# Using Contracted Solution Graphs for Solving Reconfiguration Problems

**Paul Bonsma**[*1] **and Daniël Paulusma**[†2]

1 **University of Twente, Enschede, The Netherlands,** p.s.bonsma@ewi.utwente.nl
2 **Durham University, UK,** daniel.paulusma@durham.ac.uk

**Abstract**

We introduce a dynamic programming method for solving reconfiguration problems, based on *contracted solution graphs*, which are obtained from solution graphs by performing an appropriate series of edge contractions that decrease the graph size without losing any critical information needed to solve the reconfiguration problem under consideration. As an example, we consider a well-studied problem: given two $k$-colorings $\alpha$ and $\beta$ of a graph $G$, can $\alpha$ be modified into $\beta$ by recoloring one vertex of $G$ at a time, while maintaining a $k$-coloring throughout? By applying our method in combination with a thorough exploitation of the graph structure we obtain a polynomial-time algorithm for $(k-2)$-connected chordal graphs.

**1998 ACM Subject Classification** G.2.2 Graph Theory

**Keywords and phrases** reconfiguration, contraction, dynamic programming, graph coloring

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2016.21

## 1 Introduction

Given a search problem we may want to find out if one solution for a particular instance is "close" to another solution of that instance to get more insight into the solution space of the problem. Studying the solution space from this perspective could, for instance, be potentially interesting for improving the performance of corresponding heuristics [16]. Searching the solution space by making small "feasible" moves also turned out to be useful when analyzing randomized algorithms for sampling and counting $k$-colorings of a graph or when analyzing cases of Glauber dynamics in statistical physics (see Section 5 of the survey [19]).

In most general terms, the above situation can be modeled with solution graphs. We formalize this as follows: A *solution graph concept* $\mathcal{S}$ is obtained by defining a set of *instances*, *solutions* for these instances, and a (symmetric) *adjacency relation* between pairs of solutions. For every instance $G$ of the problem, this gives a *solution graph* $\mathcal{S}(G)$, also called a *reconfiguration graph*, which has as node set all solutions of $G$, with edges as defined by the given adjacency relation. (If $G$ has no solutions then $\mathcal{S}(G)$ is the empty graph.) The adjacency relation usually represents a smallest possible change (or *reconfiguration move*) between two solutions of the same instance. For example, the well-known *$k$-Color Graph concept* $\mathcal{C}_k$, related to the $k$-Colorability search problem, is defined as follows: instances are graphs $G$, and solutions are (proper) $k$-colorings of $G$. Two colorings are adjacent if and only if they differ in exactly one vertex. Note however that in general there may be more than one natural way to define the adjacency relation.

* Supported by the European Community's $7^{th}$ Framework Programme (FP7/2007-2013), grant agreement n° 317662.
† Supported by EPSRC Grant EP/K025090/1.





Solution graphs and their properties have been studied very intensively over the last couple of years for a variety of search problems, which include amongst others $k$-Coloring [3, 4, 8, 11, 12, 13, 15, 25], Satisfiability [16, 32], Independent Set [7, 9, 26], Shortest Path [5, 6, 26], List Coloring [18], List Edge Coloring [22, 23], $L(2,1)$-Labeling [24], $H$-Coloring [35] and Subset Sum [20]; see also the aforementioned survey [19]. The study of such solution graphs is commonly called *reconfiguration*.

**Reconfiguration problems.** Both algorithmic and combinatorial questions have been considered in the fast-growing area of reconfiguration. For instance, what is the diameter of $\mathcal{S}(G)$ (in terms of the size of the instance $G$) or if $\mathcal{S}(G)$ is not connected, what is the diameter of its (connected) components? In particular, is the diameter always polynomially bounded or not? This led to the introduction of the $\mathcal{S}$-Connectivity problem, which is that of deciding whether the solution graph $\mathcal{S}(G)$ of a given instance $G$ is connected. Refining this problem leads to the following problem:

$\mathcal{S}$-Reachability
*Instance:* an instance $G$ with two solutions $\alpha$ and $\beta$.
*Question:* is there a path from $\alpha$ to $\beta$ in $\mathcal{S}(G)$?

The $\mathcal{S}$-Reachability problem is a central problem in the area of reconfiguration, which has received much attention in the literature. The problem is sometimes called the *$\alpha$-$\beta$-path* problem for $\mathcal{S}$ [19], whereas the specific case of $\mathcal{C}_k$-Reachability is also known as the $k$-Color Path problem [13]. It is known that $\mathcal{S}$-Reachability is PSPACE-complete for most of the aforementioned solution graph concepts even for special graph classes [8, 17, 21, 29, 34, 36]. For instance, $\mathcal{C}_k$-Reachability is PSPACE-complete even if $k = 4$ and instances are restricted to planar bipartite graphs [8]. This explains that efficient algorithms are only known for very restricted classes of instances. Hence, there is still a need for developing general algorithmic techniques for solving these problems in practice, and for sharpening the boundary between tractable and computationally hard instance classes. Our paper can be seen as the next step in these directions.

**Method.** One important algorithmic technique is *dynamic programming* (DP). In the area of reconfiguration, there are only relatively few successful examples of nontrivial DP algorithms (such as [5, 7, 18, 29]). In this paper, we focus on a DP technique based on the concept of *contracted solution graphs*. This method was first used by Bonsma [5] to obtain an efficient algorithm for a Shortest-Path-Reachability problem restricted to planar graphs. Recently, Hatanaka, Ito and Zhou [18] used this technique for proving that List-Coloring-Reachability is polynomial-time solvable for caterpillars. We will:

1. generalize the ideas of [5, 18] to a unified dynamic programming method,
2. introduce this method in a broader setting,
3. provide useful notation, terminology and basic lemmas, and
4. illustrate the method by giving a new application.

In Section 2 we give a detailed description of the general method of contracted solution graphs. Informally speaking, in dynamic programming one first computes the required information for parts of the instance, and combines/propagates this to compute the same information for ever larger parts of the instance, until the desired information is known for the entire instance. In our case, the instance $G$ can be any relational structure on a ground set, such as (directed) graphs, hypergraphs, satisfiability formulas, or constraint satisfaction problems in general (see e.g. [10]). The order in which the information can be computed or in which parts should be considered is given by a *decomposition* of $G$. The elements of the ground set that are in a processed part $H$ and that have incidences with the unexplored part

are called *terminals*. The key idea behind the method is that reconfiguration moves in the processed part $H$ that do not involve terminals are often irrelevant. The information that is relevant is captured by the notion of a *terminal projection*. These projections assign labels to solutions, yielding so-called *label components*, which are maximally connected subgraphs of $\mathcal{S}(H)$ induced by sets of solutions that all have the same label. A contracted solution graph is obtained from $\mathcal{S}(H)$ by contracting the label components into single vertices. We stress that the general method can readily be applied to *any kind of relational structure*, but in our example we focus on graphs, just as [5] and [18].

**Relation to Other Results.** In [18] dynamic programming was done over a *path decomposition* of the given caterpillar. In [5], a layer-based decomposition of the graph was used (for every $i \in \mathbb{N}$, the subgraph $H_i$ consisted of all vertices at distance at most $i$ of the given shortest path starting from a vertex $s$), which can also be viewed as a path decomposition. Here we focus on the more general *tree decompositions* instead. For our application, we give *full* dynamic programming rules for the $\mathcal{C}_k$-Reachability problem. In particular we introduce a join rule and we allow bags of size larger than 2. Our rules can be used directly for List-Coloring-Reachability as well and thus generalize the rules of [18].

Many well-studied $\mathcal{S}$-Reachability problems (including $\mathcal{C}_k$-Reachability for an appropriate constant $k$) are PSPACE-complete already for graphs of bounded bandwidth [29, 34], and therefore also for graphs of bounded treewidth. Recently, the PSPACE-completeness results from [29, 34] were strengthened to hold even for planar graphs of bounded bandwidth and low maximum degree [36]. Hence we cannot hope to obtain polynomial-time algorithms for graphs of treewidth $w$, for every constant $w$, and certainly not fixed parameter tractable (FPT) algorithms parameterized by $w$, although such results are common when working with decision problems that are only NP-complete instead of PSPACE-complete.

One way to cope with the above problem is to restrict the problem even further. For instance, in a number of recent papers [10, 25, 26, 32, 30, 31] the *length-bounded* version of the $\mathcal{S}$-Reachability problem was studied, that is the problem of finding a path of length at most $\ell$ in the solution graph between two given solutions, in particular with an aim to determine fixed-parameter tractability (observe that the length of a path between two solutions is a natural parameter). For instance, although $\mathcal{C}_k$-Reachability is PSPACE-complete for $k \geq 4$, the length-bounded version is FPT when parameterized by the length $\ell$ [10, 25] (in addition, it is polynomial-time solvable for $k \leq 3$ [25]). In this restricted context, other dynamic programming algorithms over tree decompositions for reconfiguration problems are known: in [29] FPT algorithms are given for various length-bounded reachability problems, parameterized by both the treewidth and the length bound $\ell$. In [28], FPT algorithms are given for the reachability versions of different token reconfiguration problems for graphs of bounded degeneracy (and thus for bounded treewidth), when parameterized by the number of tokens.

Since we wish to solve $\mathcal{S}$-Reachability problems in general, we choose a different approach, and present a generally applicable method. However, because of the aforementioned PSPACE-completeness, we can obviously not guarantee that it terminates in polynomial time for all instances. Nevertheless, one can identify restricted instance classes for which it does yield polynomial-time algorithms, as illustrated by our new application and the two other examples [5, 18]. Moreover, our initial computational studies indicate that this method, with a few additions, performs well in practice for various instances of reconfiguration problems, for which the theoretical complexity status is not yet resolved.

**Our Application.** In Section 3 we illustrate the method by giving dynamic programming rules for the $\mathcal{C}_k$-Reachability problem, which describe how to compute new (larger)

contracted solution graphs from smaller ones. Recall that similar dynamic programming rules can be given for other reconfiguration problems, as done already in [5, 18]. The given rules can be used when a tree decomposition of the graph is given. We emphasize that the rules solve the $\mathcal{C}_k$-Reachability problem correctly for *every* graph $G$ (see e.g. [1, 27] for information on finding tree decompositions). Nevertheless, the algorithm is only *efficient* when the contracted solution graphs stay small enough (that is, polynomially bounded). As indicated by the PSPACE-hardness of the problem, this is not always the case. In the same section, we illustrate the DP rules and show that the size of the contracted solution graphs can grow exponentially, even for 2-connected 4-colorable unit interval graphs.

In Section 4 we apply our method to show that, for all $k \geq 3$, $\mathcal{C}_k$-Reachability is polynomially solvable for $(k-2)$-connected chordal graphs. As unit interval graphs are chordal, the result from Section 3 implies that we need to exploit the structure of chordal graphs to prove this. This is not surprising: although $\mathcal{C}_3$-Reachability can be solved in polynomial time for all graphs [13], $\mathcal{C}_k$-Reachability is PSPACE-complete even for bipartite graphs, and if $k \in \{4, 5, 6\}$ for planar graphs, and if $k = 4$ for planar bipartite graphs [8]. As the proof for the PSPACE-completeness result for bipartite graphs from [8] can be easily modified to hold for $(k-2)$-connected bipartite graphs, our result for $(k-2)$-connected chordal graphs cannot be extended to $(k-2)$-connected perfect graphs. On the positive side, $\mathcal{C}_k$-Connectivity is polynomial-time solvable on chordal graphs. This is due to a more general result of Bonamy et al. [4], which implies that for a chordal graph $G$, $\mathcal{C}_k(G)$ is connected if and only if $G$ has no clique on more than $k-1$ vertices. Hence, our result can be seen as an extension of this result if in addition $(k-2)$-connectivity is imposed. Our result on $\mathcal{C}_k$-Reachability on $(k-2)$-connected chordal graphs is also the first time that dynamic programming over tree decompositions is used to solve the general version of a PSPACE-complete reachability problem in polynomial time for a graph class strictly broader than trees. In Section 5 we discuss possible directions for future work.

**Preliminaries.** For a connected graph $G$, a *vertex cut* is a set $S \subseteq V(G)$ such that $G - S$ is disconnected. Vertices in different components of $G - S$ are *separated* by $S$. For $k \geq 1$, a (connected) graph $G$ is *$k$-connected* if $|V(G)| \geq k+1$ and every vertex cut $S$ has $|S| \geq k$. The *contraction* of an edge $uv$ of a graph $G$ replaces $u$ and $v$ by a new vertex made adjacent to precisely those vertices that were adjacent to $u$ or $v$ in $G$ (this does not create any multi-edges or loops). A graph is *chordal* if it has no induced cycle of length greater than 3.

Let $G$ be a graph. A *$k$-color assignment* of $G$ is a function $\alpha : V(G) \to \{1, \ldots, k\}$. For $v \in V(G)$, $\alpha(v)$ is called the *color* of $v$. It is a *$k$-coloring* if $\alpha(u) \neq \alpha(v)$ for every edge $uv \in E(G)$. A *coloring* of $G$ is a $k$-coloring for some value of $k$. If $\alpha$ and $\beta$ are colorings of $G$ and a subgraph $H$ of $G$, respectively, such that $\alpha|_{V(H)} = \beta$ (that is, $\alpha$ and $\beta$ coincide on $V(H)$) then $\alpha$ and $\beta$ are said to be *compatible*. For an integer $k$, the *$k$-color graph* $\mathcal{C}_k(G)$ has as nodes all (proper) $k$-colorings of $G$, such that two colorings are adjacent if and only if they differ on one vertex. A *walk* from $u$ to $v$ in $G$ is a sequence of vertices $v_0, \ldots, v_k$ with $u = v_0$, $v = v_k$, such that for all $i < k$, $v_i v_{i+1} \in E(G)$. A *pseudowalk* from $u$ to $v$ is a sequence of vertices $v_0, \ldots, v_k$ with $u = v_0$, $v = v_k$, such that for all $i < k$, either $v_i = v_{i+1}$, or $v_i v_{i+1} \in E(G)$. A *recoloring sequence* from a $k$-coloring $\alpha$ of $G$ to a $k$-coloring $\beta$ of $G$ is a pseudowalk from $\alpha$ to $\beta$ in $\mathcal{C}_k(G)$. A *labeled graph* is a pair $G, \ell$ where $G = (V, E)$ is a graph and $\ell : V \to X$ is a vertex labeling (which may assign the same label to different vertices). A *label preserving isomorphism* between two labeled graphs $G_1, \ell_1$ and $G_2, \ell_2$ is an isomorphism $\phi : V(G_1) \to V(G_2)$, such that $\ell_1(v) = \ell_2(\phi(v))$ for all $v \in V(G_1)$. Informally, two labeled graphs $G_1, \ell_1$ and $G_2, \ell_2$ are the same if there exists a label preserving isomorphism between them.

## 2 The Method of Contracted Solution Graphs

In this section we define the concept of *contracted solution graphs* (CSGs) for reconfiguration problems in general. Consider a solution graph concept $\mathcal{S}$, which for every instance $G$ of $\mathcal{S}$ defines a solution graph that is denoted by $\mathcal{S}(G)$. A *terminal projection* for $\mathcal{S}$ is a function $p$ that assigns a *label* to each tuple $(G, T, \gamma)$ consisting of an instance $G$ of $\mathcal{S}$, a set $T$ of *terminals* for $G$ and a solution $\gamma$ for $G$. Terminal projections are used to decide which nodes are "equivalent" and can be contracted. We remark that $G$ and $T$ can be anything, but in our example and in previous examples in the literature [5, 18] $G$ is always a graph, and $T$ is a subset of its vertices. We also note that a terminal projection $p$ can be seen as a node labeling for the solution graph $\mathcal{S}(G)$. So, for every instance $G$ of $\mathcal{S}$, every choice of terminals $T$ may give a different node labeling for the solution graph $\mathcal{S}(G)$. When $G$ and $T$ are clear from the context, we may write $p(\gamma)$ to denote the label of a node $\gamma$ of $\mathcal{S}(G)$.

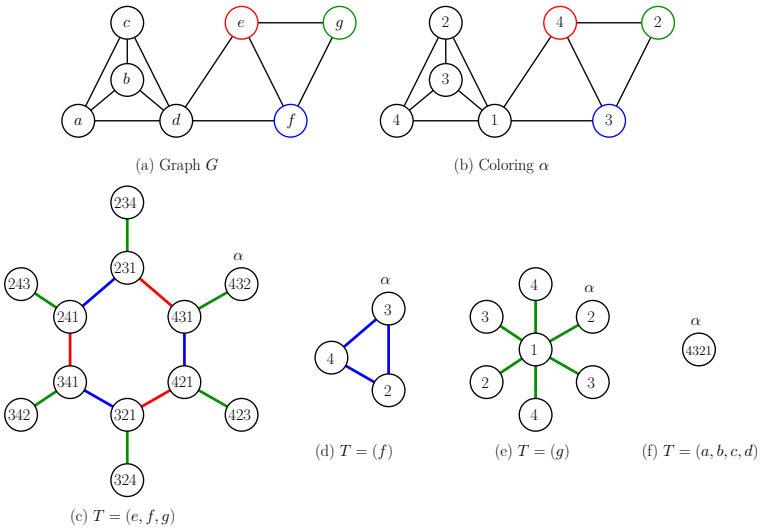
*Example.* Consider the $k$-color graph concept $\mathcal{C}_k$. Let $G$ be a graph. We can define a terminal projection $p$ as follows. Let $T$ be a subset of $V(G)$. The nodes of $\mathcal{C}_k(G)$ are $k$-colorings and we give each node as label its restriction to $T$, that is, for every $k$-coloring $\gamma$ of $G$, we set $p(\gamma) = p(G, T, \gamma) = \gamma|_T$. Note that $\gamma|_T$ is a $k$-coloring of $G[T]$.

Let $p$ be a terminal projection for a solution graph concept $\mathcal{S}$. For an instance $G$ of $\mathcal{S}$ and a terminal set $T$, a *label component* $C$ of $\mathcal{S}(G)$ is a maximal set of nodes $\gamma$ that all have the same label $p(\gamma)$ and that induce a connected subgraph of $\mathcal{S}(G)$. It is easy to see that every solution $\gamma$ of $G$ is part of exactly one label component, or in other words: the label components partition the node set of $\mathcal{S}(G)$. The *contracted solution graph (CSG)* $\mathcal{S}^c(G, T)$ is a labeled graph that has a node set that corresponds bijectively to the set of label components of $G$. For a node $x$ of $\mathcal{S}^c(G, T)$, we denote by $S_x$ the corresponding label component. Two distinct nodes $x_1$ and $x_2$ of $\mathcal{S}^c(G, T)$ are adjacent if and only if there exist solutions $\gamma_1 \in S_{x_1}$ and $\gamma_2 \in S_{x_2}$ such that $\gamma_1$ and $\gamma_2$ are adjacent in $\mathcal{S}(G)$. We define a label function $\ell^*$ for nodes of $\mathcal{S}^c(G, T)$ to denote the corresponding label in $\mathcal{S}(G)$. More precisely: for a node $x$ of $\mathcal{S}^c(G, T)$, the label $\ell^*(x)$ is chosen such that $\ell^*(x) = p(\gamma)$ for all $\gamma \in S_x$. Note that the contracted solution graph $\mathcal{S}^c(G, T)$ can also be obtained from $\mathcal{S}(G)$ by contracting all label components into single nodes and choosing node labels appropriately.

*Example.* Figure 1(c) shows one component of $\mathcal{C}_4(G)$ for the (4-colorable) graph $G$ from Figure 1(a). This is the component that contains all colorings of $G$ whose vertices $a, b, c, d$ are colored with colors $4, 3, 2, 1$, respectively (note that it is not possible to recolor any of these four vertices if one may recolor only one vertex at a time). So in Figure 1(c) the colors of the vertices $a, b, c, d$ are omitted in the node labels, which only indicate the colors of $e, f, g$, in this order. For terminal set $T = \{f\}$, this component contains three label components (of equal size), and contracting them yields the CSG $\mathcal{C}_4^c(G, \{f\})$ shown in Figure 1(d). For $T = \{g\}$, there are seven label components, and the corresponding CSG $\mathcal{C}_4^c(G, \{g\})$ is shown in Figure 1(e). Note that $\mathcal{C}_4^c(G, \{g\})$ contains different nodes with the same label.

We stress that the CSG $\mathcal{S}^c(G, T)$ is a labeled graph that includes the label function $\ell^*$ defined above. However, to keep its size reasonable, the CSG itself does not include the solution sets $S_x$ for each node that were used to define it. For proving the correctness of dynamic programming rules for CSGs the following alternative characterization of CSGs (proof omitted) is useful; note that the sets $S_x$ correspond exactly to the label components.

▶ **Lemma 1.** *Consider an instance $G$ of a solution graph concept $\mathcal{S}$, terminal set $T$ and terminal projection $p$. Let $H, \ell$ be a labeled graph. Then $H, \ell = \mathcal{S}^c(G, T)$ if and only if one can define nonempty sets of solutions $S_x$ for each node $x \in V(H)$ such that the following properties hold:*




(a) Graph $G$ (b) Coloring $\alpha$ (c) $T = (e, f, g)$ (d) $T = (f)$ (e) $T = (g)$ (f) $T = (a, b, c, d)$

**Figure 1** (a) A 4-colorable chordal graph $G$ with $V(G) = \{a, b, c, d, e, f, g\}$. (b) a 4-coloring $\alpha$, and one component of the CSGs of $G$ for four different terminal sets $T$: (c) $\mathcal{C}_4^c(G, \{e, f, g\})$, (d) $\mathcal{C}_4^c(G, \{f\})$, (e) $\mathcal{C}_4^c(G, \{g\})$ and (f) $\mathcal{C}_4^c(G, \{a, b, c, d\})$. The $G[T]$-colorings in the node labels are given as sequences of colors, for the (ordered version of) $T$ as indicated below each CSG. Example (c) can also be seen as the component of $\mathcal{C}_4(G)$ where vertices $a, b, c, d$ receive colors $4, 3, 2, 1$.

1. *$\{S_x \mid x \in V(H)\}$ is a partition of the nodes of $\mathcal{S}(G)$ (the solutions of $G$).*
2. *For every $x \in V(H)$ and every solution $\gamma \in S_x$: $p(G, T, \gamma) = \ell(x)$.*
3. *For every edge $xy \in E(H)$: $\ell(x) \neq \ell(y)$.*
4. *For every $x \in V(H)$: $S_x$ induces a connected subgraph of $\mathcal{S}(G)$.*
5. *For every pair of distinct nodes $x, y \in V(H)$: $xy \in E(H)$ if and only if there exist solutions $\alpha \in S_x$ and $\beta \in S_y$ such that $\alpha$ and $\beta$ are adjacent in $\mathcal{S}(G)$.*

A mapping $S$ that assigns solution sets (or label components) $S_x$ to each node $x$ of $\mathcal{S}^c(G, T)$ that satisfies the properties given in Lemma 1 is called a *certificate* for $\mathcal{S}^c(G, T)$. Given such a certificate $S$ and a solution $\gamma$ for $G$, the *$\gamma$-node* of $\mathcal{S}^c(G, T)$ with respect to $S$ is the node $x$ with $\gamma \in S_x$. For readability, we will not always explicitly mention this certificate when talking about $\gamma$-nodes in $\mathcal{S}^c(G, T)$ (except in Lemma 2 below), but the reader should keep the following convention in mind: when $\gamma$-nodes are identified in $\mathcal{S}^c(G, T)$ for multiple solutions $\gamma$, *these are all chosen with respect to the same certificate.*

*Example.* In Figures 1(c)–(f), the $\alpha$-node for the coloring $\alpha$ shown in Figure 1(b) is marked. In particular consider $\mathcal{C}_4^c(G, \{g\})$ in Figure 1(e). Since the certificate for $\mathcal{C}_4^c(G, \{g\})$ is not actually indicated in the figure, the other leaf with label 2 can also be chosen as the $\alpha$-node (considering the nontrivial label-preserving automorphisms of the graph). Similarly, if we choose a coloring $\beta$ that coincides with $\alpha$ except on nodes $e$ and $f$, where we choose $\beta(e) = 3$ and $\beta(f) = 4$, then the same two leaves (the ones with label 2) of $\mathcal{C}_4^c(G, \{g\})$ can be chosen as the $\beta$-node. Nevertheless, if both an $\alpha$-node and $\beta$-node are marked, then this will only be correct according to the above convention when they are distinct!

The main purpose of our definitions is the following key observation (we omit its proof).

▶ **Lemma 2.** *Let $(G, T)$ be an instance of a solution graph concept $\mathcal{S}$. Let $\mathcal{S}^c(G, T)$ be the contracted solution graph for some terminal projection $p$. Let $\alpha$ and $\beta$ be two solutions and let $x$ and $y$ be the $\alpha$-node resp. $\beta$-node with respect to some certificate $S$. Then there is a path from $\alpha$ to $\beta$ in $\mathcal{S}(G)$ if and only if there is a path from $x$ to $y$ in $\mathcal{S}^c(G, T)$.*

Lemma 2 implies that for a solution graph concept $\mathcal{S}$ and *any* terminal projection $p$ and terminal set $T$, we can decide $\mathcal{S}$-Connectivity if we know $\mathcal{S}^c(G,T)$ (the answer is YES if and only if $\mathcal{S}^c(G,T)$ is connected) and the $\mathcal{S}$-Reachability problem if we know $\mathcal{S}^c(G,T)$ and the $\alpha$-node and the $\beta$-node (the answer is YES if and only if these two nodes are in the same component). However, for obtaining an *efficient* algorithm using this strategy, we must choose the terminal projection $p$ smartly: we need to throw away enough irrelevant information to ensure that $\mathcal{S}^c(G,T)$ will be significantly smaller than $\mathcal{S}(G)$, yet maintain enough information to ensure the efficient computation of $\mathcal{S}^c(G,T)$, without first constructing $\mathcal{S}(G)$. Our strategy for doing this is to use dynamic programming to compute $\mathcal{S}^c(H,T')$ for ever larger subgraphs $H$ of $G$, while ensuring that all of the CSGs stay small throughout the process. The remainder of this paper shows a successful example of this strategy.

## 3 Dynamic Programming Rules for Recoloring

The following terminology is based on widely used techniques for dynamic programming over tree decompositions; see Section 4 and [2, 27, 33] for background information. A *terminal graph* $(G,T)$ is a graph $G$ together with a vertex set $T \subseteq V(G)$, whose vertices are called the *terminals*. If $T = V(G)$, then $(G,T)$ is called a *leaf*. If $v \in T$, then we say that the new terminal graph $(G, T \setminus \{v\})$ is obtained from $(G,T)$ by *forgetting* $v$ (or *using a forget operation*). If $T \neq V(G)$, $v \in T$ and $N(v) \subseteq T$ then we say that $(G,T)$ can be obtained from $(G-v, T \setminus \{v\})$ by *introducing* $v$ (or *using an introduce operation*). Note that for a terminal graph $(G',T')$ with $T' \neq \emptyset$, different graphs can be obtained from $(G',T')$ by introducing a vertex $v$, whereas forgetting a terminal always yields a unique result. As we will see, the condition that each neighbor of the new vertex $v$ must be in $T$ is necessary. We say that $(G,T)$ is the *join of* $(G_1,T)$ *and* $(G_2,T)$ (or *can be constructed using a join operation*) if

- $G_1$ and $G_2$ are induced subgraphs of $G$,
- $V(G_1) \cap V(G_2) = T$ and $V(G_1) \cup V(G_2) = V(G)$,
- $V(G_1) \neq T$ and $V(G_2) \neq T$, and
- for every $uv \in E(G)$, it holds that $uv \in E(G_1)$ or $uv \in E(G_2)$.

We will now focus on CSGs for the $k$-color graph concept $\mathcal{C}_k$, using the terminal projection $p(G,T,\gamma) = \gamma|_T$. We will show how to compute the CSG $\mathcal{C}_k^c(G,T)$ when $(G,T)$ is obtained using a forget, introduce or join operation from a (pair of) graph(s) for which we know the CSG(s). We recall that a variant of these CSGs have been considered before by Hatanaka, Ito and Zhou [18], namely for the case that $|T| = 1$ in the context of list colorings of caterpillars. Similar dynamic programming rules were given in [18]: for the case that $|T| = 1$, they presented a combined introduce and forget rule, and a restricted type of join rule.

We start by stating a trivial rule for computing $\mathcal{C}_k^c(G,T)$ for leaves, which follows from the facts that $\mathcal{C}_k(G)$ has $k$-colorings of $G$ as nodes and that the label $\ell(x)$ of a node $x$ in $\mathcal{C}_k^c(G,T)$ is a $k$-coloring of $G[T]$.

▶ **Lemma 3** (Leaf). *Let $(G,T)$ be a terminal graph with $T = V(G)$. Then $\mathcal{C}_k^c(G,T)$ is isomorphic to $\mathcal{C}_k(G)$ and its label function $\ell$ is the isomorphism from $\mathcal{C}_k^c(G,T)$ to $\mathcal{C}_k(G)$. Moreover, for every $k$-coloring $\gamma$ of $G$, the $\gamma$-node of $\mathcal{C}_k^c(G,T)$ is the node $v$ with $\ell(v) = \gamma$.*

We now give the rules for the forget, introduce and join operations. Figure 2 illustrates the first two rules. We show how Lemma 1 can be applied to prove Lemma 4; the other proofs are similar.

▶ **Lemma 4** (Forget). *Let $(G,T)$ be a terminal graph. For every $v \in T$, it holds that $H', \ell' = \mathcal{C}_k^c(G, T \setminus \{v\})$ can be computed from $H, \ell = \mathcal{C}_k^c(G,T)$ as follows:*

- *For every node $x$ in $H$ with $\ell(x) = \gamma$, let $\ell'(x) = \gamma|_{T \setminus \{v\}}$.*
- *Iteratively contract every edge between two nodes $x$ and $y$ with $\ell'(x) = \ell'(y)$ and assign label $\ell'(z) := \ell'(x)$ to the resulting node $z$.*

*Moreover, for any coloring $\gamma$ of $G$, the $\gamma$-node of $\mathcal{C}_k^c(G, T \setminus \{v\})$ is the node that results from contracting the set of nodes that includes the $\gamma$-node of $\mathcal{C}_k^c(G, T)$.*

**Proof sketch:** Let $S$ denote the certificate for $H, \ell$, so for every node $x$ of $H$, $S_x$ denotes the set of $k$-colorings of $G$ (or *solutions*), such that these sets satisfy the properties stated in Lemma 1. In addition, for every coloring $\gamma$ for which a $\gamma$-node $x$ has been marked in $H$, we may assume that $\gamma \in S_x$. We will prove the statement using Lemma 1 again, by giving a certificate $S'$ for $H', \ell'$, and proving that the five properties hold for these.

The graph $H'$ is obtained by iteratively contracting edges of $H$, so every node $y$ of $H'$ corresponds to a connected set of nodes of $H$, which we will denote by $M_y$. So $\{M_y \mid y \in V(H')\}$ is a partition of $V(H)$. For every node $y \in V(H')$, we define $S'_y = \cup_{x \in M_y} S_x$. For every $k$-coloring $\gamma$ of $G$ such that the $\gamma$-node $x \in V(H)$ is marked, we define the $\gamma$-node of $H'$ to be the node $y$ with $x \in M_y$. Clearly, $\gamma \in S'_y$ then holds, so this is correct. One can now verify that the solution sets $S'_x$ satisfy the five properties stated in Lemma 1. ◀

▶ **Lemma 5** (Introduce). *Let $(G, T)$ be a terminal graph obtained from a terminal graph $(G - v, T \setminus \{v\})$ by introducing $v$. Then $H', \ell' = C_k^c(G, T)$ can be computed as follows from $H, \ell = \mathcal{C}_k^c(G - v, T \setminus \{v\})$:*

- *For every node $x$ of $H$ with label $\ell(x)$, and every color $c \in \{1, \ldots, k\}$: if the (unique) function $\delta : T \to \{1, \ldots, k\}$ with $\delta(v) = c$ and $\delta|_T = \ell(x)$ is a coloring of $G[T]$ then introduce a node $x_c$ with label $\ell'(x_c) = \delta$.*
- *For every pair of distinct nodes $x_c$ and $y_d$: add an edge between them if and only if (1) $x = y$ or (2) $xy$ is an edge in $H$ and $c = d$.*

*Moreover, for every $k$-coloring $\gamma$ of $G$, if $x$ is the $\gamma|_{V(G) \setminus \{v\}}$-node in $H$ and $\gamma(v) = c$, then $x_c$ is the $\gamma$-node of $H'$.*

▶ **Lemma 6** (Join). *Let $(G, T)$ be a terminal graph that is the join of terminal graphs $(G_1, T)$ and $(G_2, T)$. Let $H_1, \ell_1 = C_k^c(G_1, T)$ and $H_2, \ell_2 = C_k^c(G_2, T)$. Then $H, \ell = C_k^c(G, T)$ can be computed as follows:*

- *For every pair of nodes $x \in V(H_1)$ and $y \in V(H_2)$: if $\ell_1(x) = \ell_2(y)$ then introduce a node $(x, y)$ with $\ell((x, y)) = \ell_1(x)$.*
- *For two distinct nodes $(x, y)$ and $(x', y')$, add an edge between them if and only if $xx'$ is an edge in $H_1$ and $yy'$ is an edge in $H_2$.*

*Moreover, for every $k$-coloring $\gamma$ of $G$, if $x$ is the $\gamma|_{V(G_1)}$-node in $H_1$ and $y$ is the $\gamma|_{V(G_2)}$-node in $H_2$, then $(x, y)$ is the $\gamma$-node in $H$.*

*Remark 1.* The DP rules in this section can be generalized further to capture the rules of [18] for the *list coloring* generalization $\mathcal{C}_L$ of $\mathcal{C}_k$. In this generalization, an instance $G, L$ consists of a graph $G$ together with color lists $L(v) \subseteq \{1, \ldots, k\}$ for each $v \in V(G)$. Solutions are now list colorings, which are colorings $\alpha$ of $G$ such that $\alpha(v) \in L(v)$ for each $v \in V(G)$. Adjacency is defined as before. So the *list coloring solution graph* $\mathcal{C}_L(G, L)$ is an induced subgraph of $\mathcal{C}_k(G)$. Hence, it is straightforward to generalize our DP rules to $\mathcal{C}_L$, namely by simply omitting all nodes that correspond to invalid vertex colors.

We now show that components of $\mathcal{C}_k^c(G)$ can grow exponentially even if $G$ is chordal and $k = 4$. First, when considering 4-colorable chordal graphs with cut vertices, it is easy to obtain CSGs that have exponentially large components: take $p$ copies of the graph shown in Figure 1(a),

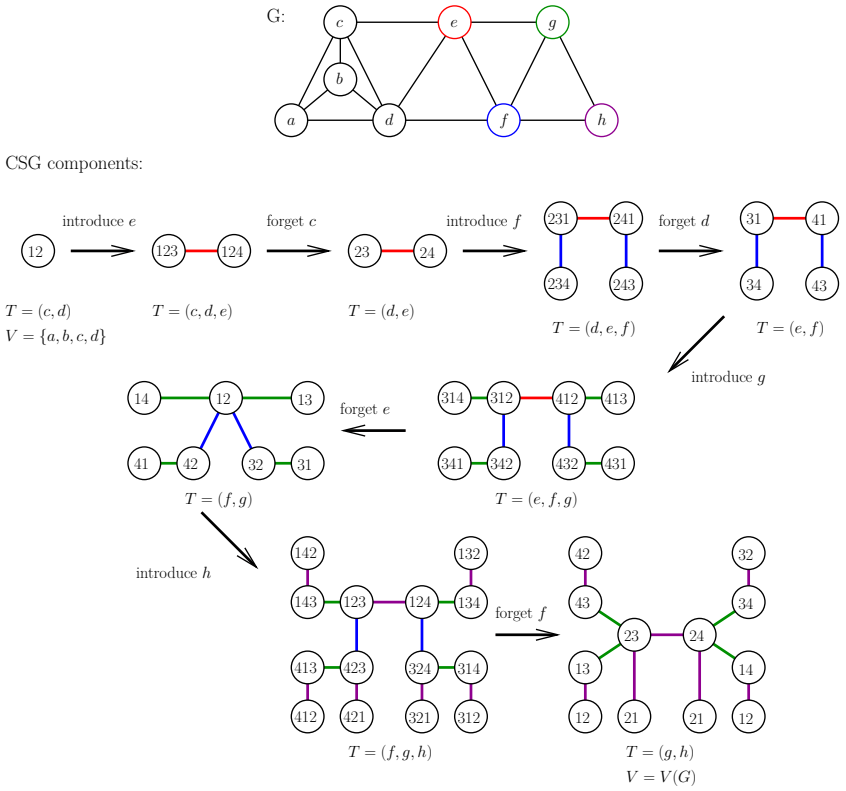



**Figure 2** An example of computing CSGs using forget and introduce operations. A 4-colorable 2-connected chordal graph $G$ with $V(G) = \{a, b, c, d, e, f, g, h\}$ is shown. Note that $G$ is in fact unit interval and isomorphic to the graph $G_8^I$ defined in Section 3. Starting with one component of the CSG $\mathcal{C}_4^c(G[\{a, b, c, d\}], \{c, d\})$, the corresponding component of $\mathcal{C}_4^c(G, \{g, h\})$ is computed, using four forget and introduce operations. The $G[T]$-colorings in the node labels are given as sequences of colors for the ordered version of $T$ as indicated below each CSG. For instance, for $T = (c, d)$, the node label 12 indicates the coloring $\gamma$ with $\gamma(c) = 1$ and $\gamma(d) = 2$.

and identify the $g$-vertices of all of these graphs. Call the resulting graph $G_p^*$. We can show that, for every integer $p \geq 1$, $\mathcal{C}_4^c(G_p^*, \{g\})$ has a component with $1 + 3 \cdot 2^p$ nodes.
We can construct CSGs with exponentially large components for $(k-2)$-connected $k$-colorable chordal graphs, or even 2-connected 4-colorable unit interval graphs, as follows. For $p \geq 4$, let the graph $G_p^I$ have vertex set $\{v_0, \ldots, v_{p-1}\}$, and edge set $\{v_0v_3\} \cup \{v_iv_{i+1} \mid 0 \leq i \leq p-2\} \cup \{v_iv_{i+2} \mid 0 \leq i \leq p-3\}$. A graph isomorphic to $G_8^I$ is shown in Figure 2. Note that each $G_p^I$ is unit interval. To state our claim more precisely, for every $p = 4q + 4$ with $q \in \mathbb{N}$, we can show that the CSG $\mathcal{C}_4^c(G_p^I, \{v_{p-2}, v_{p-1}\})$ has 4! components on at least $2^q$ nodes.

Both examples show that we need to do more than only computing CSGs to solve the problem for $(k-2)$-connected chordal graphs. Next, we will characterize the CSGs and show that it suffices to compute only a part of them.

## 4 Recoloring Chordal Graphs

We will show that CSGs can be used to efficiently decide the $\mathcal{C}_k$-Reachability problem for $(k-2)$-connected chordal graphs. To prove this we use the fact that for a chordal graph $G$ and any clique $T$ of $G$, the terminal graph $(G, T)$ can recursively be constructed from simple

cliques using a polynomial number of clique-based introduce, forget and join operations. We remark that some statements given here are similar to (and can alternatively be deduced from) well-known facts about tree decompositions [14] and nice tree decompositions [27]. However, for readability, and since we need to prove a new bound on the size of any tree decomposition, we give a self-contained presentation.

A *nice tree decomposition* of a terminal graph $(G, T)$ (where $G$ is not necessarily chordal and $T$ may not be a clique) is a tuple $(\mathcal{T}, X, r)$, where $\mathcal{T}$ is a tree with root $r$ and $X$ is an assignment of *bags* $X_u \subseteq V(G)$ for each $u \in V(\mathcal{T})$ that can be defined recursively as follows:

(1) If $T = V(G)$, then the tree $\mathcal{T}$ consists of one (root) node $r$ with bag $X_r = T$.
(2) If $v \in V(G) \setminus T$ and $(\mathcal{T}', X, r')$ is a nice tree decomposition of $(G, T \cup \{v\})$, then a nice tree decomposition for $(G, T)$ can be obtained by adding a new root $r$ with $X_r = T$, and adding the edge $rr'$.
(3) If $(G, T)$ can be obtained from $(G - v, T \setminus \{v\})$ using an introduce operation and $(\mathcal{T}', X, r')$ is a nice tree decomposition of $(G - v, T \setminus \{v\})$, then a nice tree decomposition for $(G, T)$ can be obtained by adding a new root $r$ with $X_r = T$, and adding the edge $rr'$.
(4) If $(G, T)$ can be obtained from $(G_1, T)$ and $(G_2, T)$ using a join operation, and $(\mathcal{T}_1, X, r_1)$ and $(\mathcal{T}_2, X, r_2)$ are nice tree decompositions of $(G_1, T)$ and $(G_2, T)$, then a nice tree decomposition for $(G, T)$ can be obtained by adding a new root $r$ with $X_r = T$ and adding edges $rr_1$ and $rr_2$.

We call a node $u \in V(\mathcal{T})$ a *leaf*, *forget node*, *introduce node* or *join node* if $u$ is added as the root in case (1), (2), (3) or (4), respectively. The *width* of $(\mathcal{T}, X, r)$ is $\max_{u \in V(\mathcal{T})} |X_u| - 1$.

▶ **Lemma 7.** (proof omitted) *Let $(\mathcal{T}, X, r)$ be a nice tree decomposition of $(G, T)$ of width at most $w \geq 1$, and let $n = |V(G)| \geq 1$. Then $|V(\mathcal{T})| \leq (w + 4)n$.*

The bound from Lemma 7 holds for *any* nice tree decomposition, in contrast to the (stronger) bound of [27] which states that for any graph $G$, a nice tree decomposition of $G$ of width at most $4n$ can be constructed (for an appropriate choice of the terminal set $T$). A nice tree decomposition $(\mathcal{T}, X, r)$ of $(G, T)$ is *chordal* if for every node $u \in V(\mathcal{T})$, $X_u$ is a clique of $G$. If $(\mathcal{T}, X, r)$ is a chordal nice tree decomposition of a $k$-colorable graph $G$, then the width of $(\mathcal{T}, X, r)$ is at most $k - 1$. Hence, Lemma 7 shows that any chordal nice tree decomposition has at most $(k + 3)n$ nodes. This bound is asymptotically sharp.

▶ **Theorem 8.** (proof omitted) *There are $k$-colorable chordal graphs $G$ for which any chordal nice tree decomposition has at least $\Omega(kn)$ nodes.*

In order to show how to find a chordal nice tree decomposition in polynomial time we need the following lemma (proof omitted), which tells us how to select the proper type of root node when constructing such a tree decomposition. A terminal graph $(G_1, T_1)$ is called *smaller* than another terminal graph $(G_2, T_2)$ if $2|V(G_1)| - |T_1| < 2|V(G_2)| - |T_2|$.

▶ **Lemma 9.** *Let $(G, T)$ be a terminal graph where $G = (V, E)$ is a chordal graph, and $T$ is a clique with $T \neq V$. If $G - T$ is disconnected, then $(G, T)$ can be obtained from a pair of smaller chordal terminal graphs $(G_1, T)$ and $(G_2, T)$ using a join operation. Otherwise, $(G, T)$ can be obtained from a smaller chordal terminal graph $(G', T')$ using either a forget or introduce operation, where $T'$ is again a clique. For every such $(G, T)$, the relevant operation and subgraph(s) can be found in polynomial time.*

By combining Lemmas 7 and 9 we obtain the following result.

▶ **Corollary 10.** *Let $G$ be a chordal $k$-colorable graph on $n$ vertices, and let $T$ be a clique of $G$. In polynomial time, we can find a chordal nice tree decomposition of $(G, T)$ on at most $(k + 3)n$ nodes.*

**Proof.** Lemma 9 shows how we can choose the proper type of root node. We can build the chordal nice tree decomposition by adding this node to the tree decomposition(s) of (a) smaller graph(s). The entire chordal nice tree decomposition is constructed by continuing this process recursively. Lemma 7 shows that the resulting chordal nice tree decomposition has at most $(w+4)n$ nodes, where $w+1$ is the maximum bag size. Since every bag is a clique of $G$ and the graph is $k$-colorable, we have $w+1 \leq k$, so there are at most $(k+3)n$ nodes. Since we have a polynomial number of nodes, and for every node we spend polynomial time (Lemma 9), the entire process terminates in polynomial time. ◀

The precise complexity bound in Corollary 10 depends on implementation details beyond the scope of this paper.

Using an inductive proof based on Lemma 9, we will now characterize the shape of CSGs for $(k-2)$-connected $k$-colorable chordal graphs. For integers $m, k$ with $1 \leq m \leq k$, a labeled graph $H, \ell$ is an $(m,k)$-*color-complete graph* if there exists a set $T$ with $|T| = m$ such that:

- for all vertices $v \in V(H)$, $\ell(v)$ is a $k$-coloring of a complete graph on vertex set $T$,
- every such $k$-coloring of $T$ appears at exactly one vertex of $H$, and
- two vertices of $H$ are adjacent if and only if their labels differ on exactly one element of $T$.

From this definition it follows that for every pair of integers $m$ and $k$, there is a unique $(m,k)$-color complete graph, up to the choice of $T$. An $(m,k)$-color-complete graph has $k!/(k-m)!$ vertices (this is the number of ways to $k$-color a complete graph on $m$ vertices), and every vertex has degree $m(k-m)$. In particular, if $m = k$ then the graph consists of $k!$ isolated vertices (which is a forest). A labeled graph $H, \ell$ is said to satisfy the *injective neighborhood property (INP)* if for every vertex $u \in V(H)$ and every pair of distinct neighbors $v, w \in N(u)$, it holds that $\ell(v) \neq \ell(w)$. Note that $(m,k)$-color-complete graphs trivially satisfy the INP. We prove Theorem 12 by first showing that for our graphs the following invariant (Theorem 11) is maintained by introduce, forget and join operations. This invariant can be proven by induction based on Lemma 9, using the rules from Section 3.

▶ **Theorem 11.** *Let $k \geq 3$. Let $G = (V,E)$ be a $(k-2)$-connected $k$-colorable chordal graph, and let $T \subseteq V(G)$ be a clique of $G$ with $m = |T| \geq k-2$. Then $\mathcal{C}_k^c(G,T)$ is an $(m,k)$-color-complete graph, or it is a forest that satisfies the injective neighborhood property.*

*Remark 2.* Figure 1 shows that if we relax the connectivity requirement to $(k-3)$-connectedness, the above property does not necessarily hold anymore: the examples in Figure 1(c) and (d) are not forests, and the example in Figure 1(e) does not satisfy the INP.

The characterization of $\mathcal{C}_k^c(G,T)$ in Theorem 11 does not yet guarantee that simply keeping track of the (relevant component of the) CSG yields a polynomial-time algorithm, as shown by the second example in Section 3. However, we will now show that it suffices to only keep track of the following essential information, which remains polynomially bounded.

Let $G = (V,E)$ be a graph with $T \subseteq V$, and let $\alpha$ and $\beta$ be $k$-colorings of a supergraph of $G$. ($G$ should be viewed as a subgraph that occurs during the dynamic programming, while $\alpha$ and $\beta$ are the colorings of the full graph.) Let $\alpha' = \alpha|_V$ and $\beta' = \beta|_V$. If $\mathcal{C}_k^c(G,T)$ is a forest with the $\alpha'$-node $x$ and $\beta'$-node $y$ in the same component, then we define the *$\alpha$-$\beta$-path* to be the unique path in $\mathcal{C}_k^c(G,T)$ with end vertices $x$ and $y$ (together with its vertex labels). Given the two colorings $\alpha$ and $\beta$, the *essential information* for $\mathcal{C}_k^c(G,T)$ consists of the following:

- whether the $\alpha'$ and $\beta'$ nodes appear in the same component,
- whether $\mathcal{C}_k^c(G,T)$ is a forest, and
- in case the answers to both questions are yes: the $\alpha$-$\beta$-path in $\mathcal{C}_k^c(G,T)$.

▶ **Theorem 12.** *For a $k$-colorable $(k-2)$-connected chordal graph $G$ with two $k$-colorings $\alpha$ and $\beta$, we can decide in polynomial time whether $\mathcal{C}_k(G)$ contains an $\alpha$-$\beta$ path.*

**Proof sketch:** Corollary 10 shows that for every chordal $k$-colorable graph $G$ on $n$ vertices, we can find in polynomial time a chordal nice tree decomposition on at most $(k+3)n$ nodes. So every node of this tree decomposition corresponds to a $(k-2)$-connected chordal subgraph $H$ of $G$ with terminal set $T$, such that either $H$ is a clique with $T = V(H)$ (leaf nodes), or $(H, T)$ can be obtained from the graph(s) corresponding to its child node(s) using a forget, introduce or join operation. For every one of those terminal subgraphs, we compute the essential information, bottom up. If at any point, the $\alpha'$ and $\beta'$ nodes are separated, the answer is NO. Forget, introduce and join operations maintain a forest. The lemmas from Section 3 show how the $\alpha$-$\beta$-path can be computed. We return YES if in the root node, a color-complete graph or an $\alpha$-$\beta$-path is obtained (Lemma 2). The total number of operations (tree decomposition nodes) is $O(kn)$. For every operation, the essential information can be computed in polynomial time (in the input size, which includes the path length). One can show that the maximum length of any $\alpha$-$\beta$ path that occurs during the computation is $O(kn)$. Hence, the algorithm terminates in polynomial time. ◀

We stress that $(m, k)$-color complete graphs, which have $k!/(k-m)!$ nodes, are not computed explicitly in our algorithm. So indeed, in order to obtain a polynomial-time algorithm, we do not need to assume that $k$ is a constant.

## 5 Discussion

An obvious question is whether our polynomial-time algorithm for can be extended to all chordal graphs, or whether $\mathcal{C}_k$-Reachability is PSPACE-hard for chordal graphs. As $\mathcal{C}_3$-Reachability is polynomial-time solvable in general [13], the first open case is the complexity of $\mathcal{C}_4$-Reachability for chordal graphs (with at least one cut vertex). We refer to Remark 2 for a discussion on why our current proof technique does not work for this case. We also note that the complexity of $\mathcal{C}_4$-Reachability is open for proper interval graphs (initial experimental results for these graphs seem to suggest that even this problem is not straightforward to solve). The two most important future research goals are the following.

*1. Explore for which other solution graph concepts $\mathcal{S}$ the DP method can be used to obtain polynomial-time algorithms for the $\mathcal{S}$-Reachability problem.*

The DP method has now been used to obtain polynomial-time algorithms for several reconfiguration problems, but its true strength is not always revealed when using the viewpoint of worst-case algorithm analysis. For instance, when considering randomly generated $k$-colorable chordal or interval graphs, we observed that the method performs well on most instances, despite the fact that specialized examples can be constructed that exhibit exponential growth. As we noticed when considering other reconfiguration problems, this behavior seems to occur in general. Because of this, we will write a subsequent paper which will include computational studies, where we apply extensions of this method to various other reconfiguration problems such as well-studied variants of independent set reconfiguration problems (see e.g. [9, 26]).

*2. Explore which known reconfiguration problems can be solved efficiently using CSGs.*

The method of using CSGs can easily be applied to solve the $\mathcal{S}$-Connectivity problem. Hence, this well-studied problem is a suitable candidate problem for the second research goal.

## References


1. H.L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on computing*, 25(6):1305–1317, 1996.
2. H.L. Bodlaender, P. Bonsma, and D. Lokshtanov. The fine details of fast dynamic programming over tree decompositions. In *Proc. IPEC*, volume 8246 of *LNCS*, pages 41–53. Springer, 2013.
3. M. Bonamy and N. Bousquet. Recoloring bounded treewidth graphs. *Electronic Notes in Discrete Mathematics*, 44:257–262, 2013.
4. M. Bonamy, M. Johnson, I.M. Lignos, V. Patel, and D. Paulusma. Reconfiguration graphs for vertex colourings of chordal and chordal bipartite graphs. *Journal of Combinatorial Optimization*, 27:132–143, 2014. URL: `http://dx.doi.org/10.1007/s10878-012-9490-y`, `doi:10.1007/s10878-012-9490-y`.
5. P. Bonsma. Rerouting shortest paths in planar graphs. In *Proc. FSTTCS 2012*, volume 18 of *LIPIcs*, pages 337–349. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.
6. P. Bonsma. The complexity of rerouting shortest paths. *Theoretical Computer Science*, 510:1 – 12, 2013.
7. P. Bonsma. Independent set reconfiguration in cographs and their generalizations. *Journal of Graph Theory*, 2015. doi:10.1002/jgt.21992.
8. P. Bonsma and L. Cereceda. Finding paths between graph colourings: PSPACE-completeness and superpolynomial distances. *Theoretical Computer Science*, 410(50):5215–5226, 2009.
9. P. Bonsma, M. Kamiński, and M. Wrochna. Reconfiguring independent sets in claw-free graphs. In *Proc. SWAT 2014*, volume 8503 of *LNCS*, pages 86–97. Springer, 2014.
10. P. Bonsma, A.E. Mouawad, N. Nishimura, and V. Raman. The complexity of bounded length graph recoloring and CSP reconfiguration. In *Proc. IPEC 2014*, volume 8894 of *LNCS*, pages 110–121. Springer, 2014.
11. L. Cereceda, J. van den Heuvel, and M. Johnson. Connectedness of the graph of vertex-colourings. *Discrete Mathematics*, 308(5):913–919, 2008.
12. L. Cereceda, J. van den Heuvel, and M. Johnson. Mixing 3-colourings in bipartite graphs. *European Journal of Combinatorics*, 30(7):1593–1606, 2009.
13. L. Cereceda, J. van den Heuvel, and M. Johnson. Finding paths between 3-colorings. *Journal of Graph Theory*, 67(1):69–82, 2011.
14. R. Diestel. *Graph theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, Berlin, fourth edition, 2010.
15. C. Feghali, M. Johnson, and D. Paulusma. A reconfigurations analogue of Brooks' theorem. In *Proc. MFCS 2014*, volume 8635 of *LNCS*, pages 287–298. Springer, 2014.
16. P. Gopalan, P.G. Kolaitis, E. Maneva, and C.H. Papadimitriou. The connectivity of boolean satisfiability: Computational and structural dichotomies. *SIAM Journal on Computing*, 38(6), 2009.
17. A. Haddadan, T. Ito, A.E. Mouawad, N. Nishimura, H. Ono, A. Suzuki, and Y. Tebbal. The complexity of dominating set reconfiguration. In *Proc. WADS 2015*, volume 9214 of *LNCS*, pages 398–409. Springer, 2015.
18. T. Hatanaka, T. Ito, and X. Zhou. The list coloring reconfiguration problem for bounded pathwidth graphs. *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, 98(6):1168–1178, 2015.
19. J. van den Heuvel. The complexity of change. In *Surveys in Combinatorics 2013*, pages 127–160. Cambridge University Press, 2013.
20. T. Ito and E.D. Demaine. Approximability of the subset sum reconfiguration problem. In *TAMC 2011*, volume 6648 of *LNCS*, pages 58–69. Springer, 2011. URL: `http://dx.doi.org/10.1007/978-3-642-20877-5_7`, `doi:10.1007/978-3-642-20877-5_7`.

21 T. Ito, E.D. Demaine, N.J.A. Harvey, C.H. Papadimitriou, M. Sideri, R. Uehara, and Y. Uno. On the complexity of reconfiguration problems. *Theoretical Computer Science*, 412(12–14):1054–1065, 2011.

22 T. Ito, M. Kamiński, and E.D. Demaine. Reconfiguration of list edge-colorings in a graph. *Discrete Applied Mathematics*, 160(15):2199–2207, 2012.

23 T. Ito, K. Kawamura, and X. Zhou. An improved sufficient condition for reconfiguration of list edge-colorings in a tree. *IEICE TRANSACTIONS on Information and Systems*, 95(3):737–745, 2012.

24 Takehiro Ito, Kazuto Kawamura, Hirotaka Ono, and Xiao Zhou. Reconfiguration of list $L(2,1)$-labelings in a graph. *Theoretical Computer Science*, 544:84–97, 2014.

25 M. Johnson, D. Kratsch, S. Kratsch, V. Patel, and D. Paulusma. Finding shortest paths between graph colourings. *Algorithmica*, 75(2):295–321, 2016. URL: `http://dro.dur.ac.uk/15595/`.

26 M. Kamiński, P. Medvedev, and M. Milanič. Complexity of independent set reconfigurability problems. *Theoretical Computer Science*, 439:9–15, 2012.

27 T. Kloks. *Treewidth: computations and approximations*, volume 842 of *LNCS*. Springer, 1994.

28 D. Lokshtanov, A.E. Mouawad, F. Panolan, M.S. Ramanujan, and S. Saurabh. Reconfiguration on sparse graphs. In *Proc. WADS 2015*, volume 9214 of *LNCS*, pages 506–517. Springer, 2015.

29 A. E. Mouawad, N. Nishimura, V. Raman, and M. Wrochna. Reconfiguration over tree decompositions. In *Proc. IPEC 2014*, volume 8894 of *LNCS*, pages 246–257. Springer, 2014.

30 A.E. Mouawad, N. Nishimura, and V. Raman. Vertex cover reconfiguration and beyond. In *Proc. ISAAC 2014*, volume 8889 of *LNCS*, pages 452–463. Springer, 2014.

31 A.E. Mouawad, N. Nishimura, V. Raman, N. Simjour, and A. Suzuki. On the parameterized complexity of reconfiguration problems. In *Proc. IPEC 2013*, volume 8246 of *LNCS*, pages 281–294. Springer, 2013.

32 A.E. Mouawad, N. Nishimura, Pathak V., and V. Raman. Shortest reconfiguration paths in the solution space of boolean formulas. In *Proc. ICALP 2015*, volume 9134 of *LNCS*, pages 985–996. Springer, 2015.

33 R. Niedermeier. *Invitation to fixed-parameter algorithms*, volume 31 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, Oxford, 2006.

34 M. Wrochna. Reconfiguration in bounded bandwidth and treedepth. arXiv:1405.0847, 2014. URL: `http://arxiv.org/abs/1405.0847`.

35 M. Wrochna. Homomorphism reconfiguration via homotopy. In *Proc. STACS 2015*, volume 30 of *LIPIcs*, pages 730–742. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

36 T.C. van der Zanden. Parameterized complexity of graph constraint logic. In *Proc. IPEC 2015*, volume 43 of *LIPIcs*, pages 282–293. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.