On the Efficiency of an Order-based Representation in the Clique Covering Problem

David Chalupa
Institute of Applied Informatics
Faculty of Informatics and Information Technologies
Slovak University of Technology
Ilkovičova 3
84216 Bratislava, Slovakia
chalupa@fiit.stuba.sk

ABSTRACT

Although the (vertex) clique covering problem (CCP) is a classical NP-hard problem, it is still overlooked in the fields of heuristics and evolutionary algorithms. We present two main results concerning this problem. First, we propose a genotype-phenotype mapping algorithm for an order-based representation of the CCP, called greedy clique covering (GCC), and prove that for an arbitrary graph, there is a permutation, for which GCC constructs the optimal solution. Although the greedy graph coloring can also be used as genotype-phenotype mapping, we show that GCC is much more efficient for sparse graphs. Secondly, we adapt a mutation-based metaheuristic algorithm using the order-based representation - iterated greedy (IG), to solve the CCP. On sparse graphs with planted cliques, we provide empirical evidence that IG outperforms an exact algorithm. This result is supported by a runtime analysis of IG on several subclasses of graphs with planted cliques. We include experimental results of IG on random graphs, several DIMACS instances and social graphs. Its comparison to the related existing approaches shows that our IG algorithm outperforms the standard approaches in almost all instances.

Categories and Subject Descriptors

I.2.8 [**Problem Solving, Control Methods, and Search**]: Heuristic methods; G.2.3 [**Discrete Mathematics**]: Applications

General Terms

Algorithms, Experimentation

Keywords

Clique Covering, Order-based Representation, Permutation Mutations, Iterated Greedy, Graph Coloring

1. INTRODUCTION

Let $G = [V, E]$ be an undirected graph and let $d = \frac{|E|}{\binom{|V|}{2}}$ be its density, i.e. the number of edges, divided by the number of all possible connections. We will refer to $V_i \subset V$ as a *class* in V. Then, an *induced subgraph* $G(V_i) = [V_i, E_i]$ is a graph that contains only the edges between vertices in class V_i, i.e. $E_i = \{[v, w] \in E \mid v, w \in V_i\}$. The aim of the *(vertex) clique covering problem* (abbr. CCP, also known as the *clique cover* or the *clique decomposition* problem) is to find a partitioning of V into k classes, formally $S = \{V_1, V_2, ..., V_k\}$, such that each class V_i induces a clique, i.e. $\forall\ i = 1..k\ d(G(V_i)) = 1$. We will refer to the minimal k, for which CCP can be solved, as the *clique covering number* of G and denote it as $\vartheta(G)$.

It can be easily proved that the clique covering problem in G is equivalent to the graph coloring in the complementary graph $\overline{G}$. The clique covering number is equal to the chromatic number of the complement, i.e. $\vartheta(G) = \chi(\overline{G})$. Although the decision problem for the CCP is one of the classical Karp's 21 NP-complete problems [9], CCP is still overlooked in the heuristic and evolutionary algorithm fields.

Despite this fact, the applications of the CCP can be found in important fields such as data mining, network analysis, community detection [15] and especially in large sparse graphs, such as social networks, research citation networks or gene regulatory networks [16].

In this work, we propose a *genotype-phenotype mapping algorithm - greedy clique covering* (GCC), designed specifically for the CCP. The reason is that, when G is a very sparse graph, it might be desirable to perform the optimization over G, instead of coloring the dense graph $\overline{G}$. In addition, we study a *mutation-based metaheuristic algorithm* using the *order-based representation* - the iterated greedy (IG), on the CCP. On sparse graphs with planted cliques, we provide empirical evidence that it outperforms an exact algorithm. Using a combination of fitness-based partitions method of nature inspired algorithm analysis [13] and graph theory, we prove that under certain circumstances, IG solves the problem optimally in $\mathcal{O}(|V|^4)$ time.

2. RELATED WORK

Most of the current heuristics that are usable for the CCP, are the graph coloring heuristics over the complementary graph. In the greedy coloring, vertices are taken in a certain order and the *First Fit* method for color choice is generally used, when the first available color is assigned [17]. In

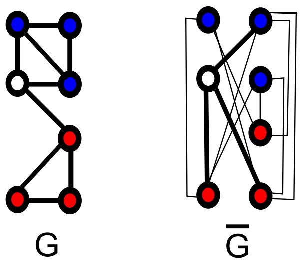

Figure 1: Illustration of the CCP in G and graph coloring in $\overline{G}$.

the terms of metaheuristics and evolutionary computation, using permutations of vertices is often referred to as the order-based representation of the problem.

An upper bound for the chromatic number $\chi(\overline{G})$ can be found using several variants of the greedy coloring. A typical method for vertex ordering is the *largest degree first* (LDF), when the vertices with more neighbors are colored first [17]. Brélaz extended this concept in an algorithm, often called *DSatur* or simply the Brélaz's heuristic, in which at each iteration, a vertex with highest *saturation* - the number of colors already used in the neighbors of a vertex, is colored next. If there are more vertices with highest saturation, LDF is used to break the tie [1]. Another interesting heuristic - RLF was proposed by Leighton, where the problem was tackled by iterative removal of large independent sets from the graph [11].

A problem is that a vast majority of modern graph coloring heuristics tackles the k-colorability decision problem, i.e. the number of colors is fixed to a constant k in the problem instance [7]. For benchmarking graphs, we know, what the k should be and this fixation significantly reduces the search space. However, for unknown instances, this would hardly be an appropriate way, how to estimate. Indeed, the problem can be solved by tackling the graph coloring for k, $k-1$, $k-2$, etc. However, when the estimate for the initial value k is very far away from the optimum, this strategy might be too exhaustive. Furthermore, when G is a sparse graph, the estimation of $\vartheta(G)$ through $\chi(\overline{G})$ might be computationally inefficient, since the computational complexity of greedy coloring for the complement with a predefined permutation of vertices is $\mathcal{O}(|E(\overline{G})|)$, where $|E(\overline{G})|$ will be very close to $\binom{|V|}{2} = \Theta(|V|^2)$ for the much denser graph $\overline{G}$. This complexity is impractical for large graphs.

Heuristics that allow a non-fixed number of colors represent a minority in the literature. Culberson and Luo have extensively studied the IG heuristic on the classical graph coloring problem [3, 4]. The IG heuristic evolves a greedy component using a neighborhood search procedure. The IG heuristic for the graph coloring relies on the fact that for every graph, there is a permutation, for which the greedy coloring produces the optimal solution with $\chi(G)$ colors. This permutation can be constructed from the optimal coloring in the way that the vertices with the same colors that form the independent sets, are adjacent in the permutation. However, when performing this transformation for a suboptimal coloring, if the independent sets are placed in a favorable order, the permutation may lead to a reduced coloring. Therefore, when beginning with a random permutation, with a sequence of permutation changes, Culberson's and Luo's method can evolve gradually better solutions. They obtained remarkable results especially for k-colorable graphs with embedded independent sets [4].

In another work, an evolutionary algorithm with non-fixed number of colors was presented, where the authors applied several types of problem-specific mutations [5], working with the order-based representation of the problem. A large spectrum of memetic algorithms for the graph coloring is also available. However, the k-fixed strategy is strongly preferred in these approaches [6, 12, 14]. In this paper, we pay special attention to sparse graphs. Previous work using heuristics designed specifically for sparse graphs can be found in [10].

3. THE ORDER-BASED REPRESENTATION OF THE CCP AND ITS EFFICIENCY

In this section, we propose our genotype-phenotype mapping algorithm and prove that for an arbitrary graph, there is a genotype, which leads to an optimal phenotype. In addition, we compare the algorithm to the existing approach to the genotype-phenotype mapping - the greedy coloring and show that our approach is much more efficient for sparse graphs, regarding both time and space complexity.

3.1 The Genotype-Phenotype Mapping Using Greedy Clique Covering (GCC)

We have already indicated that there are two different approaches available to tackle the estimation of $\vartheta(G)$. The first one is to use a graph coloring heuristic on $\overline{G}$. When using this approach, it is more efficient to store only the original graph G and handle the coloring of $\overline{G}$ by first sorting the adjacency lists (technically, when loading the graph) and then skipping the edges in G during the greedy coloring. Since G is much sparser than $\overline{G}$, this minor technical detail allows one to handle much larger graphs, due to the memory limitations. However, when using this approach, we have to take into account that for a dense graph like $\overline{G}$, complexity of $\mathcal{O}(|V|^2)$ should be accepted even for the greedy coloring.

The second way is to use a greedy algorithm that operates only on the edges of G. When looking at the graph G itself, e.g. a large social network, we can assume that it is much sparser than $\overline{G}$. The drawback, however, is that the edges in G have different interpretation than the edges in $\overline{G}$, since they do not technically represent the constraints of the problem. This is a big difference between the classical graph coloring and clique covering. However, an efficient heuristic can still be obtained to estimate the clique covering using just the original graph G.

In Figure 1, we show a simple example of the graph for the CCP (G) and the graph coloring ($\overline{G}$). In both graphs, there is a single uncolored vertex. In $\overline{G}$, we can see that it is enough to scan its neighbors to be sure that a new color will be needed, since the vertex has both a blue and a red neighbor. In G, the situation is different. The vertices that define the constraints are not adjacent to the uncolored vertex. However, suppose that we want to assign a color to the vertex in Figure 1. When we have only the graph G, it is enough to know that there are actually 3 blue and 3 red vertices. Then, if the vertex is not adjacent to all vertices of a particular color, it cannot be assigned this color. Therefore, with this additional information, a greedy heuristic would know that it needs a new color. This is the basic idea of the

Algorithm 1: Greedy Clique Covering

```
    Greedy Clique Covering
    Input: graph $G = [V, E]$
    permutation $P = [P_1, P_2, ..., P_{|V|}]$ of vertices in $V$
    Output: clique covering $S$ of $G$
1   for $c = 1..|V|$
2       $sizes(c) = 0$
3   for $i = 1..|V|$
4       $j = P_i$
5       $c = find\_equal(\Gamma(v_j, c), sizes(c))$
6       $V_c = V_c \cup \{v_j\}$
7   return $S = \{V_1, V_2, ..., V_k\}$
```

greedy clique covering (GCC), specified in the pseudocode of Algorithm 1.

The greedy clique covering is very similar to the greedy coloring. GCC processes the vertices iteratively, taking i-th component of permutation P as the current vertex in the step 4. In contrast to greedy coloring, we explicitly store an array $sizes$, in which $sizes(c)$ contains, how many vertices are already labeled with c. In this context, we will refer to colors as labels, since CCP is not a traditional perspective on the coloring problem.

Let $v_j \in V$ be a vertex to be labeled, $c \in C$ be a label and let $S = \{V_1, V_2, ..., V_k\}$ be a partial clique covering induced by the current labeling of G. Then, we define a function $\Gamma : V \times C \rightarrow N_0$ as a function, which returns the number of neighbors of v_j, which are colored with c, i.e.: $\Gamma(v_j, c) = |\{v_l \in V \mid [v_j, v_l] \in E \wedge v_l \in V_c\}|$.

When choosing a color for vertex v_j in the step 5, we verify, whether it is adjacent to all vertices that already have label c. Thus, by assigning label c to v_j, subgraph $G(V_c)$ induced by label c will still remain a clique. Practically, this means that value $\Gamma(v_j, c)$ must be equal to $sizes(c)$. If there is no such value, a new label must be chosen. The key issue in greedy clique covering is that, while scanning only the edges in G, the labeling can be constructed very quickly, even in very large graphs.

The only problem is the Γ function, used in the step 5 to determine the label to be assigned. This operation should be implemented by scanning the neighbors of the currently labeled vertex. Memorizing values of Γ is inefficient, we use it only as a formal concept.

Suppose that we implement the $find\_equal$ operator in the step 5 in the following way. We scan the neighbors of v_j. If the neighbor is already labeled, we decrement the corresponding value in $sizes()$. If the first decrementation is performed, we store the original value to be able to restore the initial values in $sizes()$. Then, if, during this process, some value $sizes(c)$ reaches 0, then c is a candidate for the label for v_j. From these candidates, we choose the lowest c, to ensure the conformity with the First Fit strategy. After this, we repeatedly scan the neighbors of v_j and restore the values in $sizes()$ that were present there before the decrementation process.

This strategy leads to an interesting complexity, since the operation in the step 4 costs $\mathcal{O}(1)$ time per iteration and the choice of label costs only $\mathcal{O}(\deg(v_j))$ time per iteration, since we iterate only over the neighbors. By using the well-known formula that $\sum_{v \in V} \deg(v) = 2|E|$, we obtain that the complexity of greedy clique covering is only $\mathcal{O}(|E|)$, which is highly favorable for sparse graphs.

Another issue is the genotype-phenotype mapping. The main question is, whether there is a permutation (genotype), for which Algorithm 1 will construct the optimal covering (phenotype). The idea here is similar to the theorem and proof of the same property for the greedy coloring [4].

Theorem 1.

For an arbitrary graph $G = [V, E]$, there is a permutation, for which the greedy clique covering will produce the optimal solution with $\vartheta(G)$ cliques.

Proof.

Let $S = \{V_1, V_2, ..., V_{\vartheta(G)}\}$ be the optimal solution to the CCP. Then, the optimal permutation P can be constructed in the way that the vertices from the same classes are next to each other in P, i.e. $P = [V_{s_1}, V_{s_2}, ..., V_{s_{\vartheta(G)}}]$, where $s_1, s_2, ..., s_{\vartheta(G)}$ is an arbitrary permutation of integers from 1 to $\vartheta(G)$. Since vertices of each of the subpermutations form the correct cliques, this permutation will surely lead to the optimal clique covering. $\square$

3.2 A Comparison to Greedy Graph Coloring

We have shown that GCC and greedy graph coloring are equivalent in the suitability as genotype-phenotype mapping algorithms. At this point, we wonder, which of the two approaches is more suitable for which type of graphs. We briefly compare the time and space complexities of these approaches.

When we look on the number of operations needed by the algorithms in more detail, we obtain the following. In the greedy graph coloring, when performed over $\overline{G}$, both the color choice and updates in an auxiliary matrix, which is needed, require an amount of operations proportional to $|V| - \delta(G) - 1$, where $\delta(G)$ is the average degree of a vertex in G. The exact number of operations depends on a particular implementation. However, for our implementation we obtained $|V|[4|V| - 4\delta(G)]$ operations, when omitting the constant factor. In addition, greedy coloring needs $|V|[|V| - \delta(G)]$ additional storage space, to store the auxiliary matrix. For more details on this approach, the reader may refer to [1, 17].

In GCC, the amount of operations depends on the implementation details, too. Our currently best implementation performs $|V|[17\delta(G)]$ operations, where $17\delta(G)$ is the average number of operations, which are done during the neighbor scans. Again, we omitted the constant factor. In addition, instead of an auxiliary matrix, GCC uses the $sizes()$ array, which needs additional storage space only of size $|V|$.

Therefore, GCC always uses less space than the greedy graph coloring and performs faster, when $\delta(G) < \frac{4|V|}{21}$. This occurs, when the graph has a density lower than approximately $\frac{4}{21}$. However, better implementation strategies might be able to improve this factor.

4. A MUTATION-BASED APPROACH: THE IG ALGORITHM

At this point, we present an adaptation of a mutation-based stochastic search algorithm in the order-based representation - the iterated greedy (IG). In addition, a brief empirical study of the performance of IG is presented on

Algorithm 2: The IG Algorithm for the CCP

```
     The IG heuristic for the CCP
     Input: graph $G = [V, E]$
     Output: clique covering $S$ of $G$
1    $P = random\_permutation(1, 2, ..., |V|)$
2    for $i = 1..I_{max}$
3        $\{V_1, V_2, ..., V_k\} = greedy\_clique\_covering(G, P)$
4        if $\vartheta^*(G)$ is known and $k = \vartheta^*(G)$
5            return $S = \{V_1, V_2, ..., V_k\}$
6        with $p_{rev}$ probability
7            $P = [V_k, V_{k-1}, ..., V_1]$
8        else
9            $P = random\_permutation(V_1, V_2, ..., V_k)$
10   return $S = \{V_1, V_2, ..., V_k\}$
```

graphs with planted cliques, which represent a simple model of clustered graphs.

4.1 The IG Algorithm

In the IG for the graph coloring, Culberson and Luo used three different mutations - moves of the independent sets - *random*, *reverse* and *largest first* [4]. The random move reordered the independent sets entirely randomly. The reverse move reordered them from the last to the first in the previous coloring. And the largest first move reordered them according to their size, from the largest to the smallest. They used a probabilistic choice of the move to be performed, generally a ratio of *50:50:30* (*largest first:reverse:random*) was shown to be efficient [3].

These moves can be easily reused to solve the CCP, by simply using the reorderings on cliques, instead of independent sets. For the clique covering, reverse and random moves require only $\mathcal{O}(|V|)$ time. The largest first would require sorting of the cliques to a non-increasing sequence according to their size, which can be done in $\mathcal{O}(|V| \log |V|)$ time using Merge sort or Heap sort or, with some additional space, $\mathcal{O}(|V|)$ time can be also achieved with Counting sort [2]. However, for simplicity, we have decided to use only the reverse and random moves. These were verified in our preliminary experiments to provide enough dynamics to use the IG as a neighborhood search algorithm.

In Algorithm 2, our IG for the CCP is specified. First, a random permutation P of vertices is generated. Then, at each iteration, P is used in greedy clique covering to generate a solution. Building blocks V_1, V_2, ..., V_k, representing the clique-inducing classes, are identified in the constructed solution (the index of the class represents the label of the clique). If we know the optimal number of cliques - $\vartheta^*(G)$, we verify the current k in the step 4. With probability p_{rev}, the reverse move is performed on the classes in the step 7, where the new permutation is generated in the way that the vertices of the clique labeled with the highest index come first and the vertices of the clique labeled with the lowest index come last. Otherwise, the cliques in the permutation are reordered randomly in the step 9, in the way that the equally labeled vertices should be still adjacent in P. We note that I_{max} is used in Algorithm 2 only as a formal concept. In the following, we use a similar symbol I to refer to the number of iterations, which are in fact performed by IG to obtain the optimal solution.

At this point, let us move on to the amount of operations, needed by IG. A random permutation of vertices can be generated in $\mathcal{O}(|V|)$ time. In the iterative procedure, the greedy clique covering takes $\mathcal{O}(|E|)$ time and the moves require no more than $\mathcal{O}(|V|)$ time per iteration. Therefore, if the IG for the CCP converges to the optimal solution, it will do so in $\mathcal{O}(I(|E| + |V|))$ time, which can be further reduced to $\mathcal{O}(I|E|)$ if $|E| \geq |V|$.

4.2 IG on the Graphs with Planted Cliques

The previous analysis implies an interesting question. If we fix the instance type to a topology, for which the IG will converge to the optimal clique covering, then we wonder, how does the computational complexity of IG behave.

For this purpose, we consider the *graphs with planted r-cliques*. Suppose that the graph contains cliques of size r and each vertex is in exactly one clique. Then, r must be a divisor of $|V|$ and the number of cliques is $\vartheta = \frac{|V|}{r}$. The edges between vertices in different cliques are generated with some relatively small probability p_{out}, to be able to practically guarantee that no pair of cliques will be merged during the edge generation process. This occurs when $p_{out}^{r^2}|V|(|V|-r)/2 << 1$, where $<<$ means "much smaller".

For simplicity, suppose that no pairs of cliques were merged and suppose that we know r. Under these circumstances, a simple exact algorithm can be found to solve the CCP. It just scans each r-tuple of vertices in the graph and verifies, whether they form a clique. It can be easily seen that this approach needs $\binom{|V|}{r} = \Theta(|V|^r)$ time and is polynomial, when r is a known constant.

Now, let us move on to the behavior of IG on the graphs with planted r-cliques. Generally, when r is small enough, IG has no problems converging to the global optimum within a small number of iterations. An interesting property is that if p_{out} is small, i.e. the graph is sparse, then the larger the cliques are, the stronger are also the attractors in the search space of IG. Therefore, for sparse graphs, smaller cliques require more "sorting", until the optimal permutation is found. However, when p_{out} is larger, the region of hard instances of the problem, which was a called a ridge by Culberson and Luo [4], moves to higher values of r.

Since for the graphs with planted r-cliques, we can generally say that $|E| \geq |V|$ and $|E| = \Theta(|V|^2)$, the complexity of IG on this type of graphs can be expressed as $\mathcal{O}(I|V|^2)$. The only question here is, how does the number of iterations I grow with $|V|$, under the consideration that p_{out} is small enough. Figures 2 and 3 shed a bit more light on the I function. It displays the growth of I as $|V|$ grows from 3×10^3 to 30×10^3 (with 3×10^3 intervals). We used the IG on graphs with planted r-cliques. These graphs were generated for $r = 3, 4, 5, 6$ and 8 with inter-clique edge generation probability $p_{out} = 10^{-3}$. The ratio for the reverse and random moves was $50 : 30$ and the values displayed are the average values over 20 runs.

Figures 2 and 3 show that the growth of I is apparently very slow, enough to consider IG to be a promising tool for locating high local density anomalies in graphs with low global density (this is a very important property, especially for applications in data mining). For $r > 4$, IG required only tens of iterations to find the optimal covering. With growing r, the number of iterations decreases, which can be explained by the fact that larger cliques in a sparse graph tend to be more "visible" and therefore, are easier to be located. This also corresponds to the results presented by

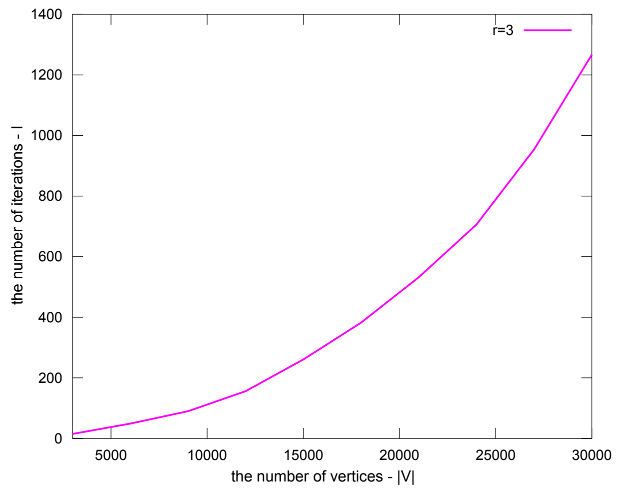

Figure 2: The number of iterations required by IG to solve the CCP over graphs with planted 3-cliques, as a function of $|V|$.

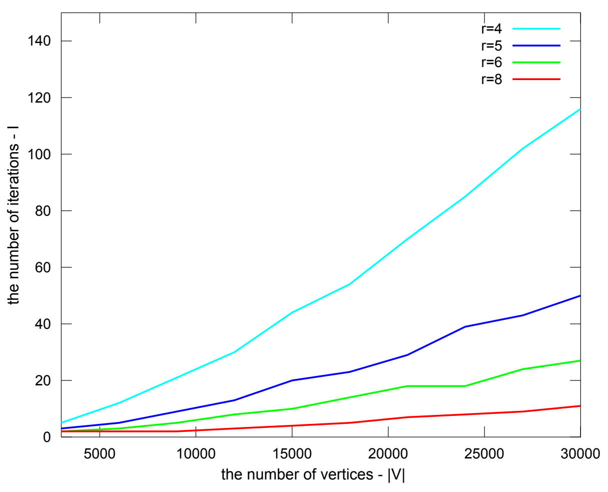

Figure 3: The number of iterations required by IG to solve the CCP over graphs with planted r-cliques ($r \geq 4$), as a function of $|V|$.

Culberson and Luo [4]. However, on the contrary, it is a reverse result to the complexity of the exact algorithm we were discussing before, where the degree of the polynomial grows with r. Perhaps an interesting observation is that the differences in $I(|V|)$ indicate mild convex behavior but with occasional local non-convex intervals, which may be caused by randomness. Maybe a more interesting information is that the quotients of the sequence, we measured for $I(|V|)$, tend to decrease with growing $|V|$. Thus, they do not indicate an exponential outbreak. A special case is $r = 3$, which seems to be on a boundary of a region of harder instances (that, in the case of small p_{out} seems to be very small). However, although this function grows fastest, the quotients here still do not indicate any exponential outbreak.

5. ANALYSIS OF IG ON GRAPHS WITH PLANTED CLIQUES

At this point, we wonder, whether it is possible to find an upper bound for the number of iterations analytically. In fact, IG is a variant of a stochastic neighborhood search algorithm. Therefore, it might be possible to analyze its behavior using the methods of evolutionary computation theory.

5.1 The Fitness-based Partitions of Planted Clique Landscapes

The appropriate method for analysis of this algorithm is the method of fitness-based partitions. In this method, the search space is divided into several partitions, according to the fitness function [13]. There are two key aspects that are important to be established for this method:

- *proper fitness-based partitions* - levels in the optimization, where the last level contains only optimal solutions
- *probability of augmentation to a new level* - the smallest probability that a mutation will lead the algorithm to augment the current solution to a higher fitness-based partition

IG is a typical algorithm for this method of analysis, since it uses only one individual and the objective function is monotonous in the search process - the algorithm never causes a degradation of fitness (in this sense, it is very similar to hill climbing). We will need the following lemma from [13]:

Lemma 1 [13].
The expected optimization time of a stochastic search algorithm that works at each time step with a population of size 1 and produces at each time step a new solution from the current solution is upper bounded by:

$$I \leq \sum_{i=1}^{m-1} \frac{1}{p_i}.$$

In this lemma, m is the number of fitness-based partitions and p_i is the smallest probability that in time step i, the mutation will cause an augmentation.

5.2 Analysis of IG on Sparse Biclique Graphs

Graphs with planted cliques can be formalized as a triple $[r, \vartheta, p_{out}]$, where r is the size of the clique, ϑ is the number of cliques and p_{out} is a probability that an inter-clique edge is generated. Indeed, for these graphs $|V| = r\vartheta$, so the number and size of cliques are divisors of the number of vertices. The expected number of inter-clique edges will be denoted as $|E|_{out}$.

In this section, we begin with the most basic case - complements of bipartite graphs, i.e. $\vartheta = \frac{|V|}{r} = 2$. For this type of graphs, the expected value of $|E|_{out} \approx p_{out}r^2$. An important assumption is that $|E|_{out} < r$. We will explain this later in more detail. For simplicity, we will refer to these graphs as sparse biclique graphs.

For practical purposes, we suppose that $r > 2$, so we consider the cliques to be at least triangles.

For the most simple situation, where $r = 3$ and $|E|_{out} = 0$, any permutation of vertices would generate the optimum. There is only one way, how to add an inter-clique edge (the other ways would always lead to isomorphic graphs). Figure

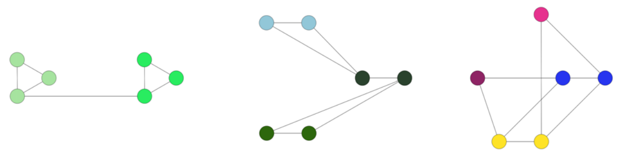

Figure 4: Two triangles with 1 inter-clique edge and their coverings with 2 and 3 cliques (on the left and in the middle) and two triangles with 2 inter-clique edges and their covering with 4 cliques (on the right).

4 shows this bridged graph with its covering with two cliques (on the left) and three cliques (in the middle).

We note that the suboptimal clique covering can be generated, when a couple of vertices, connected by the bridge (the dark green vertices in the middle of Figure 4), precedes all intra-clique couples of vertices. Therefore, it makes sense to re-represent a vertex ordering $[v_1, v_2, v_3, ..., v_{|V|}]$ as an ordering $[[v_1, v_2], [v_2, v_3], ..., [v_{|V|-1}, v_{|V|}]]$, where, instead of the vertices themselves, we have their couples. Then, a couple $[v_i, v_{i+1}]$ can fall exactly into one of the three following cases:

- (v_i, v_{i+1}) is an intra-clique edge; there are $|V|(r-1)/2$ such cases
- (v_i, v_{i+1}) is an inter-clique edge; there are $p_{out}|V|(|V|-r)/2$ such cases on average
- (v_i, v_{i+1}) is an inter-clique couple, without an edge between them; there are $(1-p_{out})|V|(|V|-r)/2$ such cases on average

When we aim our attention specifically at sparse graphs, there are two possibilities, how GCC can overestimate:

- When, in the ordering, an inter-clique edge between two cliques precedes all intra-clique edges from the cliques it connects.
- When an inter-clique couple $[v_i, v_{i+1}]$ without an edge precedes a vertex adjacent to both v_i and v_{i+1}, which is in the same clique as v_{i+1}, but the First Fit strategy will falsely put in the same clique as v_i.

Both of the cases are illustrated in Figure 4. A typical example of the first case was depicted on the left and in the middle. The second case can be illustrated by a two inter-clique edge scenario on the right.

These are the most basic worst-case scenarios for the GCC over two triangles. Now, let us move on to a formalization of these concepts.

Lemma 2.
Let $G = [V, E]$ be a graph with planted cliques for $\vartheta = \frac{|V|}{r} = 2$ and $|E|_{out} < r$. Then, GCC will be off by at most $r-1$ cliques.

Proof.
Suppose that $r = 3$. Then, there are $r^2 = 9$ inter-clique vertex couples. With the restriction that $|E|_{out} < r$, there are only 46 possible graphs and for each graph, there are $6! = 720$ permutations of vertices. This amount can be verified using an exhaustive evaluation. A typical example of the worst case is the two inter-clique edge scenario we discussed above. For this case, we have already shown that GCC uses at most $|V|/r + r - 1 = 4$ cliques.

We proceed by induction. By adding a new vertex into each of the cliques, one will obtain an analogous graph, for which, without adding a new inter-clique edge, GCC will be off by $r-2$ cliques. Similarly, the worst case here can be constructed by introducing a new inter-clique edge between the new vertices. This would cause GCC to be off by $r-1$ cliques. The non-existence of worse cases can be similarly verified using exhaustive evaluation of the possible positions of the new inter-clique edge. $\square$

Lemma 3.
Let $G = [V, E]$ be a graph with planted cliques for $\vartheta = \frac{|V|}{r} = 2$ and $|E|_{out} < r$. Then, for each clique covering generated by GCC, a random reordering of its cliques will lead to the optimum with probability at least $\frac{1}{|V|/r+r-1}$.

Proof.
As we have shown, for this type of graphs, GCC cannot use more than $|V|/r + r - 1$ cliques. Therefore, there are $(|V|/r + r - 1)!$ possible reorderings. The property that $|E|_{out} < r$ is important, since it implies that, in the current solution, there must be at least one clique that is inside one of the expected cliques.

An important property of the reordering operators is that, when the reordering starts with a clique, that is a subclique of any optimal one (i.e. does not contain any inter-clique edge and it does not contain all vertices of the expected clique), then the reordering will surely lead to a correct labeling of the two cliques being involved.

Since there surely is such a clique, we put it first and reorder the rest in any way. There are $\frac{(|V|/r+r-2)!}{(|V|/r+r-1)!}$ such permutations. Therefore, the probability of a direct jump to the optimum is at least $\frac{1}{|V|/r+r-1}$. $\square$

Theorem 2.
Let $G = [V, E]$ be a graph with planted cliques for $\vartheta = \frac{|V|}{r} = 2$ and $|E|_{out} < r$. Then, IG with GCC and random reorderings will converge to the optimal solution in $\mathcal{O}(|V|^3)$ time.

Proof.
The expected running time of IG on graphs with planted cliques is $\mathcal{O}(I|V|^2)$. By using Lemma 1, the number of iterations is upper bounded by $I(|V|) = |V|/r+r-1 = |V|/2+1$. Therefore, the running time of IG is $\mathcal{O}(|V|^3)$. $\square$

5.3 Analysis of IG on Other Cases of Graphs with Planted Cliques

The properties that $\vartheta = 2$ and $|E|_{out} < r$ lead us always to graphs, for which GCC will leave some clique to be a subclique of an expected one. This is an important property, which we can use in the running time analysis of IG.

Suppose that we have two triangles bridged by 3 inter-clique edges (each connecting a different couple). It can be easily seen that if GCC constructs a covering consisting of the three inter-clique edges, then any reordering of the initial cliques would always result in a suboptimal solution. This graph is an example of instance, for which $I = \infty$ in the worst case and the generic IG would not converge to the optimum. However, such problems can be easily overcome, e.g. by extending the clique-based reordering with a random swap of vertices in the resulting permutation. On the other hand, such extensions would make the analysis of IG even more complicated. Therefore, since these situations do not practically occur very often, we aim our attention at the more favorable scenario, when IG can improve the covering.

There are other subclasses of graphs with planted cliques, for which polynomial time upper bounds for the IG seem to be obtainable. For example, for graphs, that consist of several disjunct sparse biclique graphs, there clearly are $\vartheta/2 = |V|/2r$ fitness-based partitions. When we fix r to a constant, then we obtain an $\mathcal{O}(|V|^4)$ upper bound of the complexity.

The conditions, under which we are able to obtain an upper bound for IG analytically, can be summarized in the following theorem.

Theorem 3.

Let $G = [V, E]$ be a graph with planted cliques $K_{r,1}, K_{r,2}, ..., K_{r,|V|/r}$. Suppose that $S_i = \{V_{1,i}, V_{2,i}, ..., V_{k_i,i}\}$ is the current clique covering at the i-th iteration of IG. Furthermore, suppose that at each iteration i, there are j and m, such that there is a clique $G(V_{k_i,j}) \in S_i$, which is a subgraph of some expected clique $K_{r,m}$ $(G(V_{k_i,j}) \neq K_{r,m})$. Then, IG with GCC and random reorderings will converge to the optimal solution in $\mathcal{O}(|V|^4)$ time.

Proof.

At each iteration, there is a clique $G(V_{k_i,j})$ that is a subgraph of some of the expected cliques. Therefore, at each iteration, by putting the vertices of $V_{k_i,j}$ first in the permutation, GCC will be able to generate the whole expected clique $K_{r,m}$, because its intra-clique edges will precede the inter-clique edges being involved. This implies that each of the expected cliques is constructed in one step of IG. Thus, there are at most $|V|/r + 1$ fitness-based partitions of such landscape.

To keep the proof simple, we upper bound the current number of cliques in i-th iteration by $k_i \leq |V|$. Thus, there are at most $(|V|)!$ permutations and the probability of fitness augmentation is at least $\frac{(|V|-1)!}{(|V|)!} = \frac{1}{|V|}$. By using Lemma 1, we obtain that the number of iterations is upper bounded by $|V|^2/r$ and therefore, IG converges to the optimal solution in $\mathcal{O}(|V|^4)$ time. □

It is interesting to see that these analytical results tend to correlate well with the empirical evidence. The number of iterations really shows signs of being upper bounded by a polynomial, most probably of a second degree. Secondly, a factor of $|V|/r$ really seems to play a role in the number of fitness-based partitions of the landscape. The trend displayed in Figure 3 corresponds to this result, the number of iterations increases with $|V|$ and decreases with r.

Finally, we emphasize the fact that the complexities of IG we obtained, were never exponentially dependent on r for these sparse graphs. To obtain a polynomial-time complexity with an exact algorithm, we would have to fix r to a constant. With IG, this does not seem to be necessary. In addition, we have empirically demonstrated that the time needed by IG decreases with r, while in the exact algorithm, it is reversed. We also emphasize the fact that whenever, for a subclass of graphs with planted cliques, we can prove the $\mathcal{O}(|V|^4)$ upper bound analytically, it implies that IG outperforms the exact approach for $r > 4$.

6. EXPERIMENTAL EVALUATION

In this section, we briefly present experimental results of IG on several synthetic and real-world graphs[1]. Table 1 shows the experimental results of IG with GCC and 2 other approaches on several graphs we have chosen for the experimental evaluation. The Erdős-Rényi uniform random graphs are the most typical random graphs, generated by iterating through all couples of vertices and assigning an edge with some probability. The Leighton graphs from DIMACS instances [8] are instances, consisting of embedded cliques of different sizes[2]. These graphs are abstractions of large scheduling problems [11]. Last but not least, we have also used a sample of data from a local social network, to evaluate the performance of IG on some real-world data in the complex network domain.

The first column of the table contains the name of the graph, the other columns contain the minimal numbers of cliques obtained by the methods over 30 independent runs. BRE denotes the Brélaz's graph coloring heuristic we discussed in Section 2. This method was in fact applied to the complementary graph (using implicit mechanisms). The SAT-GCCH is a saturation-based version of GCC, in which, after assignment of a vertex into a clique, for each of its neighbors, which are adjacent to all of the vertices of the current clique, its saturation is incremented. Finally, the IG-GCC method is the IG using GCC with 50 : 30 ratio of reverse and random mutations of the clique orderings. This method was limited to 10^4 iterations.

These results demonstrate, how strongly IG outperforms the related approaches. On the random and Leighton graphs, IG shows major improvements, without fixing the number of cliques to a constant. The results on the social network data are also encouraging. Despite the fact that the difference is relatively small, IG still is able to achieve an improvement in the estimate of ϑ. These results emphasize the practical impact of this work.

7. CONCLUSION AND DISCUSSION

In this paper, we presented two contributions to the fields of heuristics and evolutionary algorithms for a classical, but still overlooked NP-hard problem [9] - the (vertex) clique covering problem (CCP).

We proposed a genotype-phenotype mapping algorithm for an order-based representation of the clique covering problem (CCP). For this algorithm - the greedy clique covering (GCC), we proved that for an arbitrary graph, there is a permutation, for which GCC constructs the optimal solution. We also showed that GCC is faster than the existing approach - greedy graph coloring, for sparse graphs. In our cur-

[1]All instances are publicly available at: `http://www.fiit.stuba.sk/~chalupa/benchmarks/ccp`.

[2]In Table 1, we present the approximations of $\vartheta(G) = \chi(\overline{G})$ for all graphs, including Leighton graphs. These numbers are different from the approximations of $\chi(G)$, which can be found in the graph coloring literature.

rently best implementation, this occurs for graphs with density less than approximately $\frac{4}{21}$. In addition, GCC always uses $\mathcal{O}(|V|)$ additional space, instead of $\mathcal{O}(|V|^2)$ needed by the greedy graph coloring. Therefore, it is highly suitable for large sparse graphs, which are typical for many real-world applications, i.e. in data mining [16], web engineering and analysis of clustered graphs [15].

In addition, we adapted a mutation-based stochastic metaheuristic approach to the CCP - iterated greedy (IG). Following-up the work by Culberson and Luo [3], we investigated the convergence ability of IG on graphs with planted cliques. We obtained interesting empirical evidence, showing that if the graph is sparse enough, IG practically guarantees the convergence to the optimum on a large spectrum of these graphs, without an evidence of an exponential outbreak in the complexity. We investigated this complexity analytically and, using the methods of evolutionary algorithm analysis, we found upper bounds for several specific types of graphs with planted cliques, mostly of $\mathcal{O}(|V|^4)$. With these results, we supported the empirical evidence that IG outperforms the exact algorithm, described in Section 4.2, on this type of graphs.

We also presented experimental evaluation of IG and its comparison to related existing approaches on random graphs, Leighton graphs and social graphs. The results show that IG in combination with GCC is a very promising method in handling large sparse graphs, regarding both the quality of results and the time needed to obtain them.

Table 1: The comparison of the approximations of $\vartheta(G) = \chi(\overline{G})$ for each graph obtained by the Brélaz's heuristic (BRE), saturation-based GCC (SAT-GCC) and the IG heuristic with GCC (IG-GCC).

G	BRE	SAT-GCC	IG-GCC
Erdős-Rényi uniform random graphs			
$unif$1000_0.1	299	310	**243**
$unif$5000_0.1	1241	1288	**1066**
$unif$10000_0.1	2326	2389	**2025**
$unif$20000_0.01	7640	7817	**6387**
Leighton graphs from DIMACS instances.			
le450_15a	85	89	**80**
le450_15b	92	90	**82**
le450_15c	68	74	**57**
le450_15d	73	73	**57**
le450_25a	**91**	92	**91**
le450_25b	81	82	**80**
le450_25c	61	59	**54**
le450_25d	60	59	**51**
Social graphs			
soc2000	**1471**	1473	**1471**
soc10000	6619	6633	**6618**
soc20000	12770	12804	**12764**

Acknowledgement

The author would like to thank prof. Jiří Pospíchal for his very helpful comments and the anonymous referees, whose remarks helped to improve this work. This contribution was supported by Grant Agency VEGA SR under the grant 1/0553/12.

8. REFERENCES

[1] D. Brélaz. New methods to color vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (3rd ed.).* MIT Press, 2009.

[3] J. C. Culberson. Iterated greedy graph coloring and the difficulty landscape. Technical report, University of Alberta, 1992.

[4] J. C. Culberson and F. Luo. Exploring the k-colorable landscape with iterated greedy. In *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, pages 245–284. American Mathematical Society, 1996.

[5] N. Drechsler, W. Günther, and R. Drechsler. Efficient graph coloring by evolutionary algorithms. In *Proceedings of the 6th International Conference on Computational Intelligence, Theory and Applications: Fuzzy Days*, pages 30–39. Springer, 1999.

[6] P. Galinier and J. K. Hao. Hybrid Evolutionary Algorithms for Graph Coloring. *Journal of Combinatorial Optimization*, 3(4):379–397, 1999.

[7] P. Galinier and A. Hertz. A survey of local search methods for graph coloring. *Computers & Operations Research*, 33(9):2547–2562, 2006.

[8] D. S. Johnson and M. Trick. *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge.* American Mathematical Society, 1996.

[9] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[10] M. Laguna and R. Martí. A grasp for coloring sparse graphs. *Computational Optimization and Applications*, 19(2):165–178, 2001.

[11] F. T. Leighton. A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards*, 84(6):489–503, 1979.

[12] Z. Lü and J. K. Hao. A Memetic Algorithm for Graph Coloring. *European Journal of Operational Research*, 203(1):241–250, 2010.

[13] F. Neumann and C. Witt. *Bioinspired Computation in Combinatorial Optimization - Algorithms and Their Computational Complexity.* Springer, 2010.

[14] D. C. Porumbel, J. K. Hao, and P. Kuntz. An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring. *Computers & Operations Research*, 37(10):1822–1832, 2010.

[15] S. E. Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007.

[16] J. Sun, Y. Xie, H. Zhang, and C. Faloutsos. Less is more: Sparse graph mining with compact matrix decomposition. *Statistical Analysis and Data Mining*, 1(1):6–22, February 2008.

[17] D. J. A. Welsh and M. B. Powell. An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal*, 10(1):85–86, 1967.