MATCH
Communications in Mathematical and in Computer Chemistry

MATCH Commun. Math. Comput. Chem. **84** (2020) 7-28

ISSN 0340 - 6253

An Improved Maximum Common Induced Subgraph Solver

Matjaž Depolli[1] , Sándor Szabó[2] , Bogdán Zaválnij[3]

[1] *Jožef Stefan Institute, Ljubljana, Slovenija*

[2] *University of Pécs, Hungary*

[3] *Alfréd Rényi Institute of Mathematics, Hungary*

(Received December 27, 2019)

Abstract

One of the traditional ways of solving the maximum common induced subgraph problem is by reduction to a maximum clique problem. The two input graphs, in which the maximum common subgraph is to be found, are multiplied to form a product graph, which is then input to the maximum clique algorithm. The result of the latter are used to identify the nodes of the input graphs that form the maximum common subgraph. Although the maximum clique problem can be solved by a modern branch-and-bound based algorithm for general graphs, such approach is far from optimal. Some special properties of the product graph can be exploited to guide the maximum clique search. Namely, the modern state-of-the-art clique search programs use coloring as auxiliary algorithm, but finding a good coloring of a graph itself a hard task. In this article, we use the exploit the nature of the product graph to provide the maximum clique algorithm with a very good initial coloring or multiple such colorings. We perform experiments on a large database of small to medium sized graphs and demonstrate the efficiency of the proposed method against a state of the art method for solving maximum common subgraph problem.

1 Introduction

1.1 Background

Given two finite simple graphs G and H. The mathematical problem we consider is to decide if G has an induced subgraph G', which is isomorphic to H. The problem called as isomorphic embedding problem as H can be isomorphically injected into G. This problem

seems to be a pure abstract mathematical problem. However, its relevance to chemical computations is a well established, well documented fact. As a matter of fact the chemical importance of the problem is the main motivation behind, to find efficient and practical algorithm to the problem. We say confidently that the chemical applications are the real motivation behind the definition of this present abstract mathematical problem.

A more general problem formulation of the above is the maximum common induced subgraph problem. In this problem we are looking for an induced subgraph G' of G, and an induced subgraph H' of H, where G' is isomorphic to H'. We are interested to find big subgraphs to optimality. The maximum common induced subgraph problem is used in chemistry as a means of comparing shapes of molecules, either as 3D scans or molecular graphs, which represent the structural formula directly [1, 2]. An example of such use is in prediction of protein function. The characteristic of proteins, which allows them to function within an organism, is their ability to bind other molecules. They bind to other molecules similarly as jigsaw puzzle fit together, by matching their shape to the shape of target molecule. The function of unknown protein can therefore be estimated by comparing its shape to shapes of known proteins with known functions.

One of the traditional ways of solving the maximum common subgraph problem is by reduction to a maximum clique problem, using auxiliary product graph. The two input graphs, in which the maximum common subgraph is to be found, are multiplied to form a product graph, which is then input to the maximum clique algorithm. The result of the latter are used to identify the nodes of the input graphs that form the maximum common subgraph. Although the maximum clique problem can be solved by a modern branch-and-bound based algorithm for general graphs, such approach is far from optimal. Some special properties of the product graph can be exploited to guide the maximum clique search. Namely, the modern state-of-the-art clique search programs use coloring as auxiliary algorithm, but finding a good coloring of a graph itself a hard task.

The product graph method mentioned above is a versatile method and can be adjusted to different scenarios. As an example, we must point out, that the graphs representing chemical molecules as a rule have labels assigned to nodes and sometimes to edges representing different atoms and bond types. One can produce the product graph just the same way only by limiting the possible pairs of labels, that is atoms and edges, that is the bonds. We will detail this difference when describing the product graph later.

In this article, we exploit the properties of the auxiliary product graph to provide the maximum clique algorithm with a set of very good initial colorings. We call these colorings *hereditary*. We perform experiments on a database of small to medium sized graphs and demonstrate the efficiency of the proposed method against a state of the art method for solving maximum common subgraph problem.

There are indications that when using these colorings, solving some of currently hard problems would become feasible.

Definition 1 *Let $G = (V, E)$ be a finite simple graph. Let D be a subset of V and let Δ be the subgraph of G induced by D. The subgraph Δ is called a clique in G if any two distinct elements of D are adjacent in G. If the set D has k elements, then we call Δ a k-clique in G. A k-clique in G called maximum clique, if there is no clique of size $k+1$ in G. The size of a maximum clique in G is referred as the clique number of G, and it is denoted by $\omega(G)$.*

Finding cliques in a given graph is an important problem in discrete applied mathematics with many applications inside and outside of mathematics. For further details see [3–8].

We formally state the following two clique search problem.

Problem 1 *Given a finite simple graph G and given a positive integer k. Decide if G contains a k-clique.*

Problem 2 *Given a finite simple graph G find the biggest positive integer k, such that G contains a k-clique.*

Many practical clique search algorithms employ coloring to speed up the computation by reducing the search space. Finding optimal or nearly optimal colorings is itself a computationally demanding problem. For this reason, the computationally less demanding greedy algorithms are used to construct suboptimal colorings in the above computations. It is customary to color the nodes of a graph G satisfying the following conditions.

- Each node of G receives exactly one color.
- Adjacent nodes in G cannot receive the same color.

This is the most commonly encountered coloring of the nodes of a graph and it is referred to as legal coloring of the nodes. It is well known that coloring can be used for

estimating clique size. For each finite simple graph G there is a well defined nonnegative integer k such that the nodes of G admit a legal coloring with k colors but the nodes of G cannot be colored legally using $k-1$ colors. This number k is called the chromatic number of the graph G and it is denoted by $\chi(G)$. Let us suppose that Δ is an l-clique in G and let us suppose that the nodes of G have a legal coloring with k colors. Then $l \leq k$ holds.

The two problems related to coloring can be formally stated as follows.

Problem 3 *Given a finite simple graph G and given a positive integer k. Decide if the nodes of G have a legal coloring using k colors.*

Problem 4 *Given a finite simple graph G find the biggest positive integer k such as the nodes of G have a legal coloring using k colors.*

Problems 1 and 3 are decision problems. From the complexity theory of computations we know that these problems belong to the NP-complete complexity class. Problems 2 and 4 are optimization problems and it can be easily seen that they are computationally as challenging as the previous NP-complete problems. That means that these problems belong to the NP-hard complexity class.

Definition 2 *Let $G = (V, E)$ be a finite simple graph and let s be a positive integer such that $s \geq 2$. A subset U of V is called an s-free set if the graph induced by U in G does not contain any s-clique. A partition $U_1, \ldots, U_r$ of V is called an s-clique free partition of V if U_i is an s-clique free subset of V for each i, $1 \leq i \leq r$.*

We can look at this partitioning as basis for an alternative coloring method and we call it s-clique free coloring [9].

1.2 Product graphs and their colorings

In this paper we are interested in some special problems. We assume that these problems can be reduced to k-clique search or maximum clique search on the given product graph. The problems of graph isomorphism and induced subgraph isomorphism are representatives of this type of problems so we will use the induced subgraph isomorphism problem to illustrate the method. Other problems can be dealt with by similar means. Induced subgraph isomorphism has important applications for example in drug design,

chemical database problems, artificial intelligence or pattern recognition. Let us state the maximum common induced subgraph isomorphism problem more formally:

Problem 5 *Let $G = (V, E)$, $G' = (V', E')$ be finite simple graphs. We are looking for the induced subgraph G_0 in G and induced subgraph G'_0 in G' such that G_0 is isomorphic to G'_0. In other words, is there a $G_0 = (V_0, E_0)$: $V_0 \subseteq V$ where v_1, $v_2 \in V_0$ and $\{v_1, v_2\} \in E$ then and only then $\{v_1, v_2\} \in E_0$ and is there a $G'_0 = (V'_0, E'_0)$: $V'_0 \subseteq V'$ where v_1, $v_2 \in V'_0$ and $\{v_1, v_2\} \in E'$ then and only then $\{v_1, v_2\} \in E'_0$ such that $G_0 \cong G'_0$? The problem is to find the biggest possible number $k = |V_0|$ and the corresponding subgraph G_0 fulfilling this property.*

A special version of the above problem occurs when G_0 is isomorphic to the G' itself. We will call this problem as the isomorphic embedding problem to distinguish it from the previous problem.

Problem 6 *Let $G = (V, E)$, $G' = (V', E')$ be finite simple graphs. Is there a induced subgraph G_0 in G such that G_0 is isomorphic to G'. In other words is there a $G_0 = (V_0, E_0)$: $V_0 \subseteq V$ where v_1, $v_2 \in V_0$ and $\{v_1, v_2\} \in E$ then and only then $\{v_1, v_2\} \in E_0$ such that $G_0 \cong G'$?*

A possible method of solving these problems is to construct an auxiliary graph $\Gamma = (W, F)$ where $|W| = |V||V'|$. The nodes of the graph Γ are labeled by ordered pairs of nodes from G and G'. That is if a $a_1 \in V$, $b_1 \in V'$ then $(a_1, b_1) \in W$. The edges of the graph Γ are constructed as follows. Let us consider (a_1, b_1), (a_2, b_2) as two distinct nodes of Γ. We put an edge between them if $\{a_1, a_2\} \in E$ and $\{b_1, b_2\} \in E'$. We also put and edge between them if $\{a_1, a_2\} \notin E$ and $\{b_1, b_2\} \notin E'$ for $a_1 \neq a_2$, $b_1 \neq b_2$. This means that a_1, a_2 and b_1, b_2 both should be connected or both should not be connected. A k-clique where $k = |V_0|$ in the graph Γ represents the function $f : V_0 \to V'_0$ such that $b_1 = f(a_1), b_2 = f(a_2), \{a_1, a_2\} \in E \Leftrightarrow \{f(a_1), f(a_2)\} \in E'$. This means that finding the maximum clique in Γ will solve the maximum common induced subgraph isomorphism problem, and finding a k-clique in Γ, where $k = |V'|$ will solve the isomorphic embedding problem.

In case of labeled graph we limit the nodes of $\Gamma = (W, F)$. Instead of using all possible pairs of input nodes, that is $V \times V'$, we use only those pairs where the labels are the same.

If we have label functions $l_n : V \to \mathbb{N}$, $l'_n : V' \to \mathbb{N}$ and $a_1 \in V$, $b_1 \in V'$, $l_n(a_1) = l'_n(b_1)$ then $(a_1, b_1) \in W$. Also, we consider labeling for edges if one is present. Consider that the labeling of edges are $l_e : E \to \mathbb{N}, l'_e : E' \to \mathbb{N}$. We put an edge between two distinct nodes of Γ (a_1, b_1), (a_2, b_2) if in both cases there is an edge with the same label, that is $\{a_1, a_2\} \in E$ and $\{b_1, b_2\} \in E'$ and $l(\{a_1, a_2\}) = l'(\{b_1, b_2\})$. We also put and edge between them if there is no edge present in both cases, that is $\{a_1, a_2\} \notin E$ and $\{b_1, b_2\} \notin E'$ for $a_1 \neq a_2$, $b_1 \neq b_2$.

It is well known, that the k-clique and maximum clique search algorithm can be sped up by using a good coloring of the given graph. We will describe several coloring schemes in the following subsections. We called these colorings *hereditary* because of the fact that they derive solely from the two input graphs and the constructing method of the auxiliary product graph.

1.3 First hereditary coloring scheme

Note that the nodes $(a_1, b_1), (a_1, b_2), (a_1, b_3), \ldots, (a_1, b_{|V'|})$ of the graph Γ form an independent set. Intuitively this means that the node a_1 can only be paired with one of the nodes from V' at the same time. Thus we can define $|V|$ number of color classes in Γ where the nodes labeled by the same node from G fall into one color class. That is, the first color class will consist of nodes $(a_1, b_1), (a_1, b_2), (a_1, b_3), \ldots$, the second will consist of nodes (a_2, b_1), (a_2, b_2), $(a_2, b_3), \ldots$, the third of nodes $(a_3, b_1), (a_3, b_2), (a_3, b_3), \ldots$, and so on.

Similarly, the nodes $(a_1, b_1), (a_2, b_1), (a_3, b_1), \ldots, (a_{|V|}, b_1)$ of the graph Γ form an independent set. Thus we can define $|V'|$ number of color classes in Γ where the nodes labeled by the same node from G' fall into one color class.

Note that if there is a solution to the isomorphic embedding problem then the second coloring defines also a "best" coloring, since it uses the number of colors equal to the chromatic number, because $k = |V'| = \chi(\Gamma)$.

We also should point out an interesting phenomenon from real life. If one constructs a product graph for a given problem, then the nodes of this product graph will be listed in such order that they will be also listed by color classes by one of the above scheme. It is because one lists the nodes of the product graph by a double nested for loop listing the nodes of one graph and the nodes of the other graph. From this it follows that pro-

grams that are using sequential greedy coloring as the initial coloring may result in the best possible coloring and will run extremely fast comparing to other programs which use other coloring methods. By our knowledge, this phenomenon was not detected previously. Perhaps this is because the initial coloring is nearly always performed only after the vertices have been sorted in descending order according to their degrees, since this produces superior coloring results on average, that is on all graph types, not only product graphs.

1.4 Second hereditary coloring scheme

We take an independent set $I \subseteq V$ from G and a clique $K \subseteq V'$ from G'. Note that nodes of Γ $(a, b) \in W, a \in I, b \in K$ form an independent set. This follows from the fact that all the nodes of I are independent, while all the nodes of K are connected. Thus no (a_1, b_1) and (a_2, b_2) pair can be connected as $\{a_1, a_2\} \notin E$ and $\{b_1, b_2\} \in E'$. If we partition the nodes from G into i independent sets and partition the nodes of G' into k cliques then we can define $i \times k$ color classes in Γ, where the color classes are formed by pairs of an independent set from G and a clique from G'.

Obviously we can partition G into cliques and G' into independent sets as well.

The described method can be used with many different partitioning resulting in several different colorings. We cannot guarantee though the number of resulting color classes opposed to the first coloring scheme.

1.5 Third hereditary coloring scheme

Similarly to the previous method we partition the set of nodes of G and G' graphs. But instead of independent sets in G we shall use s-clique free set I_s, and instead of cliques in G' – that is equivalent to an independent set in the $\bar{G}'$ complement graph – we shall use an r-clique free set K_r in $\bar{G}'$. Using nodes from these two sets, $a \in I_s, b \in K_r$ the nodes (a, b) in Γ form an $(s + r - 1)$-clique free set. For further details on s-clique free coloring in clique search see [9].

1.6 Other hereditary coloring scheme

Note, that the previous hereditary schemes involve invariant transformations of the graph. Thus other invariances similarly can be of use, for example such as a distance between two nodes. In this case, following the notation of previous subsections, I can be a set of nodes of G, where the smallest distance between two nodes is at least k for a given

number k. While K shall be a set of nodes in G', where the smallest distance between two nodes is less than k. Note that $k = 2$ gives us the second scheme described above.

This scheme probably can be well used in chemical compound similarity search, where the distances between the constituent atoms bear more relevant information than the connection between them.

2 Algorithm

An algorithm is proposed for finding maximum clique. For the initial experiments for exploration of hereditary coloring schemes, we select the first coloring scheme. A parallel program for solving the maximum clique problem [10] is taken as a starting point, which implements a state of the art branch and bound maximum clique algorithm. Ideas from [11] were also included.

First, the program is modified for solving maximum common subgraph problems. A new initial step is constructed, which performs graph multiplication, and maximum clique algorithm is then executed on the product graph. Thus a procedure is created, that is very similar to the one used in ProBiS [1, 2] a program for local comparison of protein structures.

Next, the algorithm is modified to use the first of the presented coloring schemes for pruning the search tree.

2.1 Implementation of the hereditary coloring scheme

To use the hereditary coloring scheme, the following items are of most importance:

- Coloring has to be applied when the product graph is being constructed, since afterwards, the details of the original input graphs are lost.
- The maximum clique algorithm must be *multiple-colorings aware*, that is, if multiple colorings are used, then the algorithm must use and modify them simultaneously in all the operations that would otherwise use the single coloring.

In the initial step, when the product graphs is constructed, either one or two hereditary colorings are applied. Note that the all state-of-the-art graph-based maximum clique algorithms include an initial coloring phase, which cannot be used when coloring is externally provided as it is in our case and is therefore skipped in the proposed algorithm. The

resulting product graph and its colorings are fed to the maximum clique algorithm that is multiple-colorings aware. To be able to use two distinct colorings, the algorithm keeps them both in memory and uses them as appropriate. Since the colorings are initially also optimal, no re-coloring is performed within the clique search.

The largest change in the algorithm is in the point of branching, i.e., the point where the algorithm holds a candidate clique and a set of candidate nodes for *exhaustion*. The process of exhaustion aims to increase the candidate clique by recursively picking the candidate nodes, adding them to the active clique and removing their non-neighbours from the remaining candidate nodes. At this point of the algorithm, coloring is used for two goals.

First, the results of coloring are used in the bounding mechanism, which allows the branching to commence only when the number of remaining colors in the candidate set of nodes is high enough that it can theoretically lead to a maximal clique that is larger than the largest already known clique. That is, the size difference between the largest known clique and the candidate clique (k_{diff}) is used as the bounding criterion. If the candidate nodes can be colored with less than k_{diff} colors, then this branch cannot lead to a new maximal clique, the algorithm can safely prune it and then backtrack to find a more promising branch.

Second, coloring plays a role in minimizing the branching factor [12], i.e., the number of branching points generated by a particular candidate node. Branching factor can be minimized by setting the exhaustion order of the candidate nodes in such a way that leads to the largest number of branches being pruned. The number of branches pruned equals the number of nodes that can be colored with k_{diff} colors. A greedy method of maximizing this number of nodes is to exhaust nodes in the order defined by their colors, and only exhaust the nodes that are colored with the rarest colors until only k_{diff} different colors remain.

To be used efficiently for both goals, the proposed algorithm has to select a single colorings to base this branching decisions on.

- If less than or equal to k_{diff} color bins remain in *any* of the memorized colorings, then the bounding condition is satisfied, branching is canceled, and the algorithm backtracks.
- If k_{diff} is 0, i.e., the candidate clique is also the largest clique found so far, then the

coloring with the smaller number of colors is selected. In case of a tie, the coloring with the rarest color (the color that is used to color the smallest number of nodes) is selected. If there is further tie, then any coloring scheme can be selected. To achieve repeatability of the results, we opt to always select the first coloring in such a case.

- If more than k_{diff} different colors remain for all the memorized colorings, then only the nodes that are not colored by one of the k_{diff} most frequent colors in both colorings are counted. Then the coloring with the lower number of counted nodes is selected. In case of a tie, the coloring with the rarest color is selected. If there is further tie, again any coloring can be selected, and we opt for the first coloring.

After selecting a particular coloring, a node from candidates has to be selected for the branching to be performed. Selection of the node is made based on its color and the initial vertex ordering; first, the nodes with the rarest color are considered, but within the nodes of the same color, the node with the lowest index is selected. This represents a simple but suboptimal solution. Implementing a more elaborate selection of the node would represent a trade-off between the added computational overhead and the possible reduction of the search tree. We leave this topic for further work.

2.2 Effects of the multiple-coloring awareness

On one hand, the additional pruning that is made possible by the inclusion of multiple colorings is certainly beneficial. On the other hand, the inclusion of multiple colorings into the algorithm causes some significant overheads:

- Multiple colorings have to be stored in memory in every recursion of the algorithm. For each recursion, the maximum clique algorithm has to hold three memory expensive variables: current clique, candidate set of vertices, and the set of colorings of the candidates. The latter is the most expansive of the three, and each coloring takes about k times as much memory as the candidate set of vertices, where k is the number of colors remaining in the coloring. In practice, the memory overhead results in slower execution time, since less of the data can be stored in local processor caches.

- Multiple colorings have to be processed in each recursion. Although always only a

single vertex from the candidate set is selected to be added to the current clique, it is removed from all the memorized colorings, which must then be adjusted. Adjustment of colorings (i.e., removal of the selected vertex and all its non-neighbours form the candidate set and color recount) is the most time demanding step. Selection of a vertex for branching is also more demanding, since a non-trivial procedure for selection of coloring is introduced. In total, increasing the number of colorings from one to two about doubles the amount of required processing.

The benefits have to be weighted against the overheads to determine which is more pronounced. The logic dictates that the very large and therefore difficult to solve cases should benefit more, while very small cases should experience mostly overheads. Therefore, in addition to experimenting with the hereditary coloring scheme, we prepare experiments also to establish the problem size at which the use of multiple colorings becomes beneficial (if at all).

To have a meaningful comparison, we make two separate implementations of the algorithm. In addition to simultaneous dual-coloring algorithm (we shall refer to it as *Simultaneous coloring*), we implement also an algorithm in which only one of the possible hereditary colorings is kept while the other is discarded. This implementation leverages the benefits of reduced memory usage and the simpler processing by the use of single coloring throughout the algorithm. We call this flavor of the algorithm *initial coloring* algorithm, since hereditary coloring on it is performed in its initial phase.

3 Experiments

We compare the proposed algorithm against the state-of-the-art reference [13], which is a modern subgraph isomorphism algorithm that implements search via a bit-parallel maximum clique search algorithm [14]. We use the c++ code that was provided as a supplement to the referenced article, without any changes and we compile it with same compiler (GCC 8.3) and same set of compiler switches (`-O3 -march=native -std=c++14`). A small note must be made for the reference algorithm - in its implementation, the timing is performed with the precision of milliseconds. On the smallest problem instances that is insufficient (execution time is always reported as 1 ms) and comparison to the proposed algorithms on those problem instances is therefore limited.

Of the proposed algorithm, we prepared two flavors:

- initial coloring, which uses a simpler algorithm that supports only one coloring, which must be selected at the beginning, and
- simultaneous coloring, which uses two hereditary colorings throughout the algorithm.

To be able to execute the large amount of experiments, we perform them on a cluster of 20 4-core Intel Xeon E5520 based computers. The experiments run in parallel, with each one occupying a single core of a single computer. That is, to avoid any kind od interference between program instances, we allow only one instance at a time per computer. And to make a meaningful comparison of algorithms, not the parallel implementation, we use sequential (single-threaded) execution.

We perform experiments on two datasets:

- A dataset of random graphs [15], from which we extract a subset of problems from the MCS90 directory. This directory contains pairs of 2D graphs of predefined size (both graphs are of the same size) and variable size of the common induced subgraph. We divide the pairs into subsets according to the size (number of nodes) of their input graphs: $|V| = [10, 20, 30, 40]$. Each subset contains 100 problem instances. We do not experiment on larger problems, since execution on several of the included problem instances already takes more than a week to complete. Note that for the hardest of instances execution time of the selected problem instances already surpasses one week.
- A dataset of 857 small molecule structures from Drugbank [16]. We use molecules with accession numbers in the range of $[DB00000 - -DB01020]$ and with less than 60 atoms. We treat molecule structures as labelled graphs, i.e. the vertices are atoms, therefore we used chemical elements as labels of vertices. When generating the product graph, only Cartesian products of same chemical elements are allowed as elements of the product graph. Since labelling reduces the size of product graph it allowed us to use a bit larger larger input graphs, with up to 60 vertices instead of 40 that were used in the random graph database. In contrast to the random database, which is divided into pairs of graphs for testing, database of molecules has no such structure. We performed experiments on all pairs, where one of the molecules is either:

- Pyridoxal phosphate (DB00114, 16 vertices),
- Tetrahydrofolic acid (DB00116, 32 vertices),
- Histidine (DB00117, 11 vertices),
- Ademetionine (DB00118, 27 vertices),
- Pyruvic acid (DB00119, 6 vertices),
- Phenylalanine (DB00120, 12 vertices),
- Biotin (DB00121, 18 vertices),
- Choline (DB00122, 7 vertices),
- L-Lysine (DB00123, 10 vertices),
- Arginine (DB00125, 12 vertices),
- Ascorbic acid (DB00126, 13 vertices),
- Spermine (DB00127 14 vertices),
- Aspartic acid (DB00128, 9 vertices),
- Ornithine (DB00129, 9 vertices),
- L-Glutamine (DB00130, 10 vertices), and
- Adenosine phosphate (DB00131, 23 vertices).

Thus, a set of 13712 pairs of molecules were created. Note that unlike in the random dataset, the pairs of molecule dataset are not divided into groups, since the in general, the two graphs from individual pair are not of equal size. To limit the total time required for these experiments, we limit the execution time to 20 minutes per instance.

We execute all three algorithms (one reference and two proposed flavors) on both datasets. Raw results - execution times, are plotted in Figure 1 and Figure 2, and provide a statistical view on the performance of the algorithms. The problem instances are sorted for all plots by their execution time and cannot be directly compared between the plots. Note that the reported execution time for the reference algorithm is in millisecond resolution, which is not enough for the problem instances of the easiest subset (denoted as *10 nodes* on the graphs). The reference algorithm does seem to be faster than the simultaneous coloring

algorithm for those problem instances though. Note also that the molecule database run times are clamped to 1200 seconds and that for each instance where execution time is 1200 seconds is actually unsolved.

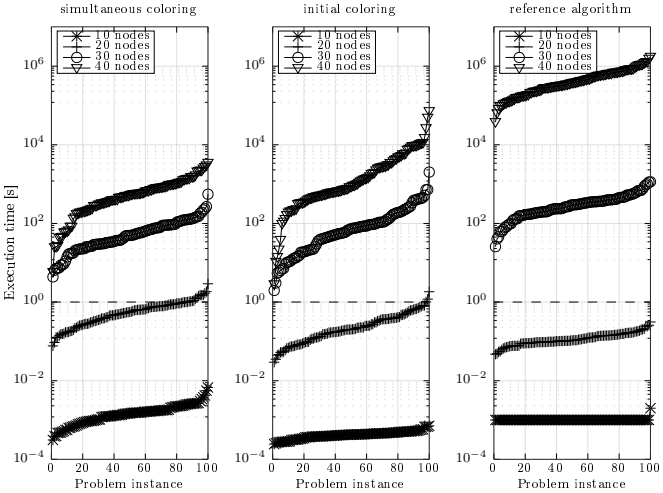

Figure 1. Execution times for all three algorithms used in the comparisons on the general graph dataset

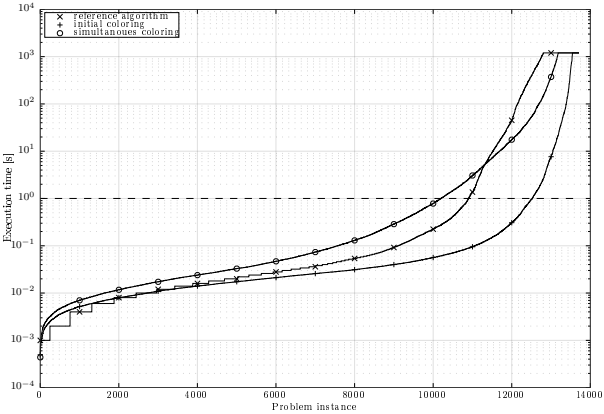

Figure 2. Execution times for all three algorithms used in the comparisons on the molecule dataset. The line of reference algorithm execution times seems jagged on the left, since these execution times are quantized to 1 ms, while others are not.

In the next three subsections, we present the obtained results in greater detail; first we make a comparison between the proposed and the reference algorithms, then we make a comparison between the two proposed flavors of algorithm, and lastly we compare the performance on the general graphs to that on labelled graphs such as the molecular structures.

When presenting the results on molecule dataset, we only present the results where all algorithms solved the problem instance before the timeout of 1200 seconds was reached. Observing the statistics for individual algorithms, the number of solved problem instances by the initial coloring flavor is 98.8 %, by the simultaneous dual coloring flavor is 96.1 % and at by the reference algorithm is 93.4 %.

3.1 Results: hereditary versus reference

The results of the first comparison are presented as speedups of hereditary algorithms relative to the reference algorithm. Figure 3 shows the speedup of the simultaneous coloring and Figure 4 shows the speedup of the initial coloring.

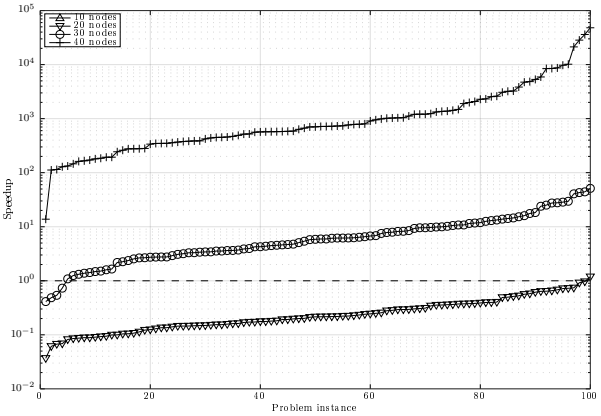

Figure 3. Speedup of the simultaneous coloring flavor of the proposed algorithm relative to the reference algorithm for the three problem subsets that produce a valid comparison. Speedup is sorted by value for easier visualization. The graph line for *10 nodes* is not plotted, since the comparison could not be made accurately.

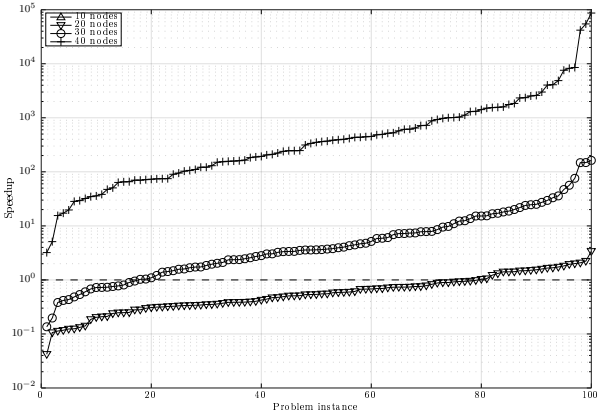

Figure 4. Speedup of the initial coloring flavor of the proposed algorithm relative to the reference algorithm for the three problem subsets that produce a valid comparison. Speedup is sorted by value for easier visualization. The graph line for *10 nodes* is not plotted, since the comparison could not be made accurately

Figure 5 shows the speedup of both flavors of the proposed algorithm relative to the reference algorithm. Results on this figure are only sorted by speedup value and are not further grouped.

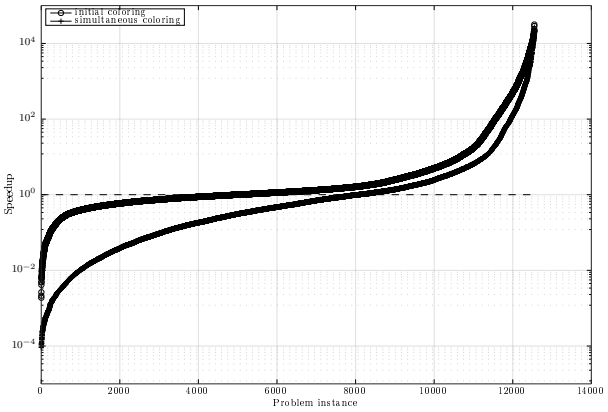

Figure 5. Speedup of both flavors of the proposed algorithm relative to the reference algorithm. Speedup is sorted by value for easier visualization.

On the figures, speedup of 1, i.e. identical execution time for both algorithms, is marked by a dashed line, and speedup above 1 indicates that the proposed algorithm

is faster than reference. Note that the results for instances with $|V| = 10$ on random graph dataset are not shown, since the reference algorithm timing resolution of 1 ms is inadequate for calculation of speedups.

The results show that the execution speed of algorithms is highly correlated with the problem size. The proposed algorithm is slower for nearly all of the smaller problem sizes, i.e. for input graphs of size 10 and 20, and faster for nearly all larger problem sizes, i.e. for input graphs of sizes 30 and 40. Furthermore, the speedups for large problems get exceptionally high for some instances, and reduce execution time from nearly unusable (in the order of days or even weeks) down to very usable (order of minutes).

3.2 Results: simultaneous versus initial coloring

To weight the overheads of multiple colorings against their benefits, we directly compare the two algorithm flavors. The results obtained on the random graph dataset, in form of simultaneous coloring speedup with initial coloring as a reference are shown in Figure 6.

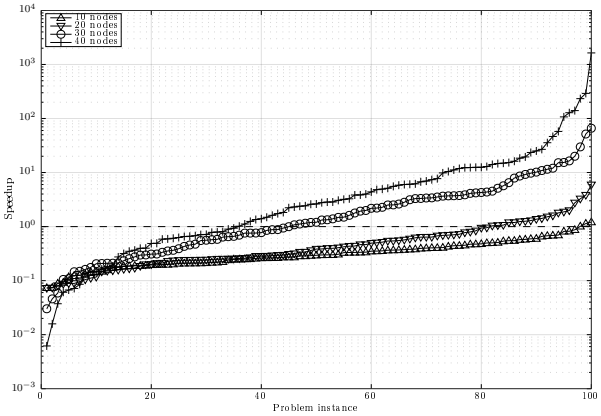

Figure 6. Speedup of the multiple colorings algorithm versus a single coloring, on the random graph dataset.

As expected, on average, multiple colorings performance is better on larger problem instances, while initial coloring is better on smaller problem instances. The divide seems to be between $|V| = 20$ and $|V| = 30$, the same as when comparing the multicoloring algorithm to the reference algorithm.

The results obtained from the molecule dataset, shown on Figure 7 are less expected, though. They show that simultaneous coloring is very rarely the more efficient algorithm.

The experiments are performed mostly on relatively small graphs and simultaneous coloring performance might increase with much larger problem sizes.

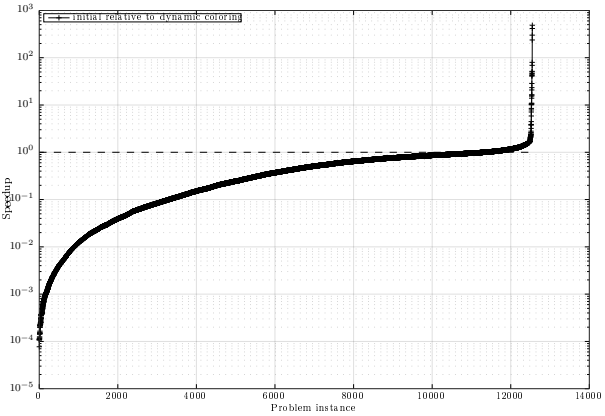

Figure 7. Speedup of the multiple colorings algorithm versus a single coloring, on the molecule dataset.

A plot of execution times tells a similar story. A direct comparison of execution times on random graph dataset is plotted on Figure 8 and The trend is clearly visible for the random graph dataset; simultaneous coloring is gaining advantage as the execution time lengthens. Although the initial coloring is faster on 258 problem instances, while the simultaneous coloring is faster on 142 instances, the latter is much faster on average. Mean execution time of the simultaneous coloring is 197 seconds, while the mean execution time for initial coloring is 993 seconds. While these statistics depend mostly on the difficulty of the presented problem instances, it shows that the simultaneous coloring is generally not faster, but when it is, it can make a significant difference.

For molecule dataset, not much new information can be gathered from Figure 9. Execution times spread out more for the more difficult problem instances, but otherwise there is no apparent trend in the collected data. Since on the molecule dataset, in contrast to random graph dataset, common induced subgraph is sought between different-sized graphs, simultaneous coloring does not bring any advantage. Recall that for simultaneous coloring flavor, two colorings are performed in advance, one per input graph. These colorings are comprised of as many colors as their source input graph has vertices. Number of colors in each of these colorings corresponds to the number of vertices in first or

second input graph. Since input graphs are of very different size, the two colorings are therefore not equally promising in terms of helping to efficiently bound the search space of the algorithm. Therefore, it seems, that working with both colorings, as in the case of simultaneous coloring algorithm flavor is not beneficial at all for regular instance sizes. For very large instances, there could still be some benefits in using simultaneous coloring. but these might be diminishing, as due to NP complexity of the problem. The molecule dataset is, however, more realistic than the random graph dataset. It is hardly ever expected that one would be interested in only searching for common subgraphs in molecules or other graphs of the exactly same size.

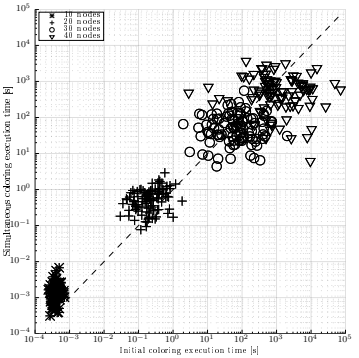

Figure 8. Comparison of execution times of the multiple and initial colorings algorithms. Dashed line represents equal times.

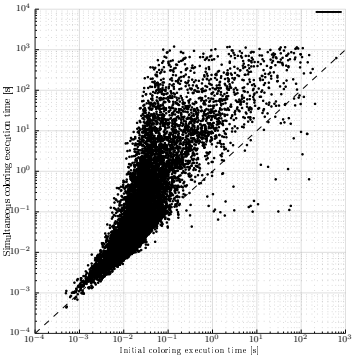

Figure 9. Comparison of execution times of the multiple and initial colorings algorithms. Dashed line represents equal times.

4 Conclusions

In this paper we propose a hereditary coloring scheme for improving the efficiency of solving problems of maximum common induced subgraph using a maximum clique algorithm. We experiment with two flavors of algorithm, that differ in their computational complexity. We observe that the proposed methodology works remarkably well when using the first and most straight-forward of the proposed hereditary coloring schemes. Obtained results show large speedups over the reference algorithm but also a dependence of the results on the size of the problem. The proposed algorithm is faster than the reference on nearly all easy problem instances, while it is faster on nearly all difficult problem instances.

Within the two presented flavors of the proposed algorithm, the one that uses dual coloring is faster, when solving difficult problem instances of equal input graph size. For use in real life, the initial coloring algorithm seems to present a more viable approach, as it offers the most performance on most real-life problem instances that are small that solving them is attempted at all.

The implemented color scheme has room left for improvements, such as re-coloring before branching, or a more elaborate selection of the node within a coloring bin to branch on. Further experimentation in this area is required to determine which steps of the algorithm should be left as simple as possible and which would benefit from additional processing.

We expect that adding other hereditary coloring schemes would also provide good results, on par or better than those already observed here. Primarily, they might make simultaneous coloring less dependent on the equal size of input graphs. The other schemes would also present some trade-offs between the complexity of the coloring procedure and the quality of the coloring with respect to being used as the bounding criterion within the branch-and-bound maximum clique algorithm. Therefore we expect that for various forms of problems (regarding their complexity, e.g. input graph sizes, densities, symmetries, labelling, etc), different combinations of hereditary coloring schemes would prove as the best for the job. Further experiments, though, are planned for the future work.

Acknowledgments: This work was funded by the Slovenian Research Agency (research core funding No. N2-0053) and by Hungarian National Research, Development and Innovation Office – NKFIH Fund No. SNN-117879.

References

[1] J. Konc, D. Janežič, ProBiS algorithm for detection of structurally similar protein binding sites by local structural alignment, *Bioinformatics* **26** (2010) 1160–1168.

[2] J. Konc, M. Depolli, R. Trobec, K. Rozman, D. Janežič, Parallel-ProBiS: Fast parallel algorithm for local structural comparison of protein structures and binding sites, *J. Comput. Chem.* **33** (2012) 2199–2203.

[3] E. Balas, J. Xue, Weighted and unweighted maximum clique algorithms with upper bounds from fractional coloring, *Algorithmica* **15** (1996) 397–412.

[4] R. Carraghan, P. M. Pardalos, An exact algorithm for the maximum clique problem, *Oper. Res. Lett.* **9** (1990) 375–382.

[5] P. R. J. Östergård, A fast algorithm for the maximum clique problem, *Discr. Appl. Math.* **120** (2002) 197–207.

[6] E. Tomita, T. Seki, An efficient branch-and-bound algorithm for finding a maximum clique, in: C. S. Calude, M. J. Dinneen, V. Vajnovszki (Eds.), *Proceedings of the 4th International Conference on Discrete Mathematics and Theoretical Computer Science*, Springer-Verlag, Berlin, 2003, pp. 278–289.

[7] D. R. Wood, An algorithm for finding a maximum clique in a graph, *Oper. Res. Lett.* **21** (1997) 211–217.

[8] A. Bóta, M. Krész, A high resolution clique-based overlapping community detection algorithm for small-world networkss, *Informatica* **39** (2015) 177–187.

[9] S. Szabo, B. Zavalnij, Greedy algorithms for triangle free coloring, *AKCE Int. J. Graphs Comb.* **9** (2012) 169–186.

[10] M. Depolli, J. Konc, K. Rozman, R. Trobec, D. Janežič, Exact parallel maximum clique algorithm for general and protein graphs, *J. Chem. Inf. Model.* **53** (2013) 2217–2228.

[11] S. Szabó, B. Zaválnij, A different approach to maximum clique search, in: E. Abraham, D. Zaharie, V. Negru, D. Petcu, T. Ida, S. M. Watt (Eds.), *20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC2018)*, IEEE, Los Alamitos, 2018, pp. 310–316.

[12] S. W. Golomb, L. D. Baumert, Backtrack programming, *J. ACM* **12** (1965) 516–524.

[13] C. McCreesh, S. N. Ndiaye, P. Prosser, C. Solnon, Clique and constraint models for maximum common (connected) subgraph problems, in: M. Rueher, (Ed.) *Principles and Practice of Constraint Programming: 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings*, Springer, Cham, 2016, pp. 350–368.

[14] C. McCreesh, P. Prosser, The shape of the search tree for the maximum clique problem and the implications for parallel branch and bound, *ACM Trans. Parallel Comput.* **2** (2015) #8.

[15] D. Conte, P. Foggia, M. Vento, Challenging complexity of maximum common subgraph detection algorithms: A performance analysis of three algorithms on a wide database of graphs, *J. Graph Alg. Appl.* **11** (2007) 99–143.

[16] C. Knox, V. Law, T. Jewison, P. Liu, S. Ly, A. Frolkis, A. Pon, K. Banco, C. Mak, V. Neveu, Y. Djoumbou, R. Eisner, A. C. Guo, D. S. Wishart, DrugBank 3.0: a comprehensive resource for 'Omics' research on drugs, *Nucleic Acids Res.* **39** (2010) D1035–D1041.