Programming (MILP) or Integer Linear Programming (ILP), or Zero-One Linear Programming (0–1 LP). For combinatorial optimization or decision problems ILP or 0–1 LP is used, but there are other methods, such as Satisfiability (SAT or MaxSAT) or Constraint Programming (CP). While most times any of them can be used there are efficiency differences. These differences caused by two phenomena. First, some problems are more suitable for one model then the other, and so the software solving the problem may be more efficient. Second, some software is simply more advanced then the other, as more developers work on its perfection. Consequently, for any combinatorial optimization problem the first choice is an ILP formulation, given its versatility and easy to model feature and the very developed software. Note though, that this is not necessarily the most efficient approach, and in the case of a harder problem may lead to failure. And there is another problem, which invovles the reliability of the computations. ILP solvers use LP as an auxiliary algorithm, so rounding errors may affect the result [Aki2016]. If one needs reliable computation she or he needs to choose another solver, which is free of such defects, and of course the clique solvers are such using exclusively integer and bit computations. The present work aim to widen the possibility of modeling by showing that modeling by graphs and finding a maximum clique or k-clique of given size is a good approach for solving several problems.

We shall also in detail show that the graph formulation is more suitable for developing a massively parallel algorithm, and so using supercomputers to aid in solution of some hard problems. This task, of efficient parallelization, is usually considered very problematic in combinatorial optimization.

Definition of the problem

Let $G = (V, E)$ be a finite simple graph. Here V is the set of nodes of the graph, and E is a subset of the Cartesian product $V \times V$. The set V is finite and consequently the set E is also finite. The graph does not contain any double edges and the graph does not contain any loops. Of course the graph cannot contain any triple or quadruple edges. The edges are undirected and there are no weights assigned to the nodes nor to the edges.

Consider a subgraph $\Delta = (U, F)$ of G. We say that Δ is a clique in G if $F = U \times U$. In other words Δ is a clique in G if each two distinct nodes of Δ are adjacent in G. The number of nodes of Δ, that is, the size of the set U is called the size of the clique Δ. Instead of saying that Δ is a clique of size k we sometimes say that Δ is a k-clique in the graph G.

Problem 1. *Given a finite simple graph G and given a positive integer k. The task is to decide if G contains a k-clique.*

The k-clique problem is a well known NP-complete problem, and appears 3rd among Karp's original 21 NP-complete problems [Karp1972].

Theses

1st thesis

In our work we showed that several problems can be modeled by graphs and solved using a k-clique solver. These problems arise from various fields. First, we showed reformulation for Latin square, Sudoku game, the problem of non attacking queens, Costas Arrays and combinatorial problems arising from coding theory. Second, we detailed a group of real life problems that connected to subgraph isomorphism, like protein docking, molecule search, fingerprint and image recognition. Third, we detailed the reformulation for scheduling problems, namely open shop, flow shop and job shop problems. Finally, we described how clique search can be of use in network analysis using market graphs or brain graphs.

In a few cases we did also some numerical experiments. The goal in that was not to beat other methods, but to show that these reformulations can solve non trivial problems of these kind in comparable manner then other methods and solvers.

We picked one problem to show our results, the Costas array problem. The proposed representation – to our knowledge – is not known in the literature. We propose a graph representation to the Costas array problem

[Cost1965]. This problem derives from radar and sonar technology, namely phased array radar engineering. The solution helps generating radar and sonar signals with ideal ambiguity functions. The formalization of the problem as follows. Given an $n \times n$ array one needs to fill it in with n dots. The constrain is that no two dots may lay on the same row or column, and the displacement vectors of any two pairs of dots must be distinct from all the other such displacement vectors.

As the constrain of different displacement vectors instruct us about pairs of dots (4 coordinates altogether in one pair), the nodes of the auxiliary graph $G(V, E)$ shall be denoted by quadruples (x_1, y_1, x_2, y_2), where the coordinates of the represented dot-pair are (x_1, y_1) and (x_2, y_2). One could list all possibilities, but clearly only pairs of agreeable nodes needed to be listed, thus the rows and columns of the two dots must be different. That means that we omit for example the quadruple $(1, 1, 1, 3)$, because the two dots lie in the same column. The size of this graph is $n^2(n-1)^2/2$. The edges of the graph represent the agreement between two dot-pairs. We need to take special care of the case when one of the dots of a pair coincides with a dot from the other dot-pair. In this case we have 3 instead of 4 distinct dots.

Formally there should be *no* edge between two nodes (x_1, y_1, x_2, y_2) and (a_1, b_1, a_2, b_2), iff:

1. If two pairs of dots have same row or column;
2. If one pair of dots have same row or column and there are 4 distinct nodes;
3. If one pair of dots have same row or column and there are 3 distinct nodes, and some of the displacement vectors of these three are the same;
4. If no pair of dots have same row or column and there are 4 distinct nodes, and some of the displacement vectors of these four are the same.

In other cases there will be an edge between these two nodes. A k-clique of size $k = n(n-1)/2$ represents all pairs from n dots, and thus it is a solution to the problem in question.

With the help of this simple formulation – even without using any kernelization techniques – one can find one solution of the non trivial 14×14 array in few seconds and calculate all possible solutions in half an hour.

2^{nd} thesis

We choose one problem class for extended demonstration of modeling power of graphs and k-cliques, namely various graph coloring problems. We detailed reformulation for the problem of k coloring of the nodes of a graph, 3-clique free coloring and coloring of hypergraphs. With the aid of the last one we could solve some hard hypergraph coloring problem and aid an open question by Voloshin. The question asks about colorability of hypergraph having C edges – nodes cannot receive all different colors –, and D edges – nodes cannot receive all the same color. The construction is interesting, because there are hypergraphs that cannot be colored at all. We performed a series of calculation from witch one can conclude about probability of edge C and D where this transition from colorable to uncolorable happens.

As an example we picked the reformulation of 3-clique free coloring [Szab2012] to k-clique. The definition of this coloring as follows:

1. Each node of G receives exactly one color.
2. The three nodes of a 3-clique in G cannot receive the same color.

This problem can be reduced to Problem 1. Starting with the the graph $G = (V, E)$ and the positive integer k we construct an auxiliary graph $\Gamma = (W, F)$. The nodes of Γ are the triples

$$(\{u, v\}, a, b), \quad \text{where} \quad \{u, v\} \in E,\ 1 \leq a, b, \leq k.$$

Let m be the number of edges of G, that is, let $m = |E|$. The number of the triples is equal to mk^2.

The triple $(\{u, v\}, a, b)$ intends to code the information that the end points u, v of the edge $\{u, v\}$ are colored with the colors a, b respectively. In this section we assume that each node of the graph G is end point of some edge of G. In other words we assume that the graph G does not contain isolated nodes.

Let us consider two distinct nodes

$$w_1 = (\{u_1, v_1\}, a_1, b_1) \quad \text{and} \quad w_2 = (\{u_2, v_2\}, a_2, b_2)$$

of Γ. Set

$$X = \{u_1, v_1\} \cup \{u_2, v_2\} = \{u_1, v_1, u_2, v_2\}.$$

It is clear that $|X| \leq 4$ and since $u_1 \neq v_1$ we get that $|X| \geq 2$. Thus $2 \leq |X| \leq 4$. Let H_X be the subgraph of G induced by X. The nodes u_1, v_1, u_2, v_2 receive the colors a_1, b_1, a_2, b_2, respectively in the graph H_X.

When $|X| \leq 3$, then these nodes are not pair-wise distinct and it may happen that two distinct colors are assigned to a node in H_X. In this case we call the graph H_X a non-qualifying graph.

It also may happen that there is a 3-clique in H_X and all the three nodes of this 3-clique receive the same color. In this situation again we call the graph H_X a non-qualifying graph. In all the other cases H_X is called a qualifying graph.

When we construct the graph Γ we connect the nodes w_1, w_2 by an edge in Γ if H_X is a qualifying graph.

Observation 1. *If the nodes of G have a 3-clique free coloring with k colors, then the graph Γ contains an m-clique.*

Proof. Suppose that the nodes of the graph G have a 3-clique free coloring using k colors. Let $f : V \to \{1, \ldots, k\}$ be a function that codes this coloring. Set

$$D = \{(\{u,v\}, f(u), f(v)) : \ \{u,v\} \in E\}$$

and let Δ be the subgraph of Γ induced by D. It is clear that $|D| = m$. We claim that Δ is a clique in Γ.

In order to verify the claim let us choose two distinct nodes w_1, w_2 from D. Let us consider the subgraph H_X associated with w_1, w_2. Since f is a function, each node of H_X receives exactly one color. As f describes a 3-clique free coloring of the nodes of G, it follows that the restriction of f to the nodes of H_X is a 3-clique free coloring of the nodes of H_X. Thus H_X is a qualifying graph. Consequently, we connected w_1, w_2 by an edge in Γ when we constructed Γ. $\square$

Observation 2. *If the auxiliary graph Γ contains an m-clique, then the nodes of the graph G have a 3-clique free coloring with k colors.*

Proof. Suppose that Γ contains an m-clique Δ and D is the set of nodes of Δ. Now $|D| = m$.

Set

$$I_{\{u,v\}} = \{(\{u,v\}, a, b) : \ 1 \leq a, b, \leq k\}$$

for each $\{u,v\} \in E$. Obviously, $|I_{\{u,v\}}| = k^2$. Note that the sets $I_{\{u,v\}}$, $\{u,v\} \in E$ are pair-wise disjoint independent sets in Γ.

Indeed, if

$$w_1 = (\{u,v\}, a_1, b_1) \quad \text{and} \quad w_2 = (\{u,v\}, a_2, b_2)$$

are distinct elements of $I_{\{u,v\}}$, then the graph H_X associated with w_1, w_2 has two nodes. From $w_1 \neq w_2$ it follows that $a_1 = a_2$, $b_1 = b_2$ cannot hold. Thus

H_X is not qualifying. This means when we constructed Γ we did not connect w_1, w_2 by an edge in Γ.

The nodes of Γ have a legal coloring using m colors. The independent sets $I_{\{u,v\}}$, $\{u,v\} \in E$ can play the roles of the color classes.

As Δ is a clique in Γ each color class contains at most one element from D. Using the cardinality of D we can conclude that D is a complete set of representatives of the color classes.

Set

$$T = \{\{u,v\} : (\{u,v\}, a, b) \in D\}.$$

It follows that $E = T$. Consequently, each node of G which is an end point of at least one edge of G receives at least one color. We claim that each node receives exactly one color.

In order to prove the claim assume on the contrary that more than one colors are assigned to a node of G. In this case there are distinct nodes w_1, w_2 of Δ such that a node receives more than one color in the subgraph H_X associated with w_1, w_2. This means that H_X is not qualifying. On the other hand when we constructed Γ we connected w_1, w_2 by an edge on the base that the subgraph H_X was qualifying.

We may summarize our consideration by saying that we can define a function $f : V \to \{1, \ldots, k\}$ by setting $f(u) = b$ whenever $(\{u,v\}, a, b)$ is a node of Δ. It remains to show that the coloring of the nodes of G described by the function f is a 3-clique free coloring.

Suppose there is a 3-clique Ω in G whose nodes receive the same color. There are distinct nodes w_1, w_2 of Δ such that Ω is a 3-clique in the subgraph H_X associated with w_1, w_2. This means that H_X is not qualifying. On the other hand when we constructed Γ we connected w_1, w_2 by an edge in Γ because the subgraph H_X was qualifying. □

Theorem 1. *Given a graph G with m edges and an integer k and a auxiliary graph Γ described above. There is a 3-free coloring of nodes of G with k colors iff there is an m-clique in Γ.*

Proof. Follows from Observation 1 and 2. □

3rd thesis

After listing different sequential maximum clique solvers we look for the most helpful auxiliary algorithms that can help us to solve these problems. First, we compared algorithms that produce upper bounds on clique size, and as such can be used inside a maximum or k-clique solver as a bounding or cutting function. Our measurements help to see how good is the produced bound and how long it takes to calculate it. Second, we showed some kernelization steps that can help in reducing the size of the graph.

Following auxiliary algorithms were listed as upper bounds for clique size:

- **Coloring the nodes of a graph.** As the problem itself NP-hard we used two heuristic approach, DSatur by Brélaz [Brél1979] and the iterative coloring heuristic from Culberson [Culb1992]. The bound is $\chi(G) \geq \omega(G)$.

- ***b*-fold coloring.** An assignment of a set of b colors to every one of its vertices such that adjacent vertices receive disjoint sets. The bound is $\omega(G) \leq \frac{\chi_b(G)}{b}$. We used a reformulation as a legal node coloring of a graph by using an auxiliary graph $\Gamma = (W, F)$ constructed from the given $G = (V, E)$ graph. The nodes of Γ are ordered pairs $(v_i, k) \in W, v_i \in V, 1 \leq k \leq b$. The edges are defined as follows.

$$F = \begin{cases} \{(v_i, k), (v_i, l)\} & \text{if } k \neq l \quad\quad 1 \leq k, l \leq b \\ \{(v_i, k), (v_j, l)\} & \text{if } \{v_i, v_j\} \in E \quad 1 \leq k, l \leq b \end{cases} \tag{1}$$

- **Edge coloring.** An assignment of color numbers to the edges of G [Szab2014c] such that:

  1. Each edge of G receives exactly one color.
  2. If x, y, z are distinct nodes of a 3-clique in G, then the edges $\{x, y\}$, $\{y, z\}$, $\{x, z\}$ must receive three distinct colors.
  3. If x, y, u, v are distinct nodes of a 4-clique in G, then the edges $\{x, y\}$, $\{x, u\}$, $\{x, v\}$, $\{y, u\}$, $\{y, v\}$, $\{u, v\}$ must receive six distinct colors.

  Let Δ be an l-clique in a graph G, and let G be edge-colorable with k colors. Then $l(l-1)/2 \leq k$ holds. The procedure to color the edges of a graph G is to use an auxiliary graph $\Gamma = (W, F)$. Each edge of G is represented by a node in Γ. We connect the nodes of Γ according the rules above, that is two nodes in Γ should be connected if the corresponding edges in G forming a 3- or a 4-clique. It is easy to see that any legal coloring of the nodes of Γ represents a legal edge coloring of G.

- **Lovász number.** A real number that is an upper bound on the Shannon capacity of the graph [Lov1976].

- **The partial MaxSAT bound,** from [Li2010a].

We listed kernelization techniques, such as:

- **Struction,** from [Ebe1984].
- **Color indices.** When we can delete a node (or an edge), if the neighborhood if that node (or edge) cannot be colored by $(k-1)$ (or $(k-2)$) colors.
- **Node dominance.** Let G be a graph and let a, b be distinct nodes of G. Node b dominates node a if a and b are not adjacent and $N(a) \subseteq N(b)$, and the node a can be canceled from G when we are deciding if G contains a k-clique.
- **Edge dominance.** Edge $\{u, v\}$ dominates edge $\{x, y\}$ if $\{u, x\}$ or $\{u, y\}$ is not edge of G, and $\{v, x\}$ or $\{v, y\}$ is not edge of G, and $N(x) \cap N(y) \subseteq N(u) \cap N(v)$. As a consequence, the edge $\{x, y\}$ can be canceled from G when we are deciding if G contains a k-clique.

4th thesis

We designed and implemented a specialized k-clique search program, and detailed its building blocks. Using this program we built a maximum clique solver, which could be compared to other state-of-the art solvers. Our approach proved successful, and we could show that our program is among the best ones. The main building blocks of our algorithm as follows.

Branching and Bounding

It is well known, that the coloring gives us an upper limit for the clique size. Thus if given the value of k of the searched cliquek size and a coloring with c colors $(c \geq k)$, then we can choose the smallest $c-(k-1)$ color classes, and use those nodes in them for branching – as a *branching rule.* As a terminology we call these nodes the k-clique covering node set. The importance of this comes from the nature of the Branch and Bound algorithms. These algorithms sort out the nodes already examined, meaning that they are not taken into account in the future search. Thus if all these nodes are eliminated, then the remaining nodes can be colored with $(k-1)$ colors, so there cannot be any k-clique present. Note, that without the value of k one cannot make this branching rule, and need to branch on all nodes. The size of the k-clique covering node set is the branching factor, and finding the smallest possible of such set can aid us in bounding the size of our search tree.

Efficient coloring

It is well known from the literature, that coloring of the nodes can speed-up the clique search. Using Dsatur at all levels reduces the size of the search tree, but costs in time. An efficient algorithm can use a costly DSatur [Brél1979] coloring at the top of the search tree, and later it can switch to the cheaper sequential greedy coloring. Although Dsatur gives us a good coloring it is often quite far from the optimum. So we used in addition another technique, the Iterated Coloring presented by Culberson [Culb1992]. This technique uses reordering the color classes and using a sequential coloring several times. The result cannot be worse than the previous coloring in terms of the number of colors, but it can be better. Thus we started from a Dsatur coloring and performed iterated coloring. Our stopping criteria was if the number of colors did nod decreased after 1000 iterations, and we used it on the top of the search tree.

Recoloring the nodes

So during the Branch and Bound procedure, when there are less and less nodes as we go down on the search tree, we can use the coloring of the previous level, and use the repacking feature of the sequential greedy coloring. We sort the color classes by their size, and start a greedy sequential coloring from the biggest color class. As the k-clique covering node set is actually the set of smallest color classes, the nodes from them moved ahead to the bigger color classes. So this procedure directly reduces the size of the k-clique covering node set and so the branching factor. As this coloring is performed on each node of the search tree, we needed a fast implementation of the sequential greedy coloring. Our original version was of $O(c|V|)$ for c colors, and proved quite fast. Later we implemented this coloring using bitsets.

Rearranging branching nodes

From previous results [Zav2014a, Zav2014b, Zav2015] on parallel clique search algorithms we concluded, that the branching is even more important than it was thought before. It seems that the sequence of the nodes by which we proceed in the branch has a big effect on the search tree size if pruning is present. We use a very basic reordering rule. We proceed with the nodes with the smallest degree in the remaining subgraph. That is we ordered the nodes by node degree in *increasing* order. By doing this we solve first the more easy problems and reduce the size of the later ones. Although simple and “cheap” this approach had quite a good effect on the size of the search tree.

5th thesis

We introduced the concept of disturbing structures, which helped us to build a k-clique search algorithm, and helps on parallelizing such algorithms.

Let $G = (V, E)$ be a finite simple graph and let k be a positive integer. Let $W \subseteq V$. If each k-clique in G has at least one node in W, then we call W a k-clique covering node set of G. Let $W = \{w_1, w_2, \ldots, w_n\}$ be a k-clique covering node set in G. Consider the subgraph H_i of G denote the graph induced by the neighbor set $N(w_i)$ in G for each i, $1 \leq i \leq n$. Let Δ be a k-clique in G. The definition of W states that w_i must be a node of Δ for some i, $1 \leq i \leq n$. Consequently, the subgraph H_i contains exactly $k - 1$ nodes of the clique Δ. This observation has a clear intuitive meaning: the problem of determining the existence of a k-clique in G can be reduced to a list of smaller problems of determining the existence of a $(k - 1)$-clique in the subgraphs H_i for each i, $1 \leq i \leq n$.

We can extend this result. Let $G = (V, E)$ be a finite simple graph and let k be a positive integer. Let F be a subset of all s-cliques in G. If each k-clique in G has at least one s-clique in F, then we call F a *k-clique covering s-clique set* of G. In particular, when $s = 2$, then F is an *k-clique covering edge set.* Let F be an s-clique cover of all the k-cliques in G, and let

$$c_i = \{u_{i,1}, u_{i,2}, \ldots, u_{i,s}\}, 1 \leq i \leq |F|$$

be all s-cliques in F. Let H_i be the subgraphs spanned by the sets of nodes

$$H_i = \bigcap_j N(u_{i,j}) \quad 1 \leq j \leq s, \tag{2}$$

and let Δ be a k-clique in G. According to the definition of F, there must be a c_i that is an s-clique of Δ for some i, $1 \leq i \leq |F|$. Consequently, the subgraph H_i contains exactly $k - s$ nodes of Δ. This observation has a clear intuitive meaning: the problem of determining the existence of a k-clique in G can be reduced to determining the existence of a $(k - s)$-clique in a series of graphs spanned by each of the subgraphs H_i, $1 \leq i \leq |F|$.

6th thesis

We implemented a massively parallel program for k-clique search. We examined the importance of the deletion sequence, that is the ordering of subproblems. Driven by this we constructed the so called Las Vegas method of parallelization, which helps us find such ordering that can reduce the search place. We also could show that the running times for different subproblems may differ in 4 magnitudes, and this is exactly where the Las Vegas method and solver instance restarting helps. We compared three methods:

1. The *a priory* deletion sequence, where we list edges in a fixed way, and delete the edges representing an earlier (in the sequence) subproblem from a later (in the sequence) problem.
2. The Las Vegas method, where we do not delete any edge from the subproblems. We start the solvers in parallel manner, and if a subproblem finished we delete the edge representing this subproblem from all subproblems – either waiting for starting to be solved either if a solver presently working on it.
3. The Las Vegas method with restarting is at first the same as the previous one. The difference comes into play at the very end, when we solved most of the subproblems, and the remaining (running) subproblems are less then the number of processing elements. In this case we restart a subproblem – keeping the same subproblem being solved at the same time –, so there will be two instances of solvers solving the same subproblem. One have the advantage of being half through solving it, the other have the advantage of being restarted, and so having a better preconditioning.

We examined closely one problem, the `monoton-9`. On Figure 1 we reordered the problems by the magnitude of the solving times. The time scale is logarithmic, so the actual differences are of several magnitudes. In this graph one can see, that all three algorithms are dominated by the longest subproblem, although the restarting Las Vegas can smooth out this problem the best, reducing the variance of running times by more then 2 magnitudes.

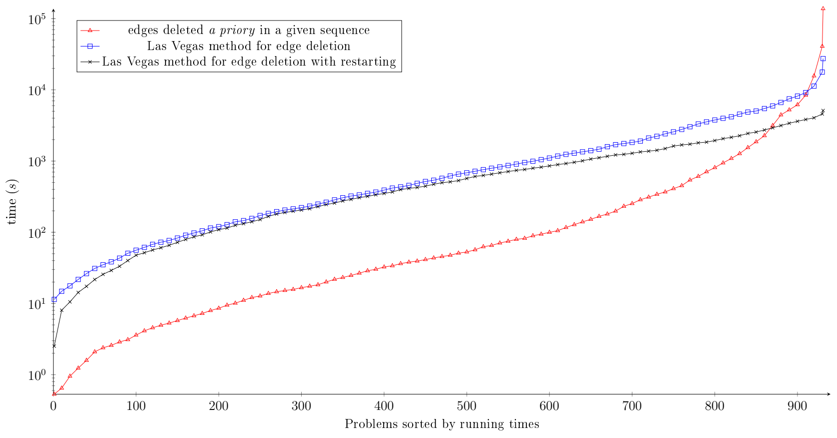

Figure 1: The sorted running times of the `monoton-9` subproblems.

Author's own results

1. The formulation of the Costas Array problem by graphs in 1st thesis and in Section 2.1.4 is my own result. It is not yet published.
2. The formulation of the flow shop, open shop and job shop problems in 1st thesis and in Section 2.3 is the joint work with Sandor Szabo. It is not yet published.
3. The k-clique approach of the 3-free coloring in 2nd thesis and in Section 3.2 is my own result. It was published in [Szab2016b].
4. The k-clique approach to hypergraph coloring including its application to Voloshin's problem in 2nd thesis and in Section 3.3 is my own result. It was published in [Szab2019b].
5. Measurements and comparison of different upper bounds in 3rd thesis and in Section 4.2, including sequential and parallel implementation of DSatur and Iterated Coloring algorithms, and also its application to edge and b-fold coloring is my own result. It was published in [Szab2017, Marg2019].
6. The sequential k-clique program in 4th thesis and in Chapter 5 was first developed by Sandor Szabo and later extendedly developed by myself including addition of Culberson's Iterative Coloring, node rearrangement, parallelization by bitsets and transformation to a maximum clique search algorithm. It was published in [Szab2018a].
7. The theoretical background of parallelization which we call "disturbing structures" in 5th thesis and in Chapter 7 is mostly my own result. It was published in [Szab2018b].
8. The parallel algorithm and program detailed in 6th thesis and in Chapter 8 is based on the idea from Sandor Szabo, while the Las Vegas approach and the measurements proving its effectiveness was done by myself. It was published in [Zav2014a, Zav2014b, Zav2015].

References

[Aki2016] Akiba, T. and Iwata, Y. “Branch-and-reduce exponential/fpt algorithms in practice: A case study of vertex cover.” *Theoretical Computer Science.* 609:211–225, 2016.

[Brél1979] Brélaz, D. “New Methods to Color the Vertices of a Graph.” *Communications of the ACM.* 1979. 22. 4. pp. 251–257.

[Cost1965] Costas, J.P. *Medium constraints on sonar design and performance.* Class 1 Report R65EMH33, G.E. Corporation. 1965.

[Culb1992] Culberson, J.C. *Iterated Greedy Graph Coloring and the Difficulty Landscape.* Technical Rep. University of Alberta. 1992.

[Ebe1984] Ebenegger, Ch. Hammer, P.L. and de Werra, D. “Pseudo-boolean functions and stability of graphs.” *North-Holland mathematics studies.* volume 95, pp. 83–97. 1984.

[Karp1972] Karp, R.M. *Reducibility Among Combinatorial Problems.* In: Complexity of Computer Computations. Eds: R. E. Miller and J. W. Thatcher. New York. Plenum. pp. 85–103.

[Li2010a] Li, C.-M. and Quan, Z. *An Efficient Branch-and-Bound Algorithm Based on MaxSAT for the Maximum Clique Problem.* In: Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence. (AAAI-10), pp. 128–133.

[Lov1976] Lovasz L. “On the Shannon Capacity of a Graph,” *IEEE Transactions on Information Theory.* 25, 1, 1979. pp. 1–7.

[Marg2019] Margenov, S. et al. *Applications for ultrascale systems.* In: Ultrascale Computing Systems. Institution of Engineering and Technology (IET), London, (2019) pp. 189–244.

[Szab2012] Szabó S. and Zaválnij B. “Greedy Algorithms for Triangle Free Coloring.” *AKCE Int. J. Graphs Comb.,* 9, No. 2 (2012), pp. 169–186.

[Szab2014a] Szabó S. and Zaválnij B. “Coloring the edges of a directed graph.” *Indian Journal of Pure and Applied Mathematics.* April 2014, Volume 45, Issue 2, pp 239–260.

[Szab2014b] Szabó S. and Zaválnij B. “Coloring the nodes of a directed graph.” *Acta Univ. Sapientiae, Informatica.* 6, 1 (2014) 117—131.

[Szab2014c] Szabó S. and Zaválnij B. “Estimating clique size via coloring edges in graphs.” *AKCE Int. J. Graphs Comb.* (submitted.)

[Szab2016b] Szabó S. and Zaválnij B. “Reducing Graph Coloring to Clique Search,” *Asia Pacific Journal of Mathematics*, 3 (2016), pp. 64–85.

[Szab2017] San Segundo, P., Szabó S. and Zaválnij B. “Parallelization of the clique search problem using sub-chromatic bounds.” NESUS. Technical Report. January 17, 2017.

[Szab2018a] Szabo S. and Zavalnij B. *A different approach to maximum clique search.* In: 20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing. (SYNASC2018) IEEE Proceedings, (2018) pp. 310–316.

[Szab2018b] Szabo S. and Zavalnij B. “Decomposing clique search problems into smaller instances based on node and edge colorings.” *Discrete Applied Mathematics.* 242 (2018) pp. 118–129.

[Szab2019a] Szabo S. and Zavalnij B. “Benchmark Problems for Exhaustive Exact Maximum Clique Search Algorithms.” *Informatica (Ljubljana).* 43 : 2 (2019) pp. 177–186.

[Szab2019b] Szabo S. and Zavalnij B. “Reducing hypergraph coloring to clique search.” *Discrete Applied Mathematics.* 264. (2019) pp. 196–207.

[Zav2014a] Zaválnij B. “Three Versions of Clique Search Parallelization.” *Journal of Computer Science and Information Technology.* Vol. 2:(No. 2) pp. 9–20. (2014)

[Zav2014b] Zavalnij, B. *The Las Vegas method of parallelization.* In: Information Society 2014 – IS 2014: Volume A; Intelligent Systems. pp. 105–108. 2014.

[Zav2015] Zavalnij, B. *Speeding up Parallel Combinatorial Optimization Algorithms with Las Vegas Method.* In: 10th International Conference on Large-Scale Scientific Computations. Lecture Notes in Computer Science (LNCS) 2015.