Incremental List Coloring of Graphs, Parameterized by Conservation[☆]

Sepp Hartung[a], Rolf Niedermeier[a]

[a]*Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, D-10587 Berlin, Germany*

Abstract

Incrementally k-list coloring a graph means that a graph is given by adding vertices step by step, 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). The "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 even on biparite graphs the problem is NP-hard for $k \geq 3$ and W[1]-hard for an unbounded number of colors when parameterized by c. In contrast, also on general graphs 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 $\mathsf{NP} \subseteq \mathsf{coNP/poly}$. Furthermore, we investigate the parameterized complexity on various subclasses of perfect graphs. We show fixed-parameter tractability for the combined parameter treewidth and number k of colors. Finally, we provide empirical findings on the practical relevance of our approach in terms of an effective graph coloring heuristic.

[☆]A preliminary version of this paper appeared in the proceedings of the 7th Annual Conference on Theory and Applications of Models of Computation (TAMC'10), held in Prague, Czech Republic, July 7–11, 2010 [29]. Significant parts of this work were done while both authors were with the Friedrich-Schiller-Universität Jena. The first author was partially supported by a fellowship by IBM Germany, Böblingen.

Email addresses: sepp.hartung@tu-berlin.de (Sepp Hartung), rolf.niedermeier@tu-berlin.de (Rolf Niedermeier)

Keywords: parameterized complexity, data reduction, problem kernel, local search, incremental clustering, precoloring extension

---

1. Introduction

We study an incremental version of the graph coloring problem:

> Incremental Conservative k-List Coloring (IC k-List Coloring)
> **Input:** A graph $G = (V, E)$, $x \in V$, a k-list coloring f for $G[V \setminus \{x\}]$ with respect to the color lists $L(v) \subseteq \{1, \ldots, k\}$ $\forall v \in V$, and $c \in \mathbb{N}$.
> **Question:** Is there a k-list coloring f' for G such that the cardinality of $\{v \in V \setminus \{x\} \mid f(v) \neq f'(v)\}$ is at most c?

Herein, a function $f : V \to \{1, 2, \ldots, k\}$ 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 $G = (V, E)$ from a known k-list coloring of $G[V \setminus \{x\}]$. It can be understood as an incremental version of k-List Coloring, where one has to find a k-list coloring from scratch. As will turn 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 by describing the related incremental clustering problem Incremental Constrained k-Center. Here, we are given a pairwise distance function on objects and a partition of the objects into k clusters, a so-called k-clustering, such that the objective function "maximum distance between objects in the same 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 and the value of the objective function is not getting worse. 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. Incremental Constrained k-Center can be interpreted as a special case of IC k-Coloring as follows: Model the objects as vertices of a graph such that there is an edge between two vertices if the distance between the corresponding objects is greater than the value of the objective function.

Incremental Constrained k-Center is only one example from the field of constrained clustering [1], where an incremental and also conservative approach as introduced here seems promising. Roughly speaking, in constrained clustering one tries to make use of additional information about the problem domain where the objects to be clustered come from. The additional information is provided by so-called constraints where the most prominent are must-link (two objects must belong to the same cluster) and cannot-link constraints (two objects have to be in different clusters). Furthermore, in the field of dynamic graph clustering, algorithms that locally optimize a clustering for a changing set of objects instead of computing it from scratch have been identified as useful, not only measured in terms of running time but also in terms of clustering quality [26]. Refer to Charikar et al. [11] for a broader view on incremental clustering and an analysis in terms of performance ratio of polynomial-time approximation algorithms.

The k-List Coloring problem is a well-studied generalized graph coloring problem, where a graph together with a list of admissible colors for each vertex is given and one has to decide whether the graph admits a k-list coloring or not. In contrast to k-Coloring, where all colors are admissible for each vertex, k-List Coloring remains NP-hard even on subclasses of perfect graphs such as planar bipartite, (unit) interval and (complete) split graphs [9]. From a parameterized point of view, it has been shown that k-List Coloring is W[1]-hard with respect to treewidth [19] and, recently, even for vertex cover size [22].

Incremental coloring is closely related to the Precoloring Extension problem (PrExt), which is the special case of k-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 the parameter treewidth [19] but, unlike k-List Coloring, it becomes fixed-parameter tractable when parameterized by the vertex cover size [22]. Moreover, PrExt

is NP-complete on planar bipartite graphs [34] and W[1]-hard with respect to the number of precolored vertices for interval graphs [35]. We will show that the NP-hardness results can be transferred to IC k-List Coloring for all hereditary graph classes but it becomes tractable when adding a conservation parameter.

Our incremental coloring setting based on the conservation parameter c can also be interpreted as a local search approach 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 [20, 27, 37]. 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, for instance, in the application of the reconfiguration of data placements [41].

A further related field of studies is reoptimization [3]. 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. We shall show that without adding the conservation parameter the reoptimization of coloring problems remains hard even on special graph classes.

Our Results. We begin a study of IC k-List Coloring in terms of parameterized complexity [17, 24, 38], 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 kernel and that there is (under some reasonable complexity-theoretic assumptions) no polynomial kernel with respect to (k, c). Furthermore, we initiate the study of IC k-List Coloring's parameterized complexity on special graph classes. For instance, we show that IC k-List Coloring is polynomial-time solvable on trees and, in general, it is fixed-parameter tractable with respect to the combined parameter k and treewidth. In addition, we show that IC k-List Coloring admits a polynomial kernel on unit interval graphs whereas it, by reasonable complexity-theoretic assumptions, does not on bipartite graphs. Finally, meant as a proof of concept we provide first empirical evidence for the practical relevance of "parameterizing by conservation", demonstrating how our algorithms can be successfully

employed as subroutines in an effective iterative local search heuristic for graph coloring.

2. Preliminaries

In this paper, all graphs are simple and undirected. For a graph $G = (V, E)$, we set $V(G) := V$ and $E(G) := E$. Analogously, for a path $P = [v_1, \ldots, v_j]$, where we interpret the edges to be directed from start to the end vertex, we write $V(P) := \{v_1, \ldots, v_j\}$ and $E(P) := \{(v_i, v_{i+1}) \mid 1 \leq i < j\}$ 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, $G[S] := (S, \{e \in E \mid e \subseteq S\})$. If the graph $G[S]$ is edgeless, we call S an independent set. We define the (open) neighborhood of a vertex v by $N(v) := \{u \in V \mid \{u, v\} \in E\}$. Moreover, for a given k-coloring f of G and a color i, we set $N(v, i) := \{u \in N(v) \mid f(u) = i\}$. For a vertex $v \in V$, we write $G - v$ instead of $G[V \setminus \{v\}]$ and, correspondingly, for a vertex set $S \subseteq V$ we write $G - S$ instead of $G[V \setminus S]$. For a vertex subset $S \subset V$ we briefly write $E(S)$ for $E(G[S])$.

A graph class $\mathcal{F}$ is *hereditary* if it is closed under taking induced subgraphs, that is for all $G \in \mathcal{F}$ and vertex subsets $S \subseteq V(G)$ it holds that $G[S] \in \mathcal{F}$. Moreover, we call a graph class $\mathcal{F}$ *closed under attaching pendant vertices* when for all graphs $G \in \mathcal{F}$ and $v \in V(G)$, after adding a new vertex v' and connecting it to v, the modified graph remains in $\mathcal{F}$.

For a finite alphabet Σ, consider a *parameterized language* $L \subseteq \Sigma^* \times \mathbb{N}$. The corresponding decision problem is called the *parameterized problem* and it is *fixed-parameter tractable* if all instances $(I, k) \in \Sigma^* \times \mathbb{N}$ can be decided in $f(k) \cdot |I|^{O(1)}$ time, where k is the problem parameter and the computable function f solely depends on k [17, 24, 38]. The corresponding algorithm is called a *fixed-parameter algorithm*. A recent development extends parameterized complexity analysis into a multivariate complexity analysis where the parameter typically consists of more than one number [18, 39].

A *parameterized reduction* from a parameterized language L to another parameterized language L' is a function that, given an instance (I, k) of L, computes in $f(k) \cdot |I|^{O(1)}$ time an instance (I', k') of L' (with k' only depending on k) such that $(I, k) \in L \Leftrightarrow (I', k') \in L'$. Based on this, Downey and Fellows [17] established a hierarchy of complexity classes in order to classify (likely) fixed-parameter intractable problems. The basic class of parameterized intractability is W[1] and there is strong evidence that a problem shown to be W[1]-hard is not fixed-parameter tractable [17, 24, 38].

Problem kernelization is a core tool in the design and analysis of fixed-parameter algorithms [5, 28]. A kernelization of a parameterized problem is a polynomial-time computable function transforming an instance (I, k) into another instance (I', k') of the same problem such that $k' \leq k$, $(I, k) \in L \Leftrightarrow (I', k') \in L$, and $|I'| \leq g(k)$ for a computable function g [17, 24, 38]. Furthermore, we refer to the reduced instance (I', k') as problem kernel and g measures its size. If g is a polynomial function then we call (I', k') a polynomial kernel. The process of kernelization is often specified by so-called *data reduction rules*.

3. Complexity of IC k-List Coloring

In this section, we first show that IC k-Coloring and thus also IC k-List Coloring is NP-complete even on bipartite graphs for any $k \geq 3$ (see Section 3.1). In the second part (Section 3.2), we start showing W[1]-hardness for IC k-List Coloring with respect to the conservation parameter c. Since we encounter computational hardness with respect to these "natural" parameterizations, we then 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). Moreover, we examine the complexity of IC k-Coloring in comparison to IC k-List Coloring. Finally, we investigate the dividing line between tractability and hardness for IC k-List Coloring when restricted to special graph classes.

3.1. NP-Completeness

Proposition 1. *IC k-List Coloring (IC k-Coloring) is NP-complete under Turing-reductions on all hereditary graph classes where k-List Coloring (k-Coloring) is NP-complete.*

Proof. Containment in NP is obvious. For the hardness proof, we Turing-reduce k-Coloring to IC k-Coloring on a hereditary graph class $\mathcal{F}$ as follows. Let $G = (V, E) \in \mathcal{F}$ with $V = \{v_1, \ldots, v_n\}$ be an instance of k-Coloring on $\mathcal{F}$.

Set $G_i := G[\{v_1, \ldots, v_i\}]$ for $1 \leq i \leq n$. Note that $G_n = G$ and $G_i \in \mathcal{F}$ for all $1 \leq i \leq n$. To decide the k-colorability of G, we proceed inductively: Obviously, if some G_i, $1 \leq i \leq n$, is not k-colorable, then G is neither. Assume that G_{i-1} is k-colorable. Choosing $c := i$, which allows to recolor all vertices, it follows for all $1 \leq i < n$ that G_{i+1} is k-colorable iff G_{i+1} can be

incrementally colored by recoloring at most c vertices in a k-coloring for G_i. Thus, we can decide the question about the k-colorability of G inductively by deciding at most n recoloring problems. This implies the NP-hardness under Turing-reductions of IC k-Coloring on the graph class $\mathcal{F}$ if k-Coloring is NP-hard on $\mathcal{F}$.

The presented construction also works for the Turing-reduction from k-List Coloring to IC k-List Coloring. □

Proposition 1 implies that IC k-List Coloring is NP-complete (under Turing-reductions) on complete bipartite, chordal, and (unit) interval graphs. In Section 4.3, we investigate the complexity of IC k-List Coloring on special graph classes in more detail. The strategy behind the proof of Proposition 1 is also used in Section 5 to devise an empirically effective heuristic for graph coloring. Furthermore, with the same strategy it can be shown that the W[1]-hardness results for k-List Coloring with respect to treewidth [19] and with respect to vertex cover size [22] carry over to IC k-List Coloring.

Note that Proposition 1 only proves NP-hardness with respect to Turing-reductions. Like many-to-one reductions, they also imply the non-existence of a polynomial-time algorithm, unless NP = P. For our further theorectical studies concerning polynomial kernelizability, however, we need NP-hardness with respect to many-to-one reductions. The following theorem states that even in the case $k = 3$ and without color lists, the resulting IC 3-Coloring problem is NP-complete.

Theorem 2. IC 3-Coloring *is NP-complete on bipartite graphs.*

Proof. Containment in NP is obvious. We next prove NP-hardness. Bodlaender et al. [7, Theorem 1] have shown that PrExt is NP-complete for $k = 3$ on bipartite graphs. More specifically, given a bipartite graph $G = (V_1 \cup V_2, E)$ and three distinguished vertices v_1, v_2, and v_3 in G, it is NP-hard to decide whether there is a 3-coloring f of G such that $f(v_i) = i$ for all $i \in \{1, 2, 3\}$. Since each bipartite graph is 2-colorable, there is a trivial solution for PrExt with $k = 3$ on bipartite graphs when the vertices $\{v_1, v_2, v_3\}$ are not contained in the same partition (either V_1 or V_2). Thus, we can assume that $v_1, v_2, v_3 \in V_1$. In order to construct an equivalent IC 3-Coloring instance (G', x, f', c), let G' be a copy of G and let f' be a 2-coloring of G' where all vertices in V_1 receive color 1 and, correspondingly, all vertices in V_2 have color 2.

Next, we extend G' and, correspondingly, f'. To this end, *forbidding a color $i \in \{1,2,3\}$ for a vertex v* means that we add c degree-one neighbors to v where all of them have color i. Clearly, this operation can be always performed without destroying bipartiteness and vertex v cannot have color i in any solution for IC 3-Coloring, since before recoloring v to i one has to recolor all of the at least c neighbors of v with color i. We now add the vertex x to G' and we forbid color 1 and 3 for x. Moreover, we add a degree-two neighbor u_2 to x which is also adjacent to v_2. We set the color of u_2 to 2 and forbid color 3 for u_2. Observe that, since x can be only colored with 2, this forces a recoloring of u_2 to 1 and thus forces v_2 to be recolored. By forbidding color 3 for v_2 we force that v_2 gets color 2 in any solution. Symmetrically, in order to force that v_3 gets color 3, we add a degree-two vertex u_3 of color 2 which is adjacent to x and v_3. Moreover, we forbid color 3 for u_3 and color 2 for v_3. Observe that G' is still bipartite. We forbid color 2 and 3 for vertex v_1, and thus v_1 cannot be recolored. Finally, we set $c := |V_1 \cup V_2| + 2$ and, thus, we allow to recolor all vertices in G' that correspond to G plus the two vertices $\{u_1, u_2\}$.

By the construction above, it is clear that the vertices $\{v_2, v_3\}$ are forced to recolor such that v_2 gets color 2 and v_3 gets color 3. Moreover, the color of v_1 cannot be changed. Then, the value of c allows to recolor all other vertices that corresponds to G and, thus, any recoloring of G' with at most c recolorings corresponds to a 3-coloring of G where the color of v_i is i for all $i \in \{1,2,3\}$. □

Clearly, if IC k-Coloring is NP-hard on a graph class $\mathcal{F}$, then IC k-List Coloring is NP-hard on $\mathcal{F}$ as well. Herein, the corresponding reduction simply adds a list of all colors to each vertex.

Corollary 3. IC k-List Coloring *is NP-complete on bipartite graphs.*

Having shown that all hardness results for IC k-Coloring transfer to IC k-List Coloring, the following theorem states that the reverse direction holds on graph classes that are closed under attaching pendant vertices.

Theorem 4. *If* IC k-List Coloring *is NP-hard on a graph class $\mathcal{F}$ that is closed under attaching pendant vertices, then* IC k-Coloring *is also NP-hard on $\mathcal{F}$.*

Proof. Consider an instance (G, f, L, c, x) of IC k-List Coloring on a graph class $\mathcal{F}$ which is closed under attaching pendant vertices. The graph G',

which is intended to be part of an equivalent IC k-Coloring instance, is initialized with G. Then, for each vertex $v \in V(G)$ and each color $i \notin L(v)$, add c pendant vertices (each of these vertices is only adjacent to v) of color i to the graph. Obviously, these vertices prevent a recoloring of v with color i, meaning that each recoloring of G' is also a valid recoloring of G and vice versa. In total, because there are at most $(k-1)$ "missing colors" in each color list, we added at most $c \cdot (k-1) \cdot V(G)$ pendant vertices by this construction. Finally, the parameter c for the IC k-Coloring instance is equal to the given one in the IC k-List Coloring instance. □

For example, by Theorem 4 IC k-Coloring is NP-complete on planar bipartite and chordal graphs. Again, we refer to Section 4.3 for a deeper investigation of the complexity of IC k-Coloring on special graph classes.

3.2. Parameterized Complexity

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, IC k-List Coloring clearly becomes polynomial-time solvable; this immediately follows from performing a straightforward brute-force algorithm where the degree of the polynomial depends on c. However, we show that IC k-List Coloring is W[1]-hard with respect to the parameter c, excluding hope for fixed-parameter tractability.

In order to show the W[1]-hardness with respect to c, we present a parameterized reduction from the W[1]-complete k-Multicolored Independent Set problem [21]. This is the problem to decide for a given k-coloring f of a graph G whether there is a *k-multicolored independent set*, that is, an independent set $S \subseteq V(G)$ with $|S| = k$ and $\forall u, v \in S : f(u) \neq f(v)$.

Theorem 5. 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', x, f', L, c) is an equivalent instance of IC k-List Coloring. Herein, f' is a list coloring of $G' - x$ with $n + 1$ colors and L represents 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 $V' := V \cup \{x, s_1, \ldots, s_k\}$.

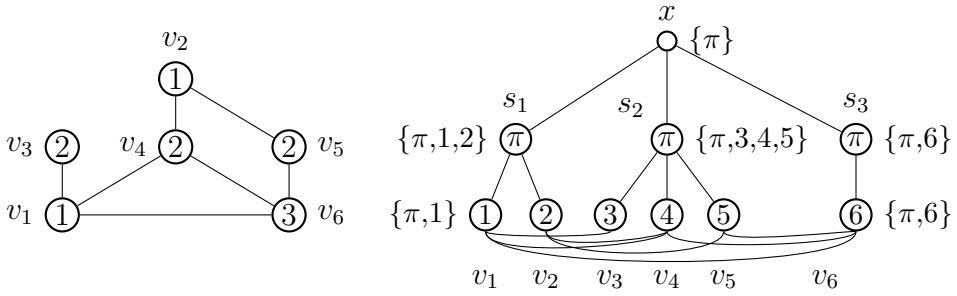

Figure 1: Illustration of the parameterized reduction in the proof of Theorem 5. On the left-hand side, an instance of k-Multicolored Independent Set, consisting of a graph on the vertex set $\{v_1, \ldots, v_6\}$ and a 3-coloring, is depicted. The instance of IC k-List Coloring constructed in the reduction is shown on the right-hand side. The vertex subset $\{v_2, v_3, v_6\}$ forms a 3-colored independent set for the graph on the left-hand side and thus recoloring the vertices s_1 to color 2, s_2 to 3, s_3 to 6, and all vertices from $\{v_2, v_3, v_6\}$ to π, results in a proper coloring of the graph on the right-hand side.

Denoting the $(n+1)$st color in f' by π, for $v_i \in V$ we set $f'(v_i) := i$ and $L(v_i) := \{i, \pi\}$.

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, $1 \leq i \leq k$. To this end, note that each vertex s_i shall one-to-one correspond to the subset of vertices from V colored i. Set $f'(s_i) := \pi$ for $1 \leq i \leq k$ and $L(s_i) := \{f'(v) \mid v \in V_i\} \cup \{\pi\}$, where $V_i := \{v \in V \mid f(v) = i\}$. Note that $L(s_i) \cap L(s_j) = \{\pi\}$ for $i \neq j$. Moreover, we add edges between s_i and all vertices from V_i. Finally, we set $f'(x) := \pi$, $L(x) := \{\pi\}$, and make x adjacent to all vertices from $\{s_1, \ldots, s_k\}$. This completes the construction, illustrated in Figure 1.

The idea behind the construction of G' is that those vertices in $V \subset V'$ that can be recolored to the color π one-to-one correspond to the vertices in a k-multicolored independent set. To show the correctness of the described parameterized reduction, assume that (G', x, f', L, c) as given above is a yes-instance of IC k-List Coloring. Consider the set of recolored vertices, that is, $S' := \{v \in V' \setminus \{x\} \mid f'(v) \neq \tilde{f}(v)\}$, where $\tilde{f}$ is the coloring obtained after the recoloring process has taken place. By construction, $\tilde{f}(x) = \pi$ and this implies that $\{s_1, \ldots, s_k\} \subseteq S'$. Since for each s_i all vertices in V_i are differently colored and s_i has a neighbor of each color in $L(s_i) \setminus \{\pi\}$, 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 π and, because $E \subset E'$, they must form a size-k independent set.

For the reverse direction, let S be a k-multicolored independent set in G. Obviously, recoloring the vertices in S to π, leads to a recoloring of the instance (G', f', L, c). Thereby, we need k recolorings for $\{s_1, \ldots, s_k\}$ and k recolorings for the vertices in S and thus $c = 2k$ recoloring operations. □

The *exponential time hypothesis* (ETH) due to Impagliazzo and Paturi [31] postulates that 3-Sat cannot be solved in subexponential time. For intuition, the ETH denies the existence of a $2^{o(n)}$-time algorithm for 3-Sat, where n is the number of variables in the given 3-Cnf-formula. It was shown that k-Independent Set cannot be solved within $\mathcal{O}(n^{o(k)})$ time unless the ETH fails [12, 23]. Observe that the parameterized reduction from k-Multicolored Independent Set to IC k-List Coloring as described in the proof of Theorem 5 increases the parameter c only by a factor of two. This implies the following.

Corollary 6. *Unless the ETH fails, there is no $\mathcal{O}(n^{o(c)})$-time algorithm for* IC k-List Coloring.

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

Theorem 7. IC k-List Coloring *can be solved in $\mathcal{O}(k \cdot (k-1)^c \cdot (|V|+|E|))$ time, that is, it is fixed-parameter tractable with respect to the combined parameter (k, c).*

Proof. Given a graph $G = (V, E)$ and a k-list coloring f for $G - x$ with respect to the color lists $L(v)$, $v \in V$, observe that if for the inserted vertex x it holds that $\{f(v) \mid v \in N(x)\} \subset L(x)$, then there is a "free color" left for x and x can be colored using this free color. Otherwise, $\{f(v) \mid v \in N(x)\} \supseteq L(x)$. 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 $\mathcal{O}(k \cdot (k-1)^c \cdot (|V| + |E|))$ follows. □

We come back to this search tree strategy in Section 5 when dealing with the implementation of our approach and corresponding empirical studies on benchmark instances.

By applying the construction of adding "pendent vertices" (as introduced in the proof of Theorem 4), the W[1]-hardness result from Theorem 5 along with Corollary 6 can be transferred to IC k-Coloring. Moreover, it is straightforward to adapt the search tree algorithm given in the proof of Theorem 7 for IC k-Coloring. However, our major focus is on IC k-List Coloring since this more general problem allows for an elegant formulation of data reduction rules. The following section will deal with this.

4. Kernelization and Special Graph Classes

Developing a polynomial-time executable data reduction rule, next we describe how to transform an instance of IC k-List Coloring into an equivalent but size-reduced instance, known as problem kernel in parameterized algorithmics. Unfortunately, our worst-case upper bound on the kernel size is exponential in the combined parameter (k, c) (Sec. 4.1). However, we complement this result by showing that IC k-List Coloring does not admit a kernel of polynomial size unless an unlikely collapse in structural complexity theory occurs (Sec. 4.2). This negative result even holds for the special case of bipartite graphs. Finally, focusing on special graph classes, we show that IC k-List Coloring is fixed-parameter tractable with respect to the combined parameter "k and treewidth" and has a polynomial kernel when restricted to unit interval graphs (Sec. 4.3). Moreover, we survey the complexity of IC k-List Coloring and related problems on special graph classes (Table 1 in Sec. 4.3).

4.1. An Exponential-Size Kernel on General Graphs

Assume that a graph $G = (V, E)$, $x \in V$, $c \in \mathbb{N}$, and a k-list coloring f for $G - x$ form a yes-instance for IC k-List Coloring. Additionally, let a k-list coloring f' for G be a *recoloring*, that is, the cardinality of the corresponding *recoloring set* $S := \{v \in V \setminus \{x\} \mid f'(v) \neq f(v)\} \cup \{x\}$ is at most $c + 1$. The recoloring 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 from S 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 v, that is, a simple path from x to v with the special property that for every edge (u, w) on the path it holds that $f'(u) = f(w)$. 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 from S, 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 8. *For a recoloring set S of cardinality at most $c+1$, the graph $G[S]$ contains a conflict path of length at most 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, $V(P_v) \subseteq S$. 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 the number of these vertices for each edge on P_v, this can be viewed as the *costs* (number of recolored vertices) of a conflict path P_v. Utilizing this idea, our data reduction rule computes for each 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, in the process of constructing the solution for the original instance, we can reinsert v with color $f(v)$ in any solution for the kernel. Actually, the possibility of conveniently removing and reinserting 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 $f(w) \in L(u)$ for each edge $(u, w) \in E(P_v)$. Next, a cost function $l : V \to \mathbb{N}$ provides for every 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 8.

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

We compute 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 $v \in V \backslash \{x\}$. The set M contains the vertices for which the cost function is already computed. Next, we choose a vertex $v \in V \backslash M$ with minimum cost function value and

add it to M (in the first step we add x). Next, consider a neighbor u of v with $f(u) \in L(v)$. When v is the predecessor of u on a cheapest possible conflict path for u, meaning that v will be recolored to $f(u)$, then besides u all vertices in $N(v, f(u)) \setminus M$ have to be recolored. Hence, we can 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 until $M = V$.

Observe that our algorithm to compute the cost function works similarly to Djikstra's algorithm for computing all shortest paths starting at one particular vertex, say x, to all other vertices in a positively edge-weighted graph. It thus can be executed with the help of a priority queue in $\mathcal{O}(|V| \log |V| + |E|)$ time. The basic idea is the same, starting at x, maintain during an incremental process a set M of vertices whose shortest (possible conflict) path already has been found. Then, the minimum cost or length of a path from x to a vertex $v \in M$ plus the cost of an edge to a vertex $u \notin M$ must establish a shortest path for u. Other than in Dijkstra's algorithm, the cost of an edge is not given in advance, but is determined by the number of those vertices in the neighborhood of v that have to be recolored if v precedes u on a conflict path for u. Thus, the cost of the edge (v, u) is determined by $N(v, f(u))$ minus the (potentially) already recolored neighbors of v in M.

Theorem 9. 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 claimed kernel. Throughout the proof, we assume $k \geq 3$. 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 rooted tree in which the distance of all leaves to the root x_T is exactly c. We set $L(v) := \{1, \ldots, k\}$ for all $v \in V(T)$. The root has one child of each color and all other inner vertices have one child of each color except their own color. Observe that, because $|N(v, i)| = 1$ for all vertices $v \in V(T)$ and colors $i \neq f(v)$, the cost function l_T assigns each vertex its distance to the root. Thus, the instance (T, L, c) 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 $V_T^j := \{v \in V(T) \mid l_T(v) = j\}$ for $0 \leq j \leq c$. By construction, it follows that $V_T^0 = \{x_T\}$ and $|V_T^j| = k \cdot (k-1)^{j-1}$. Thus, the overall size bound

for $V(T)$ is as follows:

$$\begin{aligned} |V(T)| &= 1 + k \cdot \sum_{j=1}^{c} (k-1)^{j-1} = 1 + k \cdot \left(\frac{1-(k-1)^c}{2-k} \right) \\ &= 1 + \frac{k}{k-2}((k-1)^c - 1) \leq 3 \cdot (k-1)^c. \end{aligned}$$

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, by induction on j we shall show that $|V_G^j| \leq |V_T^j|$ for all $0 \leq j \leq c$, implying $|V(G)| \leq |V(T)|$.

Clearly, by definition $|V_G^0| = |V_T^0| = 1$. 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 such that $w \in V_G^{j-t}$ for some $1 \leq t \leq j$ and thus $l_G(v) - l_G(w) = t$. By definition of the cost function l_G, there are at most t vertices with color $f(v)$ in V_G^j whose ancestor on a cheapest possible conflict path is w. Formally, defining the ancestor function by $p(v) := w$ and $p^{-1}(w) := \{v \in V_G^j \mid p(v) = w\}$ 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 $|V_G^j| \leq |V_{\tilde{G}}^j| \leq |V_T^j|$. □

Observe that, by a fundamental theorem [10], a fixed-parameter algorithm of running time $f(k) \cdot n^{O(1)}$ implies a kernel of size $f(k)$. Thus, an exponential size kernel for IC k-List Coloring as it is presented in Theorem 9 already follows from the fixed-parameter algorithm as stated in Theorem 7. However, a direct kernelization algorithm is still of interest: First, it can establish the basis to build on for further improving the kernel size. Second, the corresponding data reduction rule can be used when solving the problem in practice, as we demonstrate in Section 5.

We close this section by showing how positive kernelization results (and corresponding efficient kernelization procedures) for IC k-List Coloring (such

as Theorem 9) can be transferred to IC k-Coloring. Clearly, IC k-Coloring can be reduced to IC k-List Coloring by adding a color list that contains all colors to each vertex. Then, from the proof of Theorem 4 we can conclude that each kernelization algorithm for IC k-List Coloring together with a straightforward procedure which, afterwards, replaces each "missing color" in the color lists by pendant vertices, is also a valid kernelization for IC k-Coloring. Consequently, if IC k-List Coloring admits a polynomial kernel on a graph class $\mathcal{F}$ that is closed under attaching pendant vertices, then IC k-Coloring also admits a polynomial kernel on $\mathcal{F}$.

Corollary 10. *If* IC k-List Coloring *admits a problem kernel of size $g(k, c)$ on a graph class that is closed under attaching pendant vertices, then* IC k-Coloring *admits a problem kernel of size $c \cdot (k-1) \cdot g(k, c)$ for this graph class.*

4.2. Non-Existence of a Polynomial Kernel

We have shown that IC k-List Coloring admits a $(3 \cdot (k-1)^c)$-vertex kernel. Using a recently developed framework [6, 25], here we prove that there is no hope to improve this result to a polynomial kernel.

Bodlaender et al. [6] and Fortnow and Santhanam [25] showed that a *compositional* parameterized problem (whose unparameterized variant is NP-complete) does not have a polynomial kernel, unless NP $\subseteq$ coNP/poly. One of the most prominent (but not the strongest) consequence of NP $\subseteq$ coNP/poly would be that the polynomial hierarchy collapses to the third level. 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 for IC k-List Coloring, we first show that IC 3-Coloring is compositional with respect to the conservation parameter c. This proves together with the NP-hardness (see Theorem 2) that, unless NP $\subseteq$ coNP/poly, IC 3-Coloring does not admit a polynomial kernel. Using this result, we then show by a so-called *polynomial parameter transformation* [8] that the same holds for the general IC k-List Coloring problem.

Theorem 11. IC 3-Coloring *has no polynomial kernel with respect to the conservation parameter c, unless NP $\subseteq$ coNP/poly.*

Proof. Suppose that we are given a sequence of IC 3-Coloring instances $(G_1, x_1, c), \ldots, (G_r, x_r, c)$ for an arbitrarily large positive integer r. Adopting an idea due to Dom et al. [16], our composition algorithm makes a case distinction depending on the size of r relative to c.

Case 1 ($\boldsymbol{3r > 2^c}$): Use the search tree algorithm described in Section 3 and simply solve all instances. Depending on whether one encountered a yes-instance or not, construct a trivial yes- or no-instance as a result of the 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, due to our case assumption by $\mathcal{O}(r^2 \cdot \max_{1 \leq j \leq r} |G_j|)$.

Case 2 ($\boldsymbol{3r \leq 2^c}$): In this more complicated case, we construct a new graph G by connecting the graphs $G_1, \ldots, G_r$ as the leaves of a binary tree. The rough idea is as follows. Starting at the root x of the binary tree, one can 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.

Starting with an empty graph G, first insert a *constant vertex* s_i of color i for each color $i \in \{1, 2, 3\}$. Let $c' := c + \lceil \log(3r) \rceil$ be the conservation parameter that forms together with G the result of our reduction. By adding c' neighbors of each color (except for s_i's color), make sure that the color of a constant vertex will never change.

In order to insert the graph G_j, $1 \leq j \leq r$, into the graph G, one has to assign a 3-coloring to G_j. Therefore, preserving the color of the vertices, use three copies of G_j. We refer to the copies as G_j^1, G_j^2, G_j^3 and to the corresponding x-vertices as x_j^1, x_j^2, and x_j^3. Next, adjust the graphs for all $i \in \{1, 2, 3\}$ as follows: Using the permutation $\pi = (1\,2\,3)$ (cycle notation), set the color of x_j^i in G_j^i to $\pi(i)$. Furthermore, remove all edges in G_j^i between the vertex x_j^i and every vertex in $N_\pi(x_j^i) := N(x_j^i, \pi(i)) \cup N(x_j^i, \pi(\pi(i)))$. After this, G_j^i is correctly colored. To prevent a recoloring of the vertices in $N_\pi(x_j^i)$ with color i, 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 2 shows an example.

Next, for all colors $1 \leq i \leq 3$ and $1 \leq j \leq r$, 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, where we assume the x_j^is to be "leaves", connects two differently colored vertices and we assign 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.

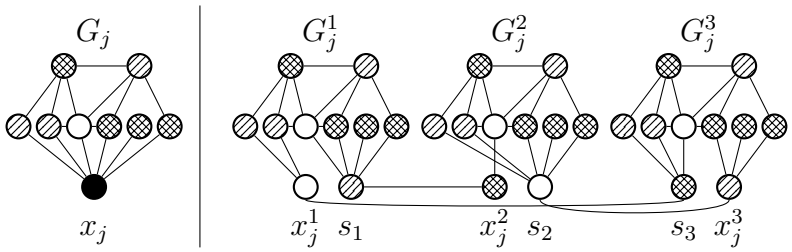

Figure 2: Example for the composition algorithm for IC 3-Coloring. The left part shows a graph G_j of an input instance. The right part shows the three duplications of G_j and the modifications on them. To keep the figure simple, the neighbors of the constant vertices $\{s_1, s_3, s_3\}$, which prevent their recoloring, are omitted.

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

It remains to show that (G, x, c') is a yes-instance iff (G_j, x_j, c) is a yes-instance for at least one $1 \leq j \leq r$. To this end, 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. Because of the second property, such a conflict path requires exactly $\lceil \log(3r) \rceil$ 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 $\{x_j^i, s_{\pi(\pi(i))}\}$, 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.

Altogether, we have shown that the problem is compositional and the claimed result follows from an application of Bodlaender et al.'s [6] and Fortnow and Santhanams [25] results. □

According to the framework of Bodlaender et al. [6, 8], 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. A polynomial parameter transformation is a parameterized

reduction restricted to run in polynomial time and the parameter value of the instance reduced to is bounded from above by a polynomial function of the parameter in the instance reduced from. Adding the color list $L(v) = \{1, 2, 3\}$ for each vertex v is a simple polynomial parameter transformation from IC 3-Coloring to IC 3-List Coloring.

Corollary 12. IC k-List Coloring *has no polynomial kernel for all* $k \geq 3$, *unless* $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.

We close the section with the following strengthening of Theorem 11, showing that there is also no hope to find a polynomial kernel even when we restrict IC k-List Coloring to bipartite graphs: The composition algorithm for IC 3-Coloring (proof of Theorem 11) composes the sequence of instances by a binary tree. Hence when the given sequence of IC 3-Coloring instances consists of bipartite graphs, then the composed graph is also bipartite. Together with Corollary 3 this implies the following.

Corollary 13. IC k-List Coloring *restricted to bipartite graphs has no polynomial kernel, unless* $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.

4.3. Improved Algorithmic Results for Special Graph Classes

In what follows, first we consider graphs of bounded treewidth and then come to unit interval graphs. Finally, in Table 2 we survey old and new results and open problems.

A Polynomial-Time Algorithm for Graphs of Bounded Treewidth. In this section, we present a dynamic programming algorithm which proves that IC k-List Coloring is fixed-parameter tractable with respect to the combined parameter (k, ω), where ω denotes the treewidth of the underlying graph. Recall that Proposition 1 implies, unless P=NP, that IC k-List Coloring is neither with respect to the parameter k nor with respect to the parameter treewidth ω fixed-parameter tractable. Thus, it is natural to consider the combined parameter.

Definition 4.1. *A* tree decomposition *of a connected graph* $G = (V, E)$ *is a tree* T *where the vertices in* $V(T)$ *are subsets* $X_1, X_2, \ldots, X_t$ *of* V, *called bags, such that*

i) *for each edge* $\{u, v\} \in E$ *there is a bag* X_i *where* $\{u, v\} \subseteq X_i$ *and*

ii) *if $u \in X_i \cap X_j$ then it follows that u is also contained in each bag on the path from X_i to X_j in T.*

The cardinality of the largest bag minus one is the treewidth *of G.*

A tree decomposition of a general graph can be computed by a fixed-parameter algorithm with respect to the treewidth [4]. Thus, in the following we may assume that the graph in an IC k-List Coloring instance is given with its tree decomposition.

Theorem 14. IC k-List Coloring *can be solved in $\mathcal{O}(k^{\omega+1}\omega^2 \cdot |V|)$ time on a graph $G = (V, E)$ with treewidth ω and thus is fixed-parameter tractable with respect to the combined parameter (k, ω).*

Proof. Let the graph $G = (V, E)$ together with a tree decomposition T, $x \in V$, the k-coloring f with respect to the color lists L, and the conservation parameter $c \in \mathbb{N}$ form an instance of IC k-List Coloring. Following a standard approach, we describe a dynamic programming algorithm which proceeds in a bottom-up manner on the graph T and actually computes a minimum number of needed recolorings. To this end, assume that T is a *nice tree decomposition*: Every bag X_i of T has at most two children; if X_i has two children X_j and X_l, then $X_j = X_l$ and we thus call X_i a *join bag*. Furthermore, if a bag X_i has only one child, say X_j, then their cardinalities differ by one and either $X_j \subset X_i$ (call X_i a *insert bag*) or $X_i \subset X_j$ (call X_i a *forget bag*). Note that an arbitrary tree decomposition can be transformed in linear time into a nice tree decomposition without an increase of the treewidth [33]. In the following we assume that T is a nice tree decomposition and, additionally, that it is rooted in an arbitrary bag.

Now, we describe our dynamic programming algorithm. The basic idea is to store for each bag X_i a table that contains all k-list colorings of $G[X_i]$ together with a value $m_i(C)$; this value indicates the number of recolorings that are necessary if one extends C to a k-list coloring of the graph corresponding to the subtree rooted at X_i. Now, the table of each leaf bag X_i is initialized with all valid k-list colorings of $G[X_i]$ and, for each k-list coloring C, the value $m_i(C)$ is determined by the number of recolored vertices in X_i.

Next, to proceed in a bottom-up manner, let X_i be a bag in T where all its children are already treated. In the first case, suppose X_i is a join node with children X_j and X_l. Clearly, by definition of a nice tree decomposition, $X_i = X_j = X_l$ and thus the corresponding tables contain the same colorings. Copy all these colorings into the table of X_i. Moreover, set $m_i(C) := m_j(C) + m_l(C)$.

In the second case, suppose that X_i is an insert node and thus has only one child X_j. Then, there is vertex u such that $u \in X_i \setminus X_j$. For each k-list coloring C in the table of X_j and for each color $i \in L(u)$, check whether C together with the coloring of u with i is a valid k-list coloring of $G[X_i]$. If so, then extend the k-list coloring C by assigning color i to the vertex u to a k-list coloring C' and store C' in the table of X_i, and if $i = f(u)$ then set $m_i(C') := m_j(C)$, otherwise set $m_i(C') := m_j(C) + 1$.

In the last case, that is X_i is a forget node with a child X_j, there is a vertex $u \in X_j \setminus X_i$. Partition all colorings in the table of X_j into sets such that within a set the colorings only differ in what color they assign to u. Now, when merging the colorings in each set by deleting the color for u, one obtains all k-list colorings for $G[X_i]$ and hence stores these colorings in the table of X_i. Herein, if the coloring C was determined by merging the colorings $C_1, \ldots, C_t$, then set $m_i(C) := \min_{1 \leq i \leq t} m_j(C_i)$. After filling up the table of the root bag X_r in T, the algorithm answers that the given IC k-List Coloring instance is a yes-instance iff there is at least one coloring C in the table of the root where $m_r(C) \leq c$.

The running time of the above described algorithm is $\mathcal{O}(k^{\omega+1}\omega^2 \cdot |V|)$ because there are at most $|V|$ bags in T and the table of each bag contains at most $k^{\omega+1}$ colorings. In addition, when computing the table of a leaf bag, then one needs $\mathcal{O}(\omega^2)$ time to check whether a coloring is feasible for the subgraph induced by the bag and to count the number of recolored vertices. The table of an insert, forget, or join node can be computed in $\mathcal{O}(k^{\omega+1}\omega)$ time.

Next, we address the correctness of the algorithm. Suppose that there is a coloring C' in the root bag X_r of T where $m_r(C') \leq c$. It is straightforward, by applying a traceback technique, to obtain a k-list coloring coloring C of the entire graph. It is obvious that C assigns a color to each vertex in V for its color list and that there are at most c vertices whose colors in C differ from f. Thus, it remains to argue that C is a feasible coloring, that is $C(u) \neq C(v)$ for all $\{u, v\} \in E$. Observe that C restricted to the vertices in a bag X_i has to be a k-list coloring for $G[X_i]$. Furthermore, because of the second condition in Definition 4.1, for each edge $\{u, v\} \in E$ there is a bag X_i such that $\{u, v\} \subseteq X_i$ and thus $C(u) \neq C(v)$. □

A Polynomial Kernel for Unit Interval Graphs. We have shown that IC k-List Coloring admits a $3 \cdot (k-1)^c$-vertex kernel on general graphs. We discussed that there cannot be a substantial improvement on this; to goal

Table 1: Overview of the complexity of IC k-List Coloring and IC k-Coloring in comparison to PrExt and k-List Coloring on subclasses of perfect graphs. Furthermore, unless NP $\subseteq$ coNP/poly, it is listed whether a polynomial kernel for IC k-List Coloring with respect to the conservation parameter c is possible or not. Non-boldfaced results can be directly deduced (partially, from Proposition 1 & Theorem 4 and the results of Section 4.3). The abbreviation "NP-c" stands for NP-complete, "NP*-c" for NP-complete under Turing-reductions, and P for polynomial-time solvable.

graph class	IC k-List Col.		IC k-Col.	PrExt	k-List Col.
	poly kernel				
trees	/	P	P	P	P [32]
complete bipartite	?	NP*-c	P	**P** [9]	**NP-c** [32]
bipartite	no	NP-c	NP-c	**NP-c** [34]	NP-c
chordal	?	NP*-c	NP-c	NP-c	NP-c
interval	?	NP*-c	?	**NP-c** [2]	NP-c
unit interval	yes	NP*-c	?	**NP-c** [36]	NP-c
cographs	?	?	?	**P** [30]	**NP-c** [32]
distance-hereditary	?	NP*-c	NP-c	**NP-c** [9]	NP-c
split	?	NP*-c	?	**P** [30]	**NP-c** [32]

of a polynomial kernel is illusory even on bipartite graphs (see Corollary 13). On the positive side, we have shown that IC k-List Coloring is polynomial-time solvable on graphs of bounded treewidth. In this section, we close our complexity studies for IC k-List Coloring on special graph classes by pointing out another positive result, that is, there is an $\mathcal{O}(k \cdot c)$-vertex kernel on unit interval graphs. Note that, since PrExt (and, thus k-List Coloring) is NP-complete on unit interval graphs, Proposition 1 implies NP-hardness (under Turing-reductions) for IC k-List Coloring on unit interval graphs. Finally, in Table 1 we summarize all our results and compare them with what is known about the closely related problems k-List Coloring and PrExt.

Roberts [40] proved that a graph $G = (V, E)$ is unit interval iff it has a compatible vertex ordering, that is, an ordering $v_1, \ldots, v_n$ of all vertices in V such that from $\{v_i, v_t\} \in E$ for $i < t$ it follows that $\{v_i, v_j\}, \{v_j, v_t\} \in E$ for all $i < j < t$. Clearly, this implies that $\{v_i, v_{i+1}, \ldots, v_t\}$ induces a clique in G and, if G is k-colorable, then $t - i < k$. For an IC k-List Coloring instance on a unit interval graph G, consider a compatible ordering of $V(G)$.

In this ordering let v_l be the first neighbor and v_r be the last neighbor of x. By Roberts' characterization, we can infer that $l - t \leq 2k - 1$ and thus there are at most $2(k-1)$ vertices "affected" when assigning a color to x. More generally, initiated by assigning a color to x, after a sequence of recolorings the position of a vertex in a compatible ordering that has to be recolored, cannot be more than $k-1$ positions away from the vertices that already have been recolored. We can infer that by performing at most c recolorings one cannot reach a vertex that is more than $c(k-1)$ positions away from x. Hence, computing a compatible ordering and then deleting all vertices (along with their color in the list of their neighbors) that are at a larger distance from x, results in a $(2c(k-1)+1)$-vertex kernel for IC k-List Coloring on unit interval graphs.

5. Implementation and Experiments

To explore the practical potential and usefulness of IC k-List Coloring, we have implemented our search tree algorithm (see Section 3) and used it as a subroutine of a popular greedy heuristic for coloring graphs. The derived algorithm outperforms an often used heuristic algorithm called *Iterated Greedy* [13] in terms of quality and running time. Moreover, we provide practical evidence that in the corresponding graph coloring approach the conservation parameter c can often be set to pretty small values, typically smaller than k (number of colors).

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) [15]. Before that, some of these graph coloring instances were studied in the *DIMACS Implementation Challenge* (1993) [14].

Altogether, the collection of instances contains 64 graphs, where the number of vertices ranges from 25 to 4730 (the average vertex number is 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 so-called Leighton, latin square, and queen graphs.[1]

[1]For more details see http://mat.gsia.cmu.edu/COLOR02/.

Implementation Details. All algorithms have been implemented in Java. The source code is open source and freely available along with a full list of results.[2] The potential speed loss in comparison to other program languages is of minor relevance since we are dealing with exponential-time algorithms. 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. We first implemented 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. In each trial the best result among these three strategies is then used to "initialize" the other two algorithms.

The greedy algorithm suffers from the fact that the color of an already colored vertex cannot be revoked in case of it is necessary in order to proceed the coloring. 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, $1 < i \leq n$, cannot be colored with the already used colors $\{1, \ldots, k\}$ since v_i has in $G_i := G[v_1, \ldots, v_i]$ at least one neighbor of each color. 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, which is 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 implies that the number of conflicts is greater than c. Thus, our search tree algorithm will never recolor a vertex which would be

[2]Incremental Graph Coloring: http://www.akt.tu-berlin.de/menue/software/

Table 2: 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 (standard deviation in brackets). For the Iterated Greedy algorithm *#iter* denotes the number of iterations. For our search tree based algorithm, c denotes the conservation parameter.

name	greedy		Iterated Greedy			search tree		
	k	time	k	#iter	time	k	c	time
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]
qg.order40	44	0.3	42.0 [0.0]	1033.8	59.9 [1.7]	41.0 [0.0]	5	4.8 [0.3]
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]

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. Based on preliminary experiments, we chose $c \leq 8$ maximal under the constraint $k \cdot (k-1)^c \leq 10^{10}$, this led to high-quality and fast results. 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* [13], 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 by Culberson and Luo [13], meaning that we adopted the strategies how generating "smart permutations", also aborting the iteration when 1000 times no better coloring was found.

Results. Our experimental findings are as follows. Table 2 contains the results for some important instances. Our algorithm applied to all 64 instances (called *search tree* in Table 2) 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% 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 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 improved for higher values of c. 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.

6. Conclusion

We believe that the incremental setting combined with "parameterization by conservation" is a natural and fruitful approach for many other optimization problems besides coloring, including clustering problems such as Incremental Constrained k-Center. However, there remain numerous challenges for future research even when restricting the focus to coloring problems. Among others, for IC k-List Coloring we leave open to achieve a problem kernel of $\mathcal{O}(c^k)$ vertices contrasting our $\mathcal{O}(k^c)$-vertex kernel. Moreover, it would be interesting to investigate the existence of a subexponential kernel, as this is not excluded by our no-polynomial size kernel result. Improving on the upper bounds of our simple search tree algorithm (Theorem 7) is desirable as well. We have shown that on unit interval graphs IC k-List Coloring admits a polynomial kernel in distinction to bipartite graphs. However, the complexity for many interesting graph classes is left open (see Table 1). To study whether improvements in terms of fixed-parameter algorithms and polynomial-time data reduction rules are achievable for such restricted graph classes is a worthwhile undertaking.

Acknowledgements. We thank Michael Wurst (IBM Germany) for helpful discussions on clustering problems. We are grateful to Johannes Uhlmann and Mathias Weller for their comments concerning Theorem 11. We thank Ondřej

Suchý who pointed us to our wrong conclusion (stated in the conference version [29]) that the proof of Theorem 11 would also work for chordal graphs. Finally, we thank two anonymous referees of *Theoretical Computer Science* for their careful and constructive feedback that helped to significantly improve our presentation.

References

[1] S. Basu, I. Davidson, and K. Wagstaff. *Constrained Clustering: Advances in Algorithms, Theory, and Applications*. Chapman & Hall, 2008. Cited on p. 3.

[2] M. Biró, M. Hujter, and Z. Tuza. Precoloring extension. I. Interval graphs. *Discrete Math.*, 100(1-3):267–279, 1992. Cited on p. 22.

[3] 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. Cited on p. 4.

[4] H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. Cited on p. 20.

[5] H. L. Bodlaender. Kernelization: New upper and lower bound techniques. In *Proc. 4th IWPEC*, volume 5917 of *LNCS*, pages 17–37. Springer, 2009. Cited on p. 6.

[6] 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. Cited on pp. 16 and 18.

[7] H. L. Bodlaender, K. Jansen, and G. J. Woeginger. Scheduling with incompatible jobs. *Discrete Appl. Math.*, 55(3):219–232, 1994. Cited on p. 7.

[8] H. L. Bodlaender, S. Thomassé, and A. Yeo. Kernel bounds for disjoint cycles and disjoint paths. *Theor. Comput. Sci.*, 412(35):4570–4578, 2011. Cited on pp. 16 and 18.

[9] F. Bonomo, G. Durán, and J. Marenco. Exploring the complexity boundary between coloring and list-coloring. *Ann. Oper. Res.*, 169(1):3–16, 2009. Cited on pp. 3 and 22.

[10] L. Cai, J. Chen, R. G. Downey, and M. R. Fellows. Advice classes of

parameterized tractability. *Ann. Pure and Appl. Logic*, 84(1):119–138, 1997. Cited on p. 15.

[11] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. *SIAM J. Comput.*, 33(6):1417–1440, 2004. Cited on p. 3.

[12] J. Chen, B. Chor, M. Fellows, X. Huang, D. W. Juedes, I. A. Kanj, and G. Xia. Tight lower bounds for certain parameterized NP-hard problems. *Inform. and Comput.*, 201(2):216–231, 2005. Cited on p. 11.

[13] 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. Cited on pp. 23 and 25.

[14] DIMACS. Maximum clique, graph coloring, and satisfiability. Second DIMACS implementation challenge, 1995. Accessed Dec. 2009. Cited on p. 23.

[15] DIMACS. Graph coloring and its generalizations, 2002. Accessed Dec. 2009. Cited on p. 23.

[16] M. Dom, D. Lokshtanov, and S. Saurabh. Incompressibility through colors and IDs. In *Proc. 36th ICALP*, volume 5555 of *LNCS*, pages 378–389. Springer, 2009. Cited on p. 17.

[17] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999. Cited on pp. 4, 5, and 6.

[18] M. R. Fellows. Towards fully multivariate algorithmics: Some new results and directions in parameter ecology. In *Proc. 20th IWOCA*, volume 5874 of *LNCS*, pages 2–10. Springer, 2009. Cited on p. 5.

[19] 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. *Inform. and Comput.*, 209(2):143–153, 2011. Cited on pp. 3 and 7.

[20] M. R. Fellows, F. V. Fomin, D. Lokshtanov, F. A. Rosamond, S. Saurabh, and Y. Villanger. Local search: Is brute-force avoidable? *J. Comput. Syst. Sci.*, 78(3):707–719, 2012. Cited on p. 4.

[21] 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. Cited on p. 9.

[22] J. Fiala, P. A. Golovach, and J. Kratochvíl. Parameterized complexity of

coloring problems: Treewidth versus vertex cover. *Theor. Comput. Sci.*, 412(23):2513–2523, 2011. Cited on pp. 3 and 7.

[23] J. Flum and M. Grohe. Parameterized complexity and subexponential time. *EATCS Bulletin*, 84:71–100, 2004. Cited on p. 11.

[24] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006. Cited on pp. 4, 5, and 6.

[25] L. Fortnow and R. Santhanam. Infeasibility of instance compression and succinct PCPs for NP. *J. Comput. System Sci.*, 77(1):91–106, 2011. Cited on pp. 16 and 18.

[26] R. Görke, P. Maillard, C. Staudt, and D. Wagner. Modularity-driven clustering of dynamic graphs. In *Proc. 9th SEA*, volume 6049 of *LNCS*, pages 436–448. Springer, 2010. Cited on p. 3.

[27] J. Guo, S. Hartung, R. Niedermeier, and O. Suchý. The parameterized complexity of local search for TSP, more refined. *Algorithmica*, 2013. To appear. Cited on p. 4.

[28] J. Guo and R. Niedermeier. Invitation to data reduction and problem kernelization. *SIGACT News*, 38(1):31–45, 2007. Cited on p. 6.

[29] S. Hartung and R. Niedermeier. Incremental list coloring of graphs, parameterized by conservation. In *Proc. 7th TAMC*, volume 6108 of *LNCS*, pages 258–270. Springer, 2010. Cited on pp. 1 and 27.

[30] M. Hujter and Z. Tuza. Precoloring extension. III. Classes of perfect graphs. *Combinatorics, Probability & Computing*, 5:35–56, 1996. Cited on p. 22.

[31] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *J. Comput. System Sci.*, 63(4):512–530, 2001. Cited on p. 11.

[32] K. Jansen and P. Scheffler. Generalized coloring for tree-like graphs. *Discrete Appl. Math.*, 75(2):135–155, 1997. Cited on p. 22.

[33] T. Kloks. *Treewidth. Computations and Approximations*, volume 842 of *LNCS*. Springer, 1994. Cited on p. 20.

[34] J. Kratochvíl. Precoloring extension with fixed color bound. *Acta Math. Uni. Comenianae*, 62(2):139–153, 1993. Cited on pp. 4 and 22.

[35] D. Marx. Parameterized coloring problems on chordal graphs. *Theor. Comput. Sci.*, 351(3):407–424, 2006. Cited on p. 4.

[36] D. Marx. Precoloring extension on unit interval graphs. *Discrete Appl.*

Math., 154(6):995–1002, 2006. Cited on p. 22.

[37] D. Marx. Searching the k-change neighborhood for TSP is W[1]-hard. *Oper. Res. Lett.*, 36(1):31–36, 2008. Cited on p. 4.

[38] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006. Cited on pp. 4, 5, and 6.

[39] R. Niedermeier. Reflections on multivariate algorithmics and problem parameterization. In *Proc. 27th STACS*, volume 5 of *LIPIcs*, pages 17–32. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010. Cited on p. 5.

[40] F. S. Roberts. On the compatibility between a graph and a simple order. *J. Combin. Theory Ser. B*, 11(1):28–38, 1971. Cited on p. 22.

[41] H. Shachnai, G. Tamir, and T. Tamir. Minimal cost reconfiguration of data placement in a storage area network. *Theor. Comput. Sci.*, 460:42–53, 2012. Cited on p. 4.