Noname manuscript No.
(will be inserted by the editor)

A new heuristic for finding verifiable k-vertex-critical subgraphs

Alex Gliesch · Marcus Ritt

the date of receipt and acceptance should be inserted later

Abstract Given graph G, a k-vertex-critical subgraph (k-VCS) $H \subseteq G$ is a subgraph with chromatic number $\chi(H) = k$, for which no vertex can be removed without decreasing its chromatic number. The main motivation for finding a k-VCS is to prove k is a lower bound on $\chi(G)$. A graph may have several k-VCSs, and the k-Vertex-Critical Subgraph Problem asks for one with the least possible vertices. We propose a new heuristic for this problem. Differently from typical approaches that modify candidate subgraphs on a vertex-by-vertex basis, it generates new subgraphs by a heuristic that optimizes for maximum edges. We show this strategy has several advantages, as it allows a greater focus on smaller subgraphs for which computing χ is less of a bottleneck. Experimentally the proposed method matches or improves previous results in nearly all cases, and more often finds solutions that are provenly k-VCSs. We find new best k-VCSs for several DIMACS instances, and further improve known lower bounds for the chromatic number in two open instances, also fixing their chromatic numbers by matching existing upper bounds.

Keywords critical subgraphs · graph coloring · heuristic algorithm · iterated tabu search

1 Introduction

A *coloring* $c : V \to \mathbb{Z}$ of an undirected graph $G = (V, E)$ is a function that assigns a color to each vertex, and a *k-coloring* is a coloring which uses exactly k distinct colors, i.e. $|c(V)| = k$. A coloring is *valid* if $c(i) \neq c(j)$ for all $\{i, j\} \in E$. If there exists a valid k-coloring for G, it is *k-colorable*. The *chromatic number* $\chi(G)$ of a graph G is the smallest k for which G is k-colorable. A graph G is *k-chromatic* if $\chi(G) = k$. The problem of finding a coloring witnessing the chromatic number is known as the *Vertex Coloring Problem* (VCP) and, the *k-Vertex Coloring Problem*

Alex Gliesch · Marcus Ritt
Federal University of Rio Grande do Sul, Brazil
E-mail: alex.gliesch,marcus.ritt@inf.ufrgs.br

asks for a valid k-coloring given some integer k. Both are NP-hard (Garey and Johnson 1979). Several exact as well as heuristic solution methods have been proposed to solve them (see Malaguti and Toth (2010) for an excellent overview). In practice, heuristic methods can usually find upper bounds on χ for instances of several thousands of vertices, but exact methods are limited to instances of about 100 vertices in reasonable time (Malaguti and Toth 2010).

A *k-vertex-critical subgraph* (k-VCS, for short) of a graph G is a subgraph $H \subseteq G$ such that $\chi(H) = k$ and $\chi(H') < k$ for all $H' \subset H$. Every graph G has a $\chi(G)$-critical subgraph (Hajós 1961). The *k-Vertex-Critical Subgraph Problem* (k-VCSP) asks for the k-VCS $H^* \subseteq G$ of a given graph G with the least number of vertices, and is NP-hard (Desrosiers et al. 2008). Given vertex $v \in G$, we say v is *k-critical* in G if G has a k-critical subgraph that contains v. Note that a k-critical vertex is not necessarily part of every k-VCS.

The main reason to search for k-VCS is to obtain lower bounds on the chromatic number: if there exists a k-VCS for graph G, then $\chi(G) \geq k$ (Hajós 1961). If k is also equal to an upper bound obtained by some heuristic coloring algorithm, then G must be k-chromatic. The advantage of this approach over simply using an exact algorithm to compute χ is that k-VCSs are typically much smaller than the original graphs, and are often solvable by exponential-time exact algorithms when the original graphs are not.

Herrmann and Hertz (2000) were the first to propose algorithms for the k-VCSP with the goal of finding the chromatic number χ. Their method iteratively removes vertices from the input graph while its chromatic number (computed by the heuristic coloring algorithm TabuCol (Hertz and Werra 1987)) remains unchanged, and stops when no further vertices can be removed. It then uses an exact algorithm to solve the $(k-1)$-VCP on the resulting subgraph; if it proves that no such coloring exists, then k is a lower bound on χ. Otherwise, it continues by iteratively adding vertices to the solution until its chromatic number is proved to be k.

Finding a critical subgraph is equivalent to finding an *irreducible inconsistent set* (IIS) of variables in *constraint satisfaction problems* (CSPs) (Tsang 1993), as shown by Galinier and Hertz (2007). An IIS of variables of a CSP is an infeasible set of variables which becomes feasible once any variable is removed. Desrosiers et al. (2008) proposed three algorithms for finding vertex-critical as well as edge-critical graphs (where deleting any edge causes χ to decrease): "insertion", "removal" and "hitting set", extending the work of Galinier and Hertz (2007) on finding IISs for CSPs towards graph coloring. The algorithms iteratively solve the so-called "minimum partial legal weighted k-coloring" subproblem to identify a set of vertices which must belong to at least one k-VCS, then update vertex weights to indicate their insertion or removal correspondingly. The coloring subproblem is solved for the full input graph G at each iteration. If chromatic numbers can be computed exactly, the authors show that removal-based algorithms are guaranteed to find a k-VCS. Further, the method uses a pre-filtering mechanism which solves up to $|V|$ coloring subproblems to exclude vertices which cannot be part of any k-VCS, as well as a neighborhood weight heuristic that biases vertex selection towards denser solutions. In experiments they report to have improved known lower bounds for 5 instances of the DIMACS (Johnson and Trick 1996) data set.

Sun et al. (2019) propose an extension of the removal algorithm of Desrosiers et al. (2008). The method uses a backtracking procedure to correct false positives that occur when the heuristic coloring algorithm fails to provide a $(k-1)$-coloring of a given subgraph, thus classifying it as a k-VCS, when such a coloring does in fact exist but was not found by the heuristic. In this case, the backtracking procedure reinserts previously removed vertices into the subgraph until a k-VCS is detected. The algorithm also uses a stochastic perturbation scheme as a diversification mechanism. In experiments, the method is reported to improve results from Desrosiers et al. (2008) in almost all instances, and improve best known lower bounds for six DIMACS instances. However, these lower bounds cannot be verified since they rely on the results of the heuristic.

Recently, de Grey (2018) made a breakthrough in the Hadwiger-Nelson problem (Soifer 2009), which asks for the chromatic number of the plane. He was able to increase the lower bound from 4 to 5 colors by providing a family of unit-distance plane graphs which are not 4-colorable. The original counterexample has 20425 vertices; de Grey (2018) further found a 5-vertex-critical subgraph of 1581 vertices using computational simplification techniques. Since then, a polymath project (Polymath Wiki contributors 2020) has been created with the goal of finding even smaller counter-examples (i.e., 5-VCSs) of the so-called "de Grey graph". Heule (2019a) found a witness of size 553 by applying clausal minimization methods in a SAT formulation of the vertex coloring problem for the de Grey graph, and Heule (2019b) reduced the witness further to 529 vertices using a similar method, though with significant computational effort. At the time of writing, the best result is by Jaan Parts (Parts 2019) with 510 vertices.

1.1 Contributions of this paper

The major shortcoming of existing methods for the k-VCSP is that they rely on heuristics for the k-VCP and, as observed by Sun et al. (2019) and noted earlier, these heuristics often report false positives. This may lead removal-based algorithms to incorrectly discard too many vertices, and return subgraphs whose chromatic numbers are less than k. Because heuristics are inexact, these methods also do not guarantee that solutions obtained are critical or even k-chromatic, unless confirmed by an exact coloring algorithm afterwards. Another issue with heuristics is that, if no valid k-coloring exists, a heuristic method will typically exhaust all of its allocated resources (either a time limit or a maximum number of operations) before providing an inconclusive answer, which is generally undesirable. Finally, methods based on vertex removal such as Herrmann and Hertz (2000) and Sun et al. (2019) optimize from the input graph downwards, and so must solve the VCP for the full graph G in early iterations which, for larger instances, often cannot be done effectively within reasonable time.

In this paper we propose a method for the k-VCSP that addresses the above issues. In a first phase it tries to obtain an initial upper bound by generating subgraphs of geometrically increasing sizes, and, in a second phase, iteratively generates subgraphs

of the same size as the upper bound, and improves them by vertex removal search. Our main contributions are:

1. A bottom-up heuristic that quickly generates k-chromatic subgraphs of a target size from the ground up. It maximizes edge density as a proxy objective. We show that this is often similar to optimizing for χ itself, but much faster. This has a number of advantages over existing approaches which grow or shrink solutions one vertex at a time and must solve a VCP at each iteration. In particular, a bottom-up strategy lets us focus first on smaller subgraphs which are easier to color optimally, and only rely on heuristics in later stages. To our knowledge, this is the first method to use an alternative fitness function to build subgraphs in the context of critical coloring.
2. A hybrid strategy for computing chromatic numbers that chooses between an exact or a heuristic algorithm, depending on the input subgraph. It computes χ exactly in solutions when this can be done quickly, and in this way avoids potential false positives due to the inexactness of heuristics. We show that, due to the hardness of the VCP, this allows the proposed method to avoid false positives in almost all cases where one can prove the solution is k-chromatic. In contrast, existing methods are highly susceptible to this effect. Moreover, a hybrid strategy often yields solutions which are verifiably k-chromatic, making ours the only available implementation that does this without relying on a post-hoc check.
3. A fine-tuning step based on vertex removal search that is run on each subgraph found which, when χ can be computed exactly, is guaranteed to obtain a k-VCS. While vertex removal is not a new concept, in the second phase of our method we exploit the speed and randomness of our subgraph generation procedure to execute removal search on a very large number of candidate subgraphs, and thus increase our likelihood of reaching better local minima.
4. Two novel pre-processing techniques, including the detection of maximum cliques, allowing us to solve within milliseconds a number of benchmark instances that other methods often struggle with.

Experimentally the proposed method consistently finds verifiably k-chromatic subgraphs more often than other approaches, and with generally fewer vertices. On the DIMACS data set we were able to witness the current best lower bounds on χ in 108 instances through k-VCSs, and have found the best known k-VCSs for 37 instances. We have further improved the best known lower bounds on the chromatic number χ for two instances, 4-FullIns_5 and abb313GPIA, also fixing their optimal chromatic numbers since the new lower bounds match existing upper bounds from the literature. In an ablation study, we see that each major component of our method contributes to some degree to its effectiveness.

In the following section we describe our main algorithm in detail, as well as each of its components. Section 2.1 explains a pre-processing step which is executed prior to the main algorithm, followed by Section 2.2 which overviews the proposed algorithm, and Sections 2.3–5 detailing its three main components: the heuristic used to generate candidate subgraphs, the hybrid strategy for computing chromatic numbers, and the removal search step. Next, in Section 3 we calibrate the parameters of our

method experimentally and report on further experiments that assess its effectiveness. We conclude in Section 4.

2 Proposed method

In what follows, we say that a graph G' is an s-subgraph of graph G if $G' \subseteq G$ and $|V(G')| = s$. For brevity, we use the notation $|G|$ to denote the number of vertices of a graph G, and n for the size of the input graph. Further, for a given positive integer i we use the notation $[i] = \{1, \ldots, i\}$.

2.1 Pre-processing

We start by trying to reduce the input graph G by removing vertices of degree smaller than $k - 1$. As shown by Brooks (1941), such vertices cannot be k-critical. Because the removal of a vertex can reduce the degree of its neighbors, we repeat this process iteratively until no further vertices can be removed. In instances of the DIMACS data set, this allows us to reduce the number of vertices of 59 instances by about 40%, on average. In the rest of this paper, we always assume input graph G has already undergone this step.

Next, we search for a k-clique in G. If we find one, then the clique itself clearly is a minimal k-VCS and is therefore optimal. In practice, this step alone is enough to solve 71 out of the 136 DIMACS graphs in a matter of milliseconds, since many of them are weakly perfect (i.e., $\omega(G) = \chi(G)$, where the *clique number* $\omega(G)$ is the size of the maximum clique in G)) and maximum cliques can be computed very efficiently. In our implementation, we use the heuristic algorithm of Wu et al. (2012) to find cliques, executed with a time limit of 2 seconds and a fixed seed.

2.2 Main algorithm

As mentioned previously, we compute vertex colorings with a hybrid procedure that, depending on the subgraph size, uses either an exact or a heuristic coloring algorithm. Since the latter may not always yield optimal results, in the following we denote the resulting number of colors by $\chi_h(H)$ for a given subgraph $H \subseteq G$, to indicate that it may not be a true chromatic number. We explain in detail how $\chi_h(H)$ is computed in Section 2.5.

Algorithm 1 outlines our main method. It has two phases. In the first phase, we try to obtain an initial k-VCS by considering different subgraph sizes in an geometric progression. Starting with candidate size $s_0 = k + 2$, at each iteration we increase the candidate size by a factor μ, up to n, where μ is a parameter; i.e., $s_i = \min\{n, (k + 2)\mu^i\}$, for $i \geq 0$. For each size s_i we run a randomized heuristic GenerateSubgraph(s_i) that tries to generate a subgraph $H \subseteq G$ with s_i vertices such that $\chi_h(H) \geq k$. If GenerateSubgraph cannot find such a subgraph, we continue with the next candidate size s_{i+1}. Otherwise, we search the interval $(s_{i-1}, s_i]$ via binary search, for the least s^* such that GenerateSubgraph can find a k-chromatic subgraph.

Algorithm 1 Main algorithm overview.

Input: A graph G, and an integer $k \leq \chi(G)$.
Output: A subgraph $S \subseteq G$ with $\chi_h(S) \geq k$.

for $s_i = k+2, \mu(k+2), \mu^2(k+2), \ldots, n$ **do** ▷ first phase
  $H \leftarrow$ GenerateSubgraph(G, s_i, k)
  if $\chi_h(H) \geq k$ **then**
    $s \leftarrow \min\{s_{i-1} < s \leq s_i \mid \chi_h(\text{GenerateSubgraph}(G, s, k)) \geq k\}$ by binary search
    $S_{gen} \leftarrow$ GenerateSubgraph(G, s, k)
    break
$S \leftarrow$ IterativeRemoval(S_{gen})
while $|S| > k+2$ **and** time limit not reached **do** ▷ second phase
  $H \leftarrow \emptyset, s = \min(n, \lceil \xi |S_{gen}| \rceil)$
  while $s \geq k+2$ **do**
    $H' \leftarrow$ GenerateSubgraph(G, s, k)
    $H \leftarrow H'$ **if** $\chi_h(H') \geq k$; **else break**
    $s \leftarrow s - 1$
  if $H \neq \emptyset$ **then**
    $S_{gen} \leftarrow \arg\min |H|, |S_{gen}|$
    $S \leftarrow \arg\min |\text{IterativeRemoval}(H)|, |S|$
return S

Once s^* is found the first phase ends. In the worst case, if GenerateSubgraph fails for $s_i = n$, the algorithm returns V and halts. This is unlikely, however, and only occurs due to the inexactness of χ_h, since the pre-processing phase guarantees k is a lower bound. Heuristic GenerateSubgraph is explained in Section 2.3.

This first phase focuses only on subgraphs smaller than $s^*\mu$, and avoids larger subgraphs for which computing χ_h is difficult. For small enough s^*, in practice χ_h can usually be computed by an exact algorithm, and thus $\chi_h = \chi$, giving a correctness guarantee during execution. Note that we do not apply the binary search from the start, as this would require computing χ_h for large subgraphs of size close to n. We start from $s_0 = k+2$ as this is the smallest possible size for a k-VCS: a k-clique is already considered during pre-processing, and no k-VCS of size $k+1$ can exist (Dirac 1952). We also stop whenever any k-chromatic subgraph of size $k+2$ is found, as it must be optimal.

Let now S_{gen} hold the current best solution obtained from GenerateSubgraph, and S hold the globally best solution. As a fine-tuning step, a procedure IterativeRemoval is applied on S to further reduce it. As the name suggests, it iteratively tries to remove vertices from S while maintaining $\chi_h(S) \geq k$. If $\chi_h = \chi$, IterativeRemoval also returns a guarantee that S is a k-VCS and cannot be reduced further. We detail IterativeRemoval further in Section 2.4.

The second phase tries to improve S until a time limit is reached, by exploiting the randomness of GenerateSubgraph to generate a large variety of candidates in a multi-start fashion. Each iteration starts with target size $s = \lceil \xi |S_{gen}| \rceil$ for GenerateSubgraph, where $\xi > 1$ is a parameter, and repeatedly decrements s by one until we have $\chi_h(\text{GenerateSubgraph}(s-1)) < k$. The algorithm then executes IterativeRemoval on the last s-subgraph generated, updates S and S_{gen}, and starts the next iteration.

Here, we start each iteration at slightly higher target $s = \lceil \xi |S_{gen}| \rceil$, rather than simply $s = |S_{gen}|$, as we have found IterativeRemoval to exhibit a significant vari-

ance across different inputs, with smaller subgraphs not always leading to the smallest solutions. Considering a slightly wider range of sizes therefore provides more variability and usually helps to find a globally best solution. In Section 3 we calibrate parameter ξ experimentally and show that a larger value is indeed more effective.

2.2.1 Caching

Although procedure GenerateSubgraph is randomized, multiple executions with the same target size s can often yield the same subgraph, if they happen to converge to the same local minimum. This can happen especially if n or s are small. In order to avoid executing IterativeRemoval several times on the same graph, we maintain in the second phase a hash table of all subgraphs produced by GenerateSubgraph. If a duplicate is detected, that subgraph is skipped and GenerateSubgraph is run again. Despite IterativeRemoval also being non-deterministic due to the random seeds in the heuristic graph coloring algorithm, it is a performance bottleneck, and in practice we have found that the computational effort is better spent exploring new solutions. The number of calls to GenerateSubgraph is expected to be small (on average about 50 calls per minute experimentally in our environment), so maintaining this cache is not memory-critical.

Further, instead of discarding subgraphs generated by GenerateSubgraph in the second phase once a smaller one is found, we maintain them in a table C which holds, for each size s, at most one subgraph $C_s = \text{GenerateSubgraph}(s)$ such that $\chi_h(C_s) \geq k$ and for which $\text{IterativeRemoval}(C_s)$ has never been run. In each call to $\text{GenerateSubgraph}(s)$, if C_s is not empty we retrieve the subgraph C_s instead, and clear it.

2.3 Generating k-chromatic subgraphs of a target size

In this section we explain procedure GenerateSubgraph, which, given a target size s, searches for an s-subgraph H of G with $\chi_h(H) \geq k$. In practice, since no upper bound on $\chi_h(H)$ is required, GenerateSubgraph simply searches for an s-subgraph with as high a chromatic number as possible.

GenerateSubgraph is a local search procedure. We have observed that χ_h itself is not an ideal objective for a neighborhood search that aims for high chromatic numbers, since typical vertex swap neighborhoods do not have landscapes with clear improving paths and, more importantly, χ_h is expensive to compute. According to Smith-Miles et al. (2014), the three properties of graphs which cause the greatest disparity in the performance of state-of-the-art graph coloring algorithms are density, algebraic connectivity and energy. In preliminary experiments we have found that, out of these three, the graph density was the best predictor of high chromatic numbers, while also having the advantage of being efficient to compute and to maintain dynamically under small modifications to the graph. Therefore we maximize graph density.

In the literature, the problem of finding a subgraph of a fixed size k with the most edges is known as the *Densest k-Subgraph Problem*. It is NP-hard (Feige and Seltser

Algorithm 2 Multistart iterated tabu search method for finding k-chromatic subgraphs of a target size.

Input: a graph G, a target size s, and a number of colors $k \leq \chi(G)$.
Output: a subgraph $H \subseteq G$ with $|H| = s$ and $\chi_h(H) \geq k$, or $\emptyset$, if no such subgraph is found.

```
1: procedure GENERATESUBGRAPH(G, s, k)
2:     for i ∈ [R] do
3:         H ← Constructive(G, s)
4:         H ← IteratedTabu(G, H)
5:         return H if χ_h(H) ≥ k
6:     return ∅
```

1997). Recent works have mostly focused on approximation algorithms (Bourgeois et al. 2017; Bhaskara et al. 2010; Manurangsi 2017), though Chang et al. (2014) proposed an exact approach in time $O^*(1.7315^n)$ and Brimberg et al. (2009) proposed a variable neighborhood search heuristic for the *Heaviest k-Subgraph Problem*, the edge weighted version, which reduces to the Densest k-Subgraph Problem if all weights are equal.

Procedure GenerateSubgraph is outlined in Algorithm 2. It is a multistart iterated tabu search. Given target size s, it executes at most R iterations, where R is a parameter. In each iteration it constructs an initial s-subgraph H using a greedy constructive heuristic Constructive, and then improve it by an iterated tabu search procedure IteratedTabu. Both procedures aim for high-density subgraphs. If $\chi_h(H) \geq k$, GenerateSubgraph stops and returns H; otherwise, it continues with the next iteration. If no k-chromatic subgraph is found after R iterations, failure is indicated by returning an empty graph. Note that this strategy only executes χ_h once per subgraph generated, which is desirable since it is computationally expensive.

The two following subsections explain algorithms Constructive and IteratedTabu. For shortness, given a subgraph $H \subseteq G$ we use the notation $H+v$ for $G[V(H) \cup \{v\}]$, and $H - v$ for $G[V(H) \setminus \{v\}]$.

2.3.1 Constructing subgraphs

We have considered two constructive heuristics, *add* and *drop*, shown in Algorithm 3. Both are based on an α-greedy approach (Feo and Resende 1995). Heuristic *add* starts with a single vertex chosen by a random selection where the chance of each vertex being chosen is proportional to its degree, and iteratively includes new vertices until the desired size s is reached. At each iteration, given the current subgraph H it selects uniformly at random a vertex $v \in V \setminus H$ for which the relative deviation $(b - \mathrm{E}(H + v))/b$ from the best increase $b = \max\{\mathrm{E}(H + v) \mid v \in V \setminus H\}$ is less than a tolerance α. The *drop* heuristic works similarly, but on the other direction: starting with $H = V$ it iteratively removes vertices from it. At each iteration it selects a vertex $v \in H$ uniformly at random for which $(b - \mathrm{E}(H - v))/b < \alpha$ holds, where $b = \max\{\mathrm{E}(H - v) \mid v \in H\}$. In both heuristics, values E can be computed in amortized constant time by maintaining, for each vertex, the resulting number of edges were it to be selected. These values are updated in time $O(n)$ for every n candidate evaluations.

Algorithm 3 Constructive algorithm for dense subgraphs.

Input: a graph G, a target size s, and a number of colors $k \leq \chi(G)$.
Output: a subgraph $H \subseteq G$ with $|H| = s$; ideally E(H) is as high as possible.

procedure CONSTRUCTIVE(G, s, k)
  $(\underline{s}, \overline{s}) \leftarrow (k + 2 + \rho, n - \rho)$
  add $\leftarrow$ true with prob. $(\rho - s)/(n - 2\rho)$, false, otherwise
  if $s < \underline{s}$ or ($s \in [\underline{s}, \overline{s}]$ and add) **then** ▷ *add* strategy
    choose a random vertex $v \in V(G)$ with probability $\propto \delta(v)$
    $H \leftarrow G[\{v\}]$
    while $|H| < s$ **do**
      $b \leftarrow \max\{\mathrm{E}(H + v) \mid v \in V \setminus H\}$
      $u \leftarrow$ a unif. rand. vertex from $V \setminus H$ s.t. $(b - \mathrm{E}(H + u))/b < \alpha$
      $H \leftarrow H + u$
  else ▷ *drop* strategy
    $H \leftarrow G$
    while $|H| > s$ **do**
      $b \leftarrow \max\{\mathrm{E}(H - v) \mid v \in H\}$
      $u \leftarrow$ a unif. rand. vertex from H s.t. $(b - \mathrm{E}(H - u)/b < \alpha$
      $H \leftarrow H - u$
  return H

In practice both strategies produce solutions of similar quality, but can be quite different when it comes to running time. While *add* is fast for small s, its performance drops significantly for larger s, since it requires $O(ns)$ steps; on the other hand, by the same reasoning *drop* is efficient for large target sizes and less so for smaller ones, as it requires $O(n(n-s))$ steps. To mitigate this, we introduce a parameter ρ such that *add* is used if s is among the smallest ρ possible target sizes (i.e. $s < k + 2 + \rho$) and *drop* if s is among the ρ largest possible target sizes (i.e. $s > n - \rho$). For s in between these limits *drop* is chosen with probability $p_d = (s + \rho - n)/(k + 2 - n + 2\rho)$, and *add* with probability $1 - p_d$. To reduce the number of parameters to be tuned, in our implementation we have fixed $\rho = 0.25(n - k - 2)$ (i.e. 25% of the possible values for s), as we have found it to work well in most cases.

2.3.2 Optimizing subgraph density by iterated tabu search

Given an s-subgraph H obtained by the constructive algorithm above, we use the iterated tabu search heuristic IteratedTabu shown in Algorithm 4 to further improve its density. IteratedTabu extends the variable neighborhood search of Brimberg et al. (2009) for the Heaviest k-Subgraph Problem by a tabu strategy that allows moves that do not improve the objective function and a tabu list to avoid cycling. In Section 3.1 we calibrate the parameters of IteratedTabu, and show that using a tabu strategy is indeed more effective.

Procedure IteratedTabu considers moves that swap a vertex $u \in H$ and a vertex in $v \in H \setminus V$, setting $H \leftarrow H - u + v$. Iteratively, IteratedTabu executes the best neighboring move on the current solution, i.e., the one resulting in the maximum number of edges. Note that this neighborhood maintains $|H|$ constant throughout the search. If multiple such moves exist, one of them is selected uniformly at random. After a move, both vertices are declared *tabu* for the next τ iterations and cannot be selected again,

Algorithm 4 Iterated tabu search algorithm for optimizing the density of a given subgraph.

Input: a graph G, a subgraph $H \subseteq G$, and a number of colors $k \leq \chi(G)$.
Output: a subgraph $H' \subseteq G$ with $|H'| = |H|$ and $\mathrm{E}(H') \geq \mathrm{E}(H)$.

1: **procedure** ITERATEDTABU(H, G, k)
2: $i \leftarrow 0, p \leftarrow 0, H' \leftarrow H$
3: **while** $p < P_{max}$ **do**
4: $u, v \leftarrow \arg\min\{\mathrm{E}(H - u + v) \mid u$ and v are not tabu, $u \in G \setminus H, v \in H, \mathrm{E}(H + v) - \mathrm{E}(H) > k - 2\}$
5: **if** $\mathrm{E}(H - u + v) > \mathrm{E}(H)$ **then**
6: $i \leftarrow 0$
7: $H \leftarrow H - u + v$
8: mark u and v tabu for τ iterations
9: **else**
10: $i \leftarrow i + 1$
11: **goto** line 7 **if** $i < I_{max}$ ▷ do a non-improving move
12: $p \leftarrow p + 1$
13: perform σp random vertex swaps between H and $G \setminus H$
14: $H' \leftarrow \arg\max\{\mathrm{E}(H'), \mathrm{E}(H)\}$
15: **return** H'

where the tabu tenure τ is a parameter. After I_{max} consecutive neighboring moves without improvement to the solution, where I_{max} is a parameter, a perturbation step is applied in an attempt to diversify the search. It consists of performing $(p+1)\sigma$ random swaps on the incumbent, where p is the number of perturbations done so far and σ is a parameter which determines the perturbation strength. The search stops after a maximum P_{max} perturbation steps have been made, and returns the best solution encountered. Following Brimberg et al. (2009) we have set $P_{max} = |H|/\sigma$, that is, IteratedTabu stops when a perturbation would be equivalent to a random restart.

Additionally, IteratedTabu uses a filtering mechanism in order to reduce the size of the neighborhood. Recall from Section 2.1 that vertices of degree smaller than $k - 1$ cannot be critical. With this in mind, it ignores all incoming vertices of degree $k - 2$ or less. Although this may discard moves which could lead to an overall better solution in the long run, in practice this is rarely the case and we have found that filtering these moves out is a good compromise between performance and quality.

As in Constructive, we can determine the expected value of each candidate move in constant amortized time by a dynamic data structure which holds, for each vertex v, either $\mathrm{E}(H + v)$ if $v \notin H$ or $\mathrm{E}(H - v)$ if $v \in H$. It is updated after every move.

2.4 Removal search to fine-tune solutions

This section explains procedure IterativeRemoval, outlined in Algorithm 5, which has two functions. First, to reduce a solution to a critical subgraph, and second to try to prove the criticality of a solution, therefore avoiding to run an exact coloring algorithm as a post-hoc check. The first step is necessary, because the generation of subgraphs in the first phase is guided by the density, and often does not produce a local minimum.

Algorithm 5 Removal search step.

Input: a graph G, a subgraph $H \subseteq G$, and a number of colors $k \leq \chi(G)$.
Output: a subgraph $H' \subseteq G$ with $|H'| \leq |H|$ and $\chi_h(H') \geq k$. If χ_h is exact, H' is guaranteed to be a k-VCS.

```
1: procedure IterativeRemoval(H, G, k)
2:     C ← ∅                                   ▷ C is the set of required vertices
3:     H' ← H
4:     while C ≠ V(H) do
5:         select vertex v ∈ V(H') \ C with smallest neighborhood weight W(v)
6:         if χ_h(H' − v) ≥ k then H' ← H' − v          ▷ v is not required
7:         else C ← C ∪ {v}                            ▷ v is required
8:     return H'
```

IterativeRemoval consists of an iterated removal strategy, based on the approaches of Herrmann and Hertz (2000) and Sun et al. (2019). Given input subgraph $H \subseteq G$, initially it marks all vertices as *not required*[1]. Next, it iteratively selects a non-required vertex $v \in H$ (chosen by a greedy criterion explained below) to be considered for removal. If $\chi_h(H - v)$ is unchanged, then there exists a k-critical subgraph in H without v, so it is removed from H; otherwise, v stays in H and is marked *required*. Each $v \in H$ is examined at most once, and the algorithm stops when all vertices in H are required. As shown by Desrosiers et al. (2008), this strategy has the property that if χ_h is exact, i.e., $\chi_h(I) = \chi(I)\ \forall I \subseteq H$, then the resulting solution is guaranteed to be a k-VCS.

To determine the removal order we use the neighborhood weight heuristic of Desrosiers et al. (2008) which gives preference to denser subgraphs, as it experimentally performed better than an arbitrary order. It prioritizes non-required vertices v with smallest neighborhood weight $W(v) = |N(v)| + \sum_{u \in N(v)} (m-1)[u \in C]$, where $N(v)$ is the set of vertices sharing an edge with v[2]. If there are multiple candidates with the same weight W, one is chosen uniformly at random.

2.4.1 Finding required vertices early

Required vertices can sometimes be found early. This allows IterativeRemoval to skip trying to remove them and therefore run faster. This is achieved by checking each valid k-coloring c produced by $\chi_h(H)$ for uniquely-colored vertices, i.e., vertices $v \in V(H)$ such that $c(u) \neq c(v)$ for all $u \in V(H), u \neq v$. Such vertices can be safely marked as required, since removing them reduces $\chi_h(H)$ by one. Note that these vertices are not necessarily part of the *minimal* k-VCS, but of *some* k-VCS, so by making them required we may miss out on a potentially smaller k-VCS that does not contain them. In practice, however, we have found the performance gains offset this risk.

If a given coloring c does not have uniquely colored vertices, we try to induce them by modifying c, while keeping it valid. This is done by checking, for each

[1] We use the notation *required* instead of *critical* here to avoid confusion, since a vertex which is not k-critical in H may still be critical in G, as G can have multiple (minimal) k-VCSs.

[2] Iverson's brackets $[p]$ applied to proposition p map true to 1 and false to 0.

color, the number of vertices of that color which cannot be assigned a different color without incurring a conflict. If that number is one, then c can be transformed such that the vertex in question is uniquely colored, making it therefore critical. This check can be done in $O(n^2)$ as follows. Maintain an array $F_\#$ which stores, for each color i, the number of vertices $F_\#(i)$ that must always assume color i, and an array F which holds the index $F(i)$ of such a vertex (or is empty, if $F_\#(i) = 0$). Initially $F_\#(i)$ holds the number of vertices of color i. Iteratively, for each vertex v in the subgraph examine whether its induced neighborhood in G witnesses all colors in $Im(c)$: if so, then v is currently forced to color $c(v)$ and so set $F(c(v)) = v$; otherwise v can safely change to another color, and decrement $F_\#(c(v))$ by one. At the end, for all colors i such that $F_\#(i) = 1$, vertex $F(i)$ is made required.

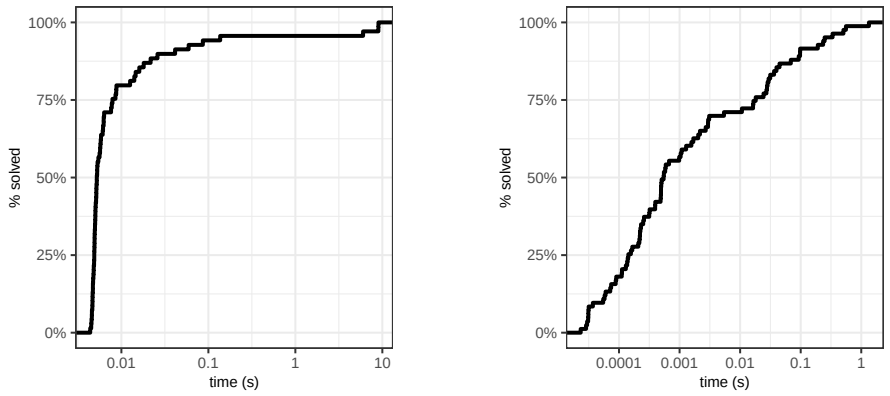
2.5 Checking for colorability

Several parts of the proposed algorithm require us to compute vertex colorings of subgraphs generated. Since these subgraphs can be large, minimal vertex colorings may not always be computable through exact algorithms, and so we have defined function χ_h which uses either an exact or a heuristic algorithm to do so, depending on the size of the subgraph. In practice, since χ_h is only used in the context of testing whether a subgraph $H \subseteq G$ has chromatic number not less than k, we simply consider here the decision problem of deciding if H has a $(k-1)$-coloring. This is done as follows.

We start by executing an exact coloring algorithm on H, with the goal of finding a $(k-1)$-coloring. If it terminates within a fixed time limit of t_e seconds, either finding a valid coloring or proving none exists, then $\chi_h = \chi$ (i.e., it is optimal) and we return the correct answer. If the exact algorithm times out, we then try a heuristic coloring algorithm, executed with a time limit of t_h seconds. If the heuristic finds a valid $(k-1)$-coloring, then $\chi_h(H) \leq k-1$ is true; otherwise, the result is inconclusive and we assume $\chi_h(H) \geq k$.

In order to avoid running the exact algorithm repeatedly on subgraphs that are too large to be colored exactly within t_e seconds, we keep track of the smallest subgraph size s_{min} for which the exact algorithm failed to find a solution. In practice, we have found that, after failing once for size s_{min}, an exact algorithm is generally unlikely to find solutions for subgraphs of s_{min} or more vertices. Therefore, when $|H| \geq s_{min}$ we skip the exact algorithm and run only the heuristic instead. This also has the positive side effect that exact coloring becomes significantly faster than heuristic coloring, in practice, since when it is not skipped it usually terminates well before t_e seconds, whereas the heuristic exhausts all t_h seconds before returning an inconclusive answer. This is the main reason we run the exact algorithm first.

In our implementation we use the exact algorithm of Korman (1979) in the implementation of Culberson (1997), dubbed BTDSatur, which uses an additional heuristic guidance strategy, and the heuristic algorithm algorithm HEA* (Galinier and Hao 1999) as implemented by Lewis et al. (2012). We chose these two implementations since they were the best performers in the extensive comparison done by Lewis et al. (2012). For HEA* we use the parameters recommended by Lewis et al. (2012) (pop-



(a) exact (b) heuristic

Fig. 1: Cumulative distributions of running times of the (a) exact and (b) heuristic algorithms for DIMACS instances.

ulation size 10, 16 iterations per cycle, GPX crossover operator and DSatur constructive algorithm). We note that our vertex removal procedure might also benefit from dynamic coloring algorithms such as Barba et al. (2017) which updates colorings under insertions or deletions.

In the tests that follow we have fixed the time limits $t_e = 1$ and $t_h = 0.5$. Figures 1a and 1b show cumulative distributions of the (log) running times of algorithms BTDSatur and HEA*, respectively, on the graphs of the DIMACS data set. Both figures report only cases where a solution was found within 10 seconds. For BTDSatur, 69 out of 136 instances were solved, with 66 (95.6%) being colored exactly in less than 0.14 seconds, while the remaining instances require 6 or more seconds to solve. HEA* found solutions to 89 instances within 10 seconds, with almost all solutions (98.7%) having been found within 1 second. Both algorithms either find solutions almost immediately or take a considerably long time, with only a few instances on the threshold of being solvable between 1 and 10 seconds. This kind of effect was also observed by Desrosiers et al. (2008) in their tests.

Because of this threshold, we also have a clear threshold in the outputs. Solutions that can be solved exactly during optimization, are provably critical, and no post-hoc test is needed. On the other hand, if the solution cannot be exactly colored during optimization, the post-hoc test is likely to also hit the time limit, and the solution has no guaranteed chromatic number or criticality. Only in the few cases whose solution time falls between the exact solution time t_e during optimization and the time limit of the post-hoc test our method may return a false positive that can be detected.

Table 1: Parameter calibration of procedure IteratedTabu: optimization ranges and best setting found by irace.

Param.	Description	Optimization range	Best value
τ	Tabu tenure	$\{0, 0.1p, 0.2p, \ldots, 1.0\} \cdot p$	$0.1p$
α	Randomization of cons. alg.	$\{0, 0.1, 0.2, \ldots, 1.0\}$	0.1
I_{max}	Max. consec. non-impr. iter.	$\{1, 2, 5, 10, 20, 50, 100\} \cdot 100$	10000
σ	Perturbation strength	$\{1, 2, 5, 10, 20, 50\}$	20

3 Experiments

We performed all experiments on a PC with an Intel i7 930 processor with 4 cores and 16 GB of main memory, running under Ubuntu Linux 18.04. For each test only one core was used. Our algorithms were implemented in C++ and compiled with GCC 7.2 with optimization level -O3. The source code, instances used and full results are available at `http://www.inf.ufrgs.br/algopt/criticalcoloring`.

3.1 Calibrating parameters of IteratedTabu

We calibrate the four parameters τ, α, I_{max} and σ of procedure IteratedTabu using the irace package in GNU R (López-Ibáñez et al. 2016). Since IteratedTabu solves the k-densest subgraph subproblem, we use a test program which only executes IteratedTabu in a multistart fashion until a time limit is reached, and reports the maximum density found.

We have executed irace with a budget of 4000 runs and 2 minutes per run, aimed at the parameter configuration which yields the maximum overall density. As test instances we generated random uniform (Erdős-Rényi) graphs with a number of vertices $n \in \{250, 500, 1000, 2500\}$, density $d \in \{0.3, 0.5, 0.7, 0.9\}$ and target subgraph size $p \in \{0.1n, 0.25n, 0.5n\}$. For each combination of (n, d, p) we have generated 5 instances with different random seeds, for a total of 240 instances. We calibrated the tabu tenure τ relative to p.

Table 1 describes each parameter, their calibration ranges and the best value found by irace. In the experiments that follow, we have fixed the four parameters to the best values found.

3.1.1 Calibrating parameters μ, R and ξ

We use irace also to calibrate parameters μ, R and ξ of the main method. Each test runs our full algorithm with a time limit and reports a fitness value $[\neg c]n + h$, where h is the size of the smallest k-VCS found and c is true when the subgraph found is guaranteed to be k-chromatic by means of the exact coloring algorithm. Irace then chooses the parameter configuration which yields the minimum overall fitness value. The fitness function effectively gives preference to parameter configurations which do offer guarantees on the chromatic numbers by penalizing the other configurations

Table 2: Calibration of main parameters: optimization ranges and best setting found by irace.

Param.	Description	Optimization range	Best value
μ	Size multiplicative of 1st phase	$\{1.1, 1.2, 1.5, 2.0\}$	1.5
R	Num. iter. of GenerateSubgraph	$\{50, 100, 200, 500\}$	200
ξ	Target size slack of 2nd phase	$\{1.01, 1.02, 1.04, 1.08\}$	1.08

by n, and break ties by preferring smaller resulting subgraphs if the guarantees are equal. We use a budget of 4000 runs with a time limit of 10 minutes per run.

Uniform random graphs are often weakly perfect, and such graphs are typically solved by the pre-processing step, which does not depend on the above parameters. Thus, to avoid runs which do not contribute to the calibration we generated a new instance set consisting only of graphs that are with high probability imperfect. Each instance is obtained by repeatedly generating a uniform random graph with given number of vertices n and density d, and computing their chromatic and clique numbers using the proposed hybrid strategy χ_h and the heuristic of Wu et al. (2012), respectively, until they are different (i.e., the generated graph is likely not weakly perfect). This typically only requires a few iterations. We use the result of χ_h as target k. For each pair $(n, d) \in \{50, 100, 150, 200, 250\} \times \{0.05, 0.1, 0.15, 0.2, 0.25\}$ we generated 5 instances with different random seeds, for a total of 125 instances. We chose this instance size range such that about half of the instances are solvable by exact algorithms, while the other half is too large.

Table 2 shows, for each parameter, the calibration range and the best value found by irace. In the experiments that follow, we have fixed these parameters to the best values found.

3.2 Comparison to state-of-the-art methods

The state-of-the-art methods for finding k-VCSs have been proposed by Sun et al. (2019) and Desrosiers et al. (2008). Unfortunately, the original implementations were not available[3]. For the comparison to Desrosiers et al. (2008) we use the values reported in their tables, since they include data on whether solutions are k-chromatic and/or critical, obtained by executing an exact coloring algorithm for 4 hours after the main algorithm.

The results reported in Sun et al. (2019) however are not comparable, since their method relies on heuristic coloring and no guarantees on criticality or chromatic numbers were reported. This means better subgraphs can be false positives caused by the inexactness of heuristic coloring, and may not have the desired chromatic numbers. Therefore, for comparison we reimplemented the algorithm of Sun et al. (2019) in C++ in the same environment as our own method. For the heuristic coloring we used HEA* (Galinier and Hao 1999), with the implementation of Lewis et al. (2012). Since

[3] Sun et al. (2019) did not respond to several requests, Desrosiers et al. (2008) have lost their implementation.

the description of the perturbation scheme of Sun et al. (2019) has an error, we have implemented the perturbation scheme of Glover et al. (2010), upon which is was based. (Namely, in the notation from Sun et al. (2019) we select γ vertices from set A with the highest scores and then move each to set B with probability P_j, where P_j is computed as described by Glover et al. (2010).) We have used the recommended parameter values $\beta = 1$, $\lambda = 1.2$ and $\gamma = 0.25$. On the following we call our implementation of this algorithm SHC. For a fair comparison, it was executed with no limit on the number of perturbations (the original method stops after 20 perturbations), but a time limit.

In the comparisons that follow, we denote our algorithm by GR. Both GR and SHC were executed for a time limit of 30 minutes, and we report averages over 10 replicates. Like Desrosiers et al. (2008) we execute a post-hoc check to determine if the solutions are indeed k-chromatic and critical. Each check executes the exact coloring algorithm (as described in Section 2.5) with a time limit of 30 minutes on the resulting subgraph S, and then considers whether it is critical by examining the removal of each vertex with a time limit of $30/|S|$ minutes, for a total of 60 minutes.

3.2.1 Test instances

We use the instances of the second DIMACS implementation challenge (Johnson and Trick 1996), which are the main benchmark for graph coloring algorithms in the literature. It consists of 136 instances from several different sources, with graphs ranging from 5 to 10000 vertices. For input parameters k we used the best known lower bounds for χ, gathered from several different sources (Johnson and Trick 1996; Mehrotra and Trick 1996; Galinier and Hao 1999; Funabiki and Higashino 2000; Gomes and Shmoys 2002; Blöchliger and Zufferey 2004; Caramia and Dell'Olmo 2004; Méndez-Díaz and Zabala 2006; Hertz et al. 2008; Blöchliger and Zufferey 2008; Caramia and Dell'Olmo 2008; Desrosiers et al. 2008; Galinier et al. 2008; Malaguti et al. 2008; Méndez-Díaz and Zabala 2008; Lü and Hao 2010; Malaguti and Toth 2010; Porumbel et al. 2010; Held et al. 2011; Malaguti et al. 2011; Titiloye and Crispin 2011b,a; Gualandi and Malucelli 2012; Hao and Wu 2012; Titiloye and Crispin 2012; Wu et al. 2012; Segundo 2012; Wu and Hao 2013; Wu 2013; Moalic and Gondran 2015; Sun et al. 2019; Tomita et al. 2017; Zhou et al. 2016, 2018; Culberson 1995) as default unless specifically stated, to ensure that the input graph has a k-VCS.

We have grouped the 136 instances in four (non-disjunct) classes:

1. Weakly-perfect instances, whose best known lower bound for χ is a clique. These instances are typically solved during the pre-processing step within milliseconds, and are thus trivial. This class has 71 instances.
2. Irreducible instances, for which no k-VCS smaller than n has been previously found. This class has 15 instances.
3. Reducible instances: those not in the two previous classes. These instances are interesting, since they potentially have non-trivial VCSs, and thus are useful to compare different approaches. This class has 50 instances.
4. Open instances, for which the optimal chromatic number is open. For these instances lower bounds can be improved by finding a k'-VCS for a k' larger than

the best known lower bound. If k' is equal to the best upper bound, one can also determine the optimal chromatic number. Note that this class overlaps with the three above. There are 20 open instances.

3.2.2 Results for DIMACS instances

We omit results for weakly-perfect instances, since they are trivially solved within milliseconds by identifying a k-clique. Even though the methods of Desrosiers et al. (2008) and Sun et al. (2019) do no such check and thus have difficulties on some instances, a comparison here is not fair since this step relies on the clique-finding algorithm, which could be added to their methods.

Tables 3 and 5 shows the results of the comparison for "irreducible" and "reducible" instances, respectively. They report, for each instance and algorithm, the current best upper bound in the literature ($\bar{\chi}$), the number of vertices (n) and edges (m) of the best subgraph found, either the running time or the time to best (ttb.) if the algorithm runs with a time limit, in seconds, and the number of replicates which had found k-chromatic solutions that could be confirmed by the post-hoc check (chr.). For GR, we also report the running time (t.p.) of the post-hoc check. For all instances, whenever GR found a solution S with guaranteed $\chi(S) \geq k$, S was critical as well, and thus for simplicity we omit the results for the criticality check[4]. If any replicate was able to confirm the chromatic number of the solution, columns (n) are only averaged over such replicates. Missing values for the results of Desrosiers et al. (2008) indicate that the authors either did not report results for that instance, or used a different value of k from the current best lower bounds (some values are smaller, reflecting best known lower bounds at the time, and some larger, since for a few instances Desrosiers et al. (2008) used k equal to an upper rather than a lower bound). In the interest of comparing the algorithms, we also report in Table 4 results for instances where Desrosiers et al. (2008) used a k smaller than the current best lower bound $\underline{\chi}$. Results where one variant is clearly better are shown in bold.

The fact that all solutions with confirmed chromatic numbers were also critical confirms the results from Figure 1 that most instances can either be exactly colored immediately or not at all within feasible time, with very little middle ground. This middle ground can be seen in some instances such as queen8_8, 3-Insertions_4, and DSJC125.5 (with $k = 14$, on Table 4), where only a few replicates were confirmed to be k-chromatic. This also indicates that the removal-based approaches proposed by Desrosiers et al. (2008) and Sun et al. (2019) work well to make subgraphs critical, if they are already small enough to be colored quickly.

In Table 3 we see an input instance being critical itself is typically detected quickly (less than 1 second), with a few exceptions on instances 2-Insertions_4 and 3-Insertions_4 which are larger, and therefore more challenging to color exactly. For all "irreducible" instances the post-hoc check proved the input graphs are indeed critical. On instances 3-Insertions_4, myciel5 and myciel6 we can observe how heuristic coloring can lead to false positives in SHC. Here, SHC found instances which are

[4] Desrosiers et al. (2008) found $\chi(S) \geq k$ but not criticality on a few instances: 3-FullIns_5, queen8_8, queen9_9, DSJC125.5, DSJC250.1, DSJC250.5, DSJC500.1, DSJR500.1c and DSJR500.5, while SHC did so on instances qg.order60, wap07a and wap08a.

Table 3: Results of GR, the reimplementation SHC of the method of Sun et al. (2019), and reported by Desrosiers et al. (2008) for "irreducible" DIMACS instances.

instance	$\bar{\chi}$	k	n	m	GR					Desrosiers et al. (2008)				SHC			
					n	m	chr.	ttb.	t.p.	n	m	chr.	time	n	m	chr.	ttb.
1-Insertions_4	5	5	67	232	67.0	232.0	10	0.0	0.0	67.0	232.0	1	0.1	67.0	232.0	10	1.5
2-Insertions_3	4	4	37	72	37.0	72.0	10	0.0	0.0	37.0	72.0	1	0.1	37.0	72.0	10	1.5
2-Insertions_4	5	5	149	541	**149.0**	**541.0**	**10**	359.3	0.0	149.0	541.0	0	0.3	**149.0**	**541.0**	**10**	1.5
3-Insertions_3	4	4	56	110	56.0	110.0	10	0.0	0.0	56.0	110.0	1	0.4	56.0	110.0	10	1.5
3-Insertions_4	5	5	281	1046	**281.0**	**1046.0**	**2**	582.2	900.0	281.0	1046.0	0	0.8	280.0	1042.0	0	438.1
4-Insertions_3	4	4	79	156	**79.0**	**156.0**	**10**	0.0	0.0	79.0	156.0	0	0.6	**79.0**	**156.0**	**10**	1.5
mug100_1	4	4	100	166	100.0	166.0	10	0.0	0.0	100.0	166.0	1	0.1	100.0	166.0	10	1.5
mug100_25	4	4	100	166	100.0	166.0	10	0.0	0.0	100.0	166.0	1	0.1	100.0	166.0	10	1.5
mug88_1	4	4	88	146	88.0	146.0	10	0.0	0.0	88.0	146.0	1	0.1	88.0	146.0	10	1.5
mug88_25	4	4	88	146	88.0	146.0	10	0.0	0.0	88.0	146.0	1	0.1	88.0	146.0	10	1.5
myciel2	3	3	5	5	5.0	5.0	10	0.0	0.0	—	—	—	—	5.0	5.0	10	1.5
myciel3	4	4	11	20	11.0	20.0	10	0.0	0.0	11.0	20.0	1	0.1	11.0	20.0	10	1.5
myciel4	5	5	23	71	23.0	71.0	10	0.0	0.0	23.0	71.0	1	0.1	23.0	71.0	10	1.5
myciel5	6	6	47	236	**47.0**	**236.0**	**10**	0.0	0.0	**47.0**	**236.0**	**1**	0.2	46.0	231.0	0	349.6
myciel6	7	7	95	755	**95.0**	**755.0**	**10**	0.0	0.0	95.0	755.0	0	0.4	92.5	739.5	0	665.3

Table 4: Results of GR, the reimplementation SHC of the methods of Sun et al. (2019), and reported by Desrosiers et al. (2008) for "reducible" instances where Desrosiers et al. (2008) used a value of k smaller than the current best lower bound.

instance	$\underline{\chi}$	$\bar{\chi}$	k	n	m	GR					Desrosiers et al. (2008)				SHC			
						n	m	chr.	ttb.	t.p.	n	m	chr.	time	n	m	chr.	ttb.
queen10_10	11	11	10	100	1470	10.0	45.0	10	0.0	0.0	10.0	45.0	1	3.8	10.0	45.0	10	45.1
DSJC125.5	17	17	14	125	3891	**64.0**	**1197.7**	3	219.7	58.5	70.0	1341.0	1	92.7	65.1	1239.0	**7**	153.2
DSJC250.5	26	28	14	250	15668	**46.2**	**717.8**	10	45.9	0.0	74.0	1505.0	1	119.2	52.4	894.6	10	618.7
DSJC500.1	9	12	6	500	12458	**17.1**	**61.4**	10	384.1	0.0	65.0	369.0	1	146.3	50.0	287.0	10	253.0
DSJR500.1c	85	85	80	500	121275	**80.0**	**3160.0**	**10**	0.0	0.0	84.0	3477.0	1	1421.5	83.0	3396.0	1	1341.9
DSJR500.5	122	122	90	500	58862	90.0	4005.0	10	0.0	0.0	90.0	4005.0	1	747.2	90.0	4005.0	10	232.9

smaller than the ones found by the other two methods, but without any guarantees on the chromatic numbers, making the results less useful. Indeed, for myciel5 and myciel6 all replicates and for 3-Insertions_4 one replicate showed chromatic numbers less than k. It is likely that the solutions obtained by the other two methods are optimal.

Turning to Table 5 we see that the results found by GR usually matched or improved the best solutions in the literature. The only exceptions are instance queen8_8, where algorithm SHC found subgraphs of size 53, while GR found subgraphs of size 53.9, on average (one of the 10 replicates did find a solution of size 53, while the other 9 found one of size 54), and in instance queen9_9, GR was not able to confirm chromatic number 10, but Desrosiers et al. (2008) did. Instance DSJC125.1 was the only case where GR found subgraphs with the same number of vertices as the second best alternative (Desrosiers et al. (2008), in this case), but with fewer edges.

Looking at column t.p. we see that the post-hoc check, in most cases, either hits the time limit or ends in less than a second. This verifies the hardness of the VCP we observed in Section 2.5. Whenever this time is 0 we were also able to confirm the chromatic numbers during execution. In a few cases, namely 4-Insertions_4, queen9_9, queen10_10, r1000.1c and DSJR500.1c we can observe a solvability threshold between t_e and 1800s, and thus these solutions have been reduced in excess due to false positives from the heuristic. .

In instances 4-FullIns_5, 5-FullIns_4 and will199GPIA we provide, to the best of our knowledge, the first confirmedly critical solutions for the values of k considered. In the first two, GR finds smaller critical subgraphs than the other methods, while for instance will199GPIA SHC finds the same result. Since instance 5-FullIns_4 is large, SHC is not able to reduce it enough within the time limit and does not find a k-chromatic solution. Additionally, we have improved the best (confirmedly) critical subgraphs in the literature for instances 2-FullIns_4, ash331GPIA and DSJC250.1.

Concerning running times, in general GR finds the best solutions faster compared to SHC (73.8%, on average, when chromatic numbers are confirmed), although since both run with a fixed time limit we find that this comparison is less relevant. We also present the times reported by Desrosiers et al. (2008); however, they are not comparable, since their algorithm was run in a different platform and the parameters with which exact colorings were computed are unknown.

Looking at Table 4 we see that GR consistently finds the smallest solutions on the instances that Desrosiers et al. (2008) have run with a different k, except instance DSJR500.5 which has a 90-clique. This suggests that GR performs well for different values of k.

Table 5: Results of GR, the reimplementation SHC of the method of Sun et al. (2019), and reported by Desrosiers et al. (2008) for "reducible" DIMACS instances.

instance	$\bar{\chi}$	k	n	m	GR					Desrosiers et al. (2008)				SHC			
					n	m	chr.	ttb.	t.p.	n	m	chr.	time	n	m	chr.	ttb.
1-FullIns_3	4	4	30	100	7.0	12.0	10	3.9	0.0	7.0	12.0	1	0.2	7.0	12.0	10	13.0
1-FullIns_4	5	5	93	593	15.0	43.0	10	6.7	0.0	15.0	43.0	1	0.5	15.0	43.0	10	44.5
1-FullIns_5	6	6	282	3247	31.0	144.0	10	10.2	0.0	31.0	144.0	1	14.6	31.0	144.0	10	140.2
2-FullIns_3	5	5	52	201	9.0	22.0	10	4.3	0.0	9.0	22.0	1	0.2	9.0	22.0	10	22.5
2-FullIns_4	6	6	212	1621	**18.9**	**74.2**	10	186.6	0.0	19.0	75.0	1	0.5	19.0	75.0	10	102.1
2-FullIns_5	7	7	852	12201	39.0	244.0	10	10.7	0.0	39.0	244.0	1	26.6	39.0	244.0	10	423.5
3-FullIns_3	6	6	80	346	11.0	35.0	10	3.9	0.0	11.0	35.0	1	0.3	11.0	35.0	10	35.5
3-FullIns_4	7	7	405	3524	23.0	116.0	10	8.1	0.0	23.0	116.0	1	2.5	23.0	116.0	10	197.9
3-FullIns_5	8	8	2030	33751	47.0	371.0	**10**	16.8	0.0	47.0	371.0	1	157.2	47.0	371.0	9	1034.9
4-FullIns_3	7	7	114	541	13.0	51.0	10	4.7	0.0	13.0	51.0	1	0.4	13.0	51.0	10	51.5
4-FullIns_4	8	8	690	6650	27.0	166.0	10	8.0	0.0	27.0	166.0	1	24.6	27.0	166.0	10	445.3
4-FullIns_5	9	7	4146	77305	**13.0**	**51.0**	**10**	7.9	0.0	—	—	—	—	1817.3	36847.6	0	1799.7
5-FullIns_3	8	8	154	792	15.0	70.0	10	4.4	0.0	15.0	70.0	1	2.4	15.0	70.0	10	70.5
5-FullIns_4	9	9	1085	11395	**30.9**	**224.2**	10	187.3	0.0	—	—	—	—	35.2	254.4	10	861.6
1-Insertions_5	6	6	202	1227	193.5	1183.0	0	528.3	1800.0	202.0	1227.0	0	0.1	199.5	1214.2	0	589.2
1-Insertions_6	7	7	607	6337	579.3	6143.4	0	789.2	1800.0	607.0	6337.0	0	99.7	600.1	6291.8	0	501.5
2-Insertions_5	6	6	597	3936	579.4	3839.4	0	618.5	1800.0	597.0	3936.0	0	16.0	592.2	3911.8	0	547.5
3-Insertions_5	6	6	1406	9695	1375.4	9522.1	0	1334.5	1800.0	1406.0	9695.0	0	541.8	1398.9	9658.6	0	234.2
4-Insertions_4	5	5	475	1795	466.2	1759.8	0	715.2	720.0	475.0	1795.0	0	1.5	473.6	1789.4	0	328.3
ash331GPIA	4	4	662	4181	**7.0**	**12.0**	10	12.3	0.0	9.0	16.0	1	3.2	9.0	18.0	10	335.0
ash608GPIA	4	4	1216	7844	9.0	18.0	10	5.1	0.0	—	—	—	—	9.0	18.0	10	647.1
ash958GPIA	4	4	1916	12506	9.0	18.0	10	5.2	0.0	—	—	—	—	9.0	18.0	10	1118.4
flat1000_50_0	50	50	1000	245000	418.4	45930.0	0	693.6	1800.0	—	—	—	—	422.2	46703.2	0	629.7
flat1000_60_0	60	60	1000	245830	528.8	72352.7	0	1319.5	1800.0	—	—	—	—	535.6	74126.3	0	446.4
flat1000_76_0	76	76	1000	246708	714.9	129577.3	0	1419.9	1800.0	—	—	—	—	721.6	132003.5	0	540.1
flat300_20_0	20	20	300	21375	125.7	4299.5	0	638.9	1800.0	—	—	—	—	129.7	4517.4	0	502.3
flat300_26_0	26	26	300	21633	194.8	9727.4	0	876.9	1800.0	—	—	—	—	196.7	9894.4	0	124.1
flat300_28_0	28	28	300	21695	219.5	12162.4	0	472.4	1800.0	—	—	—	—	222.1	12422.6	0	679.1
myciel7	8	8	191	2360	175.9	2239.0	0	741.0	0.0	191.0	2630.0	0	61.3	186.0	2323.8	0	749.1
queen6_6	7	7	36	290	**22.0**	**119.0**	10	199.6	0.0	**22.0**	**119.0**	1	1.6	24.7	148.0	10	249.7
queen8_8	9	9	64	728	53.9	543.0	**7**	229.8	5.8	54.0	538.0	1	25.2	**53.0**	**526.0**	3	751.9
queen9_9	10	10	81	1056	68.8	805.3	0	253.6	61.8	**75.0**	**897.0**	**1**	27.6	69.1	807.2	0	494.0
queen10_10	11	11	100	1470	83.1	1086.9	0	419.0	828.4	—	—	—	—	84.7	1115.8	0	948.4
r1000.1c	98	96	1000	485090	650.5	208129.1	0	1062.2	41.2	—	—	—	—	644.2	201953.4	0	1342.2
DSJC125.1	5	5	125	736	**10.0**	**25.7**	10	4.2	0.0	**10.0**	26.0	1	0.8	11.0	28.0	10	58.6
DSJC125.5	17	17	125	3891	106.3	2934.7	0	374.5	1800.0	—	—	—	—	106.9	2945.1	0	720.1
DSJC125.9	44	44	125	6961	119.9	6422.8	0	606.6	1800.0	—	—	—	—	118.5	6263.2	0	374.1
DSJC250.1	8	6	250	3218	**40.0**	**196.2**	10	1008.0	0.0	64.0	362.0	1	55.3	56.0	323.0	10	102.1
DSJC250.5	28	26	250	15668	189.0	9431.6	0	841.0	1800.0	—	—	—	—	191.4	9637.0	0	382.6
DSJC250.9	72	72	250	27897	232.0	24109.2	0	1042.3	1800.0	—	—	—	—	231.0	23917.8	0	730.1
DSJC500.1	12	9	500	12458	183.5	2398.0	0	1092.3	1800.0	—	—	—	—	191.2	2549.3	0	399.6
DSJC500.5	47	43	500	62624	352.2	32573.7	0	811.5	1800.0	—	—	—	—	354.1	32913.0	0	229.0
DSJC500.9	126	123	500	112437	422.8	80972.9	0	998.6	1800.0	—	—	—	—	425.5	82047.6	0	142.7
DSJR500.1c	85	85	500	121275	369.7	67311.2	0	1129.5	12.6	—	—	—	—	338.2	55741.7	0	1496.8
DSJC1000.1	20	9	1000	49629	152.7	1975.7	0	789.0	1800.0	—	—	—	—	160.5	2121.3	0	560.5
DSJC1000.5	82	73	1000	249826	662.6	113928.0	0	1663.6	1800.0	—	—	—	—	669.8	116402.9	0	418.6
DSJC1000.9	222	216	1000	449449	722.7	237228.2	0	1342.5	1800.0	—	—	—	—	726.5	239776.7	0	680.9
C2000.5	145	99	2000	999836	901.2	213852.7	0	1777.0	1800.0	—	—	—	—	1463.6	547491.0	0	1799.0
C4000.5	259	107	4000	4000268	944.4	239290.3	0	1800.5	1800.0	—	—	—	—	3958.6	3921406.6	0	1775.6
will199GPIA	7	7	701	6772	12.0	46.0	10	7.5	0.0	—	—	—	—	12.0	46.0	10	351.1

3.2.3 Ablation study

In this experiment we report results on an ablation study that aims to evaluate the contributions of the different components of our method. To this end, we considered 6 different versions of our algorithm, each having a major component disabled:

- noIR, in which IterativeRemoval is disabled.
- justIR, which only executes IterativeRemoval, i.e. the proposed method is almost fully disabled and is reduced to a simple vertex removal search.
- noGS, in which we substitute GenerateSubgraph for a simple alternative. Starting from a random vertex, it iteratively adds a randomly-chosen neighbor vertex to the solution S until the target size is reached, and repeats this for R iterations or until $\chi_h(S) \geq k$.
- no2nd, in which the second phase is disabled.
- noHeu, in which χ_h only runs an exact algorithm.
- noExact, in which χ_h only runs a heuristic algorithm.

For this experiment we used the "reducible" set of DIMACS instances. Each test was executed for 5 minutes and replicated 10 times, and post-hoc checks for criticality were run with a total time limit of 10 minutes. Table 6 shows the results, with columns under "Full" reporting results for the original method with no disabled components. For each version, we report the average solution size n, and the number of replicates where the solution was confirmedly k-chromatic by a post-hoc check (chr.). For brevity, we omit instances for which no version found a provenly k-chromatic subgraph.

Overall we observe that each component offers some improvement to the full algorithm, but that noIR, no2nd, noHeu and noExact still function well, showing the proposed method is robust to the removal of some components. In particular, we see that:

- noIR found mostly k-chromatic solutions, but larger than Full (26.9 versus 21.1 average size). This corroborates our previous statement that IterativeRemoval is important in bringing solutions to local minima that are not reached by GenerateSubgraph, but that these minima are not that distant.
- justIR performed very poorly, with only 136 k-chromatic solutions versus 219 of Full, showing vertex removal alone is not useful without the main method.
- noGS performed poorly on the same set of instances as noIR did, with only 158 confirmed k-chromatic solutions. This is because not optimizing for density meant we could not reliably generate k-chromatic subgraphs, and so the method boiled down to just IterativeRemoval as the first phase ended with $S = G$.
- no2nd found slightly larger solutions, of average size 21.5 versus 21.1 of Full. This suggests that the second phase fills its role in using the remaining time to further improve solutions, but is not absolutely required if time is a constraint: version no2ndterminated at 69 seconds of execution, on average, which is well before the 5-minute time limit.
- noHeu slightly outperformed Full in some instances (2-FullIns_3, 2-FullIns_4, 5-FullIns_4 and DSJC125.1), but failed to improve queen8_8 at all, which is perhaps

Table 6: Results of the ablation study on different components of GR.

Instance	n	k	Full		noIR		justIR		noGS		no2nd		noHeu		noExact	
			n	chr.	n	chr.	n	chr.	n	chr.	n	chr.	n	chr.	n	chr.
1-FullIns_3	30	4	7.0	10	7.0	10	9.8	10	7.0	10	7.0	10	7.0	10	7.0	10
1-FullIns_4	93	5	15.0	10	19.8	10	22.7	10	15.0	10	15.0	10	15.0	10	17.0	10
1-FullIns_5	282	6	31.0	10	57.7	10	96.6	7	51.0	9	31.0	10	31.0	10	34.0	10
2-FullIns_3	52	5	9.0	10	9.0	10	9.0	10	9.0	10	9.0	10	**8.9**	10	9.0	10
2-FullIns_4	212	6	19.0	10	23.8	10	24.2	10	19.0	10	19.0	10	**18.9**	10	19.8	10
2-FullIns_5	852	7	39.0	10	65.9	10	685.6	0	578.9	0	39.0	10	39.0	10	51.3	7
3-FullIns_3	80	6	11.0	10	11.0	10	12.0	10	11.0	10	11.0	10	11.0	10	11.0	10
3-FullIns_4	405	7	23.0	10	25.8	10	261.9	0	82.5	8	23.0	10	23.0	10	24.2	10
3-FullIns_5	2030	8	47.0	10	76.8	10	1855.5	0	1567.5	0	47.0	10	47.0	10	47.0	10
4-FullIns_3	114	7	**12.9**	10	13.0	10	16.1	9	13.0	10	13.0	10	**12.9**	10	13.0	10
4-FullIns_4	690	8	27.0	10	30.1	10	508.4	0	358.0	0	27.0	10	27.0	10	28.0	10
4-FullIns_5	4146	7	13.0	10	13.0	10	4036.1	0	953.7	0	13.0	10	13.0	10	13.0	10
5-FullIns_3	154	8	15.0	10	15.0	10	20.9	9	15.0	10	15.0	10	15.0	10	15.0	10
5-FullIns_4	1085	9	31.0	10	35.6	10	871.1	0	710.7	0	31.0	10	**30.9**	10	32.3	10
ash331GPIA	662	4	**7.0**	10	8.6	10	48.4	10	8.1	10	8.8	10	**7.0**	10	8.6	10
ash608GPIA	1216	4	9.0	10	10.0	10	361.6	10	9.0	10	9.0	10	9.0	10	9.7	10
ash958GPIA	1916	4	9.0	10	9.9	10	1734.6	10	9.0	10	9.0	10	9.0	10	9.8	10
DSJC125.1	125	5	10.0	10	10.0	10	63.2	10	10.0	10	10.0	10	**9.9**	10	10.0	10
DSJC250.1	250	6	**40.9**	10	46.8	10	121.0	2	102.9	7	43.9	10	**40.9**	10	48.4	10
queen6_6	36	7	22.4	10	25.9	10	26.5	10	**22.0**	10	25.7	10	22.8	10	26.0	10
queen8_8	64	9	54.0	**9**	53.9	8	56.6	6	54.6	4	**53.8**	7	64.0	0	54.0	8
will199GPIA	701	7	12.0	10	23.5	10	502.0	3	12.0	10	12.0	10	12.0	10	12.0	10
Average/Total			**21.1**	**219**	26.9	218	515.6	136	210.0	158	21.5	217	21.6	210	22.7	215

the instance in the set that is hardest to color. In all iterations of GenerateSubgraph the test $\chi_h(S) \geq k$ with the exact coloring algorithm either timed out or returned false, and so GenerateSubgraph never found a subgraph with $S < G$. When IterativeRemoval was run on G it stopped after the first iteration, since $\chi_h(G)$ could not be computed in time.

– noExact performed slightly worse in nearly all instances. We suspect this is due to a subtle effect of false positives from the heuristic, where incorrectly removing a vertex leads IterativeRemoval down an unpromising search path from which it does not escape. In 3 replicates on instance 2-FullIns_5 we observed the common effect of false positives that leads the method to discard vertices in excess, making the solution no longer k-chromatic.

3.2.4 Results for random instances

In this experiment we compare GR to SHC on the random instances described in Section 3.1.1, with the goal of gaining insight on the average behavior of the algorithms across a broader set of instances. Both algorithms were executed for 30 minutes and replicated 10 times, and post-hoc checks for criticality were run with a total time limit of 60 minutes. Table 7 shows the results. It reports, for each input size n and density d, the average value of k considered and, for each algorithm, the best subgraph size found n, the number of edges m of that subgraph, the time to best (ttb.) in seconds,

Table 7: Results of GR and SHC for randomly-generated instances.

n	d	k	GR				SHC			
			n	m	chr.	ttb.	n	m	chr.	ttb.
50	0.05	3.8	**13.6**	**25.4**	50	79.9	16.8	32.8	50	241.1
50	0.1	4.4	**11.8**	**24.4**	50	5.2	13.0	27.8	50	21.3
50	0.15	6	**18.6**	**76.0**	50	48.4	21.9	98.4	50	86.7
50	0.2	5.4	**12.0**	**35.2**	50	59.4	13.7	48.6	50	97.4
50	0.25	6	**28.0**	**131.0**	**50**	19.9	31.1	151.7	40	203.5
100	0.05	4	**22.2**	**41.4**	50	14.5	29.1	59.0	50	273.8
100	0.1	5	**38.6**	**139.6**	50	56.5	44.6	169.2	50	337.9
100	0.15	6	**48.4**	**261.3**	50	123.2	53.0	300.4	50	202.5
100	0.2	7	**57.6**	**434.5**	47	392.2	59.8	460.4	**50**	151.5
100	0.25	8	**60.7**	**573.7**	39	351.7	61.8	589.7	**41**	262.0
150	0.05	4.8	**39.3**	**210.3**	**50**	130.3	46.3	239.0	32	248.3
150	0.1	6	**85.0**	**519.0**	**19**	403.6	86.1	526.0	13	485.2
150	0.15	7.8	120.2	1235.8	0	492.4	123.4	1276.8	0	561.9
150	0.2	9	109.7	1387.1	0	380.9	111.3	1411.9	0	507.9
150	0.25	11	131.4	2288.1	0	629.7	133.4	2315.1	0	571.9
200	0.05	5	**97.7**	**407.0**	**34**	374.5	102.3	428.9	2	499.9
200	0.1	7	135.0	1150.4	0	590.2	137.5	1176.6	0	472.7
200	0.15	9	149.4	1931.7	0	638.6	153.6	2006.2	0	160.4
200	0.2	11	161.1	2841.1	0	610.3	164.9	2932.2	0	272.9
200	0.25	13	168.0	3766.6	0	646.8	171.2	3863.1	0	281.9
250	0.05	5	**66.4**	**257.2**	**50**	418.8	82.1	340.3	43	679.3
250	0.1	8	179.5	1929.5	0	714.7	185.9	2021.8	0	201.8
Average/Total			**79.7**	**893.9**	**639**	326.4	83.8	930.7	571	310.1

and the number of replicates where the solution was confirmedly k-chromatic by a post-hoc check (chr.). For both algorithms whenever a k-chromatic solution was found, it was critical as well.

Algorithm GR consistently finds smaller subgraphs ($\approx 5\%$, on average) for every n and d, and also finds confirmedly k-chromatic subgraphs more often (639 replicates versus 571 of SHC). This indicates GR is better suited for uniform random instances, and possibly new sets of instances whose structure is unknown.

We also observe a sharp decline in the number of confirmedly k-chromatic solutions as graphs get larger, in particular after $d \geq 1.5$ for $n = 150$ and $d \geq 0.1$ for $n \geq 200$: in most cases all replicates had chromatic numbers confirmed, or none did, corroborating the results of Section 2.5.

3.3 Improving lower bounds

Recall from Section 1 that any k-VCS for graph G is a witness of $\chi(G) \geq k$. With this in mind, in this experiment we attempt to improve lower bounds for DIMACS instances by looking for any k-VCS with $\underline{\chi} < k \leq \bar{\chi}$. We consider "open" instances for which GR found at least one confirmed $\underline{\chi}$-chromatic subgraph (the remaining "open" instances were omitted, since finding a $(\underline{\chi}+1)$-chromatic solution is even less likely). In a first attempt we increase k to $\underline{\chi}+1$. Since we found an 8-chromatic

Table 8: Results of GR on "open" DIMACS instances with $\underline{\chi} < k \leq \bar{\chi}$.

instance	n	m	$\underline{\chi}$	$\bar{\chi}$	k	n^*	m^*	chr.	ttb.
4-FullIns_5	4146	77305	7	9	8	**27.0**	**166.0**	**10**	16.2
4-FullIns_5	4146	77305	7	9	9	**55.0**	**525.0**	**10**	29.8
abb313GPIA	1557	53356	8	9	9	**23.0**	**159.7**	**10**	507.2
latin_square_10	900	307350	90	97	91	228.7	24497.5	0	1998.7
wap01a	2368	110871	41	43	42	267.8	12525.3	0	3476.8
wap02a	2464	111742	40	42	41	363.1	15617.5	0	4705.5
wap03a	4730	286722	40	47	41	350.4	14462.1	0	5141.7
wap04a	5231	294902	40	42	41	613.4	29454.2	0	6135.0
wap07a	1809	103368	40	41	41	210.2	8195.8	0	2564.8
wap08a	1870	104176	40	42	41	525.3	27127.2	0	4856.8
DSJC250.1	250	3218	6	8	7	104.5	858.9	0	2485.6

subgraph for instance 4-FullIns_5 with $\underline{\chi} = 7$ and $\bar{\chi} = 9$, this instance was also run with $k = \underline{\chi} + 2 = 9$.

Table 8 shows the results. We report, for each instance, the number of vertices (n) and edges (m), the current lower ($\underline{\chi}$) and upper ($\bar{\chi}$) bounds on the chromatic number from the literature, the value k used, the number of vertices (n^*) and edges (m^*) of the solution found by the heuristic, the number of replicates whose solutions were confirmedly k-chromatic (chr.), and the time to best (ttb.), in seconds. Each run was limited to 2 hours, and we report averages over 10 replicates.

We have found 9-critical subgraphs for instances 4-FullIns_5 and abb313GPIA, thereby (to the best of our knowledge) improving the current best lower bounds from 7 and 8, respectively. Since the new lower bounds match existing upper bounds, we are able to fix the exact chromatic numbers of these instances to 9. For both instances all replicates had solutions confirmedly k-chromatic and k-critical; for 4-FullIns_5 the solutions of all replicates were the same. For instance 4-FullIns_5 the solution of 55 vertices is 98.7% smaller than the original graph of 4146 vertices, while for instance abb313GPIA the solutions of 23 vertices are 98.5% smaller than the original graph of 1557 vertices. These results indicate that focusing on small subgraphs is indeed worth it, since they can be colored exactly while the original graphs are far from it.

For the remaining instances, the heuristic failed to find a $(\underline{\chi} + 1)$-VCS; the best subgraphs found are too large to be colored exactly in reasonable time.

4 Conclusion

We have proposed a new heuristic for finding minimal k-vertex-critical subgraphs. It addresses the shortcomings of previous approaches by adopting a bottom-up strategy that focuses on smaller subgraphs first, as opposed to previous approaches that optimize from the original graph downwards and must deal with larger subgraphs from the start. Because smaller subgraphs are easier to color, our method can more often compute exact chromatic numbers during its execution, and relies less on heuristic coloring algorithms that can lead to false or incorrect results. Given a target size for a

k-VCS, we use a multistart heuristic to generate a set of candidate subgraphs which considers graph density as an objective, rather than the chromatic number χ. This has shown to be more effective than other measures, and helps to significantly reduce the computational effort, since computing χ, even heuristically, is a bottleneck.

Experimentally, the proposed heuristic consistently finds smaller k-VCSs than other approaches in the literature, and also finds solutions that are guaranteed k-chromatic more often, something which other approaches had difficulties with since they relied mainly on heuristics. We have set new best k-VCSs for several instances in the DIMACS data set, and were able to find improved lower bounds on χ for two open instances, which allowed us to further fix their chromatic numbers by matching these with existing upper bounds.

Acknowledgements Our research has been supported by the funding agencies CNPq (grant 142087/2018-1), by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, and by the Google Research Latin America (grant 25111).

References

Barba L, Cardinal J, Korman M, Langerman S, Van Renssen A, Roeloffzen M, Verdonschot S (2017) Dynamic graph coloring. In: Workshop on Algorithms and Data Structures, Springer, pp 97–108, DOI 10.1109/CCCA.2011.6031437

Bhaskara A, Charikar M, Chlamtac E, Feige U, Vijayaraghavan A (2010) Detecting High Log-Densities – an $O(n^{1/4})$ Approximation for Densest k-Subgraph. In: 42nd ACM Symposium on Theory of Computing, ACM Press, New York, NY, USA, p 201, DOI 10.1145/1806689.1806719

Blöchliger I, Zufferey N (2004) A reactive tabu search using partial solutions for the graph coloring problem. Tech. Rep. 04/03, École PolyTechqnie Fédérale de Lausanne

Blöchliger I, Zufferey N (2008) A graph coloring heuristic using partial solutions and a reactive tabu scheme. Computers and Operations Research 35(3):960–975, DOI 10.1016/j.cor.2006.05.014

Bourgeois N, Giannakos A, Lucarelli G, Milis I, Paschos VT (2017) Exact and superpolynomial approximation algorithms for the densest k-subgraph problem. European Journal of Operational Research 262(3):894–903, DOI 10.1016/j.ejor.2017.04.034

Brimberg J, Mladenović N, Urošević D, Ngai E (2009) Variable neighborhood search for the heaviest k-subgraph. Computers & Operations Research 36(11):2885–2891, DOI 10.1016/j.cor.2008.12.020

Brooks RL (1941) On colouring the nodes of a network. Mathematical Proceedings of the Cambridge Philosophical Society 37(2):194–197, DOI 10.1017/S030500410002168X

Caramia M, Dell'Olmo P (2004) Bounding vertex coloring by truncated multistage branch and bound. Networks 44(4):231–242, DOI 10.1002/net.20035

Caramia M, Dell'Olmo P (2008) Coloring graphs by iterated local search traversing feasible and infeasible solutions. Discrete Applied Mathematics 156(2):201–217, DOI 10.1016/j.dam.2006.07.013

Chang M, Chen L, Hung L, Rossmanith P, Wu G (2014) Exact algorithms for problems related to the densest k-set problem. Information Processing Letters 114(9):510–513, DOI 10.1016/j.ipl.2014.04.009

Culberson JC (1995) Quasi-random coloring problem. URL `https://mat.tepper.cmu.edu/COLOR/instances.html#XXCUL`, accessed on June 9th, 2020

Culberson JC (1997) Graph coloring programs manual. URL `https://webdocs.cs.ualberta.ca/~joe/Coloring/Colorsrc/manual.html#bkdsatur`, accessed on June 9th, 2020

Desrosiers C, Galinier P, Hertz A (2008) Efficient algorithms for finding critical subgraphs. Discrete Applied Mathematics 156(2):244–266, DOI 10.1016/j.dam.2006.07.019

Dirac G (1952) Some Theorems on Abstract Graphs. Proceedings of the London Mathematical Society s3-2(1):69–81, DOI 10.1112/plms/s3-2.1.69

Feige U, Seltser M (1997) On the densest k-subgraph problem. Algorithmica 29:2001

Feo TA, Resende MG (1995) Greedy randomized adaptive search procedures. Journal of global optimization 6(2):109–133, DOI 10.1007/BF01096763

Funabiki N, Higashino T (2000) A Minimal-State Processing Search Algorithm for Graph Colorings Problems. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences E83-A

Galinier P, Hao JK (1999) Hybrid Evolutionary Algorithms for Graph Coloring. Journal of Combinatorial Optimization 3(4):379–397, DOI 10.1023/A:1009823419804

Galinier P, Hertz A (2007) Solution techniques for the Large Set Covering Problem. Discrete Applied Mathematics 155(3):312–326, DOI 10.1016/j.dam.2006.04.043

Galinier P, Hertz A, Zufferey N (2008) An adaptive memory algorithm for the k-coloring problem. Discrete Applied Mathematics 156(2):267–279, DOI 10.1016/j.dam.2006.07.017

Garey MR, Johnson DS (1979) Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA

Glover F, Lü Z, Hao JK (2010) Diversification-driven tabu search for unconstrained binary quadratic problems. 4OR 8(3):239–253, DOI 10.1007/s10288-009-0115-y

Gomes CP, Shmoys D (2002) Completing Quasigroups or Latin Squares: A Structured Graph Coloring Problem. In: Computational Symposium on Graph Coloring and Generalizations

de Grey A (2018) The chromatic number of the plane is at least 5. ArXiv:1804.02385

Gualandi S, Malucelli F (2012) Exact Solution of Graph Coloring Problems via Constraint Programming and Column Generation. INFORMS Journal on Computing 24(1):81–100, DOI 10.1287/ijoc.1100.0436

Hajós G (1961) Über eine Konstruktion nicht n-färbbarer Graphen. Wissenschaftliche Zeitschrift der Martin-Luther-Universitat Halle-Wittenberg 10:116–117

Hao JK, Wu Q (2012) Improving the extraction and expansion method for large graph coloring. Discrete Applied Mathematics 160(16-17):2397–2407, DOI 10.1016/j.dam.2012.06.007

Held S, Cook W, Sewell EC (2011) Safe Lower Bounds for Graph Coloring. In: International Conference on Integer Programming and Combinatorial Optimization, vol 6655 LNCS, pp 261–273, DOI 10.1007/978-3-642-20807-2_21

Herrmann F, Hertz A (2000) Finding the chromatic number by means of critical graphs. Electronic Notes in Discrete Mathematics 5(212):174–176, DOI 10.1145/944618.944628

Hertz A, Werra D (1987) Using tabu search techniques for graph coloring. Computing 39(4):345–351, DOI 10.1007/BF02239976

Hertz A, Plumettaz M, Zufferey N (2008) Variable space search for graph coloring. Discrete Applied Mathematics 156(13):2551–2560, DOI 10.1016/j.dam.2008.03.022

Heule MJH (2019a) Computing small unit-distance graphs with chromatic number 5. Geocombinatorics XXVIII(1):32–50

Heule MJH (2019b) Trimming graphs using clausal proof optimization. In: Proc. Int. Conf. Princ. Pract. Constr. Progr., pp 251–267, DOI 10.1007/978-3-030-30048-7_15

Johnson D, Trick M (eds) (1996) Cliques, Coloring, and Satisfiability, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol 26. American Mathematical Society, Providence, Rhode Island, DOI 10.1090/dimacs/026

Korman SM (1979) Combinatorial Optimization, Wiley, chap The graph-colouring problem, pp 211–235

Lewis R, Thompson J, Mumford C, Gillard J (2012) A wide-ranging computational comparison of high-performance graph colouring algorithms. Computers and Operations Research 39(9):1933–1950, DOI 10.1016/j.cor.2011.08.010

López-Ibáñez M, Dubois-Lacoste J, Pérez Cáceres L, Birattari M, Stützle T (2016) The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives 3:43–58, DOI 10.1016/j.orp.2016.09.002

Lü Z, Hao JK (2010) A memetic algorithm for graph coloring. European Journal of Operational Research 203(1):241–250, DOI 10.1016/j.ejor.2009.07.016

Malaguti E, Toth P (2010) A survey on vertex coloring problems. International Transactions in Operational Research 17(1):1–34, DOI 10.1111/j.1475-3995.2009.00696.x

Malaguti E, Monaci M, Toth P (2008) A metaheuristic approach for the vertex coloring problem. INFORMS Journal on Computing 20(2):302–316, DOI 10.1287/ijoc.1070.0245

Malaguti E, Monaci M, Toth P (2011) An exact approach for the Vertex Coloring Problem. Discrete Optimization 8(2):174–190, DOI 10.1016/j.disopt.2010.07.005

Manurangsi P (2017) Almost-polynomial ratio eth-hardness of approximating densest k-subgraph. In: Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, pp 954–961, DOI 10.1145/3055399.3055412

Mehrotra A, Trick MA (1996) A Column Generation Approach for Graph Coloring. INFORMS Journal on Computing 8(4):344–354, DOI 10.1287/ijoc.8.4.344

Méndez-Díaz I, Zabala P (2006) A Branch-and-Cut algorithm for graph coloring. Discrete Applied Mathematics 154(5):826–847, DOI 10.1016/j.dam.2005.05.022

Méndez-Díaz I, Zabala P (2008) A cutting plane algorithm for graph coloring. Discrete Applied Mathematics 156(2):159–179, DOI 10.1016/j.dam.2006.07.010

Moalic L, Gondran A (2015) The New Memetic Algorithm HEAD for Graph Coloring: An Easy Way for Managing Diversity. In: European Conference on Evolutionary Computation in Combinatorial Optimization, vol 9026 LNCS, pp 173–183, DOI 10.1007/978-3-319-16468-7_15

Parts J (2019) Polymath16, thirteenth thread: Bumping the deadline? – Short, Fat Matrices. URL https://dustingmixon.wordpress.com/2019/07/08/polymath16-thirteenth-thread-bumping-the-deadline/#comment-23999, accessed on June 9th, 2020

Polymath Wiki contributors (2020) Hadwiger-Nelson problem - Polymath Wiki. URL http://michaelnielsen.org/polymath1/index.php?title=Hadwiger-Nelson_problem, accessed on June 9th, 2020

Porumbel DC, Hao JK, Kuntz P (2010) An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring. Computers and Operations Research 37(10):1822–1832, DOI 10.1016/j.cor.2010.01.015

Segundo PS (2012) A new DSATUR-based algorithm for exact vertex coloring. Computers and Operation Research 39(7):1724–1733, DOI 10.1016/j.cor.2011.10.008

Smith-Miles K, Baatar D, Wreford B, Lewis R (2014) Towards objective measures of algorithm performance across instance space. Computers and Operations Research 45:12–24, DOI 10.1016/j.cor.2013.11.015

Soifer A (2009) The Mathematical Coloring Book. Springer, New York, NY, USA, DOI 10.1007/978-0-387-74642-5

Sun W, Hao JK, Caminada A (2019) Iterated backtrack removal search for finding k-vertex-critical subgraphs. Journal of Heuristics 25(4-5):565–590, DOI 10.1007/s10732-017-9358-5

Titiloye O, Crispin A (2011a) Graph coloring with a distributed hybrid quantum annealing algorithm. In: KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications, vol 6682 LNAI, pp 553–562, DOI 10.1007/978-3-642-22000-5_57

Titiloye O, Crispin A (2011b) Quantum annealing of the graph coloring problem. Discrete Optimization 8(2):376–384, DOI 10.1016/j.disopt.2010.12.001

Titiloye O, Crispin A (2012) Parameter Tuning Patterns for Random Graph Coloring with Quantum Annealing. PLoS ONE 7(11):e50060, DOI 10.1371/journal.pone.0050060

Tomita E, Matsuzaki S, Nagao A, Ito H, Wakatsuki M (2017) A much faster algorithm for finding a maximum clique with computational experiments. Journal of Information Processing 25:667–677, DOI 10.2197/ipsjjip.25.667

Tsang E (1993) Foundations of Constraint Satisfaction. Academic Press, London

Wu Q (2013) The maximum clique problems with applications to graph coloring. PhD thesis, Université d'Angers

Wu Q, Hao JK (2013) An extraction and expansion approach for graph coloring. Asia-Pacific Journal of Operational Research 30(5), DOI 10.1142/S0217595913500188

Wu Q, Hao JK, Glover F (2012) Multi-neighborhood tabu search for the maximum weight clique problem. Annals of Operations Research 196(1):611–634, DOI 10.1007/s10479-012-1124-3

Zhou Y, Hao JK, Duval B (2016) Reinforcement learning based local search for grouping problems: A case study on graph coloring. Expert Systems with Applications 64:412–422, DOI 10.1016/j.eswa.2016.07.047

Zhou Y, Duval B, Hao JK (2018) Improving probability learning based local search for graph coloring. Applied Soft Computing Journal 65:542–553, DOI 10.1016/j.asoc.2018.01.027