Incremental List Coloring of Graphs, Parameterized by Conservation

Sepp Hartung* and Rolf Niedermeier

Institut für Informatik, Friedrich-Schiller-Universität Jena,
Ernst-Abbe-Platz 2, D-07743 Jena, Germany
{sepp.hartung, rolf.niedermeier}@uni-jena.de

Abstract. Incrementally k-list coloring a graph means that a graph is given by adding stepwise one vertex after another, and for each intermediate step we ask for a vertex coloring such that each vertex has one of the colors specified by its associated list containing some of in total k colors. We introduce the "conservative version" of this problem by adding a further parameter $c \in \mathbb{N}$ specifying the maximum number of vertices to be recolored between two subsequent graphs (differing by one vertex). This "conservation parameter" c models the natural quest for a modest evolution of the coloring in the course of the incremental process instead of performing radical changes. We show that the problem is *NP*-hard for $k \geq 3$ and $W[1]$-hard when parameterized by c. In contrast, the problem becomes fixed-parameter tractable with respect to the combined parameter (k, c). We prove that the problem has an exponential-size kernel with respect to (k, c) and there is no polynomial-size kernel unless $NP \subseteq coNP/poly$. Finally, we provide empirical findings for the practical relevance of our approach in terms of a very effective graph coloring heuristic.

1 Introduction

We study an incremental version of the graph coloring problem:

> Incremental Conservative k-List Coloring (IC k-List Coloring)
> **Input:** A graph $G = (V, E)$, a k-list coloring f for $G[V \setminus \{x\}]$ and $c \in \mathbb{N}$.
> **Question:** Is there a k-list coloring f' for G such that $|\{v \in V \setminus \{x\} : f(v) \neq f'(v)\}| \leq c$?

Herein, a function $f : V \to \{1, \ldots, k\}$ is called a *k-coloring* for a graph $G = (V, E)$ when $f(u) \neq f(v)$ for all $\{u, v\} \in E$. For color lists $L(v) \subseteq \{1, \ldots, k\}$, $v \in V$, a k-coloring for G is called a *k-list coloring* when $f(v) \in L(v)$ for all $v \in V$. Occasionally, we also study IC k-Coloring, which is the special case that $L(v) = \{1, \ldots, k\}$ for all $v \in V$.

Intuitively, IC k-List Coloring models that a graph is built by sequentially adding vertices (together with the incident edges). Thereby, referring to the added vertex by x, the task is to efficiently compute a k-list coloring for G from a known k-list coloring of $G[V \setminus \{x\}]$. It can be seen as an incremental version of List Coloring, where one has to find a k-list coloring for G without the help of a known partial coloring. As will turn

* Partially supported by the IBM Germany Research and Development GmbH, Böblingen.

In the *Proceedings of the 7th Annual Conference on Theory and Applications of Models of Computation (TAMC'10)*, Prague, Czech Republic, June 2010.

out, the introduction of the *conservation parameter c* in the above definition helps in making the otherwise hard problem (fixed-parameter) tractable. Notably, conservation has a natural justification from applications where one may rather prefer an "evolution" of the coloring through the incremental process than a "revolution". We will become more specific about this in the following.

Related Work. We start with describing the related incremental clustering problem Incremental Constrained k-Center (IC k-Center). Here, we are given a discrete distance function on objects and a partition of the objects into k clusters such that the maximum distance within each cluster is minimized. Then, after adding one new object, the task is to compute a new "k-clustering" under the constraint that at most c objects of the previous clustering change their cluster. Here the conservation parameter c reflects the fact that in many settings users would not accept a radical change of the clustering, since this may cause a big loss of information acquired in a costly process for the previous clustering. It is easy to see that IC k-Center can be interpreted as a special case of IC k-List Coloring. Moreover, IC k-Center is only one example for the field of constrained clustering [1] where an incremental and also conservative approach as introduced here seems promising. Refer to Charikar et al. [5] for a broader view on incremental clustering and an analysis in terms of performance ratio of approximation algorithms.

Incremental coloring is also related to the Precoloring Extension problem (PrExt), which is the special case of List Coloring where each color list contains either one or all colors. In other words, the task is to extend a partial k-coloring to the entire graph. It has been shown that on general graphs PrExt is not fixed-parameter tractable with respect to parameter treewidth [10] but, other than List Coloring, it becomes fixed-parameter tractable when parameterized by vertex cover size [13]. Moreover, it is NP-complete for fixed $k \geq 3$ on planar bipartite graphs [15] and $W[1]$-hard with respect to the number of precolored vertices for chordal graphs [16].

Our incremental coloring setting based on the conservation parameter c can also be interpreted as a local search where c measures the degree of locality. Recently, there has been strong interest in analyzing the parameterized complexity of l-local search in terms of some locality parameter l [12, 17]. Our locality measure c can also be seen as "transition cost" between old and new solutions—to keep this cost small has been identified as an important target, e. g., in the reconfiguration of data placements [19].

A further related field of studies is reoptimization [2]. Here, starting with an optimal solution of an optimization problem, one asks how to compute a solution for a locally modified instance more efficiently by using the known solution for the old instance instead of starting the computation from scratch. Without adding the conservation parameter, however, the reoptimization of coloring problems remains hard.

Our Results. We initiate a study of IC k-List Coloring in terms of parameterized complexity [9, 14, 18], considering the two parameters k (number of colors) and c (number of recolored vertices). We show that the problem is NP-hard for fixed $k \geq 3$ and $W[1]$-hard when parameterized by c. In contrast, it becomes fixed-parameter tractable with respect to the combined parameter (k, c). We show that IC k-List Coloring has a $3(k-1)^c$-vertex problem kernel and that there is no hope for a polynomial-size problem kernel with respect to (k, c). Finally, we provide first empirical evidence for the practical relevance of "parameterizing by conservation" by demonstrating how our fixed-parameter

algorithms can be successfully employed as subroutines in a very effective (local search) heuristic for graph coloring.

Preliminaries. For a graph $G = (V, E)$, we set $V(G) := V$ and $E(G) := E$. Analogously, for a path $P = [v_1, \ldots, v_j]$, we write $V(P) := \{v_1, \ldots, v_j\}$ and $E(P) := \{(v_i, v_{i+1}) : 1 \leq i < j\}$ for all directed edges on P. For a graph $G = (V, E)$ and a vertex set $S \subseteq V$, we write $G[S]$ to denote the graph induced by S in G, that is, $G[S] := (S, \{e \in E : e \subseteq S\})$. We define the (open) neighborhood of a vertex v by $N(v) := \{u \in V : \{u, v\} \in E\}$. Moreover, for a given k-coloring f of G and a color i, we set $N(v, i) := \{u \in V : f(u) = i \wedge \{u, v\} \in E\}$.

A parameterized problem is called *fixed-parameter tractable* if it can be solved in $f(k) \cdot n^{O(1)}$ time, where f is a computable function depending only on the parameter k, not on the input size n [9, 14, 18]. A *parameterized reduction* from a language L to another parameterized language L' is a function that, given an instance (x, k), computes in $f(k) \cdot n^{O(1)}$ time an instance (x', k') (with k' only depending on k) such that $(x, k) \in L \leftrightarrow (x', k') \in L'$. Based on that, Downey and Fellows [9] established a hierarchy of complexity classes in order to classify (likely) fixed-parameter intractable problems. The basic class of parameterized intractability is $W[1]$.

Problem kernelization is a core tool to develop parameterized algorithms [9, 18]. A kernelization is often described with a set of *data reduction rules* that are applied to the instance I with parameter k and that change this instance into a smaller instance I' with parameter $k' \leq k$ in polynomial time such that (I, k) is a yes-instance if and only if (I', k') is a yes-instance. If the size of I' is bounded by a (polynomial) function in k, then the instance (I', k') is called a (polynomial) kernel.

2 Parameterized Complexity

In this section, we first study the parameterized complexity of IC k-List Coloring with respect to the single parameters k (number of colors) and c (number of recolored vertices). Since we encounter computational hardness with respect to these single parameterizations, we proceed with a simple search tree strategy showing that IC k-List Coloring is fixed-parameter tractable with respect to the combined parameter (k, c). Since k-Coloring is NP-complete for $k \geq 3$, the following may come without surprise.

Theorem 1. IC k-List Coloring *for fixed* $k \geq 3$ *is NP-complete.*

Proof. Containment in NP is obvious. We reduce k-Coloring to IC k-List Coloring as follows. Let $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$ be an instance of k-Coloring. We set $G_i := G[\{v_1, \ldots, v_i\}]$ for $i \leq n$. To decide the k-colorability of G, we proceed inductively: Obviously, if any G_i, $1 \leq i \leq n$, is not k-colorable, then G also is not. Note that $G_n = G$. Assume that G_{i-1} is k-colorable. For each vertex of G, we fix its color list to be $\{1, \ldots, k\}$. Moreover, we choose $c := |V|$. Clearly, G_i is k-colorable iff G_i can be incrementally colored by recoloring at most c vertices in a k-coloring for G_{i-1}. Thus, we can decide the question about the k-colorability of G inductively by deciding at most n recoloring problems, implying the NP-hardness of IC k-List Coloring for $k \geq 3$. □

Using the above proof strategy, the result that k-List Coloring is $W[1]$-hard with respect to the parameter treewidth [10] can be transferred to IC k-List Coloring. Moreover, it also follows that IC k-List Coloring is NP-complete for fixed $k \geq 3$ for all hereditary

graph classes[1] where the ordinary List Coloring problem is *NP*-hard for fixed $k \geq 3$, e. g., planar bipartite and chordal graphs. The proof strategy is also used in Section 4 to devise an empirically effective heuristic for graph coloring.

We proceed by considering the parameterized complexity of IC k-List Coloring with respect to the parameter c (for unbounded k). In contrast to the parameter k, when c is a constant, then IC k-List Coloring clearly becomes polynomial-time solvable. However, it is $W[1]$-hard, again excluding hope for fixed-parameter tractability.

In order to show the $W[1]$-hardness, we present a parameterized reduction from the $W[1]$-complete k-Multicolored Independent Set problem [10, 11]. The problem is to decide for a given k-coloring f for a graph $G = (V, E)$ whether there exists a *multicolored k-independent set*, that is, a vertex subset $S \subseteq V$ with $|S| = k$ such that $\forall u, v \in S : \{u, v\} \notin E \wedge f(u) \neq f(v)$.

Theorem 2. IC k-List Coloring *is* $W[1]$*-hard with respect to the parameter c.*

Proof. Let $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$ be a k-colored graph (through a coloring f), taken as an instance for k-Multicolored Independent Set. We construct a graph $G' = (V', E')$ from G such that by choosing $c := 2k$ the instance (G', f', L, c) is a yes-instance of IC k-List Coloring iff G is a yes-instance of k-Multicolored Independent Set. Herein, f' is an $(n+1)$-coloring of $G'[V' \backslash \{x\}]$ and L stands for the color lists $L(v)$, $v \in V'$. Note that x is the vertex added in the incremental process. We add $k+1$ new vertices to V, setting $V' := V \cup \{x, s_1, \ldots, s_k\}$. For $v_i \in V$, set $f'(v_i) := i$ in G'. Moreover, the color list for v_i is $L(v_i) := \{i, n+1\}$. To complete the construction, it remains to define f' for the vertices from $\{x, s_1, \ldots, s_k\}$, the edge set E' with $E \subseteq E'$, and the color lists for x and all s_i. To this end, note that each vertex s_i one-to-one corresponds to the subset of vertices from V colored i. Set $f'(s_i) := n+1$ for $1 \leq i \leq k$ and $L(s_i) := \{f'(v) : v \in V_i\} \cup \{n+1\}$, where $V_i := \{v \in V : f(v) = i\}$. Note that $L(s_i) \cap L(s_j) = \{n+1\}$ for $i \neq j$. Moreover, we add edges between s_i and all vertices from V_i. Finally, we set $f'(x) := n+1$, $L(x) := \{n+1\}$, and make x adjacent to all vertices from $\{s_i, \ldots, s_k\}$. This completes the construction.

The idea behind the construction of G' is that those vertices in $V \subset V'$ which can be recolored to the color $n+1$ one-to-one correspond to the vertices in a multicolored k-independent set. It remains to prove that the above reduction is a parameterized reduction from k-Multicolored Independent Set to IC k-List Coloring.

First, assume that (G', f', L, c) as given above is a yes-instance of IC k-List Coloring. Consider the set of recolored vertices, that is, $S' := \{v \in V' \setminus \{x\} : f'(v) \neq \tilde{f}(v)\}$, where $\tilde{f}$ is the coloring obtained after the recoloring process has taken place. By construction, $f'(x) = \tilde{f}(x) = n+1$. This implies that $\{s_1, \ldots, s_k\} \subseteq S'$. Since for each s_i we have $|L(s_i)| = |N_{G'}(s_i)|$, it follows that exactly one vertex from V_i will be recolored (each time caused by the recoloring of s_i). All these chosen "V_i-vertices" receive the color $n+1$ and thus must form a size-k independent set.

For the reverse direction, it is easy to observe that a multicolored k-independent set in G, by recoloring the vertices in the independent set to $n+1$, leads to a recoloring of the instance (G', f', L, c) with c recoloring operations (k recolorings for the vertices in $\{s_1, \ldots, s_k\}$ and k recolorings for the vertices in the independent set). □

[1] A graph class is hereditary if it is closed under taking induced subgraphs.

The results presented so far are negative in terms of fixed-parameter tractability, motivating the study of the *combined* parameter (k, c). A simple search strategy leads to fixed-parameter tractability in this case.

Theorem 3. IC k-List Coloring *can be solved in* $\mathcal{O}(k \cdot (k-1)^c \cdot |V|)$ *time, that is, it is fixed-parameter tractable with respect to the combined parameter* (k, c).

Proof. Clearly, if for the inserted vertex x it holds that $\{f(v) : v \in N(x)\} \subset L(x)$, then there is a "free color" left for x and x can be colored using this free color. Otherwise, $\{f(v) : v \in N(x)\} \supseteq L(x)$. Hence, a recoloring is necessary. First, branch into the $|L(x)| \leq k$ possibilities how to color x. In each branch, at least one of the neighbors of x has the same color as x and hence needs to be recolored. Now, we have at most $k-1$ options to do so. This process continues until all "color conflicts" have disappeared or in total c vertex recolorings have been performed without obtaining a k-coloring. It is easy to see that this strategy leads to a search tree of size $\mathcal{O}((k-1)^c)$ (depth c and branching factor $k-1$). From this, the running time $\mathcal{O}(k \cdot (k-1)^c \cdot |V|)$ follows. □

Note that all our results above also hold for IC k-Coloring; we have a stronger focus on IC k-List Coloring since this more general problem allows for an elegant formulation of data reduction rules. The following section will give more details about that.

3 Data Reduction and Kernelization

By developing a polynomial-time executable *data reduction rule*, next we describe how to the transform an instance of IC k-List Coloring into an equivalent but size-reduced instance, known as problem kernel in parameterized algorithmics.

An Exponential-Size Kernel. Assume that the graph $G = (V, E)$, $c \in \mathbb{N}$, and the k-list coloring f for $G[V \setminus \{x\}]$ form a yes-instance for IC k-List Coloring. Furthermore, let the k-list coloring f' for G be a *recoloring*, that is the number of elements in the corresponding *recoloring set* $S := \{v \in V \setminus \{x\} : f'(v) \neq f(v)\} \cup \{x\}$ is at most $c+1$. Intuitively speaking, the set S contains all recolored vertices (including x).

Our kernelization approach makes use of the following observations. If there exists a connected component Z in $G[S]$ such that $x \notin Z$, then one can simply remove Z by setting $f'(v) = f(v)$ for all $v \in Z$, obtaining a smaller recoloring set. Hence, we can assume without loss of generality that for every vertex $v \in S$ there exists a path from x to v in $G[S]$. Actually, there must exist a so-called *conflict path* for every vertex, that is, a simple path from x to v with the special property that for every edge (u, w) on the path $f'(u) = f(w)$. The non-existence of a conflict path for a vertex $v \in S$ implies $f'(u) \neq f(v)$ for all $u \in N(v)$, thus, we can again remove v, setting $f'(v) = f(v)$. Therefore, one can view the recoloring of u as the reason why w must also be recolored. The following lemma summarizes the above observations.

Lemma 1. *The graph* $G[S]$ *contains a conflict path of length* $\leq c$ *for every vertex in* S.

In order to describe the idea behind our data reduction rule, assume that $v \in S$ and denote the corresponding conflict path in $G[S]$ by P_v. Clearly, if $V(P_v)$ denotes the set of all vertices on P_v, then $V(P_v) \subseteq S$. Consider an arbitrary edge (u, w) on P_v. Since $f'(u) = f(w)$, it follows that $N(u, f(w)) \subseteq S$. By summing up these vertices, this can be

viewed as the *costs* (#recolored vertices) of a conflict path P_v. Utilizing this idea, our data reduction rule computes for some vertex a *possible conflict path* of minimum cost and removes the vertex when the costs are greater than the conservation parameter c.

Removing a vertex v means to remove v and all its incident edges and to delete the color $f(v)$ in the color list of all neighbors. This makes sure that we can reinsert v with color $f(v)$ in any solution for the kernel. Actually, the possibility to conveniently remove and reinsert a vertex is the main reason why we work with *list* coloring.

As the name suggests, a possible conflict path for a vertex v is a path which could become a conflict path for v when $v \in S$. Therefore, a possible conflict path P_v is a simple path such that $f(u) \in L(w)$ for each edge $(u, w) \in E(P_v)$. Next, a cost function $l : V \to \mathbb{N}$ gives for each vertex a lower bound for the number of vertices that need to be recolored when reaching the vertex on a possible conflict path. Using this, we now formulate our data reduction rule; its correctness can be directly inferred from Lemma 1.

Reduction Rule 1 *If there is a vertex v with $l(v) > c$, then remove v.*

We define our *cost function* l in an iterative manner. We start with an empty set M and initialize $l(x) := 0$ and $l(v) := \infty$ for all $v \in V \setminus \{x\}$. The set M contains the vertices for which the cost function is already computed. Next, we choose a vertex $v \in V \setminus M$ with minimum cost function value and add it to M (clearly, in the first step we add x). Next, consider a neighbor u of v with $f(u) \in L(v)$. When v is the ancestor of u on a cheapest possible conflict path for u, then the cost function value $l(u)$ is the sum of $l(v)$ and $|N(v, f(u)) \setminus M|$. Thus, we update the cost function value by setting $l(u) := \min\{l(u), l(v) + |N(v, f(u)) \setminus M|\}$. This process will be continued while $M \neq V$. Furthermore, with the help of a priority queue, the cost function l can be computed in $\mathcal{O}(|V| \log |V| + |E|)$ time.

Using the cost function described above, Rule 1 leads to the following.

Theorem 4. IC k-List Coloring *admits a $(3 \cdot (k-1)^c)$-vertex kernel, which can be computed in $\mathcal{O}(|V| \log |V| + |E|)$ time.*

Proof. We show that the exhaustive application of Rule 1 leads to the asserted kernel. In order to bound the size of a reduced graph G, at first we construct a worst-case graph T. Next, we prove that the size of T is bounded by the asserted kernel size. We complete our proof by showing that $|V(G)| \leq |V(T)|$.

The graph T is a tree in which the distance of all leaves to the root is exactly c. We set $L(v) := \{1, \ldots, k\}$ for all $v \in V(T)$. In addition, the root has one child of each color and all other inner vertices have one child of each color except their own color. The cost function l_T assigns each vertex its distance to the root. Thus, the instance (T, c, k) for IC k-List Coloring is reduced with respect to Rule 1.

For a reduced graph and its corresponding cost function, for instance, T and l_T, we define a partition of $V(T)$ by $V_T^j := \{v \in V(T) : l_T(v) = j\}$ for $0 \leq j \leq c$. By construction, it follows that $V_T^0 = \{x\}$ and $|V_T^j| = k \cdot (k-1)^{j-1}$. Thus, the overall size bound for $V(T)$ is as follows:

$$|V(T)| = 1 + k \cdot \sum_{j=1}^{c} (k-1)^{j-1} = 1 + k \cdot \left(\frac{1 - (k-1)^c}{2 - k} \right)$$

$$= 1 + \frac{k}{k-2}((k-1)^c - 1) \leq 3 \cdot (k-1)^c.$$

Next, we prove the kernel size for the reduced graph G. Let l_G be the corresponding cost function and consider the partition V_G^j for $0 \leq j \leq c$ of $V(G)$. In the following, we will show that $|V_G^j| \leq |V_T^j|$, implying $|V(G)| \leq |V(T)|$.

Consider a vertex $v \in V_G^j$ for some $1 \leq j \leq c$ and a cheapest possible conflict path $P_v = [x, \ldots, w, v]$ for v. Because w is the ancestor of v on P_v, it holds that $l_G(w) < l_G(v)$. Suppose that $w \in V_G^{j-t}$ for some $1 \leq t \leq j$. Then, by definition it follows that $l_G(v) - l_G(w) = t$. Thus, there exist at most t vertices with color $f(v)$ in V_G^j, whose ancestor on a cheapest conflict path is w. Formally, defining the ancestor function by $p(v) := w$ and $p^{-1}(w) := \{v \in V_G^j : p(v) = w\}$ we can infer that $|p^{-1}(w) \cap N(w, f(v))| \leq t$. Considering all colors, it follows that $|p^{-1}(w)| \leq t \cdot (k-1)$.

To show $|V_G^j| \leq |V_T^j|$, next, by removing all vertices in $p^{-1}(w)$ from G we construct a graph $\tilde{G}$. Then, we insert tree T_w with w as root into $\tilde{G}$. Similarly to the structure of T, every inner node of T_w has exactly one child of each color and all leaves of T_w have distance exactly t to the root w. Thus, $V_{\tilde{G}}^j$ contains all $(k-1)^t$ leaves of T_w. Assuming $k \geq 3$, it follows that $(k-1)^t \geq t \cdot (k-1) \geq |p^{-1}(w)|$ and, by this, we can infer that $|V_G^j| \leq |V_{\tilde{G}}^j|$. This process can be executed for all ancestors w on a cheapest possible conflict path for each $v \in V_G^j$. Since the structure of T_w is similar to that of T, we obtain that $|V_G^j| \leq |V_{\tilde{G}}^j| \leq |V_T^j|$. □

Non-Existence of a Polynomial Kernel. Bodlaender et al. [4] showed that a *compositional* parameterized problem (whose unparameterized formulation is *NP*-complete) does not have a polynomial kernel, unless $NP \subseteq coNP/poly$. A parameterized problem is compositional if there exists an algorithm which receives as input a sequence of instances $(I_1, c), \ldots, (I_r, c)$ and outputs an instance (I, c') such that (I, c') is a yes-instance iff (I_j, c) is a yes-instance for some $1 \leq j \leq r$. Furthermore, the running time of the algorithm has to be bounded by a polynomial in $\sum_{j=1}^{r} |I_j| + c$ and $c' \leq \text{poly}(c)$.

To prove the non-existence of a polynomial kernel, we first show that IC 3-Coloring is compositional.

Theorem 5. IC 3-Coloring *has no polynomial kernel, unless* $NP \subseteq coNP/poly$.

Proof. Suppose that we are given a sequence of instances $(G_1, x_1, c), \ldots, (G_r, x_r, c)$ of the IC 3-Coloring problem. Our composition algorithm makes a case distinction depending on the size of r relative to c.

Case 1: $3r > 2^c$.

We use the search tree algorithm introduced in Section 2 and simply solve all instances. Depending on whether we found a yes-instance or not, we construct a trivial yes- or no-instance as a result of our composition algorithm. We need $\mathcal{O}(2^c \cdot |G_j|)$ time per instance, hence, the overall running time is bounded by $\mathcal{O}(r \cdot 2^c \cdot \max_{1 \leq j \leq r} |G_j|)$ or $\mathcal{O}(r^2 \cdot \max_{1 \leq j \leq r} |G_j|)$.

Case 2: $3r \leq 2^c$.

In this more complicated case, we construct a new graph G by connecting the graphs $G_1, \ldots, G_r$ by a binary tree. The rough idea is as follows. Starting at the root x of the binary tree, it is then possible to traverse the tree on a conflict path for reaching any subgraph G_j. Then, every recoloring of G_j for some $1 \leq j \leq r$ is a recoloring for G, and vice versa. We first describe the construction of G in detail and then prove its correctness.

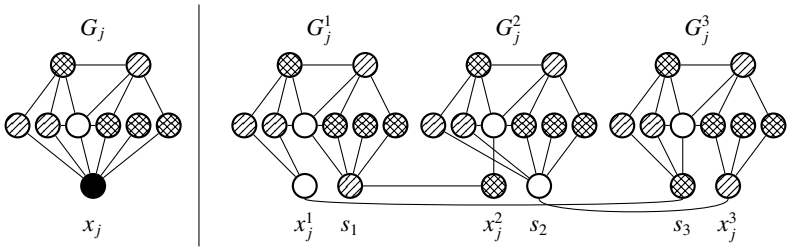

Fig. 1. Example for the composition algorithm for IC 3-Coloring. The left part shows a graph G_j of an input instance. The right part shows the three duplications of G_j and the modifications on them. To keep the figure simple, the neighbors of the constant vertices, which prevent their recoloring, are omitted.

Starting with an empty graph G, we first insert a *constant vertex* s_i of color i for each color $i \in \{1, 2, 3\}$. By adding $c' + 1$ neighbors of each color (except for s_i's color), we make sure that the color of a constant vertex will never change.

In order to insert the graph G_j for some $1 \leq j \leq r$ into the graph G, we have to assign a 3-coloring to G_j. Therefore, preserving the color of the vertices, we duplicate the graph G_j three times (referred to as G_j^1, G_j^2, G_j^3). We adjust the graphs for all $i \in \{1, 2, 3\}$ as follows: Using the permutation $\pi = (1, 2, 3)$ (cycle notation), we set the color of x_j^i in G_j^i to $\pi(i)$. Furthermore, we remove all edges in G_j^i between the vertex x_j^i and every vertex in $N_\pi(x_j^i) := N(x_j^i, \pi(i)) \cup N(x_j^i, \pi^2(i))$. After this, G_j^i is correctly colored. To prevent a recoloring of the vertices in $N_\pi(x_j^i)$ with color i, we insert an edge from the constant vertex s_i to each vertex in $N_\pi(x_j^i)$. Adding the edge $\{x_j^i, s_{\pi^2(i)}\}$ then completes the construction of G_j^i. Figure 1 shows an example.

Next, for all colors $1 \leq i \leq 3$ and $1 \leq j \leq r$, we insert the graphs $\{G_j^i\}$ into G and connect the vertices $\{x_j^i\}$ through a balanced binary tree. The first property of the tree is that every inner vertex connects two differently colored vertices and we set the third color to it. Second, balanced means that each vertex x_j^i has the same distance to the root. Both properties can be fulfilled by "filling up" the tree with constant vertices.

Next, we prove the correctness of our composition algorithm. The algorithm outputs the instance (G, x, c') with $c' := c + \lceil \log(3r) \rceil$. Clearly, the construction of G can be done in polynomial time and by our assumption $3r \leq 2^c$ it follows that $c' \in \mathcal{O}(c^2)$.

In order to show equivalence, the first property of the binary tree implies that (starting at the root x) each inner vertex serves as a "switch" between on which of its both children a conflict path can continue. Hence, by a path through the tree we can reach each vertex x_j^i. Note that, because of the second property, such a conflict path requires $\lceil \log(3r) \rceil$ recoloring operations.

Consider the situation where a conflict path through the tree reaches x_j^i. Recalling that x_j^i has color $\pi(i)$ and G contains the edge $\{x_j^i, s_{\pi^2(i)}\}$, one has to recolor x_j^i with color i. Now, when G_j can be recolored by at most c changes such that x_j obtains color i, then this recoloring is also a proper recoloring for G_j^i. The reverse direction also holds, because we excluded the possibility of recoloring the vertices in $N_\pi(x_j^i)$ with color i. □

According to the framework of Bodlaender et al. [3, 4], there are two ways to show that a problem does not admit a polynomial kernel. First, as we did for IC 3-Coloring, one can show that the problem is compositional. Second, one can reduce by a polynomial parameter transformation a problem for which the non-existence of a polynomial kernel is already known to the problem in question. In the following, we choose the latter approach for IC k-List Coloring.

Corollary 1. IC k-List Coloring *has no polynomial kernel, unless* $NP \subseteq coNP/poly$.

Proof. By simply adding the color list $L(v) = \{1, 2, 3\}$ for each vertex v, an instance for IC 3-Coloring can be reduced to an equivalent instance of IC k-List Coloring. This is a polynomial parameter transformation, which together with Theorem 1 and Theorem 5 implies the claim. □

We close the section with the following strengthening of Theorem 5, showing that there is also no hope to find a polynomial kernel even when we restrict IC k-List Coloring to planar bipartite or chordal graphs.

Corollary 2. IC k-List Coloring *for fixed* $k \geq 3$ *restricted to planar bipartite or chordal graphs has no polynomial kernel, unless* $NP \subseteq coNP/poly$.

Proof. The composition algorithm for IC 3-Coloring (proof of Theorem 5) composes the sequence of instances by a binary tree. Thus, when the instances consist of planar bipartite (chordal) graphs, the composed graph is also planar bipartite (chordal). □

4 Implementation and Experiments

To demonstrate the practical relevance and usefulness of IC k-List Coloring, we have implemented our search tree algorithm (see Section 2) as a subroutine of a popular greedy algorithm for the graph coloring problem. We will show that the derived algorithm outperforms an often used heuristic algorithm called *Iterated Greedy* [6] in terms of quality and running time. Furthermore, we provide practical evidence that in this graph coloring approach the conservation parameter c can often be set to pretty small values, typically smaller than k (the number of overall colors used).

Graph Coloring Instances. We performed our tests on a collection of graph coloring instances, previously used in the "Graph Coloring and its Generalizations"-Symposium (2002) [7]. Before that, some of these graph coloring instances were studied in the *DIMACS Implementation Challenge* (1993) [8].

Altogether, the collection of instances contains 64 graphs, where the number of vertices ranges from 25 to 4730 (avg: 1067). The average density of the graphs (ratio of the number of vertices to the number of edges) is 15%. The instances cover a wide range of graph classes such as *Leighton graphs, latin square graphs* and *queen graphs*.

Implementation Details. The implementation of all algorithms is written in Java, it is open source and freely available.[2] The potential speed loss in comparison to other program languages can be neglected since we are dealing with exponential-time algorithms.

[2] Source code and a full list of our results: http://theinf1.informatik.uni-jena.de/inc_cluster/

All experiments were run on an AMD Athlon 64 3700+ machine with 2.2 GHz, 1 M L2 cache, and 3 GB main memory running under the Debian GNU/Linux 5.0 operating system with Java 1.6.0_12 (option: -Xmx256M).

Next, we describe some details of the implementation of the graph coloring algorithms. Our algorithm is based on the following greedy strategy. Processing the vertices in descending order according to their degree, color a vertex with an already used color whenever possible, otherwise use a new color for the vertex. There exist many strategies how to choose a color from all possible already used colors. We implemented the strategies *Simple* (choose the first color according to any ordering), *Largest First* (choose the color which is used most often) and *Random* (random color). For all our results we ran the algorithm with all strategies and list the best result that was found during these trials.

The greedy algorithm suffers from the fact that the color of an already colored vertex cannot be revoked. This is the point where our search tree algorithm comes into play. Consider the situation where the greedy algorithm "fails", that is, during the coloring of a graph $G = (V, E)$ with $V := \{v_1, \ldots, v_n\}$ a vertex v_i for some $1 < i \leq n$ cannot be colored with the already used colors $\{1, \ldots, k\}$, since v_i has at least one neighbor of each color in $G_i := G[v_1, \ldots, v_i]$. Instead of adding a new color $k + 1$ for vertex v_i, we try to solve an IC k-List Coloring instance on the graph G_i with v_i as the uncolored vertex x by our search tree algorithm to get a k-coloring for G_i. If our search tree algorithm cannot find a k-coloring for G_i, then we color the vertex v_i with the new color $k + 1$.

Our search tree implementation incorporates the idea, also used in our data reduction rule (Rule 1), to check after each recoloring whether the number of conflicts is at most c (conservation parameter). Using our cost function (see Section 3), the fact that the cost of a cheapest possible conflict path is greater than c (Rule 1) implies that the number of conflicts is greater than c. Thus, our search tree algorithm will never recolor a vertex which would be reduced by Rule 1.

The potential to find a k-coloring for G_i (if possible) depends on the choice of the value for the conservation parameter c. On the one hand, the higher the value, the higher the potential; on the other hand, the value of c makes the "major contribution" to our algorithm's running time. Preliminary experiments showed that choosing $c \leq 8$ maximal under the constraint $k \cdot (k-1)^c \leq 10^{10}$ leads to high-quality and fast results. We remark that a more thorough investigation of this tradeoff is a promising task for future research.

We compare our above described algorithm to the *Iterated Greedy Algorithm* [6], which is also based on the described greedy algorithm. Its main idea is to iteratively run the greedy algorithm (mixing the described strategies how to choose a possible color). Thereby, the algorithm makes use of the fact that if the greedy algorithm processes the vertices with respect to an already known k-coloring, it will not produce a worse coloring. Clearly, the hope is, while using a "smart permutation" of the vertices, to get a better coloring (in terms of number of colors). We implemented Iterated Greedy in basically the same manner as proposed in [6], meaning that we adopt the strategies how generating "smart permutations" and aborting the iteration when 1000 times no better coloring was found.

Results. Our experimental findings are as follows. Table 1 contains the results for some important instances. Our algorithm (called *search tree* in Table 1) finds for 89% and Iterated Greedy for 83% of the instances a better coloring than the naive greedy algorithm. Thereby, our algorithm could decrease the number of colors by about 12%

name	greedy		Iterated Greedy			search tree		
	k	time (s)	k	#iter	time (s)	k	c	time (s)
ash608GPIA	8	0.2	5.0 [0.0]	1058.8	33.2 [1.3]	5.0 [0.0]	8	0.2 [0.0]
DSJC1000.1	29	0.1	27.5 [0.6]	1243.8	24.5 [5.0]	25.0 [0.0]	6	0.5 [0.1]
DSJC500.1	17	0.0	16.8 [0.5]	1217.5	6.7 [2.3]	14.8 [0.5]	8	0.3 [0.1]
latin_square_10	148	0.3	109 [1.4]	2765.3	68.8 [9.2]	116.8 [2.6]	4	1.5 [0.6]
le450_15a	18	0.0	18.0 [0.0]	1000	5.0 [0.0]	16.0 [0.0]	7	1.2 [0.2]
le450_5a	11	0.0	10.0 [1.4]	1333.8	5.6 [1.4]	9.0 [0.0]	8	0.2 [0.0]
qg.order40	44	0.3	42.0 [0.0]	1033.8	59.9 [1.7]	41.0 [0.0]	5	4.8 [0.3]
queen10_10	16	0.0	13.0 [0.0]	1071.3	0.4 [0.0]	12.3 [0.5]	8	0.2 [0.1]
queen16_16	26	0.0	20.3 [0.5]	1295	2.2 [0.7]	19.3 [0.5]	7	0.4 [0.2]
school1_nsh	31	0.0	14.0 [0.0]	1443.8	3.7 [0.3]	23.0 [5.4]	6	0.2 [0.1]
wap03	55	4.0	53.8 [0.5]	1006.3	475.5 [3.4]	50.0 [0.0]	5	3.9 [0.3]

Table 1. Summary of our experiments. The first column k for each algorithm denotes the best number of colors which was found and the last column provides the running time in seconds. Each value was obtained as the average over four runs (we state the standard deviation in brackets). For the Iterated Greedy algorithm *#iter* denotes the number of iterations. For our search tree based algorithm, c denotes the value of the conservation parameter in the last iteration.

and Iterated Greedy by about 11%. Furthermore, our algorithm is by a factor of 50 and Iterated Greedy is by a factor of 170 slower than the greedy algorithm. In other words, the greedy algorithm needs 23 seconds, our algorithm needs 19 minutes and Iterated Greedy needs 1 hour 5 minutes for coloring all instances.

In summary, in most cases our algorithm is clearly superior to the Iterated Greedy algorithm, both in terms of number of used colors and running time. We emphasize that the potential of our algorithm is bounded by having chosen an upper bound of 8 for the conservation parameter c. We conjecture that the quality of our results can be drastically improved for higher values of c. In summary, our results indicate that adding the conservation parameter leads to a problem formulation which can be used to attack practical instances of the classical graph coloring problem.

5 Conclusion

We believe that the incremental setting combined with the parameterization by conservation provides a fruitful approach to numerous other optimization problems besides coloring. For instance, as we discussed before, a "conservative viewpoint" is very natural in the context of clustering. There remain numerous challenges for future research even when restricting the focus to coloring problems. Among others, for IC k-List Coloring it is open to achieve a problem kernel of $\mathcal{O}(c^k)$ vertices contrasting our $\mathcal{O}(k^c)$-vertex kernel. Improving on the upper bounds of our simple search tree algorithm is desirable. By means of a reduction from List Coloring we have shown that IC k-List Coloring is also *NP*-hard for restricted graph classes such as planar bipartite or chordal graphs. However, it would be interesting to study whether improvements in terms of parameterized algorithms and data reduction rules are achievable for such restricted graph classes.

Future work may also see a more elaborated empirical study on the conservative approach to incremental graph coloring and related clustering problems, e. g. IC k-Center.

Acknowledgements. We thank Michael Wurst (IBM Germany) for helpful discussions on clustering problems. Furthermore, we are grateful to Johannes Uhlmann and Mathias Weller for their insightful comments concerning Theorem 5.

References

[1] S. Basu, I. Davidson, and K. Wagstaff. *Constrained Clustering: Advances in Algorithms, Theory, and Applications*. Chapman & Hall, 2008. on p. 2.

[2] H.-J. Böckenhauer, J. Hromkovič, T. Mömke, and P. Widmayer. On the hardness of reoptimization. In *Proc. 34th SOFSEM*, volume 4910 of *LNCS*, pages 50–65. Springer, 2008. on p. 2.

[3] H. L. Bodlaender, S. Thomasse, and A. Yeo. Analysis of data reduction: Transformations give evidence for non-existence of polynomial kernels. Technical report, UU-CS-2008-030, Institute of Information and Computing Sciences, Utrecht University, Netherlands, 2008. on p. 9.

[4] H. L. Bodlaender, R. G. Downey, M. R. Fellows, and D. Hermelin. On problems without polynomial kernels. *J. Comput. System Sci.*, 75(8):423–434, 2009. on pp. 7 and 9.

[5] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. *SIAM J. Comput.*, 33(6):1417–1440, 2004. on p. 2.

[6] J. C. Culberson and F. Luo. Exploring the k-colorable landscape with iterated greedy. *DIMACS Series in Discrete Math. and Theor. Comput. Sci.*, pages 245–284, 1996. on pp. 9 and 10.

[7] DIMACS. Graph coloring and its generalizations, 2002. URL `http://mat.gsia.cmu.edu/COLOR02`. Accessed Dec. 2009. on p. 9.

[8] DIMACS. Maximum clique, graph coloring, and satisfiability. Second DIMACS implementation challenge, 1995. URL `http://dimacs.rutgers.edu/Challenges/`. Accessed Dec. 2009. on p. 9.

[9] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999. on pp. 2 and 3.

[10] M. R. Fellows, F. V. Fomin, D. Lokshtanov, F. A. Rosamond, S. Saurabh, S. Szeider, and C. Thomassen. On the complexity of some colorful problems parameterized by treewidth. In *Proc. 1st COCOA*, volume 4616 of *LNCS*, pages 366–377. Springer, 2007. on pp. 2, 3, and 4.

[11] M. R. Fellows, D. Hermelin, F. A. Rosamond, and S. Vialette. On the parameterized complexity of multiple-interval graph problems. *Theor. Comput. Sci.*, 410(1):53–61, 2009. on p. 4.

[12] M. R. Fellows, F. A. Rosamond, F. V. Fomin, D. Lokshtanov, S. Saurabh, and Y. Villanger. Local search: Is brute-force avoidable? In *Proc. 21st IJCAI*, pages 486–491, 2009. on p. 2.

[13] J. Fiala, P. A. Golovach, and J. Kratochvíl. Parameterized complexity of coloring problems: Treewidth versus vertex cover. In *Proc. 6th TAMC*, volume 5532 of *LNCS*, pages 221–230. Springer, 2009. on p. 2.

[14] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006. on pp. 2 and 3.

[15] J. Kratochvíl. Precoloring extension with fixed color bound. *Acta Math. Uni. Comenianae*, 62(2):139–153, 1993. on p. 2.

[16] D. Marx. Parameterized coloring problems on chordal graphs. *Theor. Comput. Sci.*, 351(3): 407–424, 2006. on p. 2.

[17] D. Marx. Searching the k-change neighborhood for TSP is W[1]-hard. *Oper. Res. Lett.*, 36 (1):31–36, 2008. on p. 2.

[18] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006. on pp. 2 and 3.

[19] H. Shachnai, G. Tamir, and T. Tamir. Minimal cost reconfiguration of data placement in storage area network. In *Proc. 7th WAOA*, volume 5893 of *LNCS*, pages 148–157. Springer, 2010. on p. 2.