Combinatorial and Algorithmic aspects of Reconfiguration

Valentin Bartier

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

Spécialité : Mathématiques et Informatique

Arrêté ministériel : 25 mai 2016

Présentée par

Valentin BARTIER

Thèse dirigée par **Myriam PREISSMANN**,CR
et co-encadrée par **Nicolas BOUSQUET**, CNRS

préparée au sein du **Laboratoire Laboratoire des Sciences pour la Conception, l'Optimisation et la Production de Grenoble** dans **l'École Doctorale Mathématiques, Sciences et technologies de l'information, Informatique**

Aspects combinatoires et algorithmiques de la reconfiguration

Combinatorial and Algorithmic aspects of Reconfiguration

Thèse soutenue publiquement le **29 novembre 2021**,
devant le jury composé de :

Madame MYRIAM PREISSMANN
DIRECTEUR DE RECHERCHE EMERITE, CNRS DELEGATION ALPES, Directrice de thèse
Monsieur JEAN CARDINAL
PROFESSEUR, Université Libre de Bruxelles, Rapporteur
Monsieur SEBASTIAN SIEBERTZ
PROFESSEUR, Universität Bremen, Rapporteur
Madame NADIA BRAUNER
PROFESSEUR DES UNIVERSITES, UNIVERSITE GRENOBLE ALPES, Présidente
Monsieur FLORIAN SIKORA
MAITRE DE CONFERENCES, UNIVERSITE PARIS 9 - DAUPHINE, Examinateur
Monsieur FREDERIC HAVET
DIRECTEUR DE RECHERCHE, INRIA- SOPHIA ANTIPOLIS-MEDITERRANEE, Examinateur

Contents

Acknowledgements

Je tiens tout d'abord à remercier mes encadrants de thèse. Et pour beaucoup de choses! En premier lieu, merci de m'avoir fait initialement confiance pour cette thèse alors que je n'avais pas forcément le parcours idéal, et pour avoir maintenu cette confiance tout au long de ma thèse. Merci Nicolas pour ta pédagogie, j'ai énormément appris (a peu prés tout ce que je sais sur les graphes, en fait) en bossant avec toi et toutes les personnes que tu m'as fais rencontré, ces trois années durant. Merci également pour ta patience (je t'en ai fais répéter pas mal, mais en général ça a fini par rentrer!), pour ta présence et ton suivi sans faille! Et pour avoir su me remotiver, toujours sereinement, dans les moments ou ça allait moins bien. Un gros merci à Myriam pour avoir toujours été là dans les moments ou ça compte, même si on a moins eu l'occasion de travailler ensemble. En bref, j'espère sincèrement que de nombreux étudiants auront la chance de faire une thèse avec vous, de bénéficier de votre encadrement, vos connaissances, votre gentillesse. Merci ensuite à tous mes co-auteurs et à ceux avec qui j'ai eu la chance de travailler. Tout d'abord à Amer et Clément (et Nicolas bien entendu, mais tu as déjà eu un paragraphe plus haut!), pour cette collaboration qui dure depuis CoRE 2019, donc depuis le presque tout début de ma thèse! Et qui, je l'espère, se prolongera. Pourquoi pas en présentiel, après toutes ces visios de la période covid? Ensuite, à tous les membres de PGW 2020, Laurine, Marthe, Jonathan, Hoang, (sans oublier Peppie) pour cette semaine fructueuse et cette super ambiance. A tous les membres de notre équipe PACE, Gabriel, Nicolas, Marc, Théo et Ulysse. Et à Julien (on va le publier, cet article!). Je remercie tous les membres de mon jury, pour avoir accepter relire et/ou participer à la soutenance de ma thèse. Je souhaite également un gros merci à tous les potes du G-SCOP, ceux de mon années, les plus anciens, et les nouveaux. Vous êtes trop nombreux pour que je vous cite tous sans en oublier. Mais je souhaite tout de même vous dire à quel point les moments passés ensembles et les bonnes tranches de rigolade ont été important pour moi! Mention spéciale à toutes nos pauses à la cafet' et aux fins d'après-midi (et parfois fin de soirées) passées au Sun, qui nous a acceuilli par temps clair comme par temps gris. Sans ces moments de respirations, terminer cette thèse aurait été autrement plus difficile. J'espère qu'on aura encore de nombreuses occasions de se retrouver tous ensemble, se remémorer nos bonnes blagues et en faire des nouvelles. Bien évidemment, merci également à tous les copains de Marseille (comprenez, également ceux qui ont déménagés) pour votre soutient et pour m'avoir écouté patiemment quand j'essayais maladroitement de vulgariser ma thèse! Pour vous, pas besoin de grandes promesses de retrouvailles, je sais qu'on ne se lachera pas. Un énorme merci à toute la famille également! A mes parents, grand-parents et à Léo, pour votre soutient, votre confiance, vos conseils. J'ai de la chance de vous avoir! Enfin, merci à Marie pour cette dernière année passée ensemble et pour celles qui arrivent, pour tes rires, ton écoute patiente et nos longues discussions sur la vie, et pour tout le reste..

Résumé en français

Cette thèse traite de ce que nous appelons aujourd'hui *problèmes de reconfiguration*. Presque chacun d'entre nous a déjà essayé - si ce n'est réussi - de résoudre un tel problème. En effet, de nombreux jeux très célèbres tels que le jeu du taquin, Rush Hour, ou le plus célèbre de tous, le Rubik's cube, sont dans leur essence des problèmes de reconfiguration. Ainsi, toute personne ayant déjà joué à l'un de ces jeux, que ce soit récemment ou dans son enfance, a déjà été confrontée à certaines des questions que nous abordons dans cette thèse.
Rappelons brièvement le fonctionnement du Rubik's cube à titre d'exemple introductif. Dans ce jeu, un seul joueur reçoit un cube en trois dimensions, chaque face de ce cube étant elle-même divisée en neuf cubes plus petits, également appelés *cubelets*. Chaque cubelet est coloré soit en rouge, jaune, orange, vert, bleu ou blanc et sa couleur ne peut jamais changer. Les cubelets sont initialement placés de manière apparemment aléatoire sur le cube (nous verrons plus tard pourquoi ce n'est pas "vraiment" aléatoire). Un positionnement de tous les cubelets sur le cube est appelé une *configuration* du Rubik's cube. À chaque étape du jeu, le joueur est autorisé à faire tourner une des faces du cube d'un quart de tour, et donc à transformer la configuration actuelle en une autre. Le but est d'obtenir la configuration où chaque face contient des cubes de la même couleur. Une première chose à noter est que tout problème de reconfiguration peut en fait être énoncé de manière similaire : Étant donné une configuration initiale d'un système (ici le Rubik's cube), le but est d'atteindre une configuration cible (chaque face contient des cubelets de la même couleur) en appliquant successivement une opération atomique pour transformer une configuration en une autre (rotation d'une face du cube). Une opération qui modifie la configuration d'un système est appelée une *opération de reconfiguration*.
Revenons à l'exemple particulier du Rubik's cube. Théoriquement, on pourrait atteindre cette configuration cible en faisant tourner les faces du cube de façon aléatoire à chaque étape. Cependant, cela pourrait obliger le joueur à passer en revue les 42252003274489856000 (environ 43 quintillion, pour faire court) configurations différentes qui peuvent être atteintes par des rotations successives des faces, ce qui n'est évidemment pas humainement possible. Supposons donc que vous ayez à votre disposition un ordinateur moderne équipé d'une unité centrale de $3,0$GHz capable de faire tourner une face du Rubik's cube à chacune de ses opérations. Il faudrait encore (avec un certain nombre d'hypothèses simplificatrices) environ 13943161080 secondes, soit environ 442 années, pour que cet ordinateur visite toutes les configurations du cube. En d'autres termes, l'énumération de toutes les possibilités pour résoudre le problème est hors de portée. Le fait qu'il existe un nombre incroyablement élevé de configurations atteignables, que l'on appelle souvent l'*explosion combinatoire*, est l'une des principales difficultés auxquelles on est confronté lorsqu'on étudie les problèmes de reconfiguration. La première question qui se pose face à un Rubik's cube est alors la suivante : comment atteindre cette configuration cible en aussi peu d'étapes que

possible ?
Revenons sur un point que nous avons éludé précédemment. Comme dit dans le paragraphe précédent, la configuration initiale du cube est "apparemment" aléatoire. Cependant, une chose est cachée pour le joueur : avant de vendre le cube, le fabricant sélectionne une configuration initiale de telle sorte que vous puissiez effectivement atteindre la configuration cible en faisant tourner les faces, en partant de celle sélectionnée. Qu'est-ce que cela signifie? Supposons que l'on soit capable de détacher chaque cubelet un par un, et qu'on les rattache ensuite au hasard sur le cube. En procédant ainsi, il y a de très fortes chances que la configuration cible ne puisse plus être atteinte, quel que soit le nombre de mouvements que vous effectuez. En d'autres termes, la configuration obtenue après avoir rattaché les cubelets se trouve dans un "espace de configuration" différent de celui de la configuration cible. Une deuxième question se pose alors, qui cette fois ne vient probablement pas à l'esprit du joueur : comment le fabricant peut-il s'assurer que la configuration cible peut être atteinte à partir de la configuration initiale qu'il choisit pour son cube ?

Ces deux questions que nous avons posées à propos du jeu du Rubik's cube peuvent être généralisées et sont au cœur de l'étude des problèmes de reconfiguration. Le dernier, de nature combinatoire, est généralement appelé le *problème de l'accessibilité*

Question 1 (Accessibilité)**.** *Étant donné une configuration initiale et une configuration cible d'un système et une opération de reconfiguration, est-il possible d'atteindre la configuration cible à partir de la configuration initiale en appliquant successivement l'opération donnée ?*

Le premier, de nature algorithmique, est généralement appelé le problème du *plus court chemin*

Question 2 (Plus court chemin)**.** *Étant donné une configuration initiale et une configuration cible d'un système et une opération de reconfiguration, comment puis-je atteindre la configuration cible à partir de la configuration initiale en appliquant le moins d'opérations possible ?*

Nous définissons formellement ces deux problèmes, qui constituent le fil rouge de cette thèse, ainsi que les questions connexes dans le premier chapitre.

Notion de graphes. Dans cette thèse, nous étudions le cas particulier des problèmes de reconfiguration sur les *graphes*. Un graphe est un objet mathématique qui représente des relations entre les éléments d'un ensemble appelé *sommets*. Notez qu'un sommet peut être n'importe quoi : un lieu, une personne, une machine ou même une notion abstraite. En d'autres termes, un graphe relie les sommets deux à deux pour représenter une relation entre eux. Un lien entre deux sommets est appelé *une arête*. Deux sommets reliés par une arête sont dits *adjacents*. Un graphe peut, par exemple, représenter un réseau routier, où chaque sommet représente une ville, deux villes étant adjacentes s'il

existe une route pour aller de l'une à l'autre. Les graphes sont en particulier des outils très pratiques car ils peuvent être facilement dessinés et visualisés, comme l'illustre la figure 1.1. Mais surtout, ils permettent d'extraire les informations essentielles d'un problème : reprenons l'exemple d'un réseau routier et supposons que nous voulions trouver le chemin le plus court entre la ville A et la ville B. Il y a beaucoup de paramètres différents à prendre en compte : la position des feux de signalisation, la densité du trafic, les travaux routiers en cours entre les villes, et bien d'autres encore. Cependant, avant de pouvoir prendre en compte toutes ces informations, nous devons résoudre une question essentielle : comment choisir le chemin le plus court, parmi le nombre probablement énorme de chemins allant de A à B ? Le modèle apparemment simple que donne un graphe capture déjà une partie importante du problème et permet de concevoir un premier algorithme de haut niveau pour trouver le plus court chemin entre deux points. Cela ne signifie pas que ce premier algorithme soit nécessairement simple. Au contraire, de nombreux problèmes sur les graphes sont difficiles dans leur essence. Nous en citerons quelques-uns dans les sections suivantes.
Soulignons également le fait qu'étant des objets purement mathématiques, les graphes peuvent être étudiés en tant que tels sans nécessairement modéliser des situations ou des systèmes du monde réel. Ils sont au carrefour de nombreux domaines des mathématiques discrètes (algorithmique, combinatoire, topologie..) et permettent de développer des notions complexes, souvent nécessaires à la conception d'algorithmes efficaces.
Une grande variété de problèmes peut être représentée par des graphes. En outre, il arrive souvent que des problèmes apparemment très différents se ramènent en fait à la même question lorsqu'ils sont modélisés sous forme de graphes. Considérons par exemple les problèmes suivants :

1. Un certain nombre d'examens doivent être organisés dans des salles de classe. Le problème consiste à trouver une répartition des examens dans les salles de classe, de telle sorte que deux examens n'aient pas lieu en même temps dans la même salle.

2. Un certain nombre d'antennes sont placées sur un territoire. Le problème est d'attribuer une fréquence à chaque antenne, de manière à ce que deux antennes proches l'une de l'autre n'aient pas la même fréquence, afin d'éviter les interférences.

Donnons la représentation de ces problèmes en tant que problèmes de graphes. Pour le problème 1, chaque examen est représenté par un sommet du graphe. Deux examens sont reliés par une arête si et seulement si leurs créneaux horaires se superposent. Chaque salle de classe est représentée par une couleur unique : le problème se réduit alors à trouver une assignation des couleurs aux sommets du graphe, de telle sorte que deux sommets adjacents ne partagent pas la même couleur. Il n'est alors pas difficile de voir que le second problème est en fait le même : chaque sommet représente une antenne, et chaque couleur représente une fréquence. Deux antennes sont adjacentes

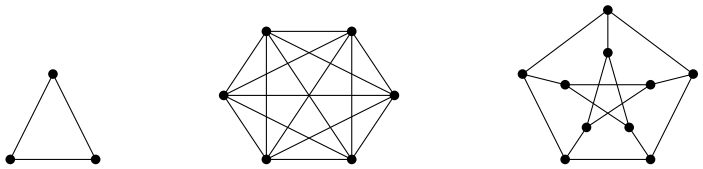


FIGURE 1: Exemples de graphes. Les sommets sont représentés par des points noirs et les arêtes par des lignes noires pleines. De gauche à droite : le cycle sur trois sommets C_3 (également appelé triangle), le graphe complet sur 6 sommets C_6, et le graphe de Petersen.

si elles sont proches l'une de l'autre et doivent avoir une fréquence différente.
Le problème consistant à trouver une affectation de couleurs aux sommets d'un graphe telle que deux sommets adjacents ne partagent pas la même couleur est sobrement intitulé le *problème de coloration*. C'est l'un des problèmes les plus célèbres et les plus étudiés en théorie des graphes, et il présente également un intérêt particulier dans cette thèse. Plus particulièrement, nous nous concentrons sur la variante reconfiguration du problème de coloration, le *problème de recoloration* que nous décrivons dans le paragraphe suivant.

L'autre problème qui nous intéresse dans cette thèse est le problème de l'*ensemble indépendant*. Expliquons-le à travers un autre exemple. Supposons que vous organisiez une fête pour votre anniversaire. Vous disposez d'une liste d'amis que vous voulez inviter et vous savez que certains d'entre eux s'entendent bien et que d'autres ne s'entendent pas. Votre objectif est d'inviter le plus grand nombre de personnes possible sans créer de conflit. Ce problème peut être modélisé comme un problème de graphe comme suit : chaque sommet représente une personne, deux personnes étant reliées par une arête s'il y a un conflit entre elles. Votre objectif est donc de sélectionner le plus grand ensemble de sommets qui ne sont pas deux à deux adjacents dans ce graphe. Un tel ensemble est appelé un ensemble indépendant (maximum). Comme pour le problème de coloration, nous considérons la version de reconfiguration du problème des ensembles indépendants, le problème de la *reconfiguration des ensembles indépendants* que nous décrivons ci-dessous.

Graphes et reconfiguration. Rappelons brièvement la définition (informelle) d'un problème de reconfiguration que nous avons donnée en utilisant l'exemple du Rubik's cube. On nous donne un système (le cube), une configuration initiale et une configuration cible (les couleurs des cubelets sur chaque face du cube) et une opération de reconfiguration (tourner une face du cube d'un quart de tour). Le but est de trouver une séquence d'opérations de reconfiguration qui transforme la configuration initiale en la configuration cible. Dans notre cas, le système sera un graphe. Les configurations et les opérations de reconfiguration dépendent du problème que nous considérons.

La première partie de cette thèse étudie le problème de recoloration de graphes. Dans ce problème, une configuration est une coloration des sommets du graphe.

Rappelons que pour qu'une telle coloration soit valide, deux sommets adjacents doivent avoir des couleurs différentes. Une coloration qui respecte cette propriété est appelée une *coloration propre* du graphe. L'opération de reconfiguration que nous considérons consiste à changer la couleur d'un unique sommet à chaque étape. Nous avons de plus deux contraintes : le nombre de couleurs que nous pouvons utiliser est limité, et les colorations successives que nous obtenons doivent rester propres. Nous pouvons maintenant définir le problème *d'accessibilité pour les colorations* :

Problem 1. *Étant donné deux colorations propres d'un graphe utilisant au plus k couleurs, est-il possible de transformer l'une en l'autre en changeant la couleur d'un sommet à la fois, tout en maintenant toujours une coloration correcte ?*

La deuxième partie de cette thèse étudie le problème de reconfiguration d'ensembles indépendants. Dans ce problème, une configuration est un ensemble indépendant du graphe. Rappelons qu'un ensemble indépendant est un ensemble de sommets du graphe deux à deux non adjacents. On nous donne ensuite un ensemble de *jetons* qui sont placés sur les sommets de la configuration actuelle (ensemble indépendant). Dans cette thèse, nous considérons différentes opérations de reconfiguration pour le problème de reconfiguration d'ensemble indépendant. Cependant, afin de garder ce résumé concis, nous n'en présentons qu'une seule ici. La description des autres opérations de reconfiguration que nous considérons est donnée dans le Chapitre 6. L'une des opérations que nous considérons consiste à déplacer un jeton le long d'une arête du graphe. En d'autres termes, un jeton peut glisser d'un sommet à un autre sommet si les deux sommets sont adjacents. Nous avons de plus la contrainte qu'à chaque étape, l'ensemble des sommets sur lesquels se trouvent les jetons doit être un ensemble indépendant. Définissons maintenant le problème *d'accessibilité pour les ensembles indépendants* :

Problem 2. *Étant donné deux ensembles indépendants d'un graphe, est-il possible de transformer l'un en l'autre en faisant glisser un jeton le long d'une arête à chaque étape, de telle sorte que les jetons soient toujours placés sur un ensemble indépendant ?*

Des exemples de séquences de recoloration et de reconfiguration d'ensembles indépendants sont donnés dans la figure 1.2 et la figure 1.3. Soulignons enfin le fait que les problèmes de reconfiguration trouvent leurs applications dans différents domaines, allant de la résolution de jeux à un seul joueur, à - de façons plus surprenante - la physique statistique. Les connexions des problèmes de recoloration de graphes et de reconfiguration d'ensembles indépendants avec ces domaines sont décrites respectivement dans les chapitres 2 et 6.

Chapter 1

Introduction

This thesis deals with what we call today *reconfiguration problems*. Almost anyone of us has already tried - if not succeeded - to solve such a problem. Indeed, many very famous games such as the 15-puzzle game, Rush Hour, or the most famous of them all, the Rubik's cube, are in their essence reconfiguration problems. Thus, anyone who has ever played to one of these games, be it recently or as a child, has already faced some of the questions that we tackle in this thesis.
Let us briefly recall how the Rubik's cube works as an introductory example. In this game, a single player is given a three dimensional cube, each face of this cube being itself divided in nine smaller cubes, also called *cubelets*. Every cubelet is colored either red, yellow, orange, green, blue or white and its color can never change. The cubelets are initially placed in a seemingly random manner on the cube (we will see later on why it is not "really" random). We call a positioning of all the cubelets on the cube a *configuration* of the Rubik's cube. At each step of the game, the player is allowed to rotate one the faces of the cube a quarter turn, and hence to transform the current configuration into another one. The goal is to obtain the configuration where every face contains cubelets of the same color. A first thing to note is that any reconfiguration problem can actually be stated in a similar way: Given an initial configuration of a system (here the Rubik's cube) the goal is to reach a target configuration (every face contains cubelets with the same color) by applying successively an atomic operation to transform a configuration into one another (rotating a face of the cube). An operation that modifies a configuration of a system is called a *reconfiguration operation*.
Let us go back to the particular example of the Rubik's cube. Theoretically, one could reach this target configuration by rotating the faces of the cube randomly at each step. However, doing so may require the player to go through the 42252003274489856000 (about 43 quintillion, for short) different configurations that can be reached by successive rotations of the faces, which is obviously not humanly possible. Suppose then that you have at your disposal a modern day computer equipped with a 3.0GHz CPU that can rotate a face of the Rubik's cube during each one of its operations. This would still require (with a number of simplifying assumptions) about 13943161080 seconds, or about 442 years, for this computer to visit all the configurations of the cube. In other words, enumerating all the possibilities to solve the problem is out of reach. The fact that there is an amazingly huge number of reachable configurations, which we

often refer to as the *combinatorial explosion*, is one of the main difficulties one faces when studying reconfiguration problems. The first question that arises when faced with a Rubik's cube is then the following: how can one reach this target configuration in as little steps as possible?
Let us come back to a point that we previously eluded. As said in the previous paragraph, the initial configuration of the cube is "seemingly" random. However, one thing is hidden: before selling the cube, the manufacturer selects an initial configuration in such a way that you can indeed reach the target configuration by rotating the faces, starting from the one selected. Suppose that you are able to detach each cubelet one by one, and that you then reattach them randomly on the cube. By doing so, there are very high chances that the target configuration cannot be reached anymore, no matter how many moves you make. In other words, the configuration obtained after reattaching the cubelets is in a component of the configuratin space that is different from the one of the target configuration. A second question then arises, which this time probably does not come into the mind of the regular player: how can the manufacturer ensure that the target configuration can be reached from the initial configuration selected for its cube?

These two questions that we asked about the Rubik's cube game can be generalized and are at the heart of the study of reconfiguration problems. The latter one, of combinatorial nature, is usually referred to as the *reachability* problem:

Question 3 (Reachability)**.** *Given an initial and a target configuration of a system and a reconfiguration operation, is it possible to reach the target configuration starting from the initial one by successively applying the given operation?*

The former one, of algorithmic nature, is usually referred to as the *shortest path* problem:

Question 4 (Shortest path)**.** *Given an initial and a target configuration of a system and a reconfiguration operation, how can one reach the target configuration from the initial one by applying as little operations as possible?*

We will define formally these two problems, which are the common thread of this thesis, as well as related questions in the next sections.

Notion of graphs. In this thesis, we investigate the particular case of reconfiguration problems on *graphs*. A graph is a mathematical object that represents pairwise relations between elements of a set called *vertices*. Note that a vertex can be anything: a location, a person, a machine or even an abstract notion. In other words, a graph links vertices two by two to represent a relation between them. A link between two vertices is called *an edge*. Two vertices that are linked by an edge are said to be *adjacent*. A graph can, for instance, represent a road network, where each vertex represents a city, two cities being adjacent if there is a road to go from one to the other.

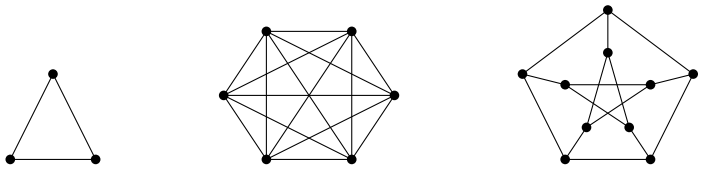


FIGURE 1.1: Examples of graphs. Vertices are represented by black dots and edges are represented by plain black lines. From left to right: the cycle on three vertices C_3 (also known as the triangle), the complete graph on 6 vertices C_6, and the Petersen graph.

Graphs are in particular very handy tools as they can be easily drawn and visualized, as illustrated in Figure 1.1. But more importantly they allow for the extraction of essential information of a problem: Consider again the example of a road network and suppose that we want to find the shortest path between the city A and the city B. There are many different things to take into account: the position of the traffic lights, the density of the traffic, the ongoing roadworks between cities, and many more. However, before we are able to take all these information into account, there is an essential question we need to solve: how can we choose the shortest path, among the possibly huge number of paths going from A to B? The seemingly simple model that a graph gives already captures a hard part of the problem and allows for the design of a first, high level algorithm to find a shortest path between two points. This does not mean that this first algorithm is necessarily simple. At the contrary, many problems on graphs are hard in their essence. We will cite a few in the next sections.

Let us also outline the fact that being purely mathematical objects, graphs can be studied as such without necessarily modeling real-world situations or systems. They are at the crossroad of many different fields of discrete mathematics (algorithmics, combinatorics, topology...) and allow for the development of complex notions, often needed for the design of efficient algorithms.

A large variety of problems can be represented as graphs. Furthermore, it is often the case that problems that are seemingly very different actually reduce to the same question when modeled as graphs problems. Consider for instance the following problems:

1. A number of exams must be held in classrooms. The problem is to find an assignement of the exams to the classrooms, such that no two exams are held at the same time in the same classroom.

2. A number of antennas are placed in an area. The problem is to assign a frequency to each antenna, in such a way that two antennas that are close to each other do not have the same frequency, in order to avoid interferences.

Let us give the graph representation of these problems. For problem 1, each exam is a vertex of the graph. Two exams are linked by an edge if and only if their time slots intersect. Each classroom is represented by a unique color: the problem then reduces to finding an assignation of the colors to the vertices of the graph, in such

a way that no two adjacent vertices share the same color. It is then not hard to see that the second problem is actually the same: each vertex represents an antenna, and each color represents a frequency. Two antennas are adjacent if they are close to each other and must have different frequency.
The problem of finding an assignation of colors to the vertices of a graph such that two adjacent vertices do not share the same color is called the *coloring problem*. It is one of the most famous and most studied problem in graph theory, and is also of particular interest in this thesis. More particularly, we focus on the reconfiguration variant of the coloring problem, the *recoloring problem* that we describe in the next paragraph.
The other problem of interest in this thesis is the *independent set* problem. Let us explain it through another example. Suppose that you are organizing a party for your birthday. You have a list of friends you want to invite and you now that between them, some get along and some do not. Your goal is to invite as many people as possible without creating any conflict. This problem can be modeled as a graph problem as follows: each vertex represents a person, two persons being linked by an edge if there is a conflict between them. Your aim is then to select the largest set of vertices that are pairwise non adjacent in this graph. Such a set is called a (maximum) *independent set*. Just as for the coloring problem, we consider the reconfiguration version of the independent set problem, the *independent set reconfiguration* problem which we describe below.

Graphs and reconfiguration. Le us briefly recall the (informal) definition of a reconfiguration problem we gave using the example of the Rubik's cube. We are given a system (the cube), an initial and a target configuration (the colors of the cubelets on each face of the cube) and a reconfiguration operation (turn a face of the cube a quarter turn). The goal is to find a sequence of reconfiguration operations that transforms the initial configuration into the target one. In our case the system will be a graph. The configurations and reconfiguration operations depend on the problem we consider.

The first part of this thesis investigates the graph recoloring problem. In this problem, a configuration is a coloring of the vertices of the graph. Recall that in order for such a coloring to be valid, two adjacent vertices must have different colors. A coloring that respects this property is called a *proper coloring* of the graph. The reconfiguration operation we consider consists in changing the color of a unique vertex at each step. We have furthermore two constraints: the number of colors we can use is limited, and the successive colorings we obtain must remain proper. We can now define the *coloring reachability* problem:

Problem 3. *Given two proper colorings of a graph using at most k colors, is it possible to transform one into the other by changing the color of one vertex at a time, while always maintaining a proper coloring?*

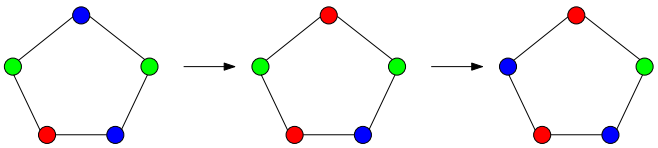


FIGURE 1.2: Example of recoloring sequence. Adjacent vertices never have the same color.

The second part of this thesis investigates the independent set reconfiguration problem. In this problem, a configuration is an independent set of the graph. Recall that an independent set is a set of vertices of the graph pairwise non adjacent. We are further given a set of *tokens* that are placed on the vertices of the current configuration (independent set). In this thesis we consider different reconfiguration operations for the independent set reconfiguration problem. However, for the sake of keeping this introduction concise we only introduce one here. The description of the other reconfiguration operations we consider is given in Chapter 6. One of the operations we consider consists in the moving of one token along an edge of the graph. In other words, a token can slide from a vertex to another vertex as long as the two vertices are adjacent. We have the constraint that, at each step, the set of vertices on which the tokens lie must be an independent set. Let us now define the *independent set reachability* problem:

Problem 4. *Given two independent sets of a graph, is it possible to transform one into the other by sliding a token along an edge at each step, in such a way that the tokens are always placed on an independent set?*

Examples of recoloring sequences and independent set reconfiguration sequences are given in Figure 1.2 and Figure 1.3. Let us finally outline the fact that reconfiguration problems find their applications in different domains, going from the resolution of single player games, to - more surprisingly - statistical physics. The connections of the graph recoloring and independent set reconfiguration problems with these fields are described in Chapter 2 and 6 respectively.
In the next section of this chapter, we give definitions about graphs and complexity that will be useful throughout the manuscript. In Section 1.2 we give a brief and more formal introduction to reconfiguration. We conclude this chapter by giving an overview of the work that is presented in this manuscript in Section 1.3.

1.1 Preliminaries

In this section, we introduce the main concepts, definitions and notations that will be useful all along the manuscript. We refer the interested reader to the classical textbooks [36, 97] for complete introductions to graph theory and to [5, 34] for complete introductions to computational complexity theory and parameterized complexity theory.

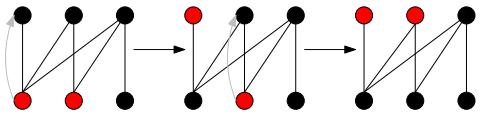


FIGURE 1.3: Example of independent set reconfiguration sequence. Tokens are represented in red and moves by gray arrows. Adjacent vertices never hold a token at the same time.

1.1.1 Computational complexity

In this section we give a short and informal introduction to complexity theory. In particular we only consider *decision problems* in this section. A *decision problem* is a question whose answer is either yes or no. An *instance* of a decision problem is an input given to an algorithm that solves the problem. We say that an algorithm $\mathcal{A}$ *decides* a decision problem π if algorithm $\mathcal{A}$ answers correctly to problem π for any instance of the problem. Such an algorithm is called a *decision algorithm*. The coloring reachability problem described in Problem 3 is an example of decision problem, where the input is a graph along with two colorings of the graph, and the question is whether one coloring can be transformed into the other as described in the previous section. One of the aim of computational complexity is to classify decision problems depending on how efficiently they can be decided. In other words, we want to measure the efficiency of decision algorithms. There are two main quantities to measure when trying to assess the efficiency of an algorithm: the amount of time the algorithm takes to solve the problem, and the amount of space it requires to do so.

A problem π belongs to the class P (which stands for *polynomial time*) if there exists an algorithm that decides π and that terminates in polynomial time (with regards to the size of the input) on any given instance. Problems that belong to this class are usually considered to be "easy", as polynomial-time is considered to be a "reasonable" amount of time (although it completely depends on the degree of the polynomial). A problem π belongs to the class NP (which stands for *non-deterministic polynomial time*) if there exists a non-deterministic algorithm that decides π and terminates in polynomial-time. Very loosely speaking, an algorithm is *non-deterministic* if it can explore every possible choice at the same time, every-time a choice has to be made during its execution. In other words we can consider that it is a deterministic algorithm that is given a sequence of choices that leads to the solution (if there exists one). Therefore, a problem belongs to NP if a deterministic algorithm can verify in polynomial time if a given candidate solution is indeed a solution or not. It is not hard to see that $\mathsf{P} \subseteq \mathsf{NP}$. However, whether this inclusion is strict or not is one of the main open problem in computer science (and is most famously known as the $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ problem).

The classes corresponding to P and NP for space complexity are respectively PSPACE and $\mathsf{NPSPACE}$. A problem belongs to PSPACE if it can be decided by an algorithm that uses a polynomial amount of space (with regards to the size of the input). A problem belongs to $\mathsf{NPSPACE}$ if it can be decided by a non-deterministic algorithm

that uses a polynomial amount of space (with the same definition of non-deterministic algorithm). Again, it is not hard to see that PSPACE $\subseteq$ NPSPACE. In a famous article, Savitch [85] actually showed that the converse inclusion is also true and hence that PSPACE = NPSPACE. We also have inclusions between time and space complexity classes. Indeed, since an algorithm that runs in polynomial time can use at most a polynomial amount of space, we have NP $\subseteq$ PSPACE. Summarizing the known results, we have the following chain of inclusions: P $\subseteq$ NP $\subseteq$ PSPACE(= NPSPACE).
There also exists distinctions between problems of the same class. A way to classify decision problems within a same class is through the use of *polynomial-time reductions*. A problem π_2 can be *reduced* to a problem π_1 if there exists a polynomial-time algorithm which given an instance $\mathcal{I}$ of the problem π_2, outputs an instance $\mathcal{I}'$ of the problem π_1 in polynomial time, such that the answer to $\mathcal{I}$ is yes if and only if the answer to $\mathcal{I}'$ is yes. Hence, if a problem π_2 is reducible to a problem π_1, any algorithm that decides π_1 can be used to decide π_2 (up to paying the "small" polynomial cost of the reduction). Given a class X, a problem π is X*-hard* if any problem of the class X can be reduced to π in polynomial time. If furthermore the problem π belongs to the class X, then π is X*-complete*. Informally speaking, an X-hard problem is at least as hard as any other problem in the class X, and being able to solve efficiently an X-hard problem means that any problem in the class can also be solved efficiently. Assuming that P $\neq$ NP, the NP-hardness of a problem π shows that there exists no polynomial-time algorithm that solves π. Cook's famous theorem [33] shows that the SAT problem is NP-complete. It is one of the most classical problem to reduce to when trying to prove NP-hardness.
It's counterpart (and generalization) for space complexity is the QUANTIFIED BOOLEAN FORMULA problem (abbreviated as QBF). A quandified boolean formula is a boolean formula where the variables are quantified with the $\forall$ and $\exists$ quantifiers. The QBF problem asks whether there exists a satisfying assignment to a given quantified boolean formula. A classical result (see for instance [5]) shows that QBF is PSPACE-complete. Just as the SAT problem, QBF is a classical starting point for reductions proving PSPACE-hardness.

As a conclusion to this section, let us note that many other complexity classes exist and that only the one relevant to this thesis have been introduced here. The interested reader is referred to the classical textbook [5] for a more complete overview of complexity theory.

1.1.2 Graphs

All along the manuscript $G = (V, E)$ denotes a graph, where the set V is the *vertex set* and the set $E \subseteq V \times V$ is the *edge set*. When the context is clear we denote the number of elements in V by n and the number of elements in E by m. Given a graph G we denote its vertex set by $V(G)$ and its edge set by $E(G)$. We only consider graphs without loop, where a *loop* is an edge that links a vertex to itself. Furthermore,

the graphs we consider are *undirected*, meaning that there is no distinction between the edge (u, v) and the edge (v, u) for any $u, v \in V$. An undirected graph without loops is called a *simple graph*. All the graphs we consider in this manuscript are simple graphs. Given two vertices $u, v \in V(G)$ such that $(u, v) \in E$ we say that u is a *neighbor* of v (and reciprocally that v is a neighbor of u). The set of neighbors of a vertex u of a graph G is called the *neighborhood* of u and is denoted by $N_G(u)$. The *closed neighborhood* $N_G[u]$ of a vertex u is the set $N_G(u) \cup \{u\}$. The *degree* of u is the number of neighbors of u is denoted by $d_G(u)$. When the graph G is clear from context we often drop the subscript G and simply write $N(u)$, $N[u]$, and $d(u)$. The *maximum degree* of a graph G, denoted by $\Delta(G)$ is the maximum degree of a vertex of G taken over all vertices of G. The *complement graph* of G denoted by $\overline{G}$ is the graph which vertex set is $V(G)$ and such that for every $u, v \in V(G)$ where $u \neq v$, $(u, v) \in E(\overline{G})$ if and only if $(u, v) \notin E(G)$. In other words, it is the graph on the same vertex set as G such that each edge of G is a non-edge of $\overline{G}$ and vice versa.

Subgraphs and induced subgraphs. A graph $G' = (V', E')$ is a *subgraph* of $G = (V, E)$ if $V \subseteq V'$ and $E \subseteq E'$. If furthermore for all $u, v \in V(G')$, we have $(u, v) \in E(G) \implies (u, v) \in E(G')$ then G' is an *induced subgraph* of G. In other words, G' is an induced subgraph of G if every edge of G with both endpoints in $V(G')$ is also an edge of G'. Given $S \subseteq V$, the induced subgraph of G containing the vertices of S is the *graph induced by S in G* and is denoted by $G[S]$. The graph $G - S$ is the graph induced by $V \setminus S$ in G.

Walks, paths and cycles. A *walk* on a graph G is a subgraph of G with vertices $u_1, \ldots, u_p$ of G such that $u_i \in N(u_{i+1})$ for all $i \in 1, \ldots, p-1$. If furthermore no other edges between the vertices of the walk exist in G then the walk is an *induced walk*. The length of a walk W is the number of vertices in W minus one. A walk W is a *path* if no two vertices in W are the same. A walk W on vertices $u_1, \ldots, u_p$ is a *cycle* if no two vertices of the walk are the same except for $u_1 = u_p$. For the sake of simplicity we call a path that is an induced walk an *induced path* and a cycle that is an induced walk an *induced cycle*. The induced path on $t \geq 0$ vertices is denoted by P_t and the induced cycle on t vertices is denoted by C_t.
If there exists a path between any two vertices of a graph G, then G is *connected* and is *disconnected* otherwise. An inclusion-wise maximal set of vertices of G that induces a connected subgraph is a *connected component* of G. A graph G has no cycle as a subgraph is *acyclic*.
The *distance* $d_G(u, v)$ between two vertices u, v of graph G is the length of a shortest path in G between u and v. If no such path exists then we set $d_G(u, v) = +\infty$. The *diameter* $diam(G)$ of a graph G is the maximum distance between any two vertices of G. In other words, $diam(G) = max_{u,v \in V(G)}\{dist_G(u, v)\}$.

Independent sets and cliques. Let $G = (V, E)$ be a graph. A set $I \subseteq V(G)$ is an *independent set* of G if the vertices in I are pairwise non-adjacent in G. The *independence number* $\alpha(G)$ of a graph G is the size of a maximum independent set of G. A set $C \subseteq V(G)$ is a *clique* of G if the vertices in C are pairwise adjacent in G. The *clique number* $\omega(G)$ of a graph G is the size of a maximum clique of G. Let us note that deciding whether the size of a maximum independent set (resp. a clique) of a graph G is larger that k for some integer k is an NP-complete problem.

Coloring and chromatic number. A function $c : V(G) \to \{1, \ldots, k\}$ is a *k-coloring* of G. If furthermore the function c satisfies $c(u) \neq c(v)$ for every $(u, v) \in E(G)$ then the coloring is *proper*. All along the manuscript we only consider proper colorings and thus omit the term proper. If we consider non-proper colorings then it will be explicitly specified. We denote the integer in $[k]$ associated with a vertex u as the *color* of u. If a graph G admits a k-coloring for an integer k then it is *k-colorable*. The *chromatic number* $\chi(G)$ of a graph G is the minimum integer k for which the graph G is k-colorable. Deciding whether the chromatic number of G is smaller than k for some integer k is an NP-complete problem. However, note that if G admits a clique of size k, then at least k different colors are needed to color the vertices of the clique. It follows that the clique number $\omega(G)$ is a trivial lower bound on the chromatic number $\chi(G)$.

Degeneracy. Let d be an integer. A graph G is *d-degenerate* if there exists an ordering $v_1, \ldots, v_n$ of the vertices of G such that for every $i \leq n-1$, v_i has at most d neighbors in the subgraph of G induced by $\{v_{i+1}, \ldots, v_n\}$. We refer to such an ordering as a *d-degenerate ordering*, or just as a *degeneracy ordering* when d is clear from context. Equivalently, a graph G is d-degenerate if every subgraph of G contains a vertex of degree at most d. All along the manuscript, we denote the degeneracy of a graph G by $d(G)$.
Degeneracy has many algorithmic applications, let us give a simple one as an example: We show below that any d-degenerate graph G admits a $(d+1)$-coloring and hence that $\chi(G) \leq d+1$. Consider a degeneracy ordering $v_1, \ldots, v_n$ of $V(G)$ as defined above. Then, coloring each vertex v_i of G in the reverse order $v_n, \ldots, v_1$ with the first color in $\{1, \ldots, d+1\}$ that does not appear in the neighborhood of v_i yields a proper coloring of G. Indeed, when v_i receives a color, only the vertices in $\{v_{i+1}, \ldots, v_n\}$ have already been colored. Since v_i has at most d neighbors in this subset of vertices, and since there are $d+1$ colors, there exists a color that is not used in the neighborhood of v_i, and that can therefore be used to color v_i. The process terminates when coloring v_1 and gives a proper $(d+1)$-coloring of the graph.

Graph classes. A *class of graph* is a not-necessarily finite set of graphs. A class $\mathcal{G}$ is *hereditary* if for every $G \in \mathcal{G}$ every induced subgraph of G also belongs to $\mathcal{G}$. Note

that a class of graph is usually defined by a property shared by all the graphs that belongs to this class. Below we describe a few graph classes, that are among the most studied and are of particular interest in this thesis.

- **Perfect graphs**. A graph G is *perfect* if for every $S \subseteq V(G)$ the subgraph induced by S in G satisfies $\chi(G[S]) = \omega(G[S])$. It contains several graph classes that are of central importance in this thesis. Perfect graphs have the particularity that many classical problems such as Maximum Independent Set, Maximum Clique or Coloring that are NP-complete in the general case become polynomial-time solvable on perfect graphs. The celebrated strong perfect graph theorem shows that perfect graphs are exactly the graphs such that neither G nor $\overline{G}$ contain an induced odd cycle of length at least five.

- **Chordal graphs.** A graph G is *chordal* if it has no cycle on four vertices or more as an induced subgraph. In other words, any cycle of length four or more in G has a *chord*, which is an edge between two non-consecutive vertices of the cycle. A graph is chordal if and only if it admits a *perfect elimination ordering*. A perfect elimination ordering is an ordering $v_1, \ldots, v_n$ of $V(G)$ such that for every $i < n$, the neighborhood of v_i in $\{v_{i+1}, \ldots, v_n\}$ induces a clique. Such an ordering of the vertices of the graph can be found, if it exists, in linear time [53, 83]. It follows that chordal graphs can be recognized in linear time. Furthermore, note that every vertex v_i has at most $\omega(G) - 1$ neighbors in $\{v_{i+1}, \ldots, v_n\}$. Hence, a perfect elimination ordering is also a $(\omega(G) - 1)$-degeneracy ordering. Since for any graph G we have $d(G) \geq \omega(G) - 1$, it follows that if G is a chordal graph then $d(G) = \omega(G) - 1$. The class of chordal graphs is a subclass of the class of perfect graphs.

- **Bipartite graphs.** A graph $G = (V, E)$ is a *bipartite graph* if its vertex set can be partitioned into two independent sets $V = A \cup B$ called the *bipartition* of G. Equivalently, the class of bipartite graphs is exactly the class of graph that do not contain any cycle of odd length has a subgraph. It is also the class of graphs that are 2-colorable.

- **H-free graphs.** Let H be a graph. A graph G is *H-free* if it does not contain no induced subgraph of G is isomorphic to H. Many well studied graph classes can be described this way. For instance, the class of *cographs* is the class of P_4-free graphs. Given a set $\mathcal{H}$ of graph, a graph G is $\mathcal{H}$-free if it does not contain H as an induced subgraph for every $H \in \mathcal{H}$.

- **Split graphs.** A graph $G = (V, E)$ is a *split graph* if its vertex set can be partitioned into an independent set I and a clique C. Split graphs are a subclass of chordal graphs.

- **Trees.** A graph is a *tree* if it is connected and acyclic. It is not hard to see that if a graph G is a tree then it satisfies $|E(G)| = |V(G)| - 1$. Trees are also a subclass of bipartite graphs and chordal graphs.

Tree decomposition and treewidth. Let $G = (V, E)$ be a graph. A *tree decomposition* of G is a pair (T, B) where $T = (W, E')$ is a tree and B is a function that associates to each node U of T a subset of vertices B_U of V (called *bag*) such that: (i) $\cup_{U \in T} B_U = V(G)$, (ii) for every edge (u, v) of G there exists a bag B_U that contains both u and v and (iii) for every $x \in V$, the subset of nodes whose bags contain x forms a non-empty subtree in T.
The *width* of a tree decomposition is the size of a largest bag B_U minus one. The *treewidth* of a graph G is the minimumm width among all possible tree decompositions of G. Informally speaking, the treewidth of a graph measures how close the graph is to being a tree, where trees are graphs of treewidth exactly 1. The smaller the treewidth of a graph is, the more "tree-like" is its structure. It follows that a number of problems that can be solved efficiently on trees such as Maximum Independent Sets can also be solved efficiently on graphs of small treewidth. As we will see in the next section, treewidth is also of central importance in parameterized complexity.

1.1.3 Parameterized complexity.

In this section, we give a brief and informal introduction to parameterized complexity in the particular framework of graph theory. The goal of parameterized complexity is to refine the knowledge we have on the complexity of decision problems, with respect to parameters depending either on the input (the graph) or on the output (the solution size, for instance). One of the motivation is the following: Many fundamental problems (Maximum clique, Maximum Independent Set, Vertex Cover...) are NP-complete. Does it mean that when such a problem is encountered, any hope of finding an efficient algorithm should be abandoned? Or can we, in some cases, still hope for an efficient resolution? This is one of the questions tackled by parameterized complexity. Furthermore, parameterized complexity can be seen as the "science of preprocessing". Often, it is possible to determine *a priori* that some parts of the input are unnecessary from an algorithmic point of view, which can be for instance vertices or edges that belong to either all or no solutions. Such parts can be safely removed from the input graph which size is then decreased, simplifying the problem. By systematically applying such a preprocessing, called a *reduction*, one is able to isolate the "hard" parts of an instance. Parameterized complexity gives a number of tools to design these reductions and to assess their efficiency.

In what follows, a *parameterized problem* denotes a decision problem π along with a parameter (an integer) k. As in the classical computational complexity framework, parameterized problems can be divided into classes which measure (so to speak) their hardness. The main difference being that the hardness of a problem also depends

on the parameter. Let π be a parameterized problem which takes as input a graph on n vertices and k be the parameter. The problem π is *fixed-parameter tractable* (abbreviated FPT) with respect to the parameter k, if there exists an algorithm $\mathcal{A}$ and a computable function f such that the algorithm $\mathcal{A}$ decides problem π in time $f(k) \cdot n^{O(1)}$. Note that in this definition, nothing prevent the growth of f from being exponential or even super-exponential. However, if the parameter k is small, then the problem π can be solved efficiently since the dependency in the size of the input is polynomial. Another way to define FPT algorithm is through the notion of *kernel*. A *kernel* for a problem π with the parameter k is an algorithm that given an instance (I, k) of π, works in polynomial time and returns an equivalent instance (I', k') of π such that the size of I' is bounded by a computable function of k. Obviously, a decidable parameterized problem that admits a kernel with respect to a parameter k is also fixed-parameter tractable with respect to k. More surprisingly, the converse direction is also true. This leads to the following theorem (see for instance [34] for a complete proof):

Theorem 1. *A decidable parameterized problem admits a kernel with respect to the parameter k if and only if it is* FPT *with respect to the parameter k.*

Let us illustrate these notions through a simple example. A *vertex cover* of a graph G is a set S of vertices of G such that for every edge $(u, v) \in E(G)$, either u or v belongs to S. Given a graph G and an integer k, the decision problem k-Vertex Cover asks whether G admits a vertex cover of size at most k. Let us give a sketch of the following folklore theorem:

Theorem 2. *Let k be an integer. k-Vertex cover admits a kernel with at most k^2 vertices. Therefore, k-Vertex cover is fixed-parameter tractable with respect to the solution size.*

Sketch of proof. Let (G, k) be an instance of Vertex Cover. To avoid cumbersome notations, we denote an instance equivalent to (G, k) by an *equivalent instance*.
Let G_1 be the graph obtained by deleting the isolated vertices of G. Since no isolated vertex is in a minimum vertex cover, (G_1, k) is an equivalent instance.
Suppose that G_1 contains a vertex v of degree at least $k+1$. Then v is necessarily in any solution of size k. Indeed, not taking v in the solution implies that all of its neighbors must be chosen, leading to a solution of size at least $k+1$. Hence, v can be safely removed from the graph and the parameter be safely decreased by 1. Let G_2 be the graph obtained after removing all the vertices of degree at least $k+1$ from G_1, and t be the corresponding parameter. If $t \leq 0$, then return a trivial no instance (for instance the graph K_2 with $k = 0$).
Otherwise $t > 0$. Every vertex in G_2 has degree at most k. Hence, a vertex set S of size at most t can be adjacent to at most $kt \leq k^2$ vertices. It follows that if G_2 contains more than k^2 vertices, then it is a no instance and we can return a trivial no instance (constructed as before). Otherwise (G_2, t) is an equivalent instance with at most k^2 vertices, as needed. □

Let us now give a brief description of the W-*hierarchy*. The *depth* of a Boolean circuit is the maximum length of a path from an input node to the output node. The *weft* of a Boolean circuit is the maximum number of gates with at least three entries on a path from the root of the circuit to a leaf. For $i, d \geq 0$ let $C_{i,d}$ be the class of Boolean circuits with weft at most i and depth at most d. A parameterized problem π with parameter k is in the class $\mathsf{W}[i]$ if there exists $d \geq 1$ such that any instance I of the problem can be transformed into a Boolean circuit of $C_{i,d}$ in FPT time, and such that I is a yes instance if and only if there exists an assignation of the entries of the circuit with at most k assignations at 1 that satisfies the circuit. Let us note that we have $\mathsf{W}[i] \subseteq \mathsf{W}[j]$ for every $i \leq j$ and that $\mathsf{FPT} = \mathsf{W}[0]$. The union of the classes $\mathsf{W}[i]$ for every i forms the class $\mathsf{W}[P]$. Let π_1, π_2 be two parameterized problems with parameter k and k' respectively. A *parameterized reduction* from π_1 to π_2 is an algorithm that given an instance I of π_1 and the parameter k outputs an instance J of π_2 with parameter k' such that:

1. (I, k) is a yes-instance of π_1 if and only if (J, k') is a yes-instance of π_2,
2. $k' \leq g(k)$ for some computable function g, and
3. the algorithm runs in time $f(k) \cdot n^{O(1)}$ where n is the size of the instance I.

As for computational complexity, we can define *hard* problems as follows: given a class X, a parameterized problem π is X-hard, if there exists a parameterized reduction from every problem of the class X to π. Recall that we have $\mathsf{FPT} \subseteq \mathsf{W}[1]$. However, as for the $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ problem, whether the converse inclusion also holds is still an open problem. However, $\mathsf{W}[1]$-hardness of a problem is often interpreted as a strong hint that no FPT algorithm exists.
Unlike Vertex Cover, many classical problems such as Independent Set or Clique are $\mathsf{W}[1]$-hard with respect to the solution size. It is also well-known that Dominating Set is $\mathsf{W}[2]$-hard, and hence is in some sense "harder" than the aforementioned problems with respect to the solution size.
Another parameter that is frequently considered in parameterized complexity is the treewidth of the input graph, which we defined in the previous section. Assuming that the treewidth of the graph is small, and thus that the input graph has a tree-like structure, allows for the efficient use of dynamic programming to design FPT algorithm. In particular, problems such as Vertex Cover, Independent Set or Dominating set are FPT when parameterized by the treewidth of the input graph. Let us also note that the treewidth is of central importance in the celebrated Courcelle's meta-theorem, which states that any graph problem that can be expressed in *monadic second-order logic* is FPT when parameterized by the treewidth.

As a conclusion, let us recall that this section is an introduction to the notion we need in this thesis. The reader is referred to the classical textbook [34] for a formal and complete introductory course to parameterized complexity.

1.2 A brief introduction to reconfiguration

A combinatorial optimization problem consists in finding a solution that satisfies a particular set of constraints among a countable set of candidates. Such a solution is said to be *feasible*. Graphs are particularly suitable to model a vast variety of real-life combinatorial problems and a considerable amount of work has been done to determine whether or not such a solution can be computed in a reasonable time and to design efficient algorithm to do so. However, finding a feasible solution is sometimes not enough to deal with the dynamic aspect of a system. It can be because the problem itself implies that the solution changes over time, or because one wants an "overview" of the solution space, as it is for instance the case in enumeration problems or problems involving random sampling.
Combinatorial reconfiguration is closely related to these questions. In this framework, we are given a problem π called the *source problem* whose feasible solutions are called *configurations*. We are also given an operation called the *reconfiguration operation* (for π) that given a configuration and some parameters (that depend on the source problem) output another configuration of π, and that can be computed in polynomial time. Such an operation defines an adjacency relation: two solutions are *adjacent* if one can be obtained from the other by applying one reconfiguration operation (in one step). We can thus define the *reconfiguration graph*, as the graph whose vertex set is the set of all configurations and where two configurations are adjacent if they differ by one reconfiguration operation. Given a source problem π and a reconfiguration operation $\mathcal{T}$ for π, we denote the corresponding reconfiguration graph as $G^{\pi}_{\mathcal{T}}$. Note that by definition of a reconfiguration operation $G^{\pi}_{\mathcal{T}}$ is a simple, undirected graph. Let us now formally describe the questions that are at the heart of the study of reconfiguration problems. Let π be a source problem and $\mathcal{T}$ be a reconfiguration operation:

- Reachability: Given two solutions S and T of π, is it possible to find a sequence of operations $\mathcal{T}$ that transforms S into T? Equivalently, is there a path from S to T in $G^{\pi}_{\mathcal{T}}$?

- Connectivity: Is it possible to find a sequence of operations $\mathcal{T}$ that transforms S into T for any two solution S, T of π? Equivalently, is $G^{\pi}_{\mathcal{T}}$ connected?

- Shortest Path: Given two solutions S and T of π, what is the shortest sequence, in terms of number of operations $\mathcal{T}$, that transforms S into T? Equivalently, what is the length of a shortest path from S to T in $G^{\pi}_{\mathcal{T}}$?

- Diameter: What is the maximum length of a shortest sequence, in terms of number of operations $\mathcal{T}$, that transforms S into T for any two solutions S, T of π? Equivalently, what is the diameter of $G^{\pi}_{\mathcal{T}}$?

Although the systematic study of reconfiguration counterparts of classical combinatorial problems (3-Sat, Independent set, Dominating set, Clique, Vertex

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

FIGURE 1.4: The target configuration of the 15-puzzle game.

COVER and so on) has only been initiated in the past ten to twenty years, some reconfiguration problems have been studied for much longer than that, even if not defined as such. The first ones we can think of are single player games, such as the ones mentioned at the very beginning of this Chapter. Consider for instance the 15-puzzle game: In this game, the player is given a 4×4 board on which 15 labeled tokens of unit size are placed, one place being left empty. The player is allowed at each step to slide a token on the empty spot, and the goal is to reach the target configuration illustrated in Figure 1.4. This puzzle, which is approximately 150 years old, has been studied theoretically for almost as many years. Indeed, Jonhson and Story [66] showed in 1879 that the reconfiguration graph of the 15-puzzle has exactly two connected components (loosely speaking, one component corresponding to the even permutations of $1, \ldots, 15$ and one corresponding to the odd permutations) therefore showing that some initial configurations of the game are "impossible" to solve.

Although more recent than the 15-puzzle games, some reconfiguration problems more closely related to our setting have also been studied for decades. We can for instance mention the case of triangulations and edge flips. A *triangulation* of a polygon is a partition of the polygonal area into triangles whose interiors are pairwise non-intersecting. Consider two triangles of a triangulation sharing an edge, thus forming a quadrilateral. The *edge-flip* operation consists in replacing this edge with the other diagonal edge of the quadrilateral. The edge flip operation is illustrated in Figure 1.5. The *flip distance* between two triangulations of a convex polygon is the length of a shortest sequence of edge flips transforming one triangulation into the other. In 1988, Sleator et al. [88] showed that the flip distance between two triangulations of a polygon with n vertices is at most $2n - 6$. However, determining the complexity of computing the exact flip distance between two triangulations was a longstanding open-problem in the field. It was shown by Lubiw and Pathak [74] to be NP-complete in 2012. A combinatorial version of the problem has also been studied for a long time, where the input is a maximal planar graph, and the edge flip operation is defined as the replacement of an edge by another edge such that the obtained graph remains maximal planar. Wagner [96] showed as far back as 1936 that given two maximal planar graphs G_1 and G_2 there always exists a sequence of flips transforming G_1 into G_2. Let us note that a considerable amount of work has been done toward triangulations and edge flip on planar graphs, be it from a geometrical or a combinatorial point of view. The interested reader is referred to [25] for a (although not recent) nice overview of the field.

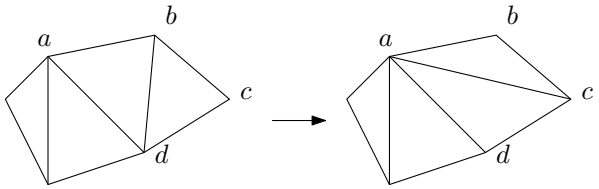


FIGURE 1.5: An example of edge flip in a triangulation of a convex polygon. The edge (b, d) is replaced by the opposite diagonal (a, c) in the quadrilateral formed by $\{a, b, c, d\}$

Another longstanding open question in reconfiguration is the one asked by Vizing towards edge-colorings of graphs. Given a graph $G = (V, E)$ and an integer k, a *k-edge-coloring* of G is a function $c : E \to [k]$ that associates a color to each edge of G, in such a way that no two adjacent edges receive the same color. The *chromatic index* $\chi'(G)$ of a graph G is the minimum k such that there exists a k-edge-coloring of G. In [95] Vizing showed the following:

Theorem 3. *For any simple graph G, $\chi'(G) \leq \Delta(G) + 1$.*

Since the maximum degree is a trivial lower bound on the chromatic index, Vizing's theorem implies that for any graph G, $\chi'(G) = \Delta(G)$ or $\chi'(G) = \Delta(G) + 1$. A *Kempe chain* is an inclusion-wise maximal connected component induced by the edges of two given colors. A *Kempe change* is the operation consisting in exchanging the colors of the edges along a Kempe chain. Two edge-colorings are *Kempe-equivalent* if one can be obtained from the other by a sequence of Kempe changes. In 1965 [94], Vizing asked the following: is it true that any k-edge-coloring of a graph G with $k > \chi'(G)$ is Kempe-equivalent to a $\chi'(G)$-edge-coloring of G?
In the proof of Theorem 3, Vizing actually showed that any k-edge-coloring with $k \geq \Delta(G) + 2$ is Kempe-equivalent to a $(\Delta(G) + 1)$-edge coloring of G, thus answering his question in the case $\chi'(G) = \Delta(G) + 1$. Despite an answer for the case $\chi'(G) = \Delta(G)$ still has to be found, partial results have been obtained over the years. Mohar showed in 2006 [77] that all $(\chi'(G) + 2)$-edge-colorings are Kempe-equivalent and conjectured that all $(\Delta(G) + 2)$-edge-coloring should be Kemp-equivalent. Later on, Vizing's question was answered positively in the case $\Delta(G) = 3$ by McDonald et al. [75] and in the case $\Delta(G) = 4$ by Asratian and Casselgren [6]. Very recently, Bonamy et al. [18] showed that it also holds for any triangle-free graph, without any restriction on the maximum degree.

All these examples show that even though the name of *reconfiguration* only emerged recently as a unifying term, related problems were already studied for a long time.
The first reconfiguration problem we are interested in this thesis is also a coloring problem: the *single-vertex recoloring problem*. In what follows we refer to this problem as the *graph recoloring* problem to avoid cumbersome notations. In the graph recoloring problem we are given a coloring of the vertices of the input graph, and the reconfiguration operation consists in changing the color of a single vertex, in such a

way that the obtained coloring remains proper. We give a detailed introduction to this problem in Chapter 2.

Another number of questions in the field of reconfiguration concerns the set of problems that we could call the *moving tokens* problems. Consider a problem π that takes as input a graph G and an integer k, and the goal is to find a subset S of vertices of $V(G)$ of size k that satisfies a property P. A handy way to consider this problem in the reconfiguration framework is to view the set S as a set of tokens that are placed on the vertices of S in G. It is then no hard task to define a reconfiguration operation as a way to move tokens on the graphs. The additional condition is that the set of vertices on which the tokens lie must satisfy the property P after each move. In particular, two rules according to which the tokens can move have been studied:

- The *token sliding* rule, where a token is allowed to move from a vertex to a neighboring vertex, or in other words to *slide* along an edge (as long as property P remains satisfied).
- The *token jumping* rule, where a token is allowed to move from a vertex to any other vertex, or in other words to *jump* anywhere on the graph (as long as property P remains satisfied).
- The *token addition-removal with threshold* ℓ rule, where ℓ is an integer that is either an upper bound or a lower bound on the number of tokens that can lie on the graph at any time, and where a token can be either added or removed from the graph at each step.

These reconfiguration problems have been studied for many different source problems π. We can mention for instance the Dominating set reconfiguration problems [13, 52, 54, 89], the Vertex cover reconfiguration [58, 62, 78] problems, or more recently the Vertex separator reconfiguration [49] problems. In this thesis, we investigate the Independent set reconfiguration problems. As we will see in the introduction to independent set reconfiguration in Chapter 6, it is one of the most studied reconfiguration problem since the late 2000's, and is still a very active field of research today.

As a conclusion to this section, let us note that many more reconfiguration problems have been studied over the years, making it difficult to give a concise summary of all the works that has been done in the field. We refer the interested reader to the very nice surveys of Van den Heuvel [57] and Nishimura [80] for the description of other classical reconfiguration problems and a more detailed historical overview of reconfiguration.

1.3 Outline of this thesis

This thesis focuses on two particular reconfiguration problems, the *graph recoloring* problem and the *independent set reconfiguration* problem.

The first part of the manuscript is devoted to the *graph recoloring* problem. In Chapter 2 we introduce the problem and its motivations. One of the most studied question in the framework of *graph recoloring* is the question of the diameter of the reconfiguration graph. As we will see, the study of this diameter allows to measure how efficiently we are able to sample at random colorings of the input graph. It is of particular interest in the field of statistical physics for the study of the behavior of spine systems at extremal temperatures.
A classical result in the field shown independently by Dyer et al. [38] and Bonsma and Cereceda [22] (the proof of which will be given in the next section) shows that for any d-degenerate graph G there exists a transformation (where the color of a single vertex is modified at each step) between any two k-colorings of G as long as $k \geq d+2$. In other words, the reconfiguration graphs whose vertex set corresponds to the k-colorings of G with $k \geq d+2$ is always connected. The proof of this result is algorithmic and outputs a sequence between any two colorings given as input. However, the length of this sequence may be exponential (in the number of vertices of G) and thus cannot be used to bound the diameter of the reconfiguration graph. A famous conjecture of Cereceda says that this diameter should be $O(n^2)$ when considering $(d+2)$-colorings only. Although still widely open, this conjecture has been shown to be true for several graph classes, in particular for chordal graphs and graphs of bounded treewidth. On the other hand, a result of Bousquet and Perarnau [29] shows that when considering $2d+2$ colorings, the diameter of the reconfiguration graph has linear diameter. Informally speaking, combining these two results confirms the intuition that the more colors are available, the easier it gets to find a sequence between two colorings. Consider chordal graphs and bounded treewidth graphs: the diameter of the reconfiguration graphs of $(d+2)$-colorings has quadratic diameter. Increasing the number of colors, we obtain a linear diameter for the reconfiguration graphs of $(2d+2)$-colorings. Hence the following question: how many colors above the $d+2$ threshold are really needed for the diameter to become linear? This is the question we investigate in this thesis. Before diving into our results, we give in Chapter 3 an overview of the proof techniques that have been employed to bound the diameter of reconfiguration graphs, and explain why most of them cannot be used to obtain linear transformations.
In [19], Bonamy et al. showed that given a chordal graph G, the diameter of the reconfiguration graph of $(d+2)$-colorings has quadratic diameter. In Chapter 4 we introduce a new proof technique to show that if G has bounded degree, then the diameter of the reconfiguration graph of $(d+4)$-colorings is linear, drastically improving on the $2d+2$ bound given by the result of Bousquet and Perarnau. Formally,

we show the following:

Theorem 4. *Let G be a d-degenerate chordal graph of maximum degree Δ. For every $k \geq d+4$, the diameter of $\mathcal{R}_k(G)$ is at most $O_\Delta(n)$. Moreover, given two colorings c_1, c_2 of G, a transformation of length at most $O_\Delta(n)$ can be found in linear time.*

Towards the same question, we consider the case of graphs of treewidth two in Chapter 5. Due to a result of Bonamy and Bousquet [15], it is known that given a graph of treewidth tw, the reconfiguration graph of $(tw+2)$-colorings has quadratic diameter. Hence, given a graph of treewidth two, the diameter of the 4-reconfiguration graph is quadratic. Furthermore, the result of Bousquet and Perarnau shows that it becomes linear when considering 6-colorings, leaving the case of 5-colorings open. In Chapter 5 we complete the picture for graphs of treewidth two and show the following:

Theorem 5. *Let G be graph of treewidth at most 2 and $k = 5$. There exists a constant c such that, for every pair of 5-colorings α, β of G, there exists a transformation from α to β recoloring each vertex at most c times.*

The second part of the manuscript is dedicated to the *independent set reconfiguration* problems. These problems find their origins in the study of single player puzzle-games which, as explained in the previous sections, have been topics of matter for mathematician for decades. We give a detailed introduction to independent set reconfiguration in Chapter 6. As we will see, a great amount of work has been done toward the reachability problem for both the sliding and the jumping rule in particular graph classes, as both rules define a PSPACE-complete problem in the general case. In particular, the case of H-free graphs has been investigated for H being a P_4 (cographs) or a claw. In Chapter 7 we give a general result on the complexity of both token jumping and token sliding in H-free graphs:

Theorem 6. *Token Jumping and Token Sliding on H-free graphs are* PSPACE*-complete, unless H is a path, a claw, or a subdivision of the claw.*

Furthermore, we show that when the source and target independent sets are maximum, token jumping and token sliding reachability are polynomial-time solvable on fork-free graph (where a fork is a claw with one subdivided edge).
In Chapter 8, we then consider graphs that forbid some cycles to appear as induced subgraph. In [72], Lokshtanov and Mouawad showed that token jumping reachability is NP-complete on bipartite graphs and that token sliding reachability is PSPACE-complete on the same class of graphs. We consider token sliding reachability on bipartite graphs and related classes from a parameterized point of view and show, among other results:

Theorem 7. *Token Sliding parameterized by k is* W[1]*-hard on bipartite graphs and is fixed-parameter tractable on bipartite C_4-free graphs.*

Finally, we introduce in Chapter 9 a generalization of the token sliding reachability problem which we call *galactic token sliding* as a tool for parameterized reductions. We make use of this model to show the following:

Theorem 8. *Token Sliding is fixed-parameter tractable when parameterized by* $k + \Delta(G)$.

And we give a conclusion to this thesis in Chapter 10.

Part I

Graph recoloring

Chapter 2

Introduction to graph recoloring

2.1 An example: the frequency assignement problem

Let us begin this chapter with an introductory example. Consider an area in which antennas have been placed. Such antennas are used for instance for radio emissions or for mobile telecommunication networks. Each antenna emits radio waves in every direction up to some fixed distance on a specific frequency. If two antennas are close to each other, their emission radius overlap and they must be assigned different frequencies in order to avoid interferences, which may severely degrade service quality. Furthermore, the usable frequency range is limited and thus the same frequency may be reused several times for different antennas. The question is the following: how to assign a frequency to each antenna in order to minimize the interferences? This problem is known in the literature as the frequency assignment problem (FAP). As we explained in the previous chapter, this problem is closely related to graph coloring: Consider the graph which set of vertices is the set of antennas, two vertices being linked by an edge if the emission radius of the corresponding antennas overlaps. If each frequency corresponds to a color, finding an assignment of frequencies to antennas with no interferences is then equivalent to finding a proper coloring of the corresponding graph.

Suppose now that such an assignment has been found and that each antenna has been set to emit on its assigned frequency. Suppose furthermore that at some point, a new antenna needs to be added on the area. If there exists a frequency that is not used by neighboring antennas, then the new one can be set to work on this frequency without causing any interference. Otherwise, the working frequencies of the other antennas need to be modified to free a frequency. Is it possible to modify the frequencies of the antennas, so that no interference is ever created, and obtain a state where there is an available frequency for the new antenna? A trivial solution to this problem consists in shutting down the whole network, compute a new coloring of the underlying graph and proceed with the modifications on all the antennas at the same time, causing the signal to be off in the whole area for the duration of the modifications. A presumably smarter solution is the following: pick an arbitrary antenna whose frequency range needs to be modified and modify its range if it creates no interference, and then proceed with another antenna. By doing so, only a small part of the area is not

covered at every step of the process. Hence the following question: can we find a sequence of modifications (where each element in the sequence corresponds to the modification of the frequency of single antenna) such that no interference is caused by any of these modifications and such that a desired working state (where a frequency is available for the new antenna) is reached after the last modification of the sequence? Note that we are allowed to change the frequency of an antenna several times in this sequence.
This new problem can be restated as follows in terms of graph coloring: starting from the coloring corresponding to the initial state of the antennas, is it possible to modify the color of one vertex at a time in such a way that each intermediate coloring remains proper and that the final coloring corresponds to the final desired state? This is one of the question at the heart of the graph recoloring problem which we formally define in the next section. Let us stress the fact that the Frequency Assignement Problem only comes as a motivation and that the main topic of this chapter is the graph recoloring problem.

Let us however note that this introductory example is not completely artificial. Indeed, the problem of handling the installation of new antennas, called the *frequency reassignement problem*, was adressed by Han in [55], in the specific case of a mobile telecommunication network. The problem of finding a sequence of frequency reassignment has also been studied in order to solve the classical frequency assignment problem. The strategy employed is an "error correction" strategy with a local search approach (see for instance [24]): Start by finding a solution that is not "too bad" (in terms of number of interferences). Then try to successively reduce the number of interferences by exploring neighboring solutions, which are solutions that differ from the current one on the frequency of a single antenna (or equivalently the color of a single vertex). Other types of local search neighborhood have been studied such has neighborhood defined by *Kempe changes* [24], which we described in the previous Chapter. The interested reader is referred to [1] for a complete survey on the frequency assignment problem.

2.2 Definitions

In this section, we describe formally the graph recoloring problem along with the main questions of the field. Let G be a graph and α, β be two proper colorings of G. A *reconfiguration sequence* $\mathcal{S} := \gamma_0, \gamma_1, \dots \gamma_{|\mathcal{S}|}$ from α to β is a sequence of colorings of G such that $\gamma_0 = \alpha$ and $\gamma_{|\mathcal{S}|} = \beta$. A reconfiguration sequence is *valid* if for every $i \leq |S|$ the coloring γ_i is proper and if γ_i and γ_{i+1} differ on the color of exactly one vertex for every $i \leq |\mathcal{S}| - 1$. The problem of deciding if such a valid sequence exists is trivial if no bound is set on the number of colors that can be used in the sequence. However, as seen in the introductory example it is natural to ask for a limitation on the number of colors that can be used. This problem is often referred to in the

literature as k-Color-path [31, 57] or as k-Coloring Reconfiguration [80] where k is an integer that denote the maximum number of available colors:

k-Color-path
Input: A graph G, an integer k and α, β two proper k-colorings of G.
Question: Does there exists a valid reconfiguration sequence of k-colorings of G from α to β?

As for any reconfiguration problem, graph recoloring problems can be restated in terms of the reconfiguration graph. In this setting, the *reconfiguration graph* $\mathcal{R}_k(G)$ is the graph which vertex set is the set of all proper k-colorings of G and where two vertices are adjacent if and only if the corresponding colorings differ on the color of exactly one vertex of G. A valid reconfiguration sequence of k-colorings of G is a path in $\mathcal{R}_k(G)$. The k-color-path problem can be restated as follows:

k-Color-path (restated)
Input: A graph G, an integer k and α, β two proper k-colorings of G.
Question: Is there a path from α to β in $\mathcal{R}_k(G)$?

A natural question that comes up is the following: for which integer k can we ensure that the answer to the k-color-path problem is always yes? That is, for which integer k is the reconfiguration graph $\mathcal{R}_k(G)$ of the input graph G connected? The corresponding decision problem is known as k-Color-Mixing:

k-Color-Mixing
Input: A graph G and an integer k.
Question: Is $\mathcal{R}_k(G)$ connected?

One of the first general result on the connectedness of the reconfiguration graph is due to Jerrum [65]. We include a proof for completeness.

Theorem 9. *For any graph G of maximum degree Δ and every integer $k \geq \Delta + 2$, the reconfiguration graph $\mathcal{R}_k(G)$ is connected.*

Proof. Let G be a graph of maximum degree Δ, let $k \geq \Delta + 2$ be an integer and α, β be two k-colorings of G. The proof goes by induction on the number of vertices of G. The result obviously holds if $n = 1$. Otherwise, let v be any vertex of G and let α', β' be the colorings induced by α and β on $G - v$. By induction there exists a recoloring sequence $\alpha'_1 := \alpha', \alpha'_2, \ldots, \alpha'_p = \beta'$ from α' to β' in $G - v$. Let us show how to extend this recoloring sequence to a sequence from α to β in G. As long as no neighbor of v gets recolored, or if a neighbor of v is recolored to a color distinct from the current color of v, the sequence is the same in $G - v$ and G (the initial color of v is $\alpha(v)$ and does not change). Suppose then that a neighbor u of v gets recolored at step i to $\alpha_i(v)$, the current color of v. Since v has at most Δ neighbors, there are at most $\Delta + 1$ distinct colors in the closed neighborhood of v (in α_i). Hence there is a color that we can recolor v with right before recoloring u to $\alpha_i(v)$. The obtained coloring is proper and we can proceed with the rest of the sequence. At the end of the sequence we can

recolor v to its target color $\beta(v)$, since all other vertices have their target color and β is proper. This gives a sequence from α to β, as needed. □

Suppose now that we are given a graph G and an integer k such that $\mathcal{R}_k(G)$ is connected. Then the graph G is said to be *k-mixing*. Another well-studied question, which is also central in this thesis, is the one of the diameter of $\mathcal{R}_k(G)$:

k-Color Diameter
Input: A graph G and an integer k such that G is k-mixing.
Question: What is the diameter of $\mathcal{R}_k(G)$?

This last problem can be understood as follows: what is the maximum length of a reconfiguration sequence transforming a proper k-coloring of G into another, over all possible pairs of proper k-colorings of G? Note that since the number of such colorings is exponential, it is often impossible in practice - or at least very difficult - to give an exact answer to the k-Coloring Diameter problem. We focus on finding bounds for the diameter, usually in terms of the number n of vertices of the input graph. Obviously we always assume in this chapter that the input graph G and the integer k satisfy $k \geq \chi(G)$. A k-coloring of a graph G is *frozen* if the k-colors appear in the closed neighborhood of every vertex of G, such that no valid reconfiguration operation can be performed starting from this coloring. Note that a frozen coloring is an isolated vertex of the reconfiguration graph.

2.3 From statistical physics to graph theory: Glaubers dynamic

In this section we motivate the main questions of the graph recoloring problem mentioned previously by describing the link between graph recoloring and statistical physics. More precisely, we explain how the study of a particular Monte-Carlo Markov Chain, the Glaubers dynamic, lead to the study of the graph recoloring problem from a purely graph-theoretic point of view.

Anti-ferromagnetic Potts model at zero temperature. The study of graph recoloring was initially motivated by enumeration and efficient sampling of proper colorings (even though it is also interesting from a purely combinatorial point of view). In particular, this latter problem is of interest in statistical physics for the study of the so-called *anti-ferromagnetic Potts model at zero temperature*. From a high level perspective, this model describes interactions between particles in a system whose temperature tends to $0K$. This system is represented as a (not necessarily finite) graph $G = (V, E)$ where each vertex represents a particle and the presence of an edge $(x, y) \in E$ indicates that the particles x and y are close enough to each other to interact. Furthermore, each particle $x \in V$ is in a state $\sigma(v)$, chosen among a finite set of possibilities $Q := \{1, 2, \ldots, q\}$. This state is drawn at random for each particle

and depends in particular on the states of its neighbors. Given a state $\sigma(x)$ for each particle x the *energy of the system* is described by the following formula:

$$E(\sigma) = -J \sum_{(x,y)\in E(G)} \delta_{\sigma(x),\sigma(y)} \tag{2.1}$$

where $J < 0$ is a constant (the case $J \geq 0$ corresponds to the *ferromagnetic model* which is outside the scope of this thesis) and $\delta_{\sigma(x),\sigma(y)} = \begin{cases} 1 \text{ if } \sigma(x) = \sigma(y) \\ 0 \text{ otherwise} \end{cases}$

To put this equation in words, it means that two neighboring particles contribute to the increase of the energy of the system if and only if they are in the same state. Given the temperature T of the system, the probability for the system to be in the state σ is proportional to $e^{\frac{-E(\sigma)}{T}}$. Hence, when the temperature T tends to 0, the probability to be in a state where any two particles interact is almost null. At zero temperature the system is completely "calm" and its energy is minimum. The states that minimize the energy of the system are the *feasible states* and each such state σ satisfies $E(\sigma) = 0$. It follows from the definition of the energy that for every feasible state σ and every two neighboring particles $(x, y) \in E(G)$, we have $\sigma(x) \neq \sigma(y)$. If we interpret the states of Q as colors, the set of feasible states of the system at zero temperature coincides exactly with the set of all proper q-colorings of the graph G that represents this system. The study of the feasible states is of central importance for the anti-ferromagnetic Potts model and it is of particular interest for statistical physicists to be able to sample these states (and hence sample proper colorings) at random. Indeed, since the number of proper colorings of a graph is usually exponential in the graph size, computing and enumerating all the feasible states of the system is too long. Sampling these states can be seen as taking a "snapshot" of the system and allows for the study of their properties and the transitions between them. A way to do so is through the use of a seemingly simple stochastic process known as the *Glauber dynamics*. In order to properly introduce the Glauber dynamics, we first need to state a few definitions and basic results about Markov chains.

Markov chains. A *(finite, discrete-time) Markov chain* $(X)_{t\geq 0}$ is a sequence of random variables over a finite state space Ω that satisfies the *Markov property*:

$$Pr(X_t = x \mid X_{t-1} = x_{t-1}, \dots X_0 = x_0) = Pr(X_t = x \mid X_{t-1} = x_{t-1}) \tag{2.2}$$

Equivalently, a discrete-time Markov chain can be seen as a random walk over Ω where the probability of moving to some state at time $t > 0$ depends only on the state the walk is in at time $t - 1$. Such a stochastic process is often described as being "memoryless": the successive states $x_0, x_1, \dots, x_{t-1}$ reached by the chain before the state x_t have no impact on the probability of the transition at time t. Hence the probabilities of transitions can be described by a single $|\Omega| \times |\Omega|$ matrix P. In what follows we denote the probability to go from the state x to the state y in one step by $P(x, y)$.

A Markov chain can be equivalently represented as a directed and weighted graph whose vertex set is Ω and such that there is an arc from the state x to the state y if the probability of transitioning from x to y is positive. The weight of the arc (x, y) is the probability of transition $P(x, y)$. Note that the transition matrix P is the weighted adjacency matrix of this graph. Let π_0 denote the initial probability distribution: for every $x \in \Omega$, $\pi_0(x)$ is the probability of being in the state x at the beginning of the walk. Furthermore, we denote by π_t the distribution obtained after t steps from π_0 for $t \geq 0$. It is not hard to see that the following holds (see for instance [70]):

$$\pi_t = P^t \pi_0 \qquad \forall t > 0 \tag{2.3}$$

A distribution is *stationary* if it satisfies $\pi = P\pi$. In other words, if a Markov chain reaches a stationary distribution π at steps t_0 then it satisfies $\pi_t = \pi_{t_0}$ for all $t \geq t_0$. A chain is *reversible* if it satisfies $P(x, y) = P(y, x)$ for all $x, y \in \Omega$. A simple example of stationary distribution is given by the following folklore result:

Proposition 1. *The uniform distribution of a reversible Markov chain is stationary.*

A Markov chain $(X)_{t\geq 0}$ is *ergodic* if it satisfies the following:

- It is *irreducible*: for any two $x, y \in \Omega$, there exists $t \geq 0$ such that $P^t(x, y) > 0$. Equivalently, the corresponding graph is strongly connected.
- It is *aperiodic*: for every $x \in \Omega$, there exists a t_0 such that for every $t > t_0$ we have $P^t(x, x) > 0$.

Finally, we need the following classical theorem before we can proceed:

Theorem 10. *An ergodic Markov chain with transition matrix P has the following properties:*

- *it has a single stationary distribution π that satisfies $\pi(x) > 0$ for every $x \in \Omega$*
- *for any initial distribution π_0 the distribution after t steps converges to the stationary distribution : $\lim_{t\to+\infty} P^t \pi_0 = \pi$*

Glauber dynamics. Let $G = (V, E)$ be a graph and $k \geq 0$ be an integer. In our setting, the state space Ω is the set of all proper k-colorings of G (recall that proper colorings correspond to feasible states defined earlier in this section). The *Glauber dynamics* is the algorithm that defines the stochastic process of transition from a proper coloring α_t to another as follows:

1. Choose a vertex $v \in V$ and a color $c \in [k]$ uniformly at random.
2. Let β be the coloring obtained by changing the color of v to c. If β is a proper coloring of G then set $\alpha_{t+1} := \beta$ and set $\alpha_{t+1} = \alpha_t$ otherwise.

Since at most one vertex is recolored at each step, the Markov chain defined by the Glauber dynamics corresponds to a random walk on the k-reconfiguration graph of G $\mathcal{R}_k(G)$. Let us denote by P the transition matrix of the corresponding Markov chain. Note that by step 1 of the process described above, we have $P(\alpha, \beta) = \frac{1}{k|V|}$ for any two proper colorings $\alpha, \beta \in \Omega$ and thus the chain is aperiodic (since the probability of recoloring a vertex with its current color is always strictly positive) and reversible. In particular this latter property implies by Proposition 1 that the uniform distribution is stationary. Finally, if $\mathcal{R}_k(G)$ is connected then the chain is ergodic: by Theorem 10 the uniform distribution over the proper colorings of G is the only stationary distribution and the Markov chain converges to it, starting from any proper k-coloring. In other words if $\mathcal{R}_k(G)$ is connected then by simulating the Glauber dynamics process "long enough", one can get as close as the uniform distribution as desired an thus sample the proper colorings of the input graph almost uniformly at random. Hence the two following questions have been at the center of attention of the recoloring community for the last decade:

1. For which value of k is $\mathcal{R}_k(G)$ connected?
2. How fast can we reach the stationary distribution of the Glauber dynamics Markov chain?

In order to answer the second question we first need to be able to assess the distance between two distributions. There are several way to do so but the classical choice is usually the *total variation distance*. We refer the interested reader to [70] for more details about distances. The total variation distance $||\mu - \nu||_{TV}$ between two distributions μ and ν over Ω is defined as follows:

$$||\mu - \nu||_{TV} = max_{A \subset \Omega} |\mu(A) - \nu(A)| \tag{2.4}$$

which simply corresponds to the maximum difference of probabilities between ν and μ among all possible events. Using this measure we can now formally define what being "close enough" from the stationary distribution means. The mixing time τ is the smallest integer t after which π_t is at a distance at most $\frac{1}{4}$ from π:

$$\tau = min\{t \geq 0 : ||\pi_t - \pi||_{TV} \leq \frac{1}{4}\} \tag{2.5}$$

Note that the constant $\frac{1}{4}$ is arbitrary and that any constant less than $\frac{1}{2}$ also yields a valid definition for the mixing time (any such constant ensures that π_t converges exponentially fast to π after time τ). If the mixing time is bounded by a polynomial in $|V|$ the chain is *rapidly mixing* and this gives an efficient sampling algorithm.
Early works towards the Glauber dynamics focused on this mixing time and studied the value of k for which $\mathcal{R}_k(G)$ is trivially connected (typically $k \geq \Delta + 2$ as shown by Theorem 9). Jerrum [65] and independently Salas and Sokal [84] showed that the Glauber dynamics Markov chain is rapidly mixing for $k \geq 2\Delta$. In the same paper, Jerrum actually formulated a conjecture that is still widely open today:

Conjecture 1. *The Glauber dynamic of a graph G is rapidly mixing for $k \geq \Delta(G) + 2$.*

The first improvement on the 2Δ bound was given by Vigoda in [93]. In this paper, Vigoda studies the *flip dynamic*, which instead of modifying the color of one vertex at each step, exchanges the two colors on a Kempe chain of bounded size (the definition of Kempe chains is given in Section 1.2). He shows that this Markov chain is rapidly mixing and this implies that the Glauber dynamics mixes in time $O(n^2 klog(n)log(k))$ provided that $k \geq (11/6)\Delta$. It remained the best known bound for more than ten years until Chen et al. improved it [32] to $(11/6 - \epsilon)$ where $\epsilon > 0$ is a constant independent from Δ. This is where the results towards Jerrum's conjecture in the general case stand today, still leaving an important gap with regards to the $\Delta + 2$ bound.
It is only more recently that the recoloring problem was considered from a purely combinatorial and graph theoretic point of view, inspired by works of Heuvel [57] and Bonsma and Cerceda [22, 31] which we describe in the next section.

2.4 Diameter of the reconfiguration graph and Cereceda's conjecture

As seen in Section 2.3, the graph recoloring problem was initially mostly studied with a number k of colors that is large enough compared to Δ to ensure that the k-reconfiguration graph is connected (and hence ensure that the Glauber dynamics Markov Chain is ergodic). It is only in the early 2000's that the connectedness of the reconfiguration graph was compared to other graph parameters that are intimately linked with graph colorings, in particular the chromatic number and the degeneracy. These new studies raised the following question that used to have a trivial answer when considering a high number of colors: what is the smallest number of colors that is needed to ensure that $\mathcal{R}_k(G)$ is connected? Which impact does lowering the number of colors have on the diameter on the reconfiguration graph? If the graph is not connected what is the "structure" of its connected components? In this section, we describe the first fundamental results obtained toward these questions and explain how they lead to Cereceda's conjecture, which is still one of the main topic of interest of the recoloring community today. We then focus on the related problem of finding *linear transformations between colorings* which is one of the main question studied in this thesis.

Chromatic number. Since we only consider proper colorings the study of the k-reconfiguration graph of a graph G only makes sense when we allow at least $k \geq \chi(G)$ colors. Since χ is defined as the smallest number of colors needed to properly color a graph it is not hard to come up with examples for which the connected components of $\mathcal{R}_\chi(G)$ are "small" and in some extreme cases are single vertices (the most obvious example being the one of cliques for which any χ-coloring is frozen). A maybe more interesting example given by Cereceda in his thesis [31] is the case of 2-colorable

graphs. Given such a graph G and a proper 2-coloring of G, the only valid recoloring that can be applied is to change the color of an isolated vertex. It follows that if G has p isolated vertices and q non trivial connected components, then $\mathcal{R}_Z(G)$ has 2^q connected components each of which is a hypercube of dimension p. Indeed in each connected component of the reconfiguration graph, the colorings of the non-trivial components of G are frozen and there are exactly two proper colorings for each such component. Furthermore, the colorings of the p isolated vertices of G can be seen as a vector of dimension p with entry $\{0, 1\}$, two colorings being adjacent if they differ on exactly one coordinate of the corresponding vector (since isolated vertices can be recolored freely). It follows that each connected component of $\mathcal{R}_k(G)$ is a hypercube of dimension p.
Note that if we allow one more color (if we consider $\mathcal{R}_3(G)$), this simple analysis completely breaks and there is a priori no reason why the reconfiguration graph should not be connected. Hence the following question: can we find an expression $f(\chi)$ such that for any graph G and for any $k \geq f(\chi)$, the reconfiguration graph $\mathcal{R}_k(G)$ is connected? Somewhat counter-intuitively, the answer to this question is negative. Cereceda [31] proved the following:

Proposition 2. *For any integer $k \geq 0$ there exists a graph G such that $\chi(G) = 2$ and $\mathcal{R}_k(G)$ is not connected.*

Proof. Let L_k be a complete bipartite graph minus a perfect matching with bipartition (A, B) such that $|A| = |B| = k$. Let $u_1, \ldots, u_k$ denote the vertices in A and $v_1, \ldots, v_k$ denote the vertices in B such that for every $i \leq k$, v_i is the only non-neighbor of u_i. Consider the k-coloring α of L_k such that for every $i \leq k$, $\alpha(u_i) = \alpha(v_i) = i$. This is a proper coloring of L_k, and it is frozen since the whole set of color appears in the closed neighborhood of each vertex. □

Degeneracy. The fundamental parameter that has been considered for the study of the k-COLOR-PATH and the k-COLOR-MIXING problems is the degeneracy of the input graph. It is a well-known fact that for any graph G, the chromatic number of G is bounded by $d(G) + 1$ (see the sketch of proof in Section 1.1.2). However, unlike the chromatic number, the degeneracy was proven to be a good parameter to describe the structure of reconfiguration graphs. The following result was shown independently in [38] and [22]:

Theorem 11. *For any graph G and any integer $k \geq d(G) + 2$ the reconfiguration graph $\mathcal{R}_k(G)$ is connected.*

Proof. The proof is the same as the the proof of Theorem 9, except that the induction is applied on $G - v$ where v is a vertex of degree at most d. □

Since the maximum degree of a graph is an upper bound on its degeneracy, it immediately follows from Theorem 11 that $\mathcal{R}_k(G)$ is connected for any $k \geq \Delta(G) + 2$. It is not hard to see that this result is best possible. Consider a clique on n vertices:

such a graph has degeneracy $n-1$ and any n coloring of this graph is frozen.
Due to its inductive nature we can easily count the number of recolorings made by the algorithm described in the proof of Theorem 11. When extending the recoloring sequence from G-v to G the vertex v is eventually recolored at each step, doubling the size of the sequence. This simple analysis shows that this algorithm performs at most 2^n recolorings to reach any coloring of G starting from any other. This yields the first upper bound on the diameter of $\mathcal{R}_k(G)$. Hence the question that has been at the center of focus of the recoloring community for the last decade: is it possible to find a better general upper bound on the diameter of the reconfiguration graph? In his seminal thesis, Cereceda [31] conjectured the following:

Conjecture 2. *For any graph G on n vertices and any integer $k \geq d(G)+2$, the diameter of $\mathcal{R}_k(G)$ is at most $O(n^2)$.*

Originally this conjecture was first formulated in [22]. In this paper, Bonsma and Cereceda investigated the complexity of the k-COLOR PATH problem and obtaind tight results:

Theorem 12. *For any 3-colorable graph G, 3-COLOR-PATH can be solved in polynomial time. Furthermore, the diameter of a connected component of $\mathcal{R}_3(G)$ is at most $O(n^2)$.*

As for the negative side, they show that:

Theorem 13. *For every integer $k \geq 4$, k-COLOR-PATH is* PSPACE*-complete.*

As the graphs constructed in the proof of Theorem 13 are planar and bipartite, they obtain straightforwardly that k-COLOR-PATH remains PSPACE-complete even for planar graphs and $4 \leq k \leq 6$, bipartite graphs and $k \geq 4$ and planar bipartite graphs when $k = 4$.
The most obvious certificate for a reconfiguration problem, and more particularly for the REACHABILITY problem, is the reconfiguration sequence itself as one only has to check that every configuration in the given sequence is valid and that two consecutive configurations are indeed adjacent. Hence showing that every reconfiguration sequence has polynomial length, or in other words that the reconfiguration graph has polynomial diameter immediately yields that the considered problem belongs to NP. Following the assumption that NP $\neq$ PSPACE, Theorem 13 implies that there exist instances of k-COLOR-PATH for which $R_k(G)$ does not have polynomial diameter. Bonsma and Cereceda conclude their work by constructing such instances and show that:

Theorem 14. *For every $k \geq 4$, there exists a class of graph such that $\mathcal{R}_k(G)$ has super-polynomial diameter. Furthermore for $4 \leq k \leq 6$ these graphs may be taken to be planar and for $k = 4$ these graphs may be taken to be planar and bipartite.*

In their construction proving Theorem 14, they observe that the main obstacle for obtaining reconfiguration graphs of planar graphs or planar bipartite graphs with super-polynomial diameter for larger value of k seems to be the degeneracy of the

considered graph classes (which is 5 and 3 respectively). This led them to conjecture that for any graph G and any integer $k \geq d(G) + 2$ it is not possible to find such graphs. In other word the reconfiguration graph is connected for such value of k (following from Theorem 11) and should have polynomial diameter. Let us remark that Conjecture 2 was initially stated with a cubic bound in [22] and was then stated as it is known today with a quadratic bound in [31]. The only clue that then led to conjecture a quadratic bound was another result of the same authors showing that for any $k \geq 2d(G) + 1$ the reconfiguration graph $\mathcal{R}_k(G)$ has diameter at most $O(n^2)$ and the fact that no counter-example was known.
This conjecture has received a considerable amount of attention in the last decade. Despite much effort it was only proven to be true in the general case for $d = 1$ (trees) [19] and then later for $d = 2$ for graphs of maximum degree three [44]. Hence the conjecture is still widely open. However, Bousquet and Heinrich [27] recently made a major breakthrough showing that the diameter of $R_{d+2}(G)$ is at most $O(n^{d+1})$ and thus is indeed polynomial. They in fact also obtained general results when the number of allowed color is higher than $d + 2$:

Theorem 15. *Let d, k be integers and G be a d-degenerate graph. Then $R_k(G)$ has diameter at most:*

- *Cn^2 if $k \geq \frac{3}{2}(d+1)$ (where C is a constant independent from k and d),*
- *$C_\epsilon n^{\lceil 1/\epsilon \rceil}$ if $k \geq (1+\epsilon)(d+2)$ and $0 < \epsilon < 1$ (where C_ϵ is a constant independent from k and d),*
- *$(Cn)^{d+1}$ for any d and $k \geq d+2$ (where C is a constant independent from k and d).*

Note in particular that the diameter of $\mathcal{R}_k(G)$ indeed becomes quadratic when $k \geq \frac{3}{2}(d(G) + 1)$. Furthermore, Cereceda's conjecture has been proven to be true for some particular graph classes such as planar bipartite graphs [27], chordal graphs [19] and then generalized to bounded treewidth graphs [15] (the proof for the case of bounded treewidth graphs was recently simplified by Feghali in [40]). Planar graphs have also been extensively studied in terms of graph recoloring in the last few years: the first subexeponential bound for the diameter $\mathcal{R}_7(G)$ was given in [39] and was improved to a polynomial bound of order $O(n^6)$ as a direct consequence of Theorem 15. Note that Theorem 15 also yields that for every $k \geq 9$ the reconfiguration graph $\mathcal{R}_k(G)$ of a planar graphs G has quadratic diameter. This result on the smallest known value of k for which reconfiguration graphs of planar graphs have quadratic diameter was improved in [42] where the author shows that for every planar graph G and every $k \geq 8$, $\mathcal{R}_k(G)$ has actually a diameter of at most $O(n \cdot polylog(n))$. This gives hope that the bound on the diameter of $R_7(G)$ can still be improved: it would indeed come as a surprise that the diameter of the reconfiguration graphs of planar graphs undergoes a jump from $O(n^6)$ to subquadratic when allowing the use of just one more color. Some sketch of proofs of results mentioned in this section are given in

Chapter 3.
Let us conclude about results toward Cereceda's conjecture by underlining the fact that if the conjecture is true then this result is best possible, in the sense that there exist $(d+2)$-colorings of a d-degenerate graph G which are at distance $\Omega(n^2)$ in $R_{d+2}(G)$. Consider the two 3-colorings of a path where vertices are colored from left to right with respectively colors 1, 2, 3 (and where this pattern repeats) and colors 3, 2, 1. It was shown in [19] that at least $\Omega(n^2)$ recolorings are needed to transform one into the other.

2.5 Linear transformations between colorings and our contributions.

As seen in Section 2.4, Cereceda's conjecture (Conjecture 2) was shown to be true (and tight) in some particular cases. Bousquet and Heinrich provided the first polynomial bound $O(n^{d+1})$ for d-degenerate graphs with $d+2$ colors in Theorem 15. Allowing for more colors above the $d+2$ threshold lowers the number of recoloring needed to go from one coloring to another, as seen for instance in the case of planar graphs. Hence the following question: what is the minimum number of colors needed to ensure that the diameter of the reconfiguration graph is linear? Note that one cannot hope for a smaller diameter, as a linear number of recolorings is always needed to go, for instance, from a coloring to one of its permutations.
By carefully analyzing and adapting the algorithm of Dyer et al [38], Bousquet and Perarnau showed in [29] that it also yields a general bound to obtain linear transformations between colorings:

Theorem 16. *Let G be a d-degenerate graph. The diameter of the $(2d+2)$-reconfiguration graph of G is at most $(d+1)n$.*

It follows that if G is a d-degenerate graph, the diameter of $R_{d+2}(G)$ is polynomial and the diameter of $R_{2d+2}(G)$ is linear. It has also been shown that there exists some d-degenerate graphs G for which $diam(\mathcal{R}_{d+2}(G)) = \Omega(n^2)$. However, there is no known example for which the bound given by Theorem 16 is tight. This raises the following question: how many additional colors are needed above the $d+2$ threshold to obtain a linear diameter for the reconfiguration graph? As far as we know, the answer to this question might as well be 1.
Results toward this line of study are relatively sparse and often come as corollaries of articles concerning Cereceda's conjecture. Linear transformations were mostly studied for their own sake for planar graphs. In particular in the last few years, several papers successively improved the $2d+2$ (i.e 12, since planar graphs are 5-degenerate) bound for this class. The best result today is due to Feghali and Dvořák [37] who showed that the diameter of $\mathcal{R}_{10}(G)$ is at most $7n$ and that $\mathcal{R}_7(G)$ has diameter at most $8n$ if G is planar and triangle-free, resolving the problem for this class of graphs.

Our contributon. The main contribution of this thesis concerning graph recoloring problems is to continue in this line of work and study other classes of graphs for which we can prove linear diameter of the reconfiguration graph with few extra colors above the $d+2$ threshold. First, we introduce a new proof technique along with an algorithm to show the following:

Theorem 4. *Let G be a d-degenerate chordal graph of maximum degree Δ. For every $k \geq d+4$, the diameter of $\mathcal{R}_k(G)$ is at most $O_\Delta(n)$. Moreover, given two colorings c_1, c_2 of G, a transformation of length at most $O_\Delta(n)$ can be found in linear time.*

The algorithm is described in Chapter 4. The results presented in Chapter 4 were obtained with Nicolas Bousquet and were accepted at the 27th Annual European Symposium on Algorithms (ESA), which was held in Munich in September 2019. The full paper is available on arXiv [26].

We then make a carefull study of the algorithm of Bousquet and Perarnau given in Theorem 16 and show that it actually achieves a linear number of recolorings for graphs of treewidth 2 with 5 colors, closing the problem for this class of graphs:

Theorem 5. *Let G be graph of treewidth at most 2 and $k = 5$. There exists a constant c such that, for every pair of 5-colorings α, β of G, there exists a transformation from α to β recoloring each vertex at most c times.*

The proof of this result is given in Chapter 5. The results presented in Chapter 5 were obtained with Nicolas Bousquet and Marc Heinrich and were accepted for publication in Discrete Mathematics. The full paper is available on arXiv [7].

Before stating formally our results and diving into the proofs, we describe in the next chapter the main proof techniques that have been used to obtain results toward - and sometimes prove - Cereceda's conjecture in particular cases. As will we see, most of the recent works make use of similar ideas that revolve around finding a nice way to "peel" the input graph as one can do with a degeneracy ordering. In the next chapter, we describe and illustrate these techniques with several examples taken from the literature. We will see that although these techniques may be very efficient to prove quadratic bounds for the diameter of reconfiguration graphs with few extra colors, most tend to fail when one wants to prove linear bounds.

Chapter 3

Proof techniques for graph recoloring

Many existing results toward Cereceda's conjecture make use of similar proof techniques. In this Chapter, we identify three different strategies that have been used several times and describe them through sketch of proofs. We also discuss whether these techniques can be refined to obtain linear bounds. We conclude this Chapter by a brief description of the very few existing proofs of linear bounds for planar graphs, and then describe our techniques.

3.1 Induction based techniques

The idea behind induction based techniques is to refine the algorithm given in Theorem 11 as follows: instead of removing a single vertex (of degree at most d) at each inductive step, try to remove a subset S of vertices as large as possible. Then recolor the graph induced by $V(G) \setminus S$, and adapt the obtained sequence to a sequence for G. Choosing the set S as large as possible limits the number of inductive steps, and thus reduces the total number of recolorings. However, the set S must satisfy several properties: the vertices in S must have small degree in $V(G) \setminus S$ to ensure that the recoloring sequence obtained by induction on $G - S$ can be extended to S, and the graph induced by S must be sparse (ideally an independent set, as we will see through an example) so that recolorings of vertices in S do not conflict between each others.

Graphs with bounded maximum average degree. The arguably simplest proof that makes use of this idea to obtain a polynomial bound is the one of Feghali for the case of graphs of bounded maximum average degree. The *maximum average degree* of a graph G is the maximum average degree of a graph H taken over all subgraphs of G, in other words:

$$m_d(G) := max_{H \subseteq G} \left\{ \frac{2|E(H)|}{|V(H)|} \right\} = max_{H \subseteq G} \left\{ \frac{\sum_{v \in V(H)} d_H(v)}{|V(H)|} \right\}$$

In [42], Feghali shows the following:

Theorem 17. *Let m_d and k be positive integers, $k \geq m_d + 1$. For every $\epsilon > 0$ and every graph G with n vertices and maximum average degree $m_d - \epsilon$, there exists a constant $c = c(m_d, \epsilon)$ such that $\mathcal{R}_k(G)$ has diameter $O(n^c)$.*

The maximum average degree of a graph is related to its degeneracy. Indeed, it is not hard to show that for any graph G, $d(G) \leq m_d(G) \leq 2d(G)$ (see for instance [36]). Note that Theorem 17 was first proved by Bousquet and Perarnau in [29]. Althought their proof makes use of similar ideas it is quite more intricate, hence we chose only to sketch Feghali's proof here. A sketch of the proof of Bousquet and Perarnau can be found in Appendix A.1.

The idea of Feghali is the following: instead of peeling the graph one vertex at a time, one can delete an independent set at each step. Let I denote the deleted independent set. When adapting the recoloring sequence of $G - I$ to a sequence of G, the recolorings of any two vertices in I are independent from each other (since a a deleted vertex only needs to be recolored whenever one of its neighbor is). However it is not possible to take any independent set I of G: each vertex $v \in I$ must have degree at most $m_d - 1$ in $G - I$ to ensure the existence of an available color to recolor v when needed. Combining these two ideas leads to the following algorithm: (i) compute an independent set composed of vertices of degree at most $d - 1$ of maximum size and delete it from the graph, and (ii) recolor the remaining graph by induction and adapt the obtained sequence to a sequence of G as done in the proof of Theorem 11. Since the input graph G satisfies $m_d(G) \leq m_d - \epsilon$, it contains an independent set I satisfying the condition in (i) of size at least $a \cdot n$ for some constant $a > 0$ depending only of m_d and ϵ. Since each vertex in I is recolored only when one of its neighbor is, it is recolored at most $f(n) \leq (d-1)f((n - an)) + 1$ times (the term $+1$ corresponding to the last recoloring of the vertex to its target colors). The claim follows by the master theorem.

Let us now briefly discuss why it would be very difficult to obtain linear bounds using the same kind of technique: In both cases, the algorithm removes a part of the graph of linear size, recolor the rest to obtain some desirable properties (actually the target coloring in Feghali's case) and then adapt this sequence for the whole graph (using assumptions on the degree of the deleted vertices). Removing a subgraph of linear size at each step is best possible, and thus one cannot hope for linear bounds without more control on the procedure recoloring the remaining graph. However, combining this idea and a degeneracy ordering, Feghali achieved an almost linear bound $O(n \cdot polylog(n))$ for the 8-reconfiguration graph of planar graphs in [42].

3.2 Identification technique

Let G be a graph and u, v be two vertices of G. The graph obtained by *identifying* u and v into a new vertex w is the graph G' on the vertex set $V(G') = V(G) \setminus \{u, v\} \cup \{w\}$

and such that for every $x, y \in V(G') \setminus \{w\}$, (x, y) is an edge of G' if and only if it is an edge of G, and $N_{G'}(w) = N_G(u) \cup N_G(v)$. Let α be a coloring of G such that $\alpha(u) = \alpha(v)$. The coloring *induced* by α on G' is the coloring α' such that for every $x \neq w$, $\alpha'(x) = \alpha(x)$, and $\alpha'(w) = \alpha(u)$ $(= \alpha(v))$. Note that α' is a proper coloring of G'.

With these definitions at hand let us briefly explain the *vertex identification* technique. Let G be a graph, α, β be two colorings of G and let $u, v \in V(G)$ be two non-neighbors. Suppose that after a number of recolorings, one is able to obtain a coloring α' such that $\alpha'(u) = \alpha'(v)$ and a coloring β' such that $\beta'(u) = \beta'(v)$. Then, let G_1 be the graph obtained by identifying u and v into a new vertex w and α_1, β_1 be the colorings induced by α' and β' on G_1 respectively. Since G_1 has one less vertex, a sequence $\mathcal{S}$ of recolorings from α_1 to β_1 can be found by induction. This sequence can be turned into a sequence from α' to β' in G as follows: each time the vertex w is recolored to some color c, replace this recoloring by two consecutive recolorings of the vertices u and v to c. Since $N(w) = N(u) \cup N(v)$ and since every coloring in $\mathcal{S}$ is proper, the sequence obtained for G is also a valid sequence. Since by supposition there exists a sequence from α to α' and a sequence from β' to β, we obtain a sequence from α to β in G.
The efficiency of the vertex identification technique relies on two points: first, when dealing with particular graph classes, the graph obtained after identifying u and v must stay into the same class to apply induction. Secondly, the number of steps needed to recolor u and v with the same color must be constant, in order to obtain a sequence of quadratic length from α to β. In what follows, we illustrate this technique by sketching a proof of Bonamy et al. [19] on chordal graphs.

Chordal graphs. The class of chordal graphs is very well-structured and has many known equivalent definitions and decompositions. In particular, it is well-known that chordal graphs admit a tree decomposition where every bag is a clique (see for instance [47]), also called a *clique-tree*. Let us sketch how Bonamy et al. make use of this decomposition theorem in order to prove the following in [19]:

Theorem 18. *Let G be a d-degenerate chordal graph on n vertices. For any integer $k \geq d + 2$, the diameter of $R_k(G)$ is $O(n^2)$. Furthermore, for any integer $d \geq 0$ and $n \geq 0$ there exists a d-degenerate chordal graph such that $R_{d+2}(G)$ has diameter $\Theta(n^2)$.*

Recall that a chordal graph is d-degenerate if and only if its clique number is $d + 1$. The first step consists in proving the theorem for the case where G is a clique:

Lemma 1. *Let C be a clique on n vertices and α, β be two $(n + 1)$-colorings of C. There exists a recoloring sequence from α to β where each vertex is recolored at most twice.*

Proof. Let α and β denote the initial and target colorings respectively. We construct a partial orientation of G as follows: if u is a neighbor of v such that $\alpha(u) = \beta(v)$, we orient the edge (u, v) from v to u. In other words, the arc (v, u) indicates that the vertex u prevents the vertex v to be recolored with its target color. Since G is a clique, each vertex has out-degree at most one and this orientation of G is a collection of directed paths and cycles. First consider a directed path P: pick the vertex of out-degree zero of P and recolor it to its target color, then continue with its unique in-neighbor. Then consider a cycle C and pick any vertex of v of C: since v has degree $n-1$ and there are $n+1$ colors, there exists a color that v can be recolored with. Then reconstruct the orientation as before: the in-neighbor of v in the previous orientation now has out-degree 0. In other words, the set of vertices in C forms a path in the new orientation and can be treated as previously. It follows that α can be transformed into β by recoloring each vertex at most twice. □

Let us now sketch the proof of Theorem 18. Let T be a clique tree of G such that all the bags of T are maximal cliques (it can be obtained from any clique tree by contracting edges between bags with a strict inclusion relation). Let B be a leaf of T and P be its parent in T. By definition of the clique-tree, $V(B) \cap V(P) \leq d$ and there exists $u \in V(B) \setminus V(P)$ and $v \in V(P) \setminus V(B)$. If $\alpha(u) \neq \alpha(v)$ then recolor u with $\alpha(v)$ if possible. Otherwise, there exists $w \in V(B) \setminus V(P)$ that is colored with $\alpha(v)$ (v is complete to $V(P) \cap V(B)$ which then contains no vertices colored with $\alpha(v)$). Since there are $d+2$ colors and $V(B)$ induces a clique on at most $d+1$ vertices, there exists a color that does not appear in $N[w]$ and w can be recolored with it. Then u can be recolored with $\alpha(v)$. Hence, after applying four recolorings, we have $\alpha(u) = \alpha(v)$ and $\beta(u) = \beta(v)$. Now let G' be the graph obtained by identifying u and v and let v' denote the corresponding vertex. It is not hard to check that G' is a d-degenerate chordal graph. Let α' and β' be the colorings induced by α and β in G' respectively. By induction on the order of G, there exists a recoloring sequence from α' to β' in which each vertex is recolored at most $n-1$ times. Finally, a recoloring sequence of G can be obtained by replacing each recoloring of v' by two consecutive recolorings of v and u. This is a valid reconfiguration sequence since $N_{G'}(v') = N_G(u) \cup N_G(v)$. Furthermore, each vertex has been recolored at most $2(n-1)+2$ times (2 recolorings per vertex are needed to transform α to α' and β to β') and the sequence has length at most $2n^2$.

Here, the graph is not exactly peeled, but is rather "folded" vertex by vertex up until a single clique remains (or a union of cliques if G is disconnected). Once a single clique is obtained and a sequence for this clique is found, one can start to unfold the graph and extend the recoloring sequence vertex by vertex as done above. Let us also stress the fact that the proof in [19] is more involved since it holds for a class of graph that contains chordal graphs. Note that increasing the number of colors cannot give a subquadratic bound with the same kind of technique. Going from $d+2$ to $d+3$ colors lowers the bound on $diam(\mathcal{R}_k(G))$from $2n^2$ to n^2 (since then no neighbor

of u needs to be recolored prior to changing the color of u to $\alpha(u)$). Increasing the number of colors above $d+3$ gives no room for a better bound since the whole graph needs to be recolored when applying induction. Stronger assumptions would be needed to ensure that only a constant number of vertices are recolored at each steps.

In Appendix A.2 we discuss the cases of *weakly chordal graphs* studied by Feghali and Fiala in [43] and *OAT graphs* studied by Biedel et al. in [12], as further examples on how to use the vertex identification technique. Both proofs follow the same path, which is very similar to the one used by the proof for chordal graphs described above. It can be summarized the following way: the considered class of graphs admits a suitable decomposition. Furthermore, two vertices u, v satisfying $N(u) \subseteq N(v)$ can be recolored the same color in few recoloring steps. These two vertices can then be identified, and the decomposition theorem ensures the membership of the newly obtained graph to the class.

3.3 List recoloring

Another technique that was proven to be successful comes from the more general *list-coloring* setting. Let G be a graph. A *list assignment* L of G is a function that associates a list of colors $L(v)$ to each vertex $v \in V(G)$. A proper coloring c of G is a L-coloring if for every $v \in G$, $c(v) \in L(v)$. The graph G is *k-choosable* if it admits an L-coloring for every list assignment L such that $|L(v)| = k$ for every $v \in G$. The *choice number* $ch(G)$ of G is the smallest integer k such that G is k-choosable. Problems related to list coloring have been extensively studied in the literature, see for instance [36] for a good introduction to the field.
As for graph coloring, we can define the reconfiguration counterpart of the list coloring problem as follows:

k-List-Color-path

Input: A graph G, a list assignment L of G such that $|L(v)| \leq k$ for every $v \in V(G)$ and two L-colorings α and β of G

Question: Is it possible to transform α into β by recoloring one vertex at a time while always maintaining a L-coloring of G?

This problem was first defined by Bonsma and Cereceda in [22], and similar definitions can be found for instance in the survey of Van Den Heuvel [57]. Bonsma and Cereceda initially introduced list recoloring as a tool for simplifying their construction of graphs that admit colorings at super-polynomial distance from each others. A way to design such graphs is to apply "constraints" on vertices, or in other words to construct the graph so that some vertices can only take few colors at each reconfiguration steps. Bonsma and Cereceda show that up to adding a clique Q of size k to a graph G, they can obtain such constraints: In any proper k-coloring of $G \cup Q$, the coloring of Q is frozen. Now, one can forbid any vertex $v \in G$ to be recolored with c by adding an

edge between v and the unique vertex $q \in Q$ colored with c. This defines a list $L(v)$ of authorized colors for every vertex $v \in G$, and the problem now reduces to the study of reconfiguration of L-colorings of G.
Even though the list-recoloring problem was introduced more than a decade ago, it was first used to obtain polynomial bounds for reconfiguration graphs only very recently. The technique was first used by Feghali in [41] to show the following:

Theorem 19. *Let G be a planar graph on n vertices. Then $R_{10}(G)$ has diameter at most n^2.*

The proof is based on the following lemma due to Thomassen [91]:

Lemma 2. *Let G be a planar graph and v be a vertex of G. Let L be a list assignement such that $|L(v)| = 1$ and $|L(u)| \leq 5$ for every vertex $u \neq v$ of G. Then G admits an L-coloring.*

We can know give the full proof of Theorem 19 as done in [41]:

Proof of Theorem 19. Let α and β be two 10-colorings of G. Since G is planar it admits a 5-degenerate ordering, that is an ordering $v_1, \ldots, v_n$ of $V(G)$ such that v_i has at most five neighbors in $\{v_1, \ldots, v_{i-1}\}$ for every $i \leq n$. Let $\ell \leq n$ be the smallest integer such that $\alpha(v_\ell) \neq \beta(v_\ell)$. Define a list assignment L of $G_\ell := G[\{v_\ell, \ldots, v_n\}]$ as follows: $L(v_\ell) = \beta(v_\ell)$ and $L(v_j) = \{1, \ldots, 10\} \setminus \{\alpha(v_i) | i < j\}$ for $j > \ell$. By definition of the ordering, L satisfies the condition of Lemma 2 with $v = v_\ell$. Hence there exists an L-coloring c. By definition of c, recoloring each vertex with its color in c starting from v_n up to v_ℓ is a valid recoloring sequence, after which v_ℓ is colored with $\beta(v_\ell)$. Note that each vertex of G is recolored at most once in this sequence.
Starting from α and repeating this process at most n times gives a recoloring sequence from α to β of length at most n^2, which concludes the proof. □

This result was then successively improved, up until very recently Dvořák and Feghali showed in [37] that the 10-reconfiguration graph of a planar graph actually has linear diameter:

Theorem 20. *Let G be a planar graph on n vertices. Then $\mathcal{R}_{10}(G)$ has diameter at most $8n$.*

The proof of Theorem 20 also makes use of list recoloring, although it does not directly relies on results from list coloring. Instead, it adapts the celebrated proof of Thomassen showing that planar graphs are 5-choosable. We give a sketch of this elegant proof in the next section.

Polynomial version of Cereceda's conjecture. Let us conclude this section by briefly discussing the proof of Theorem 15 by Bousquet and Heinrich, which we recall below:

Theorem 15. *Let d, k be integers and G be a d-degenerate graph. Then $R_k(G)$ has diameter at most:*

- *Cn^2 if $k \geq \frac{3}{2}(d+1)$ (where C is a constant independent from k and d),*
- *$C_\epsilon n^{\lceil 1/\epsilon \rceil}$ if $k \geq (1+\epsilon)(d+2)$ and $0 < \epsilon < 1$ (where C_ϵ is a constant independent from k and d),*
- *$(Cn)^{d+1}$ for any d and $k \geq d+2$ (where C is a constant independent from k and d).*

This theorem not only gives the best general bound on the diameter of the $(d+2)$-reconfiguration graphs of d-degenerate graphs, but also shows that the $\frac{3}{2}(d+1)$-reconfiguration graphs of such graphs have quadratic diameters, which is the best known upper bound toward Cereceda's conjecture known today. Let us explain roughly how Bousquet and Heinrich make use of lists to prove Theorem 15. For the sake of simplicity, we only consider the case where there are $k = d+2$ colors.
Let G be a d-degenerate graph on n vertices with degeneracy ordering $v_1, v_2, \ldots, v_n$ and let $d^+(v_i)$ be the neighbors of v_i in $\{v_{i+1}, \ldots, v_n\}$ which we denote as the *out-neighbors* of v_i. Let α, β be two $(d+2)$-colorings of G. Furthermore, each vertex v_i has a list assignment $L(v_i)$ of colors it can be recolored with, which initially contains all of the $d+2$ colors.
Recall that the classical way to use the degeneracy ordering is to delete a vertex of small degree and then to recolor the graph that remains. The key idea of Bousquet and Heinrich, is to proceed in the opposite direction to avoid the recoloring of the whole graph at each step. They rather delete the vertices $\{v_{i+1}, \ldots, v_n\}$ and consider the graph induced by $\{v_1, \ldots, v_i\}$. Then, for each vertex v_j with $j < i$ they delete the colors of $N(v_j) \cap \{v_{i+1}, \ldots, v_n\}$ in $L(v_j)$. In other words the colors of the deleted neighbors of v_j with $j < i$ are also deleted from the list $L(v_j)$. Initially each vertex v_j has at least $|d^+(v_j)| + 2$ colors in its list (since G is d-degenerate). The same goes after the deletion of $\{v_{i+1}, \ldots v_n\}$, since for every color deleted in $L(v_j)$ for $j \leq i$, at least one out neighbor of v_j is also deleted. This allows to recolor $\{v_1, \ldots, v_i\}$ only without changing the color of the other vertices, avoiding the bottleneck of the proof of Theorem 11. Note however that since they do not have a total control over the colors that remain in the list of the vertices $\{v_1, \ldots, v_i\}$, they cannot recolor these vertices directly to their target colorings, and a few more steps are needed in the complete proof.

These two examples tend to show that seeking for bounds in the more general list-recoloring setting may be one of the most promising way to prove Cereceda's conjecture and obtain linear bounds. It allows to obtain both general results as done by Bousquet and Heinrich, but also allows for some fine-grained studies as shown by Dvořák and Feghali when proving that the 10-reconfiguration graph of planar graphs has diameter at most $8n$. Furthermore it also allows to draw results from the well-studied field of list-coloring, and thus extends the basic tools one has at its disposal for the study of recoloring problems.

3.4 Proof techniques for linear bounds

As mentioned in Section 2.5 there are few results showing linear bounds for the diameter of reconfiguration graphs. Apart from Theorem 16 of Bousquet and Perarnau and the results presented in this thesis, the efforts to find such bounds were focused on planar graphs. In particular, Dvořák and Feghali gave two different proofs of Theorem 20 that we recall below:

Theorem 20. *Let G be a planar graph on n vertices. Then $\mathcal{R}_{10}(G)$ has diameter at most $8n$.*

Both proof begin in the same way, but have very different means to obtain their conclusion. The first argument used by Dvořák and Feghali is the following: finding a linear transformation between 10-colorings of a planar graph reduces to finding a linear transformation that deletes a color from the graph. By deleting a color, we mean finding a transformation from a 10-coloring γ to a coloring γ' that uses at most 9 colors, in a linear number of steps.
Suppose that we are able to do so. Let α and β be the initial and target 10-colorings of a planar graph G, and let α' and β' be the two colorings that use at most 9 colors obtained with this procedure. Up to renaming the colors, we can always assume that neither α' nor β' use color 10. In particular it means that we are free to recolor any independent set of G with color 10. By a theorem of Thomassen [92], $V(G)$ can be partitioned into an independent set I and a set D that induces a 3-degenerate graph in G. Then we can proceed as follows: from both α' and β', recolor the vertices in I with color 10. We can then apply Theorem 16 with the 9 remaining colors to transform the coloring induced by α' on $V(D)$ into the coloring induced by β' on $V(D)$ in a linear number of steps. This yields a valid recoloring sequence since the color 10 is never used, which concludes the proof.
We can now briefly sketch the two different ways to delete a color from the graph designed by Dvořák and Feghali. In both cases, the authors make use of the more general list recoloring framework.

Discharging argument. The key idea of the proof is the following: The authors do not directly try to delete color 10 by recoloring each vertex one time, to a color different from 10. Instead, they show that color 10 can be deleted from any coloring by recoloring each vertex at most twice, such that if a vertex is recolored twice it is first recolored with color 10. This is a seemingly counter-intuitive process, since such a sequence will first maximize the number of vertices colored with 10, and then delete this color from the graph. However, it also allow them to show that in a minimal counterexample G with the corresponding coloring α, the color 10 appears in the closed neighborhood of every vertex. The authors then strongly make use of this argument to determine the structure of the minimal counterexample. They then show that such a counterexample does not exists via a discharging argument.

Thomassen type proof. Being in the list recoloring framework, each vertex v of the graph is given a list assignment $L(v)$. The authors follow the lines of the proof of Thomassen showing the 5-choosability of planar graphs. Instead of assigning 10 colors to each vertex, they reduce the number of colors available for the vertices of the outer-face. This essentially allow them to peel the graph of its outerface while setting colors aside for the deleted vertices: Consider a vertex v of the outerface f. Before applying induction, first delete the current color of v and one color $c \in L(v)$ different from 10 from the list of the neighbors of v that are not on the outer-face. This allows to extend any sequence obtained when recoloring $G - v$ to a sequence for G where v gets recolored with color c (since color c is not used for the neighbor of v on the inner faces). There are however many different cases to deal with when applying such a technique, mainly due to the fact that if too many neighbors of an inner vertex u are deleted during one step, then too many colors are removed from $L(u)$, which prevents from applying induction.
In his proof, Thomassen shows that if the color of one vertex on the outer-face is fixed, he is able to extend this coloring to a list-coloring of the whole graph. One of the arguably most elegant part of the proof of Dvořák and Feghali is the adaptation of this argument for graph recoloring. Very loosely speaking they show that if they are given a recoloring sequence for the vertices of a path of the outerface, this sequence can be adapted to recolor the whole graph to delete color 10.

Despite both proofs of Dvořák and Feghali develop new techniques, they strongly make use of the planarity of the input graph. In the first case when using the decomposition Theorem due to Thomassen [92] and in the second case when making use of the outerface of the graph. Hence, it seems difficult to reuse such arguments for other classes of graph. However theses tools can most likely be used further to improve the bounds on the diameter of the reconfiguration graphs of planar graphs.

3.5 Our techniques

As a conclusion to this Chapter, we briefly mention our proof techniques which are detailed in the next two chapters. The common denominator of all the proofs we presented in this chapter is that they are "inherently global". Indeed, using induction on small subgraphs or identifying vertices always require for the whole graph to be recolored at each step of the algorithm. Although it is not enough to obtain linear bounds, list recoloring techniques allow to spare some recoloring steps as one can actually avoid to recolor parts of the graph by applying restrictions on lists, as it is done by Bousquet and Heinrich in the proof of Theorem 15.
In the next Chapter, we develop a new technique on chordal graphs which is, on the contrary, completely local. Roughly speaking, we show how to recolor small parts of the input graph directly to the target coloring, by only doing small modifications on the coloring of the neighboring parts.

Finally, in Chapter 5, we show that the algorithm designed by Bousquet and Perarnau for proving Theorem 16 is in some particular cases the best possible. Indeed, we show that it actually outputs a sequence of linear size for graphs of treewidth 2.

Chapter 4

Linear transformations between colorings of chordal graphs

The results presented in this chapter were obtained with Nicolas Bousquet and were accepted at the 27th Annual European Symposium on Algorithms (ESA) which was held in Munich in September 2019. The full paper is available on arXiv [26].

As mentioned in the previous chapter, Bonamy et. al showed in [19] that for any chordal graph G on n vertices with degeneracy d, the diameter of $\mathcal{R}_{d+2}(G)$ is at most $O(n^2)$. Furthermore, they exhibited in the same paper an infinite family of d-degenerate chordal graphs for which the diameter of $\mathcal{R}_{d+2}(G)$ is at least $O(n^2)$, showing that the quadratic bound is tight for $d+2$ colors. Bousquet and Perarnau showed that the diameter of the reconfiguration graph is linear when $k \geq 2d+2$ colors. Hence, in the case of chordal graphs, it might be true that the diameter of $\mathcal{R}_k(G)$ is linear whenever $k \geq d+3$. In this chapter, we investigate the following question, raised for instance in [27]: when does the k-recoloring diameter of d-degenerate graphs become linear? In particular, this chapter is dedicated to the proof of the following theorem:

Theorem 4. *Let G be a d-degenerate chordal graph of maximum degree Δ. For every $k \geq d+4$, the diameter of $\mathcal{R}_k(G)$ is at most $O_\Delta(n)$. Moreover, given two colorings c_1, c_2 of G, a transformation of length at most $O_\Delta(n)$ can be found in linear time.*

Note that the bound on k is almost the best possible since we know that this result cannot hold for $k \leq d+2$ as mentioned above. So there is only one remaining case which is the case $k = d+3$.

Question 5. *Is the diameter of $\mathcal{R}_{d+3}(G)$ at most $f(\Delta(G)) \cdot n$ for any d-degenerate graph G?*

In some very restricted cases (such as power of paths), our proof technique can be extended to $k = d+3$, but this is mainly due to the very strong structure of these graphs. For chordal graphs (or even interval graphs), we need at least $d+4$ colors at several steps of the proof and decreasing k to $d+3$ seems to be a challenging problem.

We also ask the following question: is it possible to remove the dependency on Δ to only obtain a dependency on the degeneracy? More formally:

Question 6. *Is the diameter of $\mathcal{R}_{d+3}(G)$ at most $f(d) \cdot n$ for any d-degenerate chordal graph G?*

Again, the best known result related to this question is given by Theorem 16 and no better bound is known even in the particular case of chordal graph. Is it possible to generalize this result to bounded treewidth graphs?

Question 7. *Is the diameter of $\mathcal{R}_{d+3}(G)$ at most $f(\Delta(G)) \cdot n$ for any bounded treewidth graph G?*

Our proof cannot be immediately adapted for bounded treewidth graphs since we use the fact that all the vertices in a bag have distinct colors. As we saw in the previous chapter, Feghali [40] proposed a method to reduce the bounded treewidth case to the chordal case for recoloring problems. However his proof technique does not work here since it may increase the maximum degree of the graph. We nevertheless think that our proof technique can be adapted in order to study other well-structured graph classes.

This proof technique is very different from the one described in the previous chapter and is rather technical. Numerous definitions are needed before being able to prove the four main lemmas leading to Theorem 4. In order for this chapter not to be too tedious to read, we first start with a detailed outline of proof, focusing on the particular case of interval graphs.

4.1 Outline of proof: a warm-up on interval graphs.

All along this section, G is a d-degenerate interval graph of maximum degree Δ. We denote the size of a maximum clique of G by ω and we recall that since G is chordal we have $\omega = d + 1$.

Let $v_1, v_2, \ldots v_n$ be a perfect elimination ordering of V. A greedy coloring of $v_n, \ldots, v_1$ gives an optimal coloring c_0 of G using only ω colors. The coloring c_0 is called the *canonical coloring of* G. The colors $c \in 1, 2 \ldots, \omega$ are called the *canonical colors* and the colors $c > \omega$ are called the *non-canonical colors*. Note that the independent sets $X_i := \{v \in V \text{ such that } c_0(v) = i\}$ for $i \leq \omega$ partition the vertex set V. These independent sets are called the *classes* of G.

Since G is an interval graph, it admits a tree decomposition $P = (\mathcal{W}, E)$ where each bag $W \in \mathcal{W}$ is a maximal clique of G, and such that the graph P is a path. We also choose P so that no two bags are included in each other (see for instance [36] on how to construct such a tree decomposition). Since P is a path, we can order the bags $W_1, \ldots, W_{|\mathcal{W}|}$ such that W_1 and $W_{|\mathcal{W}|}$ have degree 1 and $(W_i, W_{i+1}) \in E$ for every $i \leq |\mathcal{W}| - 1$. We aim to show that any $(d+4)$-coloring of G can be transformed into c_0 by performing at most $O_\Delta(n)$ recolorings.

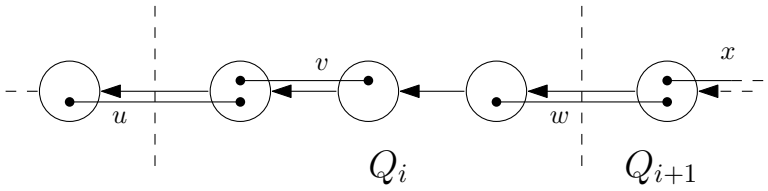


FIGURE 4.1: Example of blocks in the case $\Delta = 3$. The nodes represent cliques of G. The vertices v and w belong to Q_i. The vertex u does not belong to Q_i even if it intersects cliques of Q_i, and the vertex x belongs to Q_{i+1}.

4.1.1 Notion of blocks

With the basic definitions and notations at hand, we can now introduce two of the main definitions that are of central importance in our algorithm, namely the *blocks* and the *regions*.
Consider a vertex v of G that belongs to some bag W_i but does not belong to W_{i-1}. By definition of a tree decomposition, the bags which contains v must be consecutive and thus v does not belong to any bag W_j with $j < i$. We say that the vertex v *starts* in W_i. Since no two bags contain each other, at least one new vertex appears in each bag when we visit the bags of P from left to right starting from W_i. It follows that v belongs to at most Δ consecutive bags, and in particular that $v \notin W_{i+\Delta}$. The set of vertices that start in Δ consecutive bags is a *block* of G. Let us consider a block Q associated with bags $W_i, W_{i+1}, \ldots, W_{i+\Delta-1}$: we outline the fact that any vertex that is contained in some bags W_j with $i \leq j \leq i + \Delta - 1$ but start in a bag W_t with $t < i$ does not belong to the block Q. The following observation follows directly from the definition of a block:

Observation 1. *Every block of G is a separator of G.*

Since furthermore each block is associated to Δ consecutive cliques, we can easily introduce the notion of *consecutive blocks*. The block Q' associated with the bags $W_j, \ldots, W_{j+\Delta-1}$ is consecutive to the block Q associated to the bags $W_i, \ldots W_{i+\Delta-1}$ if $j = i + \Delta$. In other words, Q' is consecutive to Q if the last bag of Q is adjacent to the first bag of Q'. We can also easily extend this definition for a set of blocks of any size: the blocks $Q_1, \ldots Q_t$ are consecutive if Q_{i+1} is consecutive to Q_i for every $i < s$. The notions of blocks and consecutive blocks are illustrated in Figure 4.1.

Consider now three consecutive blocks Q_1, Q_2, Q_3. By Observation 1, the blocks Q_1 and Q_3 separates Q_2 from the rest of the graph. In other words, we have $N[Q_2] \subseteq Q_1 \cup Q_2 \cup Q_3$. This property, which we refer to as the *separation property* is very handy when it comes to recoloring: when recoloring a vertex of Q_2, one only has to check that the recoloring is valid in the sub-graph induced by $Q_1 \cup Q_2 \cup Q_3$ to ensure that it is valid for the whole graph G. This property will be extensively used all along the algorithm.

4.1.2 Vectorial colorings and recolorings

Recall that the class of vertex X_i for $i \leq \omega$ is the set of vertices colored with i in the canonical coloring c_0 (which we ultimately want to reach). Given a block Q, we denote by (Q, p) the (possibly empty) set of vertices of class p in the block Q. Given a proper coloring α of G, we say that the block Q is *well-colored* by α (or simply *well-colored* when α is clear from context) if for every class $p \leq \omega$, the vertices of (Q, p) have the same colors. Note that since α is a proper-coloring, the color of two classes $p \neq q$ of a well-colored block are distinct.

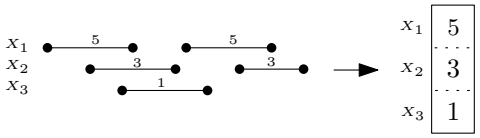


FIGURE 4.2: Example of a well-colored block in the case $\omega = 3$. Vertices are represented as intervals at the left, the color of a vertex is given above the corresponding interval. The representation of the well-colored block as a color vector is given at the right.

The coloring of a well-colored block Q can then be entirely described by a single vector ν_Q of size ω, which i-th coordinate is the color of the vertices of (Q, i). The vector ν_Q is referred to as a *color vector*. We introduce this notion as a tool to simplify the description of the successive recolorings applied by the algorithm. Indeed, the atomic operation in our framework is not the recoloring of a single vertex, but is rather the recoloring of a whole class within a block: when we say that we recolor (Q, p) with color c, we mean that we recolor all the vertices of (Q, p) one after the other with color c in an arbitrary order. Note that if a block Q is well colored with color vector ν_Q and if $c \notin \nu_Q$, then recoloring any class (Q, p) with color c yields a proper coloring of Q.

4.1.3 Regions

Let us finally introduce the notion of *region* before showing how the algorithm works. A *region* R is simply a set of three consecutive blocks (Q_A, Q_B, Q_C). In order to simplify the notations, we often refer to the three blocks constituting a region as A, B and C. The region R is well-colored for the color vectors ν_A, ν_B, ν_C if A is well-colored for ν_A, B is well-colored for ν_B and C is well-colored for ν_C. We can also consider *consecutive regions*: R_2 is consecutive to R_1 if the first block of R_2 is adjacent to the last block of R_1.

Let us now consider a set of $3\Delta N$ consecutive cliques of G for some $N > 0$, starting with some clique W_i for some $i \geq 0$ and explain how we make use of all the definitions introduced above. First, we partition this set of cliques into a set of $3N$ consecutive blocks. This set of block can be itself partitioned into a set of N consecutive regions which we denote (from left to right) as $R_1, \ldots R_N$. Note that the corresponding sequence of blocks is as follows: $A_1, B_1, C_1, A_2, B_2, C_2, \ldots$ up until A_N, B_N, C_N. Such

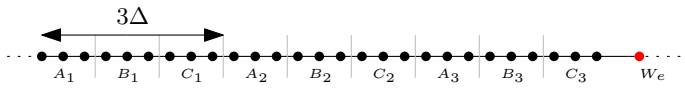


FIGURE 4.3: A buffer consisting of 3 consecutive regions R_1, R_2, and R_3 in the case $\Delta = 3$. The black dots represent maximal cliques of G. The set of vertices E are the vertices starting in the red clique.

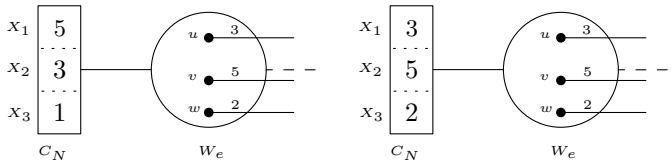


FIGURE 4.4: An example of colorings of the rightmost block of the buffer C_N and of the vertices of E at the beginning of a step (left) and at the end of a step (right). The vertices of E are u, v, and w, belong respectively to X_1, X_2 and X_3 and are initially colored with colors 3, 5, and 2 respectively.

a set of regions is called a *buffer*. Let W_e be the first clique at the right of C_N which does not belong to C_N and let E the set of vertices that start in W_e. An example of such a partitioning is given in Figure 4.3.

Suppose the following: we started from some $d+4$ coloring α of G and after a few steps of the algorithm, the coloring obtained is proper and satisfies the following properties:

1. Each vertex starting in a clique at the left of R_1 is colored with its canonical color. In other words, we are done recoloring the cliques at the left of R_1.

2. The current coloring is such that all the regions from R_1 to R_N (regions of the buffer) are well-colored.

3. Each vertex in each clique at the right of R_N is colored with its initial color in α (and in fact has never been recolored yet).

During one step of the algorithm we are only allowed to recolor vertices of the buffer. We have two objectives to fulfill: First, at the end of the step, the first clique of A_1 must be colored canonically. The second objective is to "integrate" the vertices in E to the buffer. We say that the coloring of C_N *fits with the coloring of* E if C_N is well-colored and satisfies the following property: for every color c such that E contains a vertex of class p colored with c we have $\nu_{C_N}(p) = c$. Hence, a step of the algorithm is done once the color of each class is the same in C_N and E. In other words, the second objective is to modify the coloring of the buffer so that at the end of the step, the vertices in E belong to a well-colored block. An illustration of the coloring we aim to obtain for C_N and E after one step of the algorithm is given in Figure 4.4.

Once we obtain such a coloring, one more clique is colored canonically (the first clique of A_1, in blue on Figure 4.5). We will never recolor this clique again later on during the algorithm. Furthermore, the vertices of E along with the vertices that start in the $\Delta - 1$ cliques at the left of W_e form a well-colored block, which we can integrate

into the buffer. In other word, we "slide" the buffer one clique to the right, and we can proceed with the next step. Note that since at each step one clique leaves the buffer and one clique enters the buffer, the number of cliques, and thus the number of blocks and region within a buffer remains the same all along the algorithm.
As we will see, we need for the regions of the buffer to satisfy a few more properties rather than just being well-colored. In the next subsection we run one step of the algorithm on a simple example and detail some of these properties.

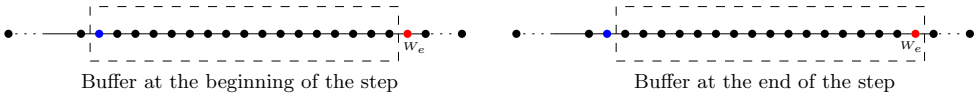


FIGURE 4.5: The clique that belong to the buffer are the cliques in the dashed rectangle. At the end of the step, the clique W_e enters into the buffer, and the leftmost clique (in blue) leaves the buffer. During this step, only vertices that start in a clique of the buffer are recolored.

As a side note, we can already explain why the total number of recoloring made by the algorithm is linear. The buffer slides one clique to the right at each step, and thus there are at most $O(n)$ steps. During one step, only vertices that start in cliques of the buffer are recolored. Furthermore, the buffer is of constant size N (in terms of number of regions), and thus a vertex is in a buffer during a constant number of steps. It follows that each vertex is recolored a constant number of times, and thus that the total number of recolorings performed by the algorithm is linear in n.

4.1.4 Example of recolorings for buffer slides

Let us summarize briefly where we stand before diving into the example. Suppose that we are at the beginning of step t. The leftmost clique of the buffer is the clique W_i for some $i \geq 0$ (as we will see later one it is not necessarily the clique W_{t+1} due to the initialization process). The regions of the buffer are denoted by $R_1, \ldots, R_N$ (and thus the leftmost clique of the block A_1 is the clique W_i). Every region of the buffer is well-colored, and we denote by ν_{A_i}, ν_{B_i} and ν_{C_i} the three color vectors of the region R_i for $1 \leq i \leq N$. As in the previous sections, we denote by E the vertices that start in the first clique W_e at the right of C_N. We have two objectives during this step:

1. Obtain the canonical coloring for the first clique of A_1
2. Recolor vertices of the buffer so that the coloring of C_N fits with the coloring of E.

The first objective is obtained easily: indeed, we always require the first region of the buffer to be colored canonically. In other words, at the beginning of the step we have $\nu_{A_1}(i) = \nu_{B_1}(i) = \nu_{C_1}(i)$ for all $i \leq \omega$. This is one of the properties of the buffer that we will maintain throughout the algorithm:

Property 1. *The region R_1 of the buffer is colored canonically*

Therefore we can focus exclusively on the second objective. Let us go back to the example of Figure 4.4. Here, we have two things to do in order to have colorings that fit. First, we have to recolor the vertices in $(C_N, 1)$ (which current color is 5) with 3 and the vertices in $(C_N, 2)$ (which current color is 3) with 5. In other word, we have to permute the color of the classes 1 and 2. Then, we need to recolor the vertices of $(C_N, 3)$ with color 2.
Note that here, none of these operations can be done without taking some precautions. Indeed, we cannot directly recolor the vertices of $(C_N, 3)$ with color 2, since vertices of $(C_N, 3)$ may have neighbors colored with 2 in the preceding block. Furthermore, permuting the color of the classes 1 and 2 can only be done as follows: recolor vertices of $(C_N, 1)$ with a color $c \notin \{1, 3, 5\}$ and such that vertices of $(C_N, 1)$ have no neighbor colored c. Then recolor vertices of $(C_N, 2)$ with 5 and finally recolor the vertices of $(C_N, 1)$ with 3. However, we do not know if such a color exists. In order to be able to fulfill objective 2, we need to apply more constrains on the coloring of the buffer.
Recall that each block is a separator of G and thus that given three consecutive blocks Q_i, Q_{i+1}, Q_{i+2} we have $N[Q_i] \subseteq Q_i \cup Q_{i+1} \cup Q_{i+2}$ (and in particular, for any region R_i we have $N[B_i] \subseteq A_i \cup B_i \cup C_i = R_i$. Hence, in order to have some freedom when recoloring, we need not too many colors to appear on consecutive blocks. We thus enforce the coloring of the buffer to satisfy the following:

Property 2 (Continuity property)**.** *For every $i \leq N-1$ the colorings of C_i and A_{i+1} are the same, or in other words $\nu_{C_i} = \nu_{A_{i+1}}$*

This property can be intuitively understood as follows: when looking at the buffer from the left to the right, the color of a class can only change in the middle block B of a region. Let us now come back to our example, and consider the coloring of the whole region R_N rather than just the coloring of its rightmost block C_N. We will consider different cases for the coloring of R_N. In order to simplify the notation, we denote the vertices of class a in a set of consecutive blocks $Q_i, Q_{i+1}, \ldots, Q_j$ by $(Q_i, \ldots, Q_j, a)$.

Suppose that the coloring of R_N is the one described in Figure 4.6a. A well-colored region R_i satisfying $\nu_{A_i} = \nu_{B_i} = \nu_{C_i}$ is a *waiting region*. We can first remark here that it is possible to recolor $(B_N, 3)$ and $(C_N, 3)$ with color 2. However, for technical reasons detailed in the full proof, we need the buffer to always satisfy the following:

Property 3. *The last region R_N of the buffer is a waiting region*

Then we cannot recolor B_N and C_N without recoloring A_N, for otherwise we obtain a recoloring that does not satisfy Property 2. Thus we need to look further at the left in the buffer to see which block we can recolor. Let us assume that the region R_{N-1} is also a waiting region, as illustrated in Figure 4.6a. Note that by the continuity property we have $\nu_{C_{N-1}} = \nu_{A_N}$ and since R_{N-1} and R_N are both waiting regions, all there blocks share the same color vector. We can then proceed as follows:

1. Recolor $(B_{N-1}, \ldots, C_N, 3)$ with color 2.

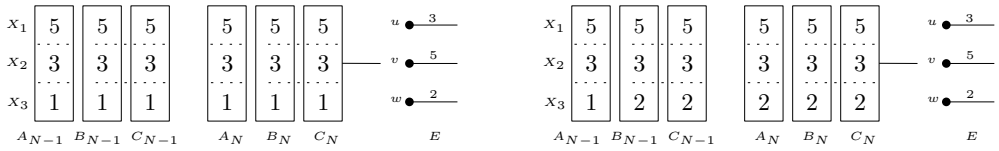


FIGURE 4.6: Coloring of the regions R_{N-1}, R_N and of the vertices in E at the beginning of the step (a) and after recoloring the class 3 (b).

We obtain the coloring illustrated in Figure 4.6b. Let us show that this is a valid recoloring operation. We only have to check that no neighbors of the vertices in $(B_{N-1}, \ldots, C_N, 3)$ are colored with color 2. The blocks B_{N-1} and C_N separates $C_{N-1}, \ldots, B_N$ from the rest of the graph, therefore only vertices in B_{N-1} and C_N can have neighbors not contained in $B_{N-1}, \ldots, C_N$, and $N[B_{N-1}] \subseteq A_{N-1} \cup B_{N-1} \cup C_{N-1}$. Furthermore, E contains a vertex of class 3: it follows that $N[(C_N, 3)] \subseteq B_N \cup C_N \cup E$. In other words, the clique W_e is the rightmost clique of the graph that contains vertices of $(C_N, 3)$. We obtain that $N[(B_{N-1}, \ldots, C_N, 3)] \subseteq A_{N-1} \cup, \ldots, \cup C_N \cup E$. Since no vertex is colored 2 in this set, the recoloring operation is valid.

A region such that the color of exactly one class differs in blocks A and B is a *color region*. In Figure 4.6b, after the recoloring, the region R_{N-1} becomes a color region and the region R_N remains a waiting region. Note that after such a recoloring, properties 1 and 2 of the buffer remain satisfied. The operation consisting into changing the color of unique class over a set of consecutive block is called a *color change*.
It still remains to permute the colors of the classes 1 and 2 to obtain a coloring of C_N that fits with the coloring of E. Assume further that R_{N-2} is also a waiting region. Due to the continuity property, we have $\nu_{C_{N-2}} = \nu_{A_{N-1}}$, and thus the coloring of R_{N-2} is completely determined by $\nu_{A_{N-1}}$. The coloring or R_{N-2}, R_{N-1} and R_N is illustrated in Figure 4.7a. Recall that we consider $d + 4 = 6$-colorings of G, and note that colors 4 and 6 are not used in the regions R_{N-2}, R_{N-1} and R_N. In order to permute the two colors we can then proceed as follows:

1. Recolor $(B_{N-2}, \ldots, C_N, 1)$ with color 6.
2. Recolor $(B_{N-2}, C_{N-2}, A_{N-1}, 2)$ with color 4.
3. Recolor $(C_{N-2}, \ldots, C_N, 2)$ with color 5.
4. Recolor $(C_{N-2}, \ldots, C_N, 1)$ with color 3.

These four recoloring steps are illustrated in Figures 4.7b, 4.7c and 4.7d. Let us show that these four recoloring steps are valid operations. As before, E contains vertices of class 1 and 2 and thus the closed neighborhoods of $(B_{N-2}, \ldots, C_N, 1)$ and $(B_{N-2}, \ldots, C_N, 2)$ are contained in $A_{N-2} \cup \ldots \cup C_N \cup E$. Since none of these vertices are colored with either 4 or 6, the steps 1 and 2 are valid operations. Let us now consider step 3: First note that the closed neighborhood of $(C_{N-2}, \ldots, C_N, 2)$ is contained in $B_{N-2} \cup \ldots \cup C_N \cup E$. After the step 2, no vertex in $B_{N-2} \cup \ldots \cup C_N$ is

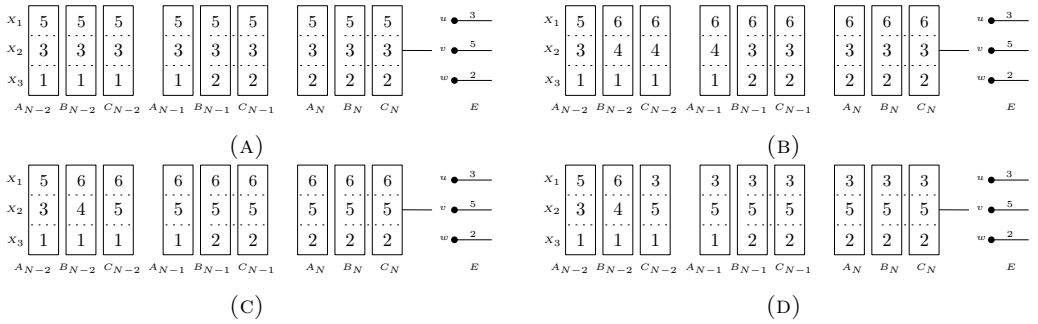


FIGURE 4.7: Coloring of the regions R_{N-2}, R_{N-1} and R_N and of the vertices in E before the first recoloring operation (a) after operations 1) and 2) (b) after recoloring 3) (c) and after recoloring 4) (d).

colored 5. Furthermore, the only vertex in E colored with 5 belongs to class 2. Since a class is an independent set, recoloring step 3 is valid. The same reasoning applies for recoloring step 4 since after step 1, 2 and 3, no vertex in $B_{N-2} \cup \ldots \cup C_N$ is colored 3 and the only vertex colored 3 in E belongs to class 1.

A region such that the color of exactly two classes is permuted in block A and C is a *transposition region*. In Figure 4.7d, after these recolorings, the region R_{N-2} is a transposition region, the region R_{N-1} is a color region and the region R_N is a waiting region. Furthermore, the buffer properties remain satisfied. The operation consisting in permuting the color of two classes is called a *transposition*.
Let us outline the fact that all along the algorithm, every region of the buffer is either a waiting region, a color region, or a transposition region.

Property 4. *Every region of a buffer is either a waiting region, a color region, or a transposition region*

After applying one color change and one transposition, we obtain a coloring of C_N that fits with the coloring of E. Let us consider the buffer at the next step. We denote its regions by $R'_1, \ldots R'_N$ and the block of the region R'_j by A'_j, B'_j and C'_j for j in $1, \ldots, N$. Since the leftmost clique of the previous buffer is W_i, the leftmost clique of the new buffer is W_{i+1}. It follows that for every j the leftmost clique of B_j belongs to A'_j, and the leftmost clique of C_j belongs to B'_j. Hence we may need to modify the coloring of vertices starting in these cliques so that the regions of the new buffer remain well-colored and satisfy properties 1 to 4 mentioned above.

By property 4 there are three cases to consider. If the region R_j is a waiting region, then no recoloring has to be made to keep R'_j well-colored. Indeed, in this case we have $\nu_{A_j} = \nu_{B_j} = \nu_{C_j}$ and therefore vertices starting in the first clique of B_j and vertices starting in the first clique of C_j already have a coloring such that R'_j is well-colored. Suppose now that R_j is a color region. Then there exists a unique p such that $\nu_{A_j}(p) \neq \nu_{B_j}(p)$ and $\nu_{B_j} = \nu_{C_j}$. This later equality ensures that since the leftmost clique of C_j belongs to B'_j, no vertex has to be recolored in this clique. Consider then the leftmost clique of B_j, which belongs to A'_j, but which is currently

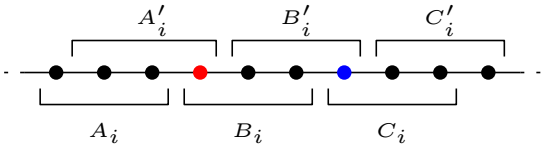


FIGURE 4.8: The three blocks A_i, B_i, C_i of the i-th region of the buffer at the beginning of step t, and the three blocks A'_i, B'_i and C'_i of the i-th region of the buffer at the beginning of step $t+1$. The vertices starting in the blue cliques and red cliques may need to be recolored before starting the step $t+1$ to maintain well-colored regions.

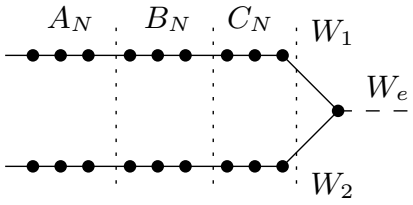


FIGURE 4.9: Example of the region R_N of a buffer in the case $\Delta = 3$ and where the clique W_e has degree 3 in the tree decomposition. The blocks are separated by dotted lines.

colored according to the color vector ν_{B_j}. We claim that recoloring the vertices of class p starting in the leftmost clique of B_j with color $\nu_{A_j}(p)$ is a valid operation. Indeed, A_j and C_j separate B_j from the rest of the graph and the only vertices to be colored with $\nu_{A_j}(p)$ in R_j belongs to class p. Their only remains to check the case where R_j is a transposition region. For the sake of keeping this example simple we postpone the proof of this case to the next section.

4.1.5 From interval graphs to chordal graphs

Throughout this section we gave some insights about how the recoloring algorithm works in the particular case where the input graph G is an interval graph. Let us briefly explain here how we can generalize this algorithm for chordal graphs, where the clique-tree of G is not restricted to be a path anymore. We first root the tree at an arbitrary clique r and start the algorithm at the leaves. Let W_ℓ be any leaf of the tree, and let W_e be its first ancestor of degree strictly more than 2 in the path from ℓ to r. The tree induced by the cliques on the path from W_ℓ to W_e is a path, therefore the graph induced by the vertices that belongs to these cliques is an interval graph. We can then apply the algorithm described previously up until the rightmost clique of the buffer is a child of W_e.

Suppose that W_e has i children $W_1, \ldots, W_i$. We then obtain i buffers, each of which contains W_i as its rightmost clique. In order to continue, we merge the i buffers into one as illustrated in Figure 4.9. To do so, we extend the notion of blocks and regions to clique-trees: a vertex *starts at height* i if the clique with the largest height containing this vertex has height i. Blocks are now constituted of vertices starting at the same height, and we can consider regions as sets of consecutive blocks, as previously. The formal definitions are given in the full proof of this algorithm.

Note however that in order for the merged buffer to be well-colored, we need all vertices that belong to the same class and start at the same height to have the same color in each buffer. We therefore need to add a recoloring step to generalize the algorithm to chordal graphs, in order to ensure that this property remains satisfied. This recoloring process is formally described in Section 4.6.

4.2 The algorithm on chordal graphs

Throughout this section, $G = (V, E)$ is a chordal graph on n vertices of maximum clique number ω and maximum degree Δ. Let $k \geq \omega + 3$ be the number of colors denoted by $1, \ldots, k$. Given two integers $x \leq y$, $[\![x, y]\!]$ is the set $\{x, x+1, \ldots, y\}$. The *closed neighbourhood* of a set $S \subseteq V$ is $N[S] := S \cup (\cup_{v \in S} N(v))$.

4.2.1 Chordal graphs and clique trees

Vertex ordering and canonical coloring. Let $v_1, v_2, \ldots v_n$ be a perfect elimination ordering of V. A greedy coloring of $v_n, v_{n-1}, \ldots, v_1$ gives an optimal coloring c_0 of G using only ω colors. The coloring c_0 is called the *canonical coloring of* G. The colors $c \in 1, 2 \ldots, \omega$ are called the *canonical colors* and the colors $c > \omega$ are called the *non-canonical colors*. Note that the independent sets $X_i := \{v \in V \text{ such that } c_0(v) = i\}$ for $i \leq \omega$, called the *classes* of G, partition the vertex set V.

Clique tree. Throughout this section, $T = (V_T, E_T)$ is a clique tree of G. We assume that T is rooted on an arbitrary node. Given a rooted tree T and a node W of T, the *height* of W denoted by $h(W)$ is the length of the path from the root to W. A clique-tree of G can be found in linear time [48].

Observation 2. *Let G be a chordal graph of maximum degree Δ and T be a clique tree of G rooted in an arbitrary node. Let x be a vertex of G and W_i, W_j be two bags of T that contain x. Then $h(W_j) - h(W_i) \leq \Delta$.*

Proof. We can assume without loss of generality (free to replace the one with the smallest height by the first common ancestor of W_i and W_j) that W_i is an ancestor of W_j (indeed, this operation can only increase the difference of height). Let P be the path of T between W_i and W_j and (U, W) be an edge of P with $h(U) < h(W)$. By assumption on the clique tree, there is a vertex y that appears in W and that does not appear in U. Since this property is true for every edge of T and since all the bags of P induce cliques and contain x, the vertex x has at least $|P|$ neighbors. □

4.2.2 Buffer, blocks and regions

Let W_e be a clique of T. We denote by T_{W_e} the subtree of T rooted in W_e and by $h_{W_e}(W)$ the height of the clique $W \in T_{W_e}$. Given a vertex $v \in T_{W_e}$, we say that v *starts* at height h if the maximum height of a clique of T_{W_e} containing v is h (in T_{W_e}).

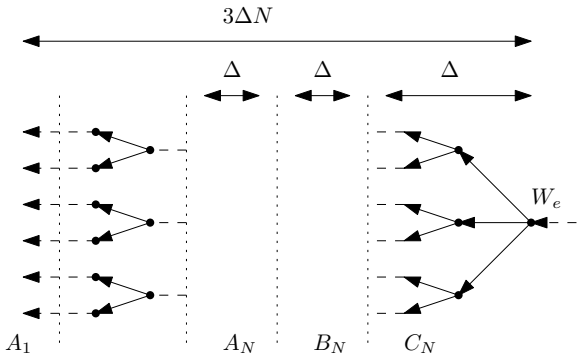


FIGURE 4.10: The buffer $\mathcal{B}$ rooted at W_e. The dots represent cliques of T and the dashed lines separates the blocks of $\mathcal{B}$.

Let $s := 3\binom{\omega}{2} + 2$ and $N = s + k - \omega + 1$ where k is the number of colors. The *buffer $\mathcal{B}$ rooted in W_e* is the set of vertices of G that start at height at most $3\Delta N - 1$ in T_{W_e}. For every $0 \leq i \leq 3N - 1$, the *block* Q_{3N-i} of $\mathcal{B}$ is the set of vertices of G that start at height h with $i\Delta \leq h \leq (i+1)\Delta - 1$. Finally, for $0 \leq i \leq N - 1$, the *region* R_i of $\mathcal{B}$ is the set of blocks $Q_{3i+1}, Q_{3i+2}, Q_{3(i+1)}$. Unless stated otherwise, we will always denote the three blocks of R_i by A_i, B_i and C_i, and the regions of a buffer $\mathcal{B}$ by $R_1, \ldots, R_N$. Given a color class X_p and $S \subseteq V$, we denote by $N[S, p]$ the set $N[S \cap X_p]$. By definition of a block and Observation 2 we have:

Observation 3. *Let W_e be a clique of T and $\mathcal{B}$ be the buffer rooted in T_{W_e}. Let Q_{i-1}, Q_i, Q_{i+1} be three consecutive blocks of $\mathcal{B}$. Then $N[Q_i] \subseteq Q_{i-1} \cup Q_i \cup Q_{i+1}$. In particular for each region $R_i = (A_i, B_i, C_i)$ of $\mathcal{B}$, $N[B_i] \subseteq R_i$.*

Proof. Let v be a vertex of Q_i. By definition of Q_i, v starts at height h with $(3N - i)\Delta \leq h \leq (3N - i + 1)\Delta - 1$. Let u be a neighbour of v. By Observation 2, u starts at height h' with $h - \Delta \leq h' \leq h + \Delta$, thus we have $(3N - i - 1)\Delta \leq h' \leq (3N - i + 2)\Delta - 1$. It follows that u belongs either to Q_{i-1}, Q_i, or Q_{i+1}. □

We refer to this property as the *separation property*. It implies that when recoloring a vertex of Q_i, one only has to show that the coloring induced on Q_{i-1}, Q_i, Q_{i+1} remains proper.

4.2.3 Vectorial coloring

Let $\mathcal{B}$ be a buffer. We denote the set of vertices of class p that belong to the sequence of blocks $Q_i, \ldots, Q_j$ of $\mathcal{B}$ by $(Q_i, \ldots, Q_j, p)$. A *color vector* ν is a vector of size ω such that $\nu(p) \in [\![1, k]\!]$ for every $p \in [\![1, \omega]\!]$, and $\nu(p) \neq \nu(q)$ for every $p \neq q \leq \omega$. A block Q is *well-colored for a color vector ν_Q* if all the vertices of (Q, p) are colored with $\nu_Q(p)$. It does not imply that all the colors are $\leq \omega$ but just that all the vertices of a same class have the same color (and vertices of different classes have different colors). For brevity, we say that (Q, ν_Q) is *well-colored* and when ν_Q is clear from context we just say that Q is *well-colored*. In particular, a well-colored block is properly colored. Since the set (Q, p) may be empty, a block may be well-colored for different vectors.

However, a color vector defines a unique coloring of the vertices of a block (if (Q, ν_Q) is well-colored then every vertex of (Q, p) has to be colored with $\nu_Q(p)$). The color vector ν is *canonical* if $\nu(p) = p$ for every $p \leq \omega$. A sequence of blocks $Q_1, \ldots, Q_r$ is *well-colored for* $(\nu_1, \ldots, \nu_r)$ if (Q_i, ν_i) is well-colored for every $i \leq r$.

Definition 1 (Waiting regions)**.** *A region R well-colored for vectors ν_A, ν_B, ν_C is a waiting region if $\nu_A = \nu_B = \nu_C$.*

Definition 2 (Color region)**.** *A region R well-colored for vectors ν_A, ν_B, ν_C is a color region if there exist a canonical color c_1, a non-canonical color z and a class p such that:*

1. *$\nu_A(m) = \nu_B(m) = \nu_C(m) \notin \{c_1, z\}$ for every $m \neq p$.*
2. *$\nu_A(p) = c_1$ and $\nu_B(p) = \nu_C(p) = z$.*

In other words, the color of exactly one class is modified from a canonical color to a non-canonical color between blocks A and C. We say that the color c_1 *disappears* in R and that the color z *appears* in R. For brevity we say that R is a color region for the class X_p and colors c_1, z.

Definition 3 (Transposition region)**.** *A region R well-colored for vectors ν_A, ν_B, ν_C is a transposition region if there exist two canonical colors $c_1 \neq c_2$, two non-canonical colors $z \neq z'$ and two distinct classes p, q such that:*

1. *$\nu_A(m) = \nu_B(m) = \nu_C(m) \notin \{c_1, c_2, z, z'\}$ and is canonical for every $m \notin \{p, q\}$.*
2. *$\nu_A(p) = c_1$, $\nu_B(p) = z$, $\nu_C(p) = c_2$.*
3. *$\nu_A(q) = c_2$, $\nu_B(q) = z'$, $\nu_C(q) = c_1$.*

Note that ν_A and ν_C only differ on the coordinates p and q which have been permuted. The colors z and z' are called the *temporary colors* of R. Note that, the coloring induced on R is proper since the separation property ensures that $N[A \cap R] \subseteq A \cup B$, $N[C \cap R] \subseteq B \cap C$ and no class in B is colored with c_1 nor c_2.

Let ν be a color vector. The color vector ν' is obtained from ν by *swapping the coordinates* $p, \ell \leq \omega$ if for every $m \notin \{p, l\}$, $\nu'(m) = \nu(m)$, $\nu'(p) = \nu(\ell)$, and $\nu'(\ell) = \nu(p)$. In other words, ν' is the vector obtained from ν by permuting the coordinates p and ℓ.
Swapping the coordinates p and ℓ in a region R well-colored for (ν_A, ν_B, ν_C) means that for every block $Q \in \{A, B, C\}$, we replace ν_Q by the color vector ν'_Q obtained by swapping the coordinates p and ℓ of ν_Q. It does not refer to a reconfiguration operation but just to an operation on the vectors.

Observation 4. *Swapping two coordinates in a waiting (resp. color, resp. transposition) region leaves a waiting (resp. color, resp transposition) region.*

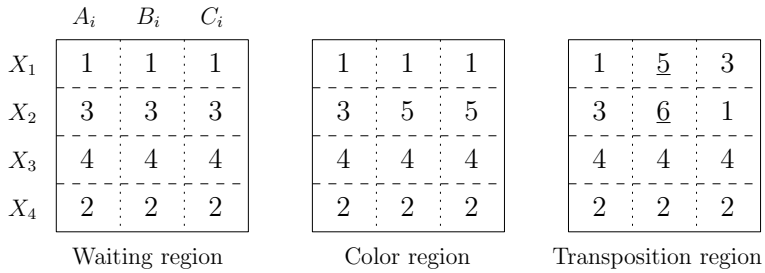


FIGURE 4.11: Example of waiting, color, and transposition regions with $\omega = 4$. Each square represents a region, the dotted lines separate the blocks and the dashed lines separate the classes. The colors 1 to 4 are the canonical colors and the colors 5 and 6 are non-canonical colors. The underlined colors in the transposition region indicate the temporary colors.

Using the following lemma, we can assume that all the transposition regions use the same temporary colors.

Lemma 3. *Let R be a transposition region with temporary colors z, z'. Let $z'' \notin \{z, z'\}$ be a non-canonical color. By recoloring the vertices of R at most once, we can assume the temporary colors are z, z''.*

Proof. Let p and q be the coordinates which are permuted in R. By definition of transposition regions, no vertex of R is colored with z''. As any class is an independent set and by the separation property, we can recolor (B, q) with z'' to obtain the desired coloring of R. □

4.2.4 Valid and almost valid buffers

In what follows, a bold symbol $\boldsymbol{\nu}$ always denote a tuple of vectors and a normal symbol ν always denotes a vector. Let $\mathcal{B} = R_1, R_2, \ldots, R_N$ be a buffer such that all the regions $R_i = A_i, B_i, C_i$ are well-colored for the vectors $\nu_{A_i}, \nu_{B_i}, \nu_{C_i}$. So $\mathcal{B}$ is well-colored for $\boldsymbol{\nu} = (\nu_{A_1}, \nu_{B_1}, \nu_{C_1}, \nu_{A_2}, \ldots, \nu_{C_N})$. The buffer $(\mathcal{B}, \boldsymbol{\nu})$ is *valid* if:

1. [*Continuity property*] For every $i \in 1, 2, ..., N-1$, $\nu_{C_i} = \nu_{A_{i+1}}$.
2. The vectors ν_{A_1}, ν_{B_1} and ν_{C_1} are canonical (and then R_1 is a waiting region).
3. The regions $R_2, \ldots, R_{s-1}$ define a *transposition buffer*, that is a sequence of consecutive regions that are either waiting regions or transposition regions using the same temporary colors.
4. The regions $R_{s+1}, \ldots, R_{N-1}$ define a *color buffer*, that is a sequence of consecutive regions that are either color regions or waiting regions.
5. The regions R_s and R_N are waiting regions.

Note that Property 1 along with the definition of well-colored regions enforce "continuity" in the coloring of the buffer: the coloring of the last block of R_i and the first block of R_{i+1} in a valid buffer have to be the same.

An *almost valid buffer* $(\mathcal{B}, \boldsymbol{\nu})$ is a buffer that satisfies Properties 1 to 4 of a valid buffer and for which Property 5 is relaxed as follows:

5'. The region R_s is a transposition region or a waiting region. R_N is a waiting region.

Let us make a few observations.

Observation 5. *Let $(\mathcal{B}, \boldsymbol{\nu})$ be an almost valid buffer. For every $i \leq s$, the color vectors ν_{A_i} and ν_{C_i} are permutations of the canonical colors.*

Proof. By induction on i. By Property 2 of almost valid buffers, it is true for every $i \leq r$. Suppose now that the property is verified for R_i with $2 < i < s$. By assumption ν_{C_i} is a permutation of $[\![1, \omega]\!]$ and the continuity property (Property 1 of almost valid buffer) ensures that $\nu_{A_{i+1}} = \nu_{C_i}$. By Property 3 we only have two cases to consider, either R_{i+1} is a waiting region and by definition $\nu_{c_{i+1}} = \nu_{A_{i+1}}$, or R_{i+1} is a transposition region. In the later case, by definition of a transposition region, $\nu_{C_{i+1}}$ is equal to $\nu_{A_{i+1}}$ up to a transposition of some classes k, ℓ and thus is a permutation of the canonical colors. □

Observation 6. *Let $(\mathcal{B}, \boldsymbol{\nu})$ be an almost valid buffer and c be a non-canonical color. There exists a unique class $p \leq \omega$ such that $\nu_{C_s}(p) = c$. Furthermore, either the class p is colored with c on all the blocks of $R_{s+1}, \ldots, R_N$, or the color c disappears in a color region for the class p.*

Proof. By Observation 5, ν_{C_s} is a permutation of the canonical colors. Thus there exists a unique class $p \leq \omega$ such that $\nu_{C_s}(p) = c$. Furthermore, by Property 4 of almost valid buffer, the regions $R_{s+1}, \ldots, R_N$ are either waiting or color regions. The continuity property then ensures that either the class p is colored with c on $R_{s+1}, \ldots, R_N$ or that the color c disappears in a color region if there exists a color region for the class p. □

Since only non-canonical colors can appear in a color region, we have the following observation:

Observation 7. *Let $(\mathcal{B}, \boldsymbol{\nu})$ be an almost valid buffer and z be a non-canonical color. Either no vertex of $R_{s+1}, \ldots, R_N$ is colored with z, or there exists a color region R_i with $s < i < N$ for the class p in which z appears. In the latter case, the vertices of the color buffer of $\mathcal{B}$ colored with z are exactly the vertices of $(B_i, C_i, \ldots, C_N, p)$.*

Finally, since the number of regions in the color buffer is the number of non-canonical colors, we have:

Observation 8. *Let $(\mathcal{B}, \boldsymbol{\nu})$ be an almost valid buffer. There exists a waiting region in $R_{s+1}, \ldots, R_{N-1}$ if and only if there exists a non-canonical color that does not appear in $R_{s+1}, \ldots, R_{N-1}$.*

4.2.5 Vectorial recoloring

Let $(\mathcal{B}, \boldsymbol{\nu})$ be a buffer. The tuple of color vectors $\boldsymbol{\nu} = (\nu_{Q_1}, \dots, \nu_{Q_{3\Delta N}})$ is a *(proper) vectorial coloring* of $\mathcal{B}$ if for every color c and every $i \leq 3\Delta N - 1$ such that c is in both ν_{Q_i} and $\nu_{Q_{i+1}}$, then there exists a unique class $p \leq \omega$ such that $\nu_{Q_i}(p) = \nu_{Q_{i+1}}(p) = c$.

Observation 9. *Any proper vectorial coloring $(\mathcal{B}, \boldsymbol{\nu})$ induces a proper coloring of $G[\mathcal{B}]$.*

Proof. Indeed, two different classes in two consecutive blocks cannot have the same color in a proper vectorial coloring. Since for any block Q_i of $\mathcal{B}$ and for any class p, $N[Q_i, p] \subseteq Q_{i-1} \cup Q_i \cup Q_{i+1}$, the coloring induced on $G[\mathcal{B}]$ is proper. □

Note that if $(\mathcal{B}, \boldsymbol{\nu})$ is an almost valid buffer then $\boldsymbol{\nu}$ is a proper vectorial coloring of $\mathcal{B}$ by the continuity property and the definition of waiting, color, and transposition regions. Since we will only consider proper vectorial colorings, we will omit the term proper for brevity.

Let ν_Q be a color vector. A color vector ν'_Q is *adjacent to* ν_Q if there exists a class p and a color $c \notin \nu_Q$ such that $\nu'_Q(p) = c$ and $\nu'_Q(m) = \nu_Q(m)$ for every $m \neq p$.

Observation 10. *Let Q be a block well-colored for ν_Q and let ν'_Q be a color vector adjacent to ν_Q such that $\nu'_Q(p) = c \neq \nu_Q(p)$. Then recoloring the vertices of (Q, p) one by one is a proper sequence of recolorings of $G[Q]$ after which Q is well-colored for ν'_Q.*

Let $(\boldsymbol{\nu}, \boldsymbol{\nu}')$ be two vectorial colorings of a buffer $\mathcal{B}$. The coloring $\boldsymbol{\nu}'$ is a *vectorial recoloring* of $\boldsymbol{\nu}$ if there exists a unique $i \in [\![1, 3\Delta N]\!]$ such that ν'_{Q_i} is adjacent to ν_{Q_i} and $\nu'_{Q_j} = \nu_{Q_j}$ for $j \neq i$. By Observation 10, we have:

Observation 11. *Let $t \geq 1$ and $(\boldsymbol{\nu}^1, \boldsymbol{\nu}^t)$ be two (proper) vectorial colorings of a buffer $\mathcal{B}$. If there exists a sequence of adjacent (proper) vectorial recolorings $\boldsymbol{\nu}^1, \boldsymbol{\nu}^2, \dots \boldsymbol{\nu}^t$, then there exists a sequence of (proper) single vertex recolorings of $G[\mathcal{B}]$ after which the coloring of $\mathcal{B}$ is well-colored for $\boldsymbol{\nu}^t$.*

Given a sequence of vectorial recolorings $\boldsymbol{\nu}^1, \boldsymbol{\nu}^2, \dots \boldsymbol{\nu}^t$, we say that each coordinate is recolored at most ℓ times if for every coordinate $p \leq \omega$ and every $r \in [\![1, 3\Delta N]\!]$, there exist at most ℓ indices $t_1, \dots, t_\ell$ such that the unique difference between $\boldsymbol{\nu}^{t_i}$ and $\boldsymbol{\nu}^{t_{i+1}}$ is the p-th coordinate of the r-th vector of the tuples.

4.3 Overview of the four steps

Let G be a chordal graph of maximum degree Δ and maximum clique size ω, T be a clique tree of G, and ϕ be any k-coloring of G. We propose an iterative algorithm that recolors the vertices of the bags of T from the leaves to the root until we obtain the canonical coloring defined in Section 4.2.1. Let S be a clique of T. A coloring α of G is *treated up to* S if:

1. Vertices starting at height more than $3\Delta N$ in T_S are colored canonically, and
2. The buffer rooted at S is valid.

Let W be a clique of T. We associate a vector ν_C of length ω to the clique W as follows. We set $\nu_W(\ell) = \alpha(v)$ if there exists $v \in X_\ell \cap W$. Then we arbitrarily complete ν_W in such a way all the coordinates of ν_W are distinct (which is possible since $|\nu_W| < k$).

Given two vectors ν and ν' the *difference* $D(\nu, \nu')$ between ν and ν' is $|\{p : \nu(p) \neq \nu'(p)\}|$, i.e. the number of coordinates on which ν and ν' differ. Given an almost valid buffer $(\mathcal{B}, \boldsymbol{\nu})$ and a vector ν_C, the *border error* $D_{\mathcal{B}}(\nu_C, \boldsymbol{\nu})$ is $D(\nu_{C_N}, \nu_C)$.

Let $\mathcal{B}$ be a buffer. The class $p \leq \omega$ is *internal* to $\mathcal{B}$ if $N[R_N, p] \subseteq R_{N-1} \cup R_N$.

We first state the main technical lemmas with their proof outlines and finally explain how we can use them to derive Theorem 4.

Lemma 4. *Let W be a clique associated with ν_W. Let S be a child of W, $\mathcal{B}$ be the buffer rooted at S and $\boldsymbol{\nu}$ be a tuple of vectors such that $(\mathcal{B}, \boldsymbol{\nu})$ is valid. If $D_{\mathcal{B}}(\nu_W, \boldsymbol{\nu}) > 0$, then there exists a recoloring sequence of $\cup_{i=s}^{N} R_i$ such that the resulting coloring $\boldsymbol{\nu}'$ satisfies $D_{\mathcal{B}}(\nu_W, \boldsymbol{\nu}') < D_{\mathcal{B}}(\nu_W, \boldsymbol{\nu})$, and $(\mathcal{B}, \boldsymbol{\nu}')$ is almost valid.*
Moreover, every coordinate of $\cup_{i=s}^{N} R_i$ is recolored at most 3 times and only internal classes are recolored.

Outline of the proof. Let ℓ be a class on which ν_W and ν_{C_N} are distinct. Then, in particular, no vertex of X_ℓ is in $C_N \cap W$ thus the class ℓ is internal. Given an internal class ℓ, if we modify $\nu_{C_N}(\ell)$ and maintain a proper vectorial coloring of the buffer $\mathcal{B}$, then the corresponding recoloring of the graph is proper. So, if we only recolor internal classes of R_N, then we simply have to check that the vectorial coloring of $\mathcal{B}$ remains proper. The proof is then based on a case study depending on whether $\nu_W(\ell)$ is canonical or not. A complete proof is given in Section 4.4. □

Lemma 5. *Let $(\mathcal{B}, \boldsymbol{\nu})$ be an almost valid buffer. There exists a recoloring sequence of $\cup_{i=2}^{s} R_i$ such that every coordinate is recolored at most 6 times and the resulting coloring $\boldsymbol{\nu}'$ is such that $(\mathcal{B}, \boldsymbol{\nu}')$ is valid.*

Outline of the proof. The proof distinguishes two cases: either there exists a waiting region in the transposition buffer or not. In the first case, we show that we can "slide" the waiting regions to the right of the transposition buffer and then ensure that R_s is a waiting region. Otherwise, because of the size of the transposition buffer, then some pair of colors has to be permuted twice. In this case, we show that these two transpositions can be replaced by waiting regions (and we can apply the first case). A complete proof is given in Section 4.5. □

Note that given a clique W and its associated vector ν_W, applying Lemma 5 to an almost valid buffer $(\mathcal{B}, \boldsymbol{\nu})$ rooted at a child S of W does not modify $D_{\mathcal{B}}(\nu_W, \boldsymbol{\nu})$ since the region R_N is not recolored.

Let W be a clique and S_1, S_2 be two children of W. For every $i \leq 2$, let $\mathcal{B}_i$ be the buffer of S_i and assume that $\mathcal{B}_i$ is valid for $\boldsymbol{\nu}^i$. We say that $\mathcal{B}_1$ and $\mathcal{B}_2$ have *the same coloring* if $\boldsymbol{\nu}^1 = \boldsymbol{\nu}^2$.

Lemma 6. *Let W be a clique associated with ν_W. Let $S_1, S_2, \dots S_e$ be the children of W, and for every $i \leq e$, $\mathcal{B}_i$ be the buffer rooted at S_i. Let $\boldsymbol{\nu}^i$ be a vectorial coloring such that $(\mathcal{B}_i, \boldsymbol{\nu}^i)$ is valid. If $D_{\mathcal{B}_i}(\nu_W, \boldsymbol{\nu}^i) = 0$ for every $i \leq e$, then there exists a recoloring sequence of $\cup_{j=2}^{N-1} R_j^i$ such that every coordinate is recolored $O(\omega^2)$ times, the final coloring of all the $\mathcal{B}_i$s is the same coloring $\boldsymbol{\nu}'$, $D_{\mathcal{B}_i}(\nu_W, \boldsymbol{\nu}') = 0$, and $(\mathcal{B}_i, \boldsymbol{\nu}')$ is valid for every $i \leq e$.*

Outline of the proof. First, we prove that it is possible to transform the coloring of $\mathcal{B}_i$ in such a way that all the color buffers have the same coloring, and that $\nu_s^1 = \nu_s^i$ for $i \in [\![2, e]\!]$.
We then have to ensure that the vectors of the transposition buffers are the same, which is more complicated. Indeed, even if we know that the vectors ν_s^1 and ν_s^i are the same, we are not sure that we use the same sequence of transpositions in the transposition buffers of $\mathcal{B}_1$ and $\mathcal{B}_i$ to obtain it. Let $\tau_1, \dots, \tau_r$ be the set of transpositions of $\mathcal{B}_1$. The proof consists in showing that we can add to $\mathcal{B}_i$ the transpositions $\tau_1, \dots, \tau_r, \tau_r^{-1}, \dots, \tau_1^{-1}$ at the beginning of the transposition buffer. It does not modify ν_{A_s} since this sequence of transpositions gives the identity. Finally, we prove that $\tau_r^{-1}, \dots, \tau_1^{-1}$ can be cancelled with the already existing transpositions of $\mathcal{B}_i$. And then the transposition buffer of $\mathcal{B}_i$ only consists of $\tau_1, \dots, \tau_r$. A complete proof is given in Section 4.6. □

Lemma 7. *Let W be a clique of T with children $S_1, S_2, \dots S_e$ and let α be a k-coloring of G treated up to S_i for every $i \in [\![1, e]\!]$. Let ν_W be a vector associated with W and $\mathcal{B}_i = R_1^i, \dots, R_N^i$ denote the buffer rooted at S_i. Assume that there exists $\boldsymbol{\nu}$ such that $(\mathcal{B}_i, \boldsymbol{\nu})$ is valid and satisfies $D_{\mathcal{B}_i}(\nu_W, \boldsymbol{\nu}) = 0$ for every $i \leq e$. Then there exists a recoloring sequence of $\cup_{j=2}^{N-1} R_j^i$ such that, for every $i \leq e$, every vertex of $\mathcal{B}_i$ is recolored at most one time and such that the resulting coloring of G is treated up to C. A complete proof is given in Section 4.7.*

Outline of the proof. This proof "only" consists in shifting the buffer of one level. We simply recolor the vertices that now start in another region (of the buffer rooted at W) with their new color. We prove that the recoloring algorithm cannot create any conflict. A complete proof of the lemma is given in Section 4.7. □

Given Lemmas 4, 5, 6 and 7 we can prove our main result:

Theorem 21. *Let Δ be a fixed constant. Let $G(V, E)$ be a d-degenerate chordal graph of maximum degree Δ and ϕ be any k-coloring of G with $k \geq d + 4$. Then we can recolor ϕ into the canonical coloring c_0 in at most $O(d^4\Delta \cdot n)$ steps. Moreover the recoloring algorithm runs in linear time.*

Proof. Let c_0 be the canonical coloring of G as defined in Section 4.2.1, and T be a clique tree of G. Let us first show that given a clique $W \in T$ with children $S_1, \dots, S_e$ and a coloring α treated up to S_i for every $i \leq e$, we can obtain a coloring of G treated up to W. Let ν_W be a vector associated with W. For every $i \leq e$, let $\mathcal{B}_i$ be the buffer

rooted in S_i and $\boldsymbol{\nu}_i$ be a vectorial coloring of $\mathcal{B}_i$ such that $(\mathcal{B}_i, \boldsymbol{\nu}^i)$ is valid. For every $i \leq e$, by applying Lemmas 4 and 5 at most $D_{\mathcal{B}_i}(\nu_W, \boldsymbol{\nu}^i)$ times to $(\mathcal{B}_i, \boldsymbol{\nu}^i)$, we obtain a vectorial coloring $\boldsymbol{\nu}^i$ such that $(\mathcal{B}_i, \boldsymbol{\nu}^i)$ is valid and $D_{\mathcal{B}_i}(\nu_W, \boldsymbol{\nu}^i) = 0$. By Lemma 6, we can recolor each $\boldsymbol{\nu}^i$ into $\boldsymbol{\nu}'$ such that for every i, $(\mathcal{B}_i, \boldsymbol{\nu}')$ is valid and $D_{\mathcal{B}_i}(\nu_W, \boldsymbol{\nu}') = 0$. Then we can apply Lemma 7 to obtain a coloring of G such that the buffer $(\mathcal{B}, \boldsymbol{\nu})$ rooted in W is valid. Since no vertex starting in cliques $U \in T_W$ with $h_W(U) > 3\Delta N$ is recolored, these vertices remain canonically colored and the resulting coloring of G is treated up to W. Note that only vertices of T_W that start in cliques of height at most $3\Delta N$ are recolored at most $O(\omega^2)$ times to obtain a coloring treated up to W.

Let us now describe the recoloring algorithm and analyze its running time. We root T at an arbitrary node W_r and orient the tree from the root to the leaves. We then do a breadth-first-search starting at W_r and store the height of each node in a table h such that $h[i]$ contains all the nodes of T of height i. Let i_h be the height of T. We apply Lemmas 4 to 7 to every $W \in h[i]$ for i from i_h to 0. Let us show that after step i, the coloring of G is treated up to W for every $W \in h[i]$. It is true for $i = i_h$ since for any $W \in h[i_h]$ the sub-tree T_W of T only contains W. Suppose it is true for some $i > 0$ and let $W \in h[i-1]$. Let $S_1, \ldots, S_e$ be the children of W. For all $j \in 1, \ldots, e$, $S_j \in h[i]$ and by assumption the current coloring is treated up to S_j after step i. Thus we can apply Lemmas 4 to 7 to W. After iteration i_h we obtain a coloring of G that is treated up to W_r. Up to adding "artificial" vertices to G, we can assume that W_r is the only clique of T adjacent to a clique path of length $3\Delta N$ (in fact we only need a tuple of $3N$ color vectors) in T and apply Lemmas 4 to 7 until we obtain a coloring such that W_r is canonically colored, and the algorithm terminates. A clique tree of G can be computed in linear time [48], as well as building the table h via a breadth-first-search. Given a clique W, we can access to the cliques of the buffer rooted at T_W in constant time by computing their height and using the table h. Furthermore, a vertex of height i is recolored during the iterations $i+1, \ldots, i+3\Delta N$ only. As each vertex is recolored at most $O(\omega^2)$ times at each iteration, it follows that the algorithm runs in linear time. Finally, as $N = 3\binom{\omega}{2} + k - \omega + 3$, each vertex is recolored at most $O(\omega^4\Delta)$ times, and thus the algorithm recolors ϕ to c_0 in at most $O(\omega^4\Delta \cdot n)$ steps. □

The proof of Theorem 4 immediately follows:

Proof of Theorem 4. Let ϕ and ψ be two k-colorings of G with $k \geq d+4$ and let c_0 be the canonical coloring of G defined in Section 4.2.1. By Theorem 21, there exists a recoloring sequence from ϕ (resp. ψ) to c_0 of length $O((d+1)^4\Delta \cdot n)$. Thus there exists a sequence of length $O(n)$ that recolors ϕ to ψ. Furthermore the recoloring sequences from ϕ to c_0 and from ψ to c_0 can be found in linear time by Theorem 21, which concludes the proof. □

4.4 Step 1: proof of Lemma 4

Recall that in a buffer $\mathcal{B}$, R_s is the region that sits between the transposition buffer and the color buffer. Before proving Lemma 4, let us first start with some observations.

Observation 12. *Let R be a well-colored region that does not contain color c. If we set $\nu_Q(p) = c$ for every block Q of R, then we obtain a waiting region if R was a waiting region or a color region for the class p, and we obtain a color region if R was a color region for a class $q \neq p$.*

Hence, modifying the color of a class on all the blocks of a region R_j with $j \geq s$ of an almost valid buffer leaves a waiting region or a color region and maintains Property 4 of almost valid buffers. The next lemma ensures that under technical assumptions the continuity property (Property 1 of valid buffers) can also be kept.

Lemma 8. *Let $(\mathcal{B}, \boldsymbol{\nu})$ be an almost valid buffer and R_i, R_j be two regions of $\mathcal{B}$ with $1 \leq i \leq j$. Let X be a block in $\{B_i, C_i\}$ and let $Y = B_j$ or $Y = C_N$. Then recoloring $(X, \dots, Y, p)$ with c preserves the continuity property.*

Proof. Since $(\mathcal{B}, \boldsymbol{\nu})$ is almost valid, for every $p \in [\![1, \omega]\!]$ and every $t \in [\![1, N-1]\!]$, we have $\nu_{C_t}(p) = \nu_{A_{t+1}}(p)$. As the sequence of recolored blocks starts in $\{B_i, C_i\}$ and ends either in B_j or in C_N, after the recoloring we still have $\nu_{C_t}(p) = \nu_{A_{t+1}}(p)$ for every $t \in [\![1, N-1]\!]$ and $p \in [\![1, \omega]\!]$ since both do not change or both are c. □

Given a color c, a class p, and a sequence of consecutive blocks $(Q_i, \dots, Q_j)$, we say that $(Q_i, \dots, Q_j, p)$ is *c-free* if no vertex of $N[\cup_{t=i}^{j} Q_t \cap X_p]$ is colored with c. Note that a sequence of (proper) vectorial recolorings of a buffer $(\mathcal{B}, \boldsymbol{\nu})$ that does not recolor A_1 and such that all the classes recolored on C_N are internal to $\mathcal{B}$ yields a (proper) sequence of single vertex recolorings of G by Observation 11. With the definition of clique-tree we can make the following observation:

Observation 13. *Let W be a clique associated with ν_W, S be a child of W, and $(\mathcal{B}, \boldsymbol{\nu})$ be the buffer rooted at S. If $\nu_{C_N}(\ell) \neq \nu_W(\ell)$ for some $\ell \leq \omega$, then the class ℓ is internal to $\mathcal{B}$.*

Proof. Suppose that the class $\ell \leq \omega$ is not internal to $\mathcal{B}$. Then there exists a vertex $u \in X_\ell \cap R_N$ which has a neighbor v that does not belong to $R_{N-1} \cup R_N$. Thus u and v must be contained in a clique W' that is an ancestor of W and then by the definition of clique tree, u must be contained in W. Then, by definition of ν_W, it must be that $\nu_W(\ell) = \nu_{C_n}(\ell)$. □

We also need the following technical lemma:

Lemma 9. *Let $(\mathcal{B}, \boldsymbol{\nu})$ be an almost valid buffer. Let $s < i < N$, c be a color and p be an internal class. If one of the following holds:*

1. *R_i is a waiting region, c is a non-canonical color that does not appear in $R_s, \dots, R_N$ and the class p is not involved in a color region, or*

2. *R_i is a color region for the class p. Moreover c is non-canonical and does not appear in $R_s, \dots, R_N$, or*

3. *R_i is a color region for the class p where c is the canonical color that disappears.*

Then changing the color of $(B_i, \dots, C_N, p)$ by c also gives an almost valid buffer and a proper coloring of G.

Proof. Let $\boldsymbol{\nu}'$ be the resulting coloring. As the class p is internal and c does not appears in $R_s, \dots R_N$, Case 1 and Case 2 describe a proper recoloring sequence. As $(\mathcal{B}, \boldsymbol{\nu})$ is almost valid, Observation 6 ensures that the only class colored with c in $R_s, \dots, R_N$ in Case 3 is the class p on the set of blocks $(A_s, \dots, A_i)$. Since the class p is internal, case 3 also defines a proper recoloring sequence.

Let us now show that $(\mathcal{B}, \boldsymbol{\nu}')$ is almost valid. First note that no block of $R_1, \dots, R_s$ is recolored thus Properties 2 and 3 of almost valid buffers are still satisfied. The recolored blocks $(B_i, \dots, C_N)$ satisfy the condition of Lemma 8 thus Property 1 also holds. Observation 12 ensures that the regions $R_{i+1}, \dots, R_N$ remain either waiting or color regions and, in particular, that R_N remains a waiting region. Along with the fact that R_s is not modified, Property 5' is satisfied. We finally have to check Property 4. By Observation 12, the regions $R_{i+1}, \dots, R_{N-1}$ remain either waiting or color regions as well as regions $R_{s+1}, \dots, R_{i-1}$ that are not modified. Let us concentrate on R_i. Note that in the three cases, $\nu_Q(m) = \nu'_Q(m)$ for every $m \neq p$ and Q block of R_i, as R_i is either a waiting region or a color region for the class p. We now have to distinguish the three cases:

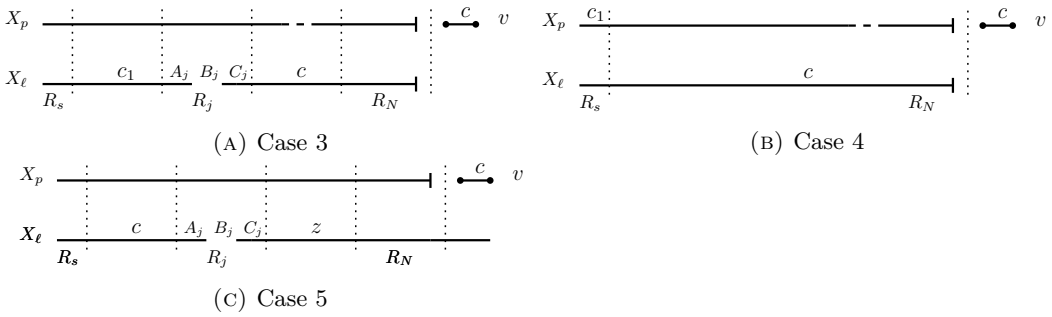
- In Case 1, let $c_1 := \nu_{C_s}(p)$. By definition of almost valid buffers, $\nu_{A_i}(p) = c_1$ and is canonical. After the recoloring, we have $\nu'_{A_i}(p) = c_1$ and $\nu'_{B_i}(p) = \nu'_{C_i}(p) = c$. Thus R_i becomes a color region in $\boldsymbol{\nu}'$ for the class p where c_1 disappears and c appears. Since by hypothesis c does not appear in $R_{s+1}, \dots R_N$ in $\boldsymbol{\nu}$ and since there is no color region for the class p in $\boldsymbol{\nu}$, there exists exactly one color region for the class p and colors c_1, c in $\boldsymbol{\nu}'$ and Property 4 follows.

- In Case 2, let $c' := \nu'_{A_i}(p)$. Since R_i is a color region for the class p in $\boldsymbol{\nu}$ where c disappears, c' is canonical. As c is non-canonical R_i becomes a color region for the class p and colors c', c in $\boldsymbol{\nu}'$. Since $\boldsymbol{\nu}$ is almost valid, R_i is the only color region for the class p in $\boldsymbol{\nu}$. Furthermore, the color c does not appear in $\boldsymbol{\nu}$. Thus, R_i is the only color region for the class p and colors c', c in $\boldsymbol{\nu}'$ and Property 4 follows.

- In Case 3, we have $\nu'_{A_i}(p) = \nu'_{B_i}(p) = \nu'_{C_i}(p) = c$. Since R_i is a color region for the class p in $\boldsymbol{\nu}$ it becomes a waiting region in $\boldsymbol{\nu}'$ and again Property 4 follows.

□

We can now give the proof of Lemma 4:



(A) Case 3

(B) Case 4

(C) Case 5

FIGURE 4.12: The initial coloring $\boldsymbol{\nu}$ for cases 3, 4, and 5 in the proof of Lemma 4. The rows represent the classes. A blank indicates a color region for the class, a dashed line indicates that the class may or may not be involved in a color region. The vertical segment at the end of the buffer indicates that the class is internal. The dotted vertical lines separate the different regions.

Proof of Lemma 4. Let W be the clique associated with vector ν_W and S be a child of W. Let $(\mathcal{B}, \boldsymbol{\nu})$ be the valid buffer rooted at S. Assume that $D_{\mathcal{B}}(\nu_W, \boldsymbol{\nu}) > 0$. Then there exists $p \leq \omega$ such that $\nu_{C_N}(p) \neq \nu_W(p) := c$ and by Observation 13 the class p is internal to $\mathcal{B}$.

The following sequences of recolorings only recolor blocks of $R_s, \ldots, R_N$ and all the recolorings fit in the framework of Lemma 8. Thus Properties 1, 2 and 3 of almost valid buffers are always satisfied. We then only have to check Properties 4 and 5' to conclude the proof. Let us distinguish several cases:

1. No class is colored with c on $R_s, \ldots, R_N$ in $\boldsymbol{\nu}$. Then c is not canonical since ν_{A_s} is a permutation of the canonical colors by Observation 5. Suppose first that there does not exist a color region for the class p in $\boldsymbol{\nu}$. Since c does not appear in the color buffer, Observation 8 ensures that there exists a waiting region R_i with $s < i < N$. Then by Lemma 9.1, we can recolor $(B_i, \ldots, C_N, p)$ with c and obtain an almost valid buffer. Suppose otherwise that there exists a color region R_j for the class p in $\boldsymbol{\nu}$. By Lemma 9.2, we can recolor $(B_j, \ldots, C_N, p)$ with c and obtain an almost valid buffer. In both cases, the border error decreases.

2. The class p is colored with c on a sequence of consecutive blocks of $R_s, \ldots, R_N$. As $c \neq \nu_{C_N}(p)$, Observation 6 ensures that c and disappears in a color region R_i for the class p and that c is canonical. By Lemma 9.3, we can recolor $(B_i, \ldots, C_N, p)$ with c and obtain an almost valid buffer where the border error has decreased.

3. A class $\ell \neq p$ is colored with c and c is not canonical (see Figure 4.12a for an illustration). Since $(\mathcal{B}, \boldsymbol{\nu})$ is valid, c appears in a color region R_j with $s < j < N$ for the class ℓ and a canonical color c_1. By Observation 7, we have $\nu_{C_N}(\ell) = \nu_W(p) = c$, thus $\nu_{C_N}(\ell) \neq \nu_W(\ell)$ which implies by Observation 13 that the class ℓ is internal. Thus, by Lemma 9.3, we can recolor $(B_j, \ldots, C_N, \ell)$ with c_1 and obtain an almost valid coloring $\boldsymbol{\nu}^{tmp}$. Since $(\mathcal{B}, \boldsymbol{\nu})$ is valid and R_s is not

recolored, $(\mathcal{B}, \boldsymbol{\nu}^{tmp})$ is valid. Moreover, since $\nu_{C_N}(\ell) \neq \nu_W(\ell)$, $D_{\mathcal{B}}(\nu_W, \boldsymbol{\nu}^{tmp}) \leq D_{\mathcal{B}}(\nu_W, \boldsymbol{\nu})$. As no class is colored with c in $\boldsymbol{\nu}^{tmp}$, we can apply case 1 to $(\mathcal{B}, \boldsymbol{\nu}^{tmp})$.

4. A class $\ell \neq p$ is colored with c, c is canonical and does not disappear in the color buffer (see Figure 4.12b for an illustration). Since c is canonical, the class ℓ is colored with c on $R_s, \dots, R_N$ by Observation 6. Since $\nu_{C_N}(\ell) = c = \nu_W(p) \neq \nu_W(\ell)$ the class ℓ is internal.
Let z, z' be two non-canonical colors and let $c_1 = \nu_{A_s}(p)$. Suppose first there exists a color y that is not contained in $B_s, C_s, R_{s+1}, \dots, R_N$ in $\boldsymbol{\nu}$. Then we apply the following recolorings:

   (1) Recolor (B_s, p) with z and (B_s, ℓ) with z',

   (2) Recolor $(C_s, \dots, C_N, \ell)$ with y,

   (3) Recolor $(C_s, \dots, C_N, p)$ with c,

   (4) Recolor $(C_s, \dots, C_N, \ell)$ with c_1.

   If such a color y does not exist then in particular all the non-canonical colors appear in $(\mathcal{B}, \boldsymbol{\nu})$. Note that in that transformation we do not assume that y is not canonical. Since $k \geq \omega + 3$ and every non-canonical color appears in at most one class of the color buffer, there exists a class $q \notin \{p, \ell\}$ and a color region R_j where some canonical color y disappears and the non-canonical color z'' appears. Free to modify z and z', we can assume that $z'' \notin \{z, z'\}$. Then we can add the following recolorings to the sequence:

   (0) Recolor $(B_s, \dots, A_j, q)$ with z'',

   (5) Recolor $(B_s, \dots, A_j, q)$ with y.

   And after recoloring 0, the color y is not contained anymore in $B_s, C_s, R_{s+1}, \dots, R_N$. So we can apply recolorings 1 to 4. Let us justify that the colorings are proper. Recoloring 0 is proper since q is the only class that contains z'' in $R_s, \dots, R_N$ by Observation 7. Recoloring 1 is proper by the separation property and the fact that after recoloring 0, the only non-canonical color in region R_s is $z'' \notin \{z, z'\}$ (if exists). Recolorings 2, 3 and 4 are proper since the classes p and ℓ are internal and, by the separation property, after recoloring 0, $(C_s, \dots, C_N, \ell)$ is y-free, after recolorings 1 and 2 $(C_s, \dots, C_N, p)$ is c-free and after recolorings 1 and 3 $(C_s, \dots, C_N, \ell)$ is c_1-free. Finally recoloring 5 is proper since after recoloring 4, the only class colored with y on $R_s, \dots, R_N$ is the class q on A_s.
   Let us show Properties 4 and 5' of almost valid buffers are satisfied. First note the colorings 0 and 5 cancel each other thus the coloring is only modified on classes p and ℓ. Furthermore recolorings 3 and 4 consist in swapping the coordinates p and ℓ on the regions $R_{s+1}, \dots, R_N$. By Observation 4, the regions $R_{s+1}, \dots, R_N$ remain either waiting or color regions (in particular R_N remains

a waiting region). As furthermore no color region is created Property 4 holds. Finally note that in $\boldsymbol{\nu}'$, R_s is a transposition region and then Property 5' follows.

5. A class $\ell \neq p$ is colored with c, c is canonical, and c disappears in the color region R_j for class ℓ where the non-canonical color z appears (see Figure 4.12c for an illustration). Let $z' \neq z$ be a non-canonical color and let $c_1 = \nu_{A_s}(p)$. We apply the following recolorings:

   (1) Recolor $(B_s, \ldots, A_j, \ell)$ with z,

   (2) Recolor (B_s, p) with z',

   (3) Recolor $(C_s, \ldots, C_N, p)$ with c,

   (4) Recolor $(C_s, \ldots, A_j, \ell)$ with c_1.

   Recoloring 1 is proper since Observation 7 ensures that the only class colored with z in $R_s, \ldots, R_N$ is the class ℓ on $(B_j, \ldots, C_N)$. By the separation property, $(B_s, \ldots, A_j, \ell)$ is z-free in $\boldsymbol{\nu}$. Recoloring 2 is proper as the only non-canonical color in R_s after recoloring 1 is $z \neq z'$. Recoloring 3 is proper as the class p is internal and thus after recoloring 1, $(C_s, \ldots, C_N, p)$ is c-free by the separation property. Finally, recoloring 4 is proper as after recolorings 2 and 3, $(C_s, \ldots, A_j, \ell)$ is c_1-free by the separation property.

   Let us show that the resulting coloring defines an almost valid buffer. First note that regions $R_{j+1}, \ldots, R_N$ are only modified by recoloring 3 and Observation 12 ensures they remain either waiting or color regions. In particular R_N remains a waiting region. Note that after recolorings 3 and 4, $\boldsymbol{\nu}'$ on regions $R_{s+1}, \ldots, R_{j-1}$ is obtained from $\boldsymbol{\nu}$ by swapping coordinates p and ℓ (on these regions). Thus the nature of these regions is maintained by Observation 4. Since R_j was a color region for the class ℓ and colors c, z in $\boldsymbol{\nu}$, B_j, C_j are not modified and since $\nu'_{A_j}(\ell) = c_1$, R_j is a color region for class ℓ and colors c_1, z in $\boldsymbol{\nu}'$. So the regions $R_{s+1}, \ldots, R_N$ remain either waiting or color regions. Furthermore no new color region is created and colors c_1, z are involved in exactly one color region thus Property 4 is satisfied. Finally, R_s is indeed a transposition region in $\boldsymbol{\nu}'$ since $\boldsymbol{\nu}$ is a valid buffer and z and z' are non-canonical colors, thus Property 5' holds.

Note that by Lemma 3 we can always suppose that up to a recoloring the temporary colors z, z' used in R_s are the same than the ones used in the transposition buffer of $(\mathcal{B}, \boldsymbol{\nu}')$. So in every case we are able to decrease the border error of $(\mathcal{B}, \boldsymbol{\nu})$ by one by recoloring each coordinate of $R_s, \ldots, R_N$ at most three times and obtain an almost valid buffer $(\mathcal{B}, \boldsymbol{\nu}')$, which completes the proof. □

4.5 Step 2: proof of Lemma 5

The proof distinguishes two cases:

Case 1: there is a region of the transposition buffer of $\mathcal{B}$ that is a waiting region. The core of the proof is the following lemma:

Lemma 10. *Let $(\mathcal{B}, \boldsymbol{\nu})$ be an almost valid buffer and R_i, R_{i+1} be two consecutive regions with $1 < i < s$ such that R_i is a waiting region and R_{i+1} is a transposition region. Then there exists a recoloring sequence of $R_i \cup R_{i+1}$ such that, in the resulting coloring $\boldsymbol{\nu}'$, R_i is a transposition region, R_{i+1} is a waiting region, and $(\mathcal{B}, \boldsymbol{\nu}')$ is almost valid. Moreover only coordinates of $R_i \cup R_{i+1}$ are recolored at most twice.*

Proof. Let p, ℓ be the classes permuted in R_{i+1}. Let $c_1 = \nu_{A_{i+1}}(\ell) = \nu_{C_{i+1}}(p)$ and $c_2 = \nu_{A_{i+1}}(p) = \nu_{C_{i+1}}(\ell)$ and z, z' be the temporary colors of R_{i+1}. Recall that since $i < s$, Observation 5 ensures that R_i only contains canonical colors. We apply the following recolorings:

1. Recolor (B_i, C_i, A_{i+1}, p) with z,
2. Recolor $(B_i, C_i, A_{i+1}, \ell)$ with z',
3. Recolor $(C_i, A_{i+1}, B_{i+1}, p)$ with c_1,
4. Recolor $(C_i, A_{i+1}, B_{i+1}, \ell)$ with c_2.

Since R_i is a waiting region that only contains canonical colors and R_{i+1} is a transposition region for classes p and ℓ in $\boldsymbol{\nu}$, recolorings 1 and 2 are proper by the separation property. After the first two recolorings, the color c_1 (resp. c_2) is only contained in (A_i, ℓ) and (C_{i+1}, p) in $R_i \cup R_{i+1}$ (resp. (A_i, p) and (C_{i+1}, ℓ)). Thus the separation property ensures that recolorings 3 and 4 are proper.

One can easily check that in $\boldsymbol{\nu}'$, R_i is a transposition region for classes p and ℓ and that R_{i+1} is a waiting region. Let us finally prove that $(\mathcal{B}, \nu')$ is almost valid. Since $\nu'_{A_i} = \nu_{A_i}$, $\nu'_{C_{i+1}} = \nu_{C_{i+1}}$ and $\nu'_{C_i} = \nu_{A_{i+1}}$, the continuity property holds. The other properties are straightforward since we only recolor the regions R_i, R_{i+1} which are waiting or transposition regions in $\boldsymbol{\nu}'$. □

By assumption there exists a waiting region in $R_2, \dots R_{s-1}$. Amongst all these regions let R_i with $1 < i < s$ be the one with the largest index. We iteratively apply Lemma 10 to the regions $(R_i, R_{i+1}), (R_{i+1}, R_{i+2}), \dots, (R_{s-1}, R_s)$. Each class is recolored at most four times and the resulting coloring $\boldsymbol{\nu}'$ defines a valid buffer.

Case 2: All the regions of the transposition buffer of $\mathcal{B}$ are transposition regions. As there are $3\binom{\omega}{2}$ regions in the transposition buffer and only $\binom{\omega}{2}$ distinct transpositions of $[\![1, \omega]\!]$, there must exist two distinct regions R_i and R_j with $1 < i < j < s$ for which the same pair of colors is transposed (note that the colors might be associated to different classes in R_i and R_j but it does not matter). Let us prove the following lemma:

Lemma 11. *Let $(\mathcal{B}, \boldsymbol{\nu})$ be an almost valid buffer and c_1, c_2 be two canonical colors. If there exist two transposition regions R_i and R_j where colors c_1 and c_2 are transposed, then there exists a sequence of recolorings of $\cup_{t=i}^{j} R_t$ such that each coordinate is recolored at most twice, R_i and R_j are waiting regions in the resulting coloring $\boldsymbol{\nu}'$, and $(\mathcal{B}, \boldsymbol{\nu}')$ is almost valid.*

Proof. Let p, ℓ (resp. p', ℓ') be the classes permuted in R_i (resp. R_j). Without loss of generality, we can assume that $i < j$, $\nu_{A_i}(p) = \nu_{C_i}(\ell) = \nu_{A_j}(\ell') = \nu_{C_j}(p') = c_1$ and $\nu_{A_i}(\ell) = \nu_{C_i}(p) = \nu_{A_j}(p') = \nu_{C_j}(\ell') = c_2$. By Property 3 of almost valid buffers, all the transposition regions use the same temporary colors. Let z, z' be these colors and let $z'' \notin \{z, z'\}$ be another non-canonical color, which exists since $k \geq \omega + 3$. Note that z'' does not appear in $R_1, \ldots, R_s$. Let I_1 (resp. I_2) be the set of blocks of $C_i, \ldots, A_j$ that contains color c_1 (resp. c_2). For each block $Q \in I_1$ (resp. $Q' \in I_2$), there exists a class p_Q (resp. $\ell_{Q'}$) such that (Q, p_Q) (resp. $(Q', \ell_{Q'})$) is colored with c_1 (resp. c_2). We apply the following recolorings:

1. For every $Q \in I_1$ recolor (Q, p_Q) with z'',
2. For every $Q \in I_2$ recolor (Q, ℓ_Q) with c_1,
3. For every $Q \in I_1$ recolor (Q, p_Q) with c_2,
4. Recolor (B_i, p) and (B_j, p') with c_1,
5. Recolor (B_i, ℓ) and (B_j, ℓ') with c_2.

Let us justify that the recolorings are proper. Recoloring 1 is proper since $(R_1, \ldots, R_s)$ is z''-free and all the blocks of Q were colored with c_1 in $\boldsymbol{\nu}$. After recoloring 1, $(C_i, \ldots, A_j)$ is c_1-free since B_i and B_j does not contain c_1 since R_i and R_j are transposition regions in $\boldsymbol{\nu}$ for the color c_1. Then recoloring 2 is proper. Recoloring 3 is proper since, after recoloring 2, $(C_i, \ldots, A_j)$ is c_2-free. At this point, the only class that contains c_1 in R_i (resp. R_j) is the class p (resp. p') and the only class that contains c_2 in R_i (resp. R_j) is the class ℓ (resp. ℓ'). Thus, the recolorings 4 and 5 are proper. Note that each coordinate of $R_i, \ldots, R_j$ is recolored at most twice in this sequence.

Let us show that $(\mathcal{B}, \boldsymbol{\nu}')$ is an almost valid buffer. Properties 2, 4 and 5' are indeed satisfied since R_1 and $R_s, \ldots, R_N$ have not been recolored. By Observation 5, for every $t \in [\![i, j-1]\!]$, the blocks C_t, A_{t+1} contain all the canonical colors, and thus belong to $I_1 \cap I_2$. As the color c_1 has been replaced by the color c_2 and conversely, and as no other color is modified on these blocks, the continuity property holds. Let us prove Property 3 of almost valid buffers holds. For every t such that $i < t < j$, the coloring of R_t in $\boldsymbol{\nu}'$ is obtained from the coloring of R_t in $\boldsymbol{\nu}$ by replacing the color c_1 by the color c_2 and conversely. One can easily check that since c_1 and c_2 are canonical, R_t remains either a waiting region or a transposition region after this operation. Finally we have that in R_i (resp. R_j), the classes p and ℓ (resp. p' and ℓ') are colored with the same colors c_1 and c_2 on the three blocks of the region. Thus the regions R_i and R_j are waiting regions and Property 3 is satisfied. □

The coloring $\boldsymbol{\nu}'$ of $\mathcal{B}$ obtained after applying Lemma 11 is almost valid and R_j is a waiting region for $\boldsymbol{\nu}'$. So case 1 applies, which concludes the proof of Lemma 5.

4.6 Step 3: proof of Lemma 6

Let W be a clique associated with a vector ν_W and $S_1, \dots, S_e$ be the children of C. We denote the buffer of S_i by $\mathcal{B}_i = R_1^i, \dots, R_N^i$ and its vectorial coloring by $\boldsymbol{\nu}^i$). By assumption, for every $i \in [\![1, e]\!]$, $(\mathcal{B}_i, \boldsymbol{\nu}^i)$ is valid and $D_{\mathcal{B}}(\nu_W, \boldsymbol{\nu}^i) = 0$.
The proof is divided in two main steps. We will first show that there exist sequences of recolorings such that the color buffer of all the $\mathcal{B}_i$s have the same coloring and match with the coloring of W. We will then show that, after this first step, there exist sequences of recolorings of the transposition buffers of the $\mathcal{B}_i$s such that the buffers all have the same coloring.

Step 1. Agreement on the color buffers
Let R_i be a color region of an almost valid buffer where the color c disappears and z appears. The following lemma shows that, up to recoloring each coordinate of the color buffer at most once, we can "choose" the index of the region in which c disappears and z appears.

Lemma 12. *Let $(\mathcal{B}, \boldsymbol{\nu})$ be a valid buffer and $s < i, j < N$ be such that R_i is a color region for a class $p \leq \omega$ and the colors c, z. There exists a sequence of recolorings of $R_{s+1}, \dots, R_{N-1}$ such that each coordinate is recolored at most once and in the resulting coloring $\boldsymbol{\nu}'$, the region R_j is a color region for the class p and colors c, z and $(\mathcal{B}, \boldsymbol{\nu}')$ is valid. Furthermore, if R_j was a waiting region in $\boldsymbol{\nu}$ then R_i is a waiting region in $\boldsymbol{\nu}'$ and if R_j was a color region for the colors c', z' in $\boldsymbol{\nu}$ then R_i is a color region for the colors c', z' in $\boldsymbol{\nu}'$. Other regions remain either waiting region or color regions for the same pair of colors.*

Proof. We have two cases to consider, either R_j is a waiting region or is a color region for a class $\ell \neq p$.

Case 1. R_j is a waiting region. In that case we simply recolor $(B_i, \dots, A_j, p)$ with c if $i < j$ or we recolor $(B_j, \dots, A_i, p)$ with z if $i > j$. Since $(\mathcal{B}, \boldsymbol{\nu})$ is valid, Observation 6 ensures that the class p is the only class colored with c in $R_s, \dots, R_N$. So the recoloring is proper. By Observation 12, the regions $R_{i+1}, \dots, R_{j-1}$ are still waiting or color regions for the same pairs of colors in $\boldsymbol{\nu}'$. One can then easily check that the region R_i is a waiting region in $\boldsymbol{\nu}'$, that R_j is a color region for the class p and colors c, z in $\boldsymbol{\nu}'$ and that $(\mathcal{B}, \boldsymbol{\nu}')$ is valid.

Case 2. R_j is a color region for the class $\ell \neq p$ and colors c', z'.
Since the recoloring will be symmetric, we assume that $i < j$. We apply the following recolorings:

1. Recolor $(B_i, \dots, A_j, \ell)$ with z'
2. Recolor $(B_i, \dots, A_j, p)$ with c

Since $(\mathcal{B}, \boldsymbol{\nu})$ is valid, Observations 6 and 7 ensures the class p (resp. ℓ) is the only class colored with c and z (resp. c' and z') in $R_s, \dots, R_N$. Thus the two recolorings are proper. The regions of $R_{s+1}, \dots, R_{i-1}$ and $R_{j+1}, \dots, R_{N-1}$ are not recolored, and

by Observation 12, the regions $R_{i+1}, \dots, R_{j-1}$ are either waiting or color regions for the same pairs of colors in $\boldsymbol{\nu}'$. One can then easily check that regions R_i is a color region for the class ℓ and colors c', z', that region R_j is a color region for the class p and colors c, z in $\boldsymbol{\nu}'$ and that $(\mathcal{B}, \boldsymbol{\nu}')$ is valid. □

The following lemma will permit to guarantee that the colorings agree on the vector ν_{C_s}.

Lemma 13. *Let W be a clique associated with a vector ν_W. Let S_1, S_2 be two children of W and $\mathcal{B}_i$ be the buffer rooted at S_i for $i \leq 2$. Assume moreover that $(\mathcal{B}_1, \boldsymbol{\nu})$ and $(\mathcal{B}_2, \boldsymbol{\mu})$ are valid and satisfy $D_{\mathcal{B}_1}(\nu_W, \boldsymbol{\nu}) = D_{\mathcal{B}_2}(\nu_C, \boldsymbol{\mu}) = 0$. Then there exists a sequence of recolorings of $\cup_{j=s}^{N-1} R_j^2$ such that in the resulting coloring $\boldsymbol{\mu}'$ of $\mathcal{B}_2$ we have $\nu_{C_s}(p) = \mu'_{C_s}(p)$ for every $p \leq \omega$, and that $(\mathcal{B}_2, \boldsymbol{\mu}')$ is valid. Moreover each coordinate is recolored at most $12(k-\omega)$ times.*

Proof. Note that, since $D(\nu_W, \boldsymbol{\nu}) = D(\nu_W, \boldsymbol{\mu}) = 0$, if $\nu_{C_N}(p)$ is canonical then there is not color regions for the class p in $\boldsymbol{\nu}$ or $\boldsymbol{\mu}$ and we have $\nu_{C_s}(p) = \nu_{C_N}(p) = \mu_{C_N}(p) = \mu_{C_s}(p)$. So if they differ on C_s for some class p, there exists a color region for p. Assume that there exists a class p such that $\mu_{C_s}(p) \neq \nu_{C_s}(p) := c$. By Observation 5, μ_{C_s} an ν_{C_s} are permutations of $\{1, \dots, \omega\}$. So there exists $\ell \neq p$ such that $\mu_{C_s}(\ell) = c$. Since $\nu_{C_N}(p) = \mu_{C_N}(p)$ and $\nu_{C_N}(\ell) = \mu_{C_N}(\ell)$, there exists color regions for the classes p and ℓ in $(\mathcal{B}_1, \boldsymbol{\nu})$ and $(\mathcal{B}_2, \boldsymbol{\mu})$.
By Lemma 12 we can assume that the color region for the class p and color c', z is R_{s+1} and that the color region for the class ℓ and colors c, z' is R_{s+2}. Since $k \geq \omega + 3$, there exists a non-canonical color $z'' \notin \{z, z'\}$ such that z'' does not appear in $R_s^2, \dots R_{s+2}^2$ in $\boldsymbol{\mu}$. We apply the following recolorings:

1. Recolor $(B_s^2, \dots, A_{s+2}^2, \ell)$ with z'',
2. Recolor (B_s^2, p) with z',
3. Recolor (C_s^2, A_{s+1}^2, p) with c,
4. Recolor $(C_s^2, \dots, A_{s+2}^2, \ell)$ with c'.

Let us call $\boldsymbol{\mu}''$ the resulting coloring. Recoloring 1 is proper since $(B_s^2, \dots, A_{s+2}^2)$ is z''-free in $\boldsymbol{\mu}$. Since $(\mathcal{B}_2, \boldsymbol{\mu})$ is valid, the only non-canonical color in R_s after recoloring 1 is $z'' \neq z$, thus recoloring 2 is proper. After recoloring 1, $(C_s^2, \dots, A_{s+1}^2)$ is c-free and after recoloring 3, $(C_s^2, \dots, A_{s+2}^2)$ is c'-free thus recolorings 3 and 4 are proper.

Let us show that $(\mathcal{B}_2, \boldsymbol{\mu}'')$ is almost valid. Properties 2 and 3 are indeed satisfied and Lemma 8 ensures the continuity property holds. Let us check Property 4. The region R_{s+1}^2 (resp. R_{s+2}^2) is a color region for the class p (resp. ℓ) and colors c, z' (resp. c', z) in $\boldsymbol{\mu}$ and thus becomes a color region for colors (c, z') (resp. c', z) after recoloring 3 (resp. 4) in $\boldsymbol{\mu}''$ and Property 4 follows. Finally one can easily check that the region R_s^2 becomes a transposition region for the classes p and ℓ. As the region R_N^2 is not recolored, Property 5' follows and $(\mathcal{B}_2, \boldsymbol{\mu}'')$ is almost valid. By Lemma 3,

we can assume that the temporary colors used in R_s are the same that the ones used in the transposition buffer of $\mathcal{B}_2$, free to recolor the coordinates of μ''_{B_s} at most twice. We can thus apply Lemma 5 to $(\mathcal{B}_2, \boldsymbol{\mu}'')$ to obtain a coloring $\boldsymbol{\mu}'$ such that $(\mathcal{B}_2, \boldsymbol{\mu}')$ is a valid buffer.

Let us count how many times a coordinate is recolored. If a class p satisfies $\nu^2_{C_s}(p) \neq \nu^1_{C_s}(p)$, then there exists a color region for the class p in $\mathcal{B}_2$ thus there are at most $(k-\omega)$ such classes. So we may have to apply the described sequence of recolorings and Lemma 5 at most $(k-\omega)$ times. As this sequence recolors each coordinate at most 6 times and as Lemma 5 recolors each coordinate at most 6 times the result follows. □

Now we apply Lemma 13 iteratively for $(\mathcal{B}_1, \boldsymbol{\nu}^1)$ and $(\mathcal{B}_i, \boldsymbol{\nu}^i)$ where we only recolor vertices in $\mathcal{B}_i \setminus R^i_N$. By the separation property, it indeed implies that no vertex of W is recolored. At the end of this procedure, we have recolored vertices of $\mathcal{B}_i$ for $i \geq 2$ at most $12(k-\omega)$ times and the resulting coloring (still denoted by $\boldsymbol{\nu}^i$ for convenience) is such that $(\mathcal{B}_i, \boldsymbol{\nu}^i)$ is a valid buffer that satisfies $\nu^1_{C_s}(p) = \nu^i_{C_s}(p)$ and $\nu^1_{C_N}(p) = \nu^i_{C_N}(p)$ for every $p \leq \omega$. In particular, by Observations 6 and 7, there exists a color region for class p in $\boldsymbol{\nu}^i$ if and only if there exists a color region for class p in $\boldsymbol{\nu}^1$. Moreover the colors that appear and disappear are the same. So in order to be sure that $\nu^1_Q(p) = \nu^i_Q(p)$ for every block Q in $R_s, \dots, R_N$, we just have to guarantee that the color change for class p are in regions with the same index in $\boldsymbol{\nu}^i$ and $\boldsymbol{\nu}^1$. We can guarantee that by iteratively applying Lemma 12. Indeed let R_j be the smallest region of the color buffer where R^1_j and R^i_j do not have the same coloring. If R^1_j is a waiting region, then there exists a waiting region $R^2_{j'}$ in $\{R^i_{j+1}, \dots, R^2_{N-1}\}$ since the number of waiting regions is the same. If R^1_j is a color region where z appears, then by minimality of i, there exists $j' > j$ such that z appears in $R^2_{j'}$. By Lemma 12, we can assume that the two colorings ν^1 and ν^i agree up to region j. We will apply at most $(k-\omega)$ times Lemma 12. And then in total, we recolor every vertex of the color buffer at most $(k-\omega)$ times.

Step 2: Agreement on the transposition buffers.
Let us first remark that since the proof of Lemma 10 is symmetric the following holds:

Lemma 14. *Let $(\mathcal{B}, \boldsymbol{\nu})$ be an almost valid buffer and R_i, R_{i+1} be two consecutive regions with $1 < i < s$ such that R_i is a transposition region and R_{i+1} is a waiting region. Then there exists a recoloring sequence of $R_i \cup R_{i+1}$ such that in the resulting coloring $\boldsymbol{\nu}'$, R_i is a waiting region, R_{i+1} is a transposition region and $(\mathcal{B}, \boldsymbol{\nu}')$ is almost valid. Moreover only coordinates of R_i, R_{i+1} are recolored at most twice.*

A valid buffer $(\mathcal{B}, \boldsymbol{\nu})$ is *well-organized* if the first $2\binom{\omega}{2}$ regions of the transposition buffer are waiting regions.

Lemma 15. *Let $(\mathcal{B}, \boldsymbol{\nu})$ be a valid buffer. There exists a sequence of recolorings of $R_2, \dots, R_{s-1}$ such that the resulting coloring $(\mathcal{B}, \boldsymbol{\nu}')$ is valid and well-organized. Moreover each coordinate is recolored $O(\omega^2)$ times.*

Proof. By Lemma 11, we can assume that (free to recolor at most $O(\omega^2)$ times each vertex of the transposition buffer) there are at most $\binom{\omega}{2}$ transposition regions in the transposition buffer of $(\mathcal{B}, \boldsymbol{\nu})$ and at least $2\binom{\omega}{2}$ waiting regions. By Lemma 14, we can then assume that the first $2\binom{\omega}{2}$ regions of the transposition buffer are waiting regions. Note that performing all these transformations require at most $O(\omega^2)$ recolorings of each coordinate. □

Lemma 15 ensures that we can assume that $(\mathcal{B}_i, \boldsymbol{\nu}^i)$ is well-organized for every $i \in [\![1, e]\!]$. Let $(\mathcal{B}, \boldsymbol{\nu})$ be a valid buffer and R_j with $1 < j < s$ be a region of the transposition buffer. By Observation 5, ν_{A_j} and ν_{C_j} are permutations of $[\![1, \omega]\!]$ and there exists a unique transposition τ_j (which might be the identity) such that $\nu_{C_j} = \tau_j \circ \nu_{A_j}$. Conversely, ν_{A_j} and τ_j define a unique coloring of R_j (since the temporary colors used in a valid buffer are fixed). In what follows, for $i \in [\![1, e]\!]$ and $j \in [\![2, s-1]\!]$, τ_j^i denotes the transposition corresponding to the region R_j^i of $(\mathcal{B}_i, \boldsymbol{\nu}^i)$. For the ease of notation, let $\Omega = \binom{\omega}{2}$. Note that after step 1, and since we can assume that all the buffers $(\mathcal{B}_i, \nu^i)$ are well-organized, we have $\prod_{j=s-1}^{2\Omega+2} \tau_j^i = \nu_{A_s}^1$ for all $i \leq e$ (where the symbol $\prod$ denotes the composition).
From now on, we will only recolor the transposition buffers of $(\mathcal{B}_i, \boldsymbol{\nu}^i)$, for $i \geq 2$ to make them agree with $(\mathcal{B}_1, \boldsymbol{\nu}^1)$. Let us start with the two following lemmas:

Lemma 16. *Let $(\mathcal{B}, \boldsymbol{\nu})$ be a valid buffer, $t_0, t_1 \in [\![2, s-1]\!]$ with $t_0 < t_1$, and $p \neq q$ be two integers in $[\![1, \omega]\!]$. Assume that $R_{t_0}, \ldots, R_{t_1}$ are waiting regions. There exists a sequence of recolorings of $\cup_{j=t_0}^{j=t_1} R_j$ such that in the resulting coloring $\boldsymbol{\nu}'$, $\tau_{t_0} = \tau_{t_1} = \tau_{p,q}$ (and then R_{t_0} and R_{t_1} are transposition regions for the classes p and q), for every $t_0 < i < t_1$ R_i is a waiting region, and $(\mathcal{B}, \boldsymbol{\nu}')$ is valid. Moreover, each coordinate is recolored at most twice.*

Proof. Since $(\mathcal{B}, \boldsymbol{\nu})$ is valid, $\nu_{A_{t_0}}$ is a permutation of the canonical colors by Observation 5. As $R_{t_0}, \ldots, R_{t_1}$ are waiting regions, the continuity property ensures that the class m is colored with the same canonical color in $R_{t_0}, \ldots, R_{t_1}$ in $\boldsymbol{\nu}$. Let $c_1 = \nu_{A_{t_0}}(p)$, $c_2 = \nu_{A_{t_0}}(q)$, and $z \neq z'$ be the (non-canonical) temporary colors used in the transposition buffer of $\mathcal{B}$. We apply the following recolorings:

1. Recolor $(B_{t_0}, \ldots, B_{t_1}, p)$ with z,
2. Recolor (B_{t_0}, q) and (B_{t_1}, q) with z',
3. Recolor $(C_{t_0}, \ldots, A_{t_1}, q)$ with c_1,
4. Recolor $(C_{t_0}, \ldots, A_{t_1}, p)$ with c_2.

Recolorings 1 and 2 are proper since $R_{t_0}, \ldots, R_{t_1}$ only contain canonical colors in $\boldsymbol{\nu}$ and $z \neq z'$ are non-canonical. After recoloring 1, $(C_{t_0}, \ldots, A_{t_1})$ is c_1-free and after recolorings 2 and 3, $(C_{t_0}, \ldots, A_{t_1})$ is c_2-free, thus recolorings 3 and 4 are proper. The regions $R_{t_0+1}, \ldots, R_{t_1-1}$ are waiting regions in $\boldsymbol{\nu}'$ by Observation 12. One can then easily check that in $\boldsymbol{\nu}'$, the regions R_{t_0} and R_{t_1} are transposition regions for the classes p and q and that $(\mathcal{B}, \boldsymbol{\nu}')$ is valid. □

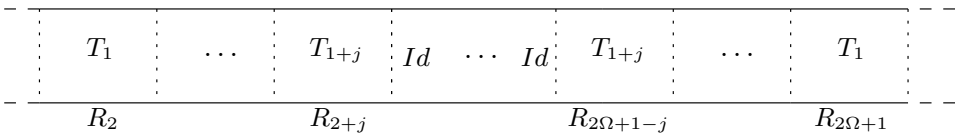


FIGURE 4.13: Coloring of regions $R_2, \dots, R_{2\Omega+1}$ of the buffer $\mathcal{B}$, after iteration j in Lemma 17.

Lemma 17. *Let $(\mathcal{B}, \boldsymbol{\nu})$ be a valid and well-organized buffer and let $T_1, \dots, T_\Omega$ be transpositions of $[\![1, \omega]\!]$. There exists a sequence of recolorings of $\cup_{j=2}^{2\Omega+1} R_j$ such that in the resulting coloring $\boldsymbol{\nu}'$ of $\mathcal{B}$, $\tau_{2+j} = \tau_{2\Omega+1-j} = T_{1+j}$ for all $j \in [\![0, \Omega-1]\!]$ and $(\mathcal{B}, \boldsymbol{\nu}')$ is valid. Moreover each coordinate is recolored at most 2Ω times.*

Proof. For $j \in [\![0, \Omega-1]\!]$ apply the following:

- If $T_{1+j} = Id$, do nothing, else
- Let $p \neq q$ be the classes that T_{1+j} permutes. Apply Lemma 16 to $\mathcal{B}$ with $t_0 = 2+j$, $t_1 = 2\Omega+1-j$ and integers p and q.

As $(\mathcal{B}, \boldsymbol{\nu})$ is well-organized, the regions $R_2, \dots, R_{2\Omega+1}$ are initially waiting regions. Then, after iteration j, we obtain a coloring of $\mathcal{B}$ such that for all $t \leq j$, $\tau_{2+t} = \tau_{2\Omega+1-t} = T_{1+j}$ and the regions $R_{2\Omega+j+2}, \dots, R_{2\Omega-j}$ are waiting regions (see figure 4.13 for an illustration). Thus we can iterate and the algorithm terminates with the desired coloring $\boldsymbol{\nu}'$. As Lemma 16 maintains a valid buffer, $(\mathcal{B}, \boldsymbol{\nu}')$ is valid. We applied Lemma 16 at most Ω times, thus each coordinate is recolored at most 2Ω times. □

For every $i \in [\![2, \omega]\!]$, we apply Lemma 17 to $(\mathcal{B}_i, \boldsymbol{\nu}^i)$ with $T_{1+j} = \tau^1_{2\Omega+2+j}$ for $j \in [\![0, \Omega-1]\!]$. Recall that $\nu^1_{A_s} = \prod_{j=s-1}^{2\Omega+2} \tau^1_j$ and that $(\nu^1_{A_s})^{-1} = \prod_{j=2\Omega+2}^{s-1} \tau^1_j$. Then after applying Lemma 17 to $\mathcal{B}_i$ with $i \geq 2$ we obtain a coloring $\boldsymbol{\nu}^i$ such that $\prod_{j=s-1}^{2} \tau^i_j = \nu^i_{A_s} \circ (\nu^1_{A_s})^{-1} \circ \nu^1_{A_s}$. We will show that we can use this coloring to "cancel" the initial transpositions τ^i_j (with $j \geq 2\Omega+2$) of $\mathcal{B}_i$. To do so, we need the following lemma:

Lemma 18. *Let R_i, R_{i+1} be two consecutive regions of a valid buffer $(\mathcal{B}, \boldsymbol{\nu})$ with $1 < i < s$ such that $\tau_{i+1} \neq Id$ and a is a class permuted by τ_{i+1}. There exists a recoloring sequence of $R_i \cup R_{i+1}$ such that in the resulting coloring $\boldsymbol{\nu}'$, τ'_i is either the identity or permutes the class a and τ'_{i+1} does not permute the class a. Moreover, $(\mathcal{B}, \boldsymbol{\nu}')$ is valid and each coordinate is recolored at most 4 times.*

Proof. Let z, z' be the temporary colors used in $(\mathcal{B}, \boldsymbol{\nu})$ and let $z'' \notin \{z, z'\}$ be another non-canonical color. We have different cases to consider:

1. $\tau_i = Id$. Then R_i is a waiting region. Let $b \neq a$ be the other class permuted by τ_{i+1}. We can apply Lemma 10 and obtain a valid buffer $(\mathcal{B}, \boldsymbol{\nu}')$ such that R_i is a transposition region for classes a and b and R_{i+1} is a waiting region. Moreover, the recolorings of Lemma 10 recolors each coordinate at most twice.

2. $\tau_i = \tau_{i+1}$. The regions R_i and R_{i+1} are consecutive and permute the same classes, thus they also permute the same colors. By Lemma 11 we can recolor R_i, R_{i+1} into waiting regions and obtain a valid buffer. Moreover each coordinate is recolored at most twice.

3. τ_i, τ_{i+1} are transpositions which permute exactly one common class. Let $b \neq c$ distinct from a be the other classes that are permuted by τ_i, τ_{i+1}. Let $c_1 = \nu_{A_i}(a)$, $c_2 = \nu_{A_i}(b)$ and $c_3 = \nu_{A_i}(c)$. As the two transpositions only involve $\{a, b, c\}$, one of the three classes is permuted in R_i and R_{i+1}, and by Lemma 3 we can suppose this class is colored with z on B_i and B_{i+1}. As z'' is not contained in the transposition buffer we can recolor this class on $(B_i, \dots, B_{i+1})$ with z''. The only remaining temporary colors on R_i, R_{i+1} is z'. Thus we can recolor one of the two other class with z on $(B_i, \dots, B_{i+1})$ and then recolor the third one with z' on the same set of blocks, and these three recolorings are proper. Let $\boldsymbol{\nu}^{tmp}$ be the coloring we obtain. The colors c_1, c_2, c_3 are not in $(B_i, \dots, B_{i+1})$ in $\boldsymbol{\nu}^{tmp}$. Suppose that $\nu^{tmp}_{C_{i+1}}(a) = c_2$. As C_{i+1} has not been recolored we have necessarily $\nu^{tmp}_{C_{i+1}}(b) = c_3$ and $\nu^{tmp}_{C_{i+1}}(c) = c_1$. We apply the following recolorings:

   (a) Recolor $(C_i, \dots, B_{i+1}, a)$ with c_2,

   (b) Recolor (C_i, A_{i+1}, b) with c_1,

   (c) Recolor $(B_i, \dots, A_{i+1}, c)$ with c_3.

   Given that colors c_1, c_2, c_3 are not contained in $(B_i, \dots, B_{i+1})$ in $\boldsymbol{\nu}^{tmp}$ and given $\nu^{tmp}_{A_i}$ and $\nu^{tmp}_{C_{i+1}}$ these recolorings are proper by the separation property. Furthermore, in the resulting coloring $\boldsymbol{\nu}'$, R_i, R_{i+1} are transposition regions and $\tau'_i = \tau_{a,b}$ and $\tau'_{i+1} = \tau_{b,c}$. The case $\nu^{tmp}_{C_{i+1}}(a) = c_3$ is symmetrical.

4. τ_i, τ_{i+1} involves four different classes $a, b, c, d \leq \omega$. We can suppose w.l.o.g that $\tau_i\tau_{i+1} = \tau_{c,d}\tau_{a,b}$ and by Lemma 3 we can assume that $(B_i, c) = (B_{i+1}, a) = z$ and $(B_i, d) = (B_{i+1}, b) = z'$. Let $c_1 = \nu_{A_i}(a)$, $c_2 = \nu_{A_i}(b)$, $c_3 = \nu_{A_i}(c)$, $c_4 = \nu_{A_i}(d)$. We apply the following recolorings:

   (a) Recolor $(B_i, \dots, B_{i+1}, a)$ with z'',

   (b) Recolor $(C_i, \dots, B_{i+1}, c)$ with z,

   (c) Recolor $(C_i, \dots, B_{i+1}, b)$ with c_1,

   (d) Recolor $(C_i, \dots, B_{i+1}, d)$ with z',

   (e) Recolor $(B_i, \dots, A_{i+1}, c)$ with c_3,

   (f) Recolor $(B_i, \dots, A_{i+1}, d)$ with c_4,

   (g) Recolor (B_i, b) with z,

   (h) Recolor $(C_i, \dots, B_{i+1}, a)$ with c_2.

   Let us justify these recolorings are proper. Recoloring (a) is valid since z'' is not contained in the transposition buffer in $\boldsymbol{\nu}$. Recolorings (b) and (c) are proper

since after recoloring (a), the only class colored with z in R_i, R_{i+1} is the class c and the only class colored with c_1 on $(B_i, \dots, C_{i+1})$ is the class b. Recoloring (d) is proper since after recoloring (c), the only class colored with z' on R_i, R_{i+1} is the class d. At this point the color c_3 (resp. c_4) is only contained in (A_i, c) and (C_{i+1}, d) (resp. (A_i, d) and (C_{i+1}, c)) in R_i, R_{i+1}, thus recolorings (e) and (f) are proper. After recoloring (e), the color z is not contained in R_i, thus recoloring (g) is proper. Finally recoloring (f) is proper, since after recoloring (g) the only class colored with c_2 on $(B_i, \dots, C_{i+1})$ is the class a. One can then easily check that in $\boldsymbol{\nu}'$, the region R_i, R_{i+1} are transposition regions, $\tau'_i = \tau_{a,b}$ and $\tau'_{i+1} = \tau_{c,d}$.

Note that in each case, Properties 2, 4 and 5 of valid buffers are indeed satisfied. Furthermore the continuity property holds as all the given recolorings fit in the framework of Lemma 8. Property 3 is satisfied since all the regions of the transposition buffer in $\boldsymbol{\nu}'$ are either waiting or transposition regions that use the same temporary colors z and z' (up to applying Lemma 3 to R_i and R_{i+1}), thus $(\mathcal{B}, \boldsymbol{\nu}')$ is valid. Moreover each coordinate is recolored at most four times.

□

We can then give the last lemma before concluding:

Lemma 19. *Let $(\mathcal{B}, \boldsymbol{\nu})$ be a valid buffer. Suppose there exists two integers $t_0, t_1 \in [\![2, s-1]\!]$ with $t_0 < t_1$ such that $\prod_{j=t_1}^{t_0} \tau_j = Id$. Then there exists a sequence of recolorings of $\cup_{j=t_0}^{t_1} R_j$ such that in the resulting colorings $\boldsymbol{\nu}'$, $R_{t_0}, \dots, R_{t_1}$ are waiting regions and $(\mathcal{B}, \boldsymbol{\nu}')$ is valid. Moreover each coordinate is recolored $O(\omega^2)$ times.*

Proof. Let $I(\boldsymbol{\nu})$ be the number of transpositions equal to the identity in $\tau_{t_0}, \dots, \tau_{t_1}$ (that is the number of waiting regions in $R_{t_0}, \dots, R_{t_1}$ for the coloring $\boldsymbol{\nu}$). If $I(\boldsymbol{\nu}) = t_1 - t_0 + 1$ we are done. Otherwise we will show that we can always recolor $\cup_{j=t_0}^{t_1} R_j$ and obtain a coloring $\boldsymbol{\nu}'$ such that the transpositions $\tau'_{t_0}, \dots, \tau'_{t_1}$ of $R_{t_0}, \dots, R_{t_1}$ in $\boldsymbol{\nu}'$ satisfy $\prod_{j=t_1}^{t_0} \tau_j = Id$ and $I(\boldsymbol{\nu}') > I(\boldsymbol{\nu})$.
Let $t_0 < r \leq t_1$ be the maximum index such that $\tau_r \neq Id$ and a be a class permuted by τ_r. Let us show that there always exists $i_0 \in [\![0, r - t_0 - 2]\!]$ such that after applying iteratively Lemma 18 to regions R_{r-i-1}, R_{r-i} and the class a for $i \in 0, 1, \dots, i_0$, we obtain a coloring $\boldsymbol{\nu}''$ such that the transposition $\tau''_{i_0}, \tau''_{i_0-1}$ of R_{i_0}, R_{i_0-1} in $\boldsymbol{\nu}''$ satisfy $\tau''_{i_0} = \tau''_{i_0-1}$.
Suppose not. Then, when applying Lemma 18 to regions R_{r-i-1}, R_{r-i} and class a for $i \in 0, 1, \dots, r - t_0 - 1$, case 2 of Lemma 18 never occures. Let $\boldsymbol{\nu}^f$ be the coloring we obtain after the last iteration and τ^f be the corresponding transpositions. Note that R_{t_1+1} is never recolored, thus $T = \prod_{j=t_1}^{t_0} \tau_j^f = Id$. By Lemma 18, $\tau_{t_0}^f$ is the only transposition that permutes the class a amongst $\tau_{t_0}^f, \dots, \tau_{t_1}^f$ and $\tau_{t_0}^f \neq Id$. Thus $T(a) \neq a$, a contradiction.
We obtain a coloring $\boldsymbol{\nu}''$ such that the transpositions $\tau''_{i_0}, \tau''_{i_0-1}$ of R_{i_0}, R_{i_0-1} in $\boldsymbol{\nu}''$ satisfy $\tau''_{i_0} = \tau''_{i_0-1}$. By applying Lemma 18 once more to $\boldsymbol{\nu}''$, R_{i_0}, R_{i_0-1} become

waiting regions and we obtain a coloring $\boldsymbol{\nu}'$ such that $I(\boldsymbol{\nu}') > I(\boldsymbol{\nu})$. Furthermore, since R_{t_1+1} is not recolored, $\prod_{j=t_1}^{t_0} \tau'_j = Id$.
We can iterate this process until we obtain a coloring $\boldsymbol{\nu}'$ (still called $\boldsymbol{\nu}'$ for convenience) such that $I(\boldsymbol{\nu}') = t_1 - t_0 + 1$. Furthermore, $(\mathcal{B}, \boldsymbol{\nu}')$ is valid since Lemma 18 maintains a valid buffer.

Let us justify the number of times a coordinate is recolored. In order to increase $I(\boldsymbol{\nu})$ at each step, we apply Lemma 18 to each region of the buffer at most twice, thus each coordinate of the transposition buffer is recolored at most 8 times. Since $I(\boldsymbol{\nu}) \leq 3\binom{\omega}{2}$, each coordinate of the buffer is recolored $O(\omega^2)$ time.

□

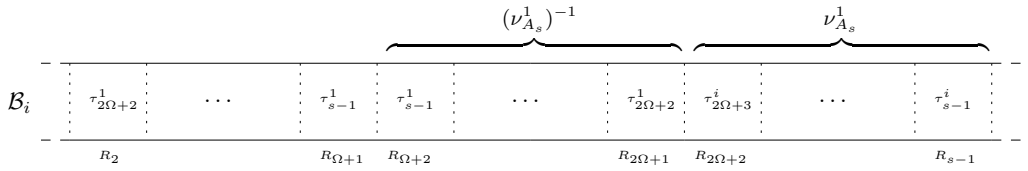


FIGURE 4.14: Coloring of the transposition buffer of $\mathcal{B}_i$ with $i \geq 2$, after step 1 and after applying Lemmas 15 and 17 in step 2. The dotted lines separate the regions.

Let us summarize the step 2 of the proof. By Lemma 15 we can suppose that all the $(\mathcal{B}_i, \boldsymbol{\nu}^i)$ are well-organized. Recall that after step 1, $\nu^i_{A_s} = \nu^1_{A_s}$ for all $i \geq 2$, thus $\prod_{j=s-1}^{2\Omega+2} \tau^i_j = \nu^1_{A_s}$. Then after applying Lemma 17 to $\mathcal{B}_i$ with $i \geq 2$ we obtain a coloring - that we still denote by $\boldsymbol{\nu}^i$ for convenience - such that for every $j \in [\![2, \Omega+1]\!]$, $\tau^i_j = \tau^1_{2\Omega+j}$ and for all $j \in [\![0, \Omega-1]\!]$, $\tau^i_{\Omega+2+j} = \tau^1_{s-1-j}$ (see figure 4.14 for an illustration). Thus we have $\prod_{j=s-1}^{\Omega+2} \tau^i_j = Id$ and we can apply Lemma 19 to $(\mathcal{B}_i, \boldsymbol{\nu}^i)$ and obtain a valid buffer coloring of $\mathcal{B}_i$ such that for all $j \in [\![2, \Omega+1]\!]$, $\tau^i_j = \tau^1_{2\Omega+j}$, and for every $j \in [\![\Omega+2, s-1]\!]$, R^i_j is a waiting region ($\tau^i_j = Id$). It just remains to switch the transpositions up in the buffer using Lemma 15 to obtain a valid buffer $(\mathcal{B}^i, \boldsymbol{\nu}^i)$ such that $\boldsymbol{\nu}^i = \boldsymbol{\nu}^1$ for every $i \in [\![2, e]\!]$.
Finally, note that for the proof of Lemma 6, we applied Lemmas 13, 17 and 19 once to each $\mathcal{B}_i$ for $i \in [\![2, e]\!]$, and we applied Lemma 15 at most twice to each $\mathcal{B}_i$ for $i \in [\![1, e]\!]$. Thus every coordinate is recolored $O(\omega^2)$ times, which concludes the proof.

4.7 Step 4: proof of Lemma 7

Let W be a clique of T with children $S_1, S_2, \ldots S_e$ and let α be a k-coloring of G treated up to S_i for $i \in 1, \ldots, e$. Let ν_W be a vector associated with W, $\mathcal{B} = R_1, \ldots, R_N$ be the buffer rooted at W, and $\mathcal{B}_i = R^i_1, \ldots, R^i_N$ be the buffer rooted at S_i. Suppose that all the $\mathcal{B}_i$s have the same coloring $\boldsymbol{\nu}$ such that $(\mathcal{B}_i, \boldsymbol{\nu})$ is a valid buffer and $D_{\mathcal{B}_i}(\nu_W, \boldsymbol{\nu}) = 0$. We will show that there exists a sequence of vertex recolorings of $\cup_{i=1}^{e} \cup_{j=2}^{N-1} R^i_j$ such that the resulting coloring of the buffer $\mathcal{B}$ rooted at W is well-colored for $\boldsymbol{\nu}$.
Let $Q_1, \ldots, Q_{3\Delta N}$ denote the blocks of $\mathcal{B}$, and $Q^i_1, \ldots, Q^i_{3\Delta N}$ denote the blocks of

$\mathcal{B}_i$ for $i \in [\![1, \omega]\!]$. Note that if a vertex starts at height h in T_{S_i} then it starts at height $h+1$ in T_W. In particular if a vertex v starts at height h in T_{S_i} such that $j\Delta < h < (j+1)\Delta - 1$ for $j \in [\![0, 3N-1]\!]$, then $v \in Q^i_{N-j} \cap Q_{N-j}$. However if a vertex v starts at height $h = j\Delta$, then $v \in Q^i_{N-j} \cap Q_{N-j-1}$, thus we may have to recolor v. Indeed, suppose that Q^i_{N-j} is the block B of a color region for the class p and colors c, z, and that $v \in (Q^i_{N-j}, p)$. Then v is colored with $\nu_B(p) = z$. As we want $\mathcal{B}$ to be well-colored for $\boldsymbol{\nu}$, Q_{N-j-1} has to be well-colored for ν_A, thus we need to recolor v with $\nu_A(p) = c$.
For the ease of notation, we denote by $f(Q^i_j)$ the set of vertices of Q^i_j which height in T_{S_i} is maximum, and $f(Q^i_j, p)$ is the set $f(Q^i_j) \cap X_p$ for $p \leq \omega$. Let z, z' be the temporary colors used in the transposition buffers of $S_1, \ldots, S_e$. For every $j \in N-1, \ldots, 2$ in the decreasing order, we apply to each region $R^1_j, \ldots R^e_j$ the following recolorings:

1. if R^i_j is a waiting region do nothing. By the continuity property and the definition of waiting regions we have $\nu_{C_{i-1}} = \nu_{A_i} = \nu_{B_i} = \nu_{C_i}$, thus R_i is well-colored for $\nu_{A_i}, \nu_{B_i}, \nu_{C_i}$.

2. if R^i_j is a color region for the class q where c_1 disappears, recolor $f(B^i_j, q)$ with c_1.
   This is a proper recoloring since by Observation 6, the only class colored with c_1 on the color buffer of $\mathcal{B}^i$ is the class q.
   For any class $m \neq q$, $\nu_{C_{i-1}}(m) = \nu_{A_i}(m) = \nu_{B_i}(m) = \nu_{C_i}(m)$ by definition of color regions and the continuity property. Furthermore, $\nu_{C_{i-1}}(q) = \nu_{A_i}(q) \neq \nu_{B_i}(q) = \nu_{C_i}(q)$. As the vertices $f(B^i_j, q)$ have been recolored with $\nu_{A_i}(q)$ for every $i \leq p$, it follows that R_i is well-colored for $\nu_{A_i}, \nu_{B_i}, \nu_{C_i}$.

3. if R^i_j is a transposition region for the classes p and q with $\nu_{A_j}(p) = c_1$ and $\nu_{C_j}(q) = c_2$ do in the following order:

   (a) Recolor $f(C^i_j, p)$ with z and recolor $f(C^i_j, q)$ with z', then

   (b) Recolor $f(B^i_j, p)$ with c_1 and recolor $f(B^i_j, q)$ with c_2.

   By the separation property $N[C^i_j] \subseteq B^i_j \cup C^i_j \cup A^i_{j+1}$. As $\boldsymbol{\nu}$ defines a valid buffer, R^i_{j+1} is either a waiting region or a transposition region, thus $\nu_{A_{i+1}}$ only contains canonical colors. The only vertices colored with z (resp. z') in $N[C^i_j]$ are thus vertices of the class p (resp. q), and recoloring (a) is proper.
   Furthermore, $N(B^i_j) \subseteq A^i_j \cup B^i_j \cup C^i_j$. By the definition of transposition regions, the only vertices colored with c_1 (resp. c_2) on $A^i_j \cup B^i_j$ are vertices of the class p (resp. q). By the separation property, $N[f(B^i_j)] \cap C^i_j \subseteq f(C^i_j)$, and after recoloring (a) no vertex of $f(C^i_j)$ is colored with c_1 nor c_2. It follows that recoloring (b) is proper.
   For any class $m \notin \{p, q\}$, $\nu_{C_{i-1}}(m) = \nu_{A_i}(m) = \nu_{B_i}(m) = \nu_{C_i}(m)$ by definition of transposition regions and the continuity property. Furthermore, $\nu_{B_i}(p) = z$ (resp. $\nu_{B_i}(q) = z'$) and the vertices $f(C^i_j, p)$ (resp. $f(C^i_j, q)$) have been recolored

with z (resp. z'). Finally, $\nu_{A_i}(p) = c_1$ (resp. $\nu_{A_i}(q) = c_2$) and the vertices $f(B_j^i, p)$ (resp. $f(B_j^i, q)$) have been recolored with c_1 (resp. c_2). It follows that the region is well-colored for $\nu_{A_i}, \nu_{B_i}, \nu_{C_i}$.

Let us finally check for the coloring of regions R_1 and R_N. We have $R_1 \subseteq \cup_{i=1}^{e} R_1^i \cup A_2^i$. As none of these vertices are recolored, and $\nu_{A_1} = \nu_{B_1} = \nu_{C_1} = \nu_{A_2}$ by the continuity property and Property 2 of valid buffers, R_1 is well-colored for $\nu_{A_1}, \nu_{B_1}, \nu_{C_1}$. We have $R_N \subseteq \cup_{i=1}^{e} R_N^i \cup W$. None of these vertices are recolored (R_N^i is a waiting region for all $i \leq e$ by Property 5 of valid buffers), and by assumption $\nu_W = \nu_{C_N}$. Thus R_N si well-colored for $\nu_{A_N}, \nu_{B_N}, \nu_{C_N}$. It follows that $(\mathcal{B}, \boldsymbol{\nu})$ is a valid buffer. As the vertices of height at least $3\Delta N$ in T_W are not recolored, they remain colored canonically. We obtain a coloring of G treated up to W, which concludes the proof.

Chapter 5

Recoloring graphs of treewidth 2

The results presented in this chapter were obtained with Nicolas Bousquet and Marc Heinrich and were accepted for publication in Discrete Mathematics. The full paper is available on arXiv [7].

In this chapter, we investigate the class of outerplanar graphs and more generally the graphs of treewidth 2. Since graphs of treewidth 2 are 2-degenerate, Theorem 16 proved by Bouquet and Perarnau in of [29] ensures that there exist linear transformations between any pair of 6-colorings. We also know that such a result is impossible for 4-colorings [15] (the diameter is quadratic). So the only open case is when $k = 5$. In this Chapter, we answer this question and show:

Theorem 5. *Let G be graph of treewidth at most 2 and $k = 5$. There exists a constant c such that, for every pair of 5-colorings α, β of G, there exists a transformation from α to β recoloring each vertex at most c times.*

Note that since outerplanar graphs have treewidth 2 and the quadratic diameter for $k = 4$ obtained in [15] also holds for outerplanar graphs, it also completely characterizes the recoloring diameter of outerplanar graphs.

One can naturally ask if the results of Theorem 5 can be extended further. In particular we ask the two following questions, where the second generalizes the first one:

Question 8. *Does there exists a constant C such that, for every graph G of treewidth at most d, the diameter of $\mathcal{R}_k(G)$ is linear when $k \geq d + C$?*

Question 9. *Does there exists a constant C such that, for every d-degenerate graph G, the diameter of $\mathcal{R}_k(G)$ is linear when $k \geq d + C$?*

Outline of the proof. Before going into the details of the proof, let us describe from a high level point of view how the proof works, and how the chapter is organized. The first step in the proof of Theorem 5 is to show that it is sufficient to prove the result for chordal graphs with cliques of size at most 3, instead of graphs of treewidth 2. This reduction to chordal graphs is done in Section 5.1. Proving that the result holds for chordal graphs is the main technical part of the paper, and is done in Section 5.2. The proof for chordal graphs is algorithmic in nature. We first describe the procedure,

introduced in [29], which builds a recoloring sequence by making greedy choices. We prove that the recoloring sequences produced by this procedure recolor each vertex of the graph a constant number of times. This is done by first proving some properties of the sequences produced by this algorithm in Section 5.2.1, and then showing in Section 5.2.2 how these properties can be used to bound the number of times that each vertex is recolored. The core of the proof proceeds by contradiction. Assuming that a vertex is recolored a large number of times puts some very strong constraints on the recoloring sequence on its neighbor. This in turn is used to show that the algorithm would recolor x a much smaller number of times in this sequence.

5.1 Reduction to chordal graphs

In this section, we show how to reduce the problem to the case where the input graph is a chordal graph of clique number at most three. The following lemma is sufficient to prove Theorem 5.

Lemma 20. *There exists a constant c such that for every chordal graph H with $\omega = 3$, there exists a transformation from any 5-coloring of H into a 3-coloring of H by recoloring each vertex at most c times.*

The proof of Lemma 20 is postponed to Section 5.2.2. Let us explain how we can use it to prove Theorem 5

Proof of Theorem 5. Let G be a graph of treewidth at most 2 and α, β be two 5-colorings of G. Let T be a tree decomposition G. Recall that such a decomposition can be found in linear time by [14].

Let us first transform G into a chordal graph H with clique number at most three. The transformation is based on a trick used in [40]. For every bag B of T, we merge the vertices of B colored the same in α. We repeat until we obtain a graph H' such that no bag contains two vertices of the same color. Note that $tw(H') \leq tw(G)$ since we only identify vertices belonging to the same bags. Since we merged vertices colored the same in α, the coloring α' which is the coloring of H' where a vertex receives the color of its color class in G is proper and well-defined. Note that, in H', vertices belonging to the same bag receive distinct colors. So α' is also a coloring of H which is the chordal graph whose clique tree is $T_{H'}$. In other words, H is the graph obtained from H' by transforming every bag of the tree decomposition of H' into cliques.

By Lemma 20, α' can be transformed into a 3-coloring of H by recoloring every vertex at most c times. Feghali observed in [40] that it implies that, in G, we can transform α into a 3-coloring γ_1 of G by recoloring every vertex of G at most c times. With a similar argument, we can transform β in a 3-coloring γ_2 by recoloring every vertex at most c times. The following claim concludes the proof of Theorem 5 by taking $d = 2$.

Claim 1. *Let G be a graph of treewidth at most d and γ_s, γ_t be two $(d+1)$-colorings of G using colors $\{1, \ldots, d+1\}$. If $k \geq 2d+1$, γ_s can be transformed into γ_t by recoloring every vertex at most twice.*

We perform the following recolorings. We denote by X_i the subset of vertices colored with i in γ_s. For every $i \leq d$, recolor one by one all the vertices of X_i with color $d+1+i$. Then recolor the vertices of X_{d+1} with their target colors. Finally recolor all the vertices of X_i for $i \leq d$ with their target color. The resulting coloring is γ_t and we claim that at every step, the current coloring is proper. Indeed, during the first phase, the color classes are the same (we simply change the color classes of the X_is). When X_{d+1} is recolored, all the vertices that receive a color in $\{1, \ldots, d+1\}$ in the current coloring have their final color. The property is kept all along the rest of the recoloring algorithm, which completes the proof since γ_t is proper. $\Diamond$ $\square$

So, in what follows, our goal is to prove Lemma 20.

5.2 Best choice algorithm

Our proof of Lemma 20 is algorithmic and is based on an algorithm already used in [29]. Our main contribution consists in making a careful analysis of its behavior for chordal graphs with clique number at most 3, and showing that the recoloring sequence produced by the algorithm recolors each vertex a constant number of times. We start by describing the algorithm and its properties.

In the following, given a graph G and two colorings α and β of G, a recoloring sequence $\mathcal{S}$ is a sequence $s_1 s_2 \ldots s_t$ where each s_i is a pair (v, c) describing the vertex v and the color c which characterize the i-th step of the sequence. All the intermediate colorings described by this sequence must be proper. The *restriction of $\mathcal{S}$ to X*, denoted $\mathcal{S}_{|X}$, is the recoloring sequence obtained from $\mathcal{S}$ by keeping only recolorings of vertices in X. In particular if X is a single vertex v, then $\mathcal{S}_{|v}$ denotes the sequence of colors taken by the vertex v in the recoloring sequence $\mathcal{S}$. With this notation, $|S_{|v}|$ is the number of times the vertex v is recolored in the sequence $\mathcal{S}$.

Part of our proof relies on the identification of patterns in a recoloring sequence $\mathcal{S}$. Given a sequence of vertices $v_1 \ldots v_\ell$, we say that $\mathcal{S}$ contains this pattern if there is an index i such that for all $j \leq \ell$, s_{i+j} recolors the vertex v_j. Given an integer r, we also denote by $v_i^{\geq r}$ the pattern where v_i is recolored r times in a row (without any other vertex interleaved in $\mathcal{S}$), and $(v_1 \ldots v_\ell)^r$ to denote that the pattern $v_1 \ldots v_\ell$ is repeated r times in a row in the sequence $\mathcal{S}$.

Consider a d-degenerate graph G, two k-colorings α, β of G with $k \geq d+2$, and a vertex u of G with degree at most d. Let α' and β' be the restrictions of α and β to $G - u$ and $\mathcal{S}$ a recoloring sequence from α' to β'. Let us now explain how we can extend this recoloring sequence to the whole graph.

Let $t_1, \ldots, t_\ell$ be all the steps where the color of any neighbor of u is modified in $\mathcal{S}$, and let c_{t_i} be the new color assigned to the recolored neighbor at step t_i. A *best choice* for u at step $t \in \{t_1, \ldots, t_\ell\}$ is:

- the color $\beta(u)$ if $\beta(u)$ is distinct from $c_t, \ldots, c_{t_\ell}$;
- or any valid choice for u at step t which is distinct from $\{c_{t_i}$ with $t_i \geq t\}$ otherwise;
- or the valid choice for u at step t that appears the latest in the sequence $(c_{t_i})_{t_i \geq t}$ otherwise.

The Local Best Choice for u extends the sequence $\mathcal{S}$ by recoloring only the vertex u. When a neighbor v of u is recolored in $\mathcal{S}$ with the current color of u, we add a recoloring for u just before this recoloring and we recolor u with a best choice. We do not perform any other recoloring for u except at the very last step to give it color $\beta(u)$ if needed.

Let G be a graph and $v_1, \ldots, v_n$ be a degeneracy ordering of G. The Best Choice Recoloring Algorithm is the algorithm which consists in making the Local Best Choice successively on $v_n, \ldots, v_1$. We call a *best choice recoloring* a sequence $\mathcal{S}$ which can be obtained from this procedure for some degeneracy ordering of G. Observe that by construction, if $\mathcal{S}$ is a best choice recoloring for some graph G and some degeneracy ordering $v_1, \ldots, v_n$, then for every index $i \geq 0$, if $V_i = \{v_i, \ldots, v_n\}$, then $\mathcal{S}_{|V_i}$ is a best choice recoloring for $G[V_i]$. In order to show how the Local Best Choice Recoloring Algorithm works, and as a warm-up, let us give a proof of Theorem 16 first stated by Bousquet and Perarnau in [29]. We first recall the theorem:

Theorem 16. *Let G be a d-degenerate graph. The diameter of the $(2d+2)$-reconfiguration graph of G is at most $(d+1)n$.*

Proof. Let α, β be a two $2d+2$ colorings of G and $v_1, \ldots, v_n$ be a degeneracy ordering of G. For $i \in 1, \ldots, n$ let $V_i = \{v_i, \ldots v_n\}$ and let α_i, β_i be the colorings induced by α and β on $G[V_i]$ respectively.

We show by induction on n that the Best Choice Algorithm outputs a reconfiguration sequence transforming α_i to β_i such that every vertex is recolored at most $d+1$ times. It is obviously true for $i = 1$. Suppose the result holds for $i = n-1$ and let $\mathcal{S}$ be a reconfiguration sequence of $G[V_{n-1}]$ transforming α_{n-1} to β_{n-1}. The Best Choice Algorithm extends this sequence to a sequence $\mathcal{S}'$ for G by recoloring v_1 with a local best choice when needed. Let $t_1, \ldots t_\ell$ be the time at which v_1 is recolored in $\mathcal{S}'$. Let us show that for any $j \leq \ell - 2$ we have $t_{j+1} - t_j \geq d$. Let $\tau_1, \ldots, \tau_q$ be the time after t_j at which a neighbor of v_1 is recolored and let $c_{\tau_1}, \ldots, c_{\tau_q}$ be the new color assigned to the recolored neighbor. Since v_1 is also recolored at step t_{j+1}, the local best choice at step t_j is the color that appears the latest in the sequence $c_{\tau_1}, \ldots, c_{\tau_q}$. Before the recoloring a time t_j, the set of distinct colors formed by the color of v_1

together with the colors of its neighbors has size at most $d+1$. It follows that there is always a valid choice of color for v_1 at time t_j that it distinct from $c_{\tau_1}, \ldots, c_{\tau_d}$ since there are $2d+2$ colors in total. Hence, the local best choice assigns some color c_{τ_p} where $p \geq d+1$ to v_1 at time t_j. It follows that there are at least $d+1$ recolorings of neighbors of v_1 before v_1 gets recolored again at time t_{j+1}.
Since each vertex is recolored at most $(d+1)$ times in $\mathcal{S}$, there are at most $d(d+1)$ recolorings of neighbors of v_1 in $\mathcal{S}$. Since v_1 is recolored only after $d+1$ recolorings of its neighbors, plus eventually one final time to its color in β, it follows that v_1 is recolored at most $d+1$ times, which concludes the proof. □

5.2.1 Properties of the Best Choice Algorithm

In what follows, we always assume that G is a chordal graph with $\omega = 3$ and $k = 5$. The ordering $v_1, \ldots, v_n$ is a perfect elimination ordering of G, and $\mathcal{S} = s_1 \ldots s_t$ is a best choice recoloring for this ordering between two 5-colorings α and β of G. We denote by V_i the set $\{v_i, \ldots, v_n\}$, and $G_i = G[V_i]$. The perfect elimination ordering of G can be used to construct an acyclic orientation of the graph, with the edge $v_i v_j$ oriented towards v_j if $j > i$. With this convention, each vertex of the graph has at most 2 out-neighbors, and for every vertex v of G, we denote by $N^+(v)$ these two out-neighbors, and $N^+[v] = N^+(v) \cup \{v\}$ the inclusive out-neighborhood. Because the ordering is a perfect elimination ordering, the out-neighbors of a vertex are adjacent.

Given a vertex v, the step i of the sequence $\mathcal{S}$ is *saved for* v if s_i recolors a vertex $w \in N^+(v)$ and one of the following holds:

- v is not recolored at steps $1, \ldots, i$;
- v is not recolored at steps $i, \ldots, t$;
- the two steps preceding s_i in $\mathcal{S}_{|N^+[v]}$ do not recolor v.

Informally, in the "worst case" scenario, when we apply the Best Choice Algorithm to extend a recoloring sequence to a new vertex v, whenever we have to recolor v because of one of its neighbors there are two possible choice of color to recolor v with. By choosing the correct color, the algorithm recolors v once every two steps at most. This is formalized in the following observation:

Observation 14. *For every vertex v and $w \in N^+(v)$, the patterns vv never occur in $\mathcal{S}_{|N^+[v]}$, and the pattern vwv can occur at most once, at the very end of the sequence $\mathcal{S}_{|N^+[v]}$.*

Saved steps quantify how many additional steps are saved by the algorithm compared to the worst case scenario. More precisely, we will show that every two saved recolorings for x reduce the number of times x is recolored by one compared with the worst case scenario. In order to prove this result, we will need one additional notion. We say that a step i recoloring a vertex v is *caused by* a vertex w if the recoloring step following s_i in $\mathcal{S}_{N^+(v)}$ recolors w. Note that by construction, this

recoloring step recolors w with the color that v had just before the transformation s_i. Moreover, all the steps recoloring v, except possibly the last one where v is given its target color, are caused by one of the out-neighbor of v.

Let us start by proving a simple inequality on the number of times that a vertex v is recolored.

Lemma 21. *For every vertex v of, if r is the number of saved recolorings for v, then the following inequality holds:*

$$|S_{|v}| \leq 1 - \frac{r}{2} + \left\lceil \frac{\sum_{w \in N^+(v)} |\mathcal{S}_{|w}|}{2} \right\rceil .$$

Proof. The statement is proved by induction on $m := \sum_{w \in N^+(v)} |S_{|w}|$. If v is recolored at most once during the sequence (which is the case when $m = 0$, i.e., when its out-neighbors are not recolored) the conclusion follows immediately. Hence, let us assume that v is recolored at least twice. If v is not the first recolored vertex of $\mathcal{S}_{|N[v]}$, then the first recoloring is saved. If we consider the coloring α' obtained after this first recoloring, and $\mathcal{S}'$ the subsequence of $\mathcal{S}$ which recolors α' to β, then by induction, v is recolored at most $\frac{(m-1)-(r-1)}{2} + 1$ times in $\mathcal{S}'$, and consequently also in $\mathcal{S}$. In this case the induction step holds, consequently we can assume in the rest of the proof that v is the first vertex recolored in $\mathcal{S}_{|N^+[v]}$.

Let us write $\mathcal{S}_{|N^+[v]} = s'_1 \dots s'_\ell$. We know that s'_1 recolors v, let us write c_0 and c_1 the colors of v respectively before and after the transformation s'_1. Since v is recolored at least twice, this recoloring must be caused by one of its out-neighbors say w, and w must be recolored in s'_2 with the color c_0. Consequently, there are still two colors that did not appeared in $N^+[v]$ in the transformation up to step 2. By Observation 14, either s'_3 recolors v, and in this case is the last step of $\mathcal{S}_{|N^+[v]}$ (i.e., $\ell = 3$); or s'_3 does not recolor v. In the first case we have $\sum_{w \in N^+(v)} |S_{|w}| = 1$, and since v is recolored twice, the inequality holds. In the second case we consider the coloring α' obtained after the transformation s'_3, and let $\mathcal{S}'$ be the subsequence starting after the step s'_3 recoloring α' into β. We know that v is recolored at one less times in $\mathcal{S}'$ than in $\mathcal{S}$. Moreover, by definition neither s'_2 nor s'_3 are saved recolorings for v, hence the number of saved recolorings for v in $\mathcal{S}'$ is still equal to r. Using the induction hypothesis on $\mathcal{S}'$, we know that v is recolored at most $\frac{m-2-r}{2} + 1$ in $\mathcal{S}'$, and since v is recolored one additional time in $\mathcal{S}$, the induction step holds. □

Hence in order to prove that a vertex is not recolored to many times, it is sufficient to show that there exists a sufficient number of saved recolorings. The following lemma gives a sufficient condition for a saved recoloring to appear.

Lemma 22. *Let u be a vertex of G and $v, w \in N^+(u)$. We write $\mathcal{S}_{|N^+[u]} = s'_1, \dots s'_\ell$. Assume that the step i of $S_{|N^+[u]}$ recolors u and is caused by v, and step $i+1$ is caused by w. Then either $i + 3 \geq \ell$ or the step $i + 3$ of $S_{|N^+[u]}$ is saved for u.*

Proof. Let c_1 (resp. c_2, resp. c_3) denotes the color of u (resp. v, resp. w) just before step i. By definition, at step $i + 1$ the vertex u takes the color c_1 (since the step i

is caused by u) and at step $i+2$, w takes the color c_2 (since $i+1$ is caused by w). Thus, after step $i+2$, the only colors which appeared for v and w at steps $i, i+1$ and $i+2$ are the colors c_1, c_2, c_3. Since we make the best choice for u and there remain two valid colors, u is not recolored at step $i+3$ (and then is saved for u) except if it is the last step of $\mathcal{S}_{|N^+[u]}$ where v_1 receives its final color. □

Lemma 23. *Let u be a vertex of G, and $v, w \in N^+(v)$. If $\mathcal{S}_{|N[u]}$ contains the pattern $uvw^{>0}u$ then the initial, intermediate and final colors of u in the subsequence corresponding to this pattern are pairwise distinct.*

Proof. If the three colors are not pairwise distinct then only the initial and final colors of u can be the same. Let c_1 be the initial color of u. Let $s'_1 \dots s'_\ell$ be the subsequence of $\mathcal{S}_{|N[v]}$ corresponding to the pattern, then s'_1 is caused by v, which implies that v is recolored in s'_2 with c_1. Since v and w are adjacent, all the subsequent recolorings of w and u use a color different from c_1, and in particular, the final color of v is different from c_1. □

Lemma 24. *Let x, u, v, w be vertices of G such that $N^+(x) = \{u, v\}$ and $N^+(u) = \{v, w\}$. If i is the first index such that $x = v_i$, let us write $c = \max_{i'>i} |\mathcal{S}_{|v_{i'}}|$. If x is recolored at least $c-1$ times in $\mathcal{S}$, then:*

1. *the pattern $uwvu$ appears at least $c - 34$ times in $\mathcal{S}_{|N^+(u)}$;*
2. *in the sequence of colors of x, there are at most 74 indices where three consecutive colors are not pairwise distinct.*

Proof. Let us prove the two points of the lemma successively.

1. Let us write $\mathcal{S}_{|N^+(u)} = s'_1, \dots, s'_\ell$. Let us start by showing that few recolorings of u are caused by v. Let i be an index such that the recoloring s'_i of u is caused by the recoloring s'_{i+1} of v.

   First observe that x is not recolored between s'_i and s'_{i+1}. Indeed, let us denote by c the color of u before s'_i. Since the step i is caused by v, this means that v is recolored with c in s'_{i+1}. Moreover, since x and u are adjacent, x is not colored c before s'_i. This implies that x is not recolored between s'_i and s'_{i+1}, as it would be a recoloring caused by v, which can only happen if v is recolored with the color of x which is not the case here.

   If s'_i is the first recoloring of a vertex in $N^+[x]$, then it is saved for x. Otherwise, the recoloring preceding s'_i in $\mathcal{S}_{|N+[x]}$ can be either:

   - a recoloring of x, in which case it is caused by u, and by Lemma 22, this implies that either s'_{i+1} is the last, or before last, step of $\mathcal{S}_{|N^+[x]}$, or the step following s'_{i+1} in $\mathcal{S}_{|N^+[x]}$ is saved for x.
   - a recoloring of either u or v, and in this case s'_{i+1} is saved for x.

This implies that every time (except once) there is a recoloring of u caused by v, we can find a saved step for x, and all these saved steps are different. Moreover, since x is recolored at least $c-1$ times, by Lemma 21 there are at most 5 saved recolorings for x, and consequently by the observation above there are at most 6 recolorings of u caused by v.

Since x is recolored at least $c-1$ times, we know by Lemma 21 that u is recolored at least $c-3$ times. At most 6 of these recolorings are caused by v by the argument above. Hence, except the last one, the other $c-10$ recolorings of u are caused by w. Moreover, since w is recolored at most c times, it also means that there are at most 10 recolorings of w which do not cause a recoloring of u.

Let us consider i and j such that s'_i and s'_j are two consecutive recolorings of u in $\mathcal{S}_{N^+[u]}$. Our goal is to show that except for a small number of choices of i, we have $j = i+3$, and $s'_i \dots s'_j$ matches the pattern $uwvu$. Observe that the following properties hold:

- By Observation 14, we have $j \geq i+2$, and $j = i+2$ can only happen once at the end of the sequence.
- If $j > i+3$, i.e., if there are at least three recolorings of v or w between s_i and s_j, then at least one of these recolorings is saved for u. This can happen at most 9 times since u is recolored at least $c-3$ times, and using Lemma 21.
- If $j = i+3$ and s'_{i+1} does not recolor w, then we have a recoloring of u which is not caused by w, which can happen at most 10 times by the arguments from the previous paragraph.
- Finally, if $j = i+3$ and s'_{i+1} recolors w, and if s'_{i+2} does not recolor v, then it must recolor w (it cannot recolor u by the assumption that $j = i+3$), in which case we have a recoloring of w which does not cause a recoloring of u, which can happen at most 10 times.

Combining all the points above, and since u is recolored at least $c-3$ times, the pattern $uwvu$ occurs at least $c-4-1-9-10-10 = c-34$ times, proving the first point of the lemma.

2. Let us now consider the second point. Since x is recolored at least $c-1$ times while its at most two out-neighbors are recolored at most c times, then by Lemma 21, there are at most 5 saved recolorings for x. Let us write $\mathcal{S}_{|N^+[x]} = s'_1 \dots s'_\ell$, and consider two indices $i < j$ such that s'_i and s'_j are two consecutive recolorings of x. Again, by Observation 14, we have $j \geq i+2$, and $j = i+2$ can occur only once at the end of the sequence. If $j > i+3$, then s_{j-1} is saved for x, which can happen at most 5 times. In all the other cases, there are exactly two recolorings of either u or v between s_i and s_j. By the point 1. above, we know that the pattern $uwvu$ appears at least $c-34$ times in $\mathcal{S}_{|N^+[u]}$. This implies that in the sequence $\mathcal{S}_{|\{u,v\}}$ there are at most 2×34 times where either two

consecutive u or two consecutive v appear as a pattern. Hence, in all but at most $68 + 5 + 1 = 74$ occurrences, the subsequence $s_i \dots s_j$ matches one of the patterns $xuvx$ or $xvux$. By Lemma 23 the three colors taken by x during this portion of the recoloring sequence are all different, which proves the second point of the lemma.

□

5.2.2 Proof of Lemma 20

Let c be a constant equal to 542. In order to prove Lemma 20, we will show that a best choice recoloring sequence recolors each vertex at most c times. This is proved by induction on the number of vertices of G. This is clearly true when G contains a single vertex since the sequence will recolor this vertex at most once. Assume that the conclusion holds for all the chordal graphs with clique number at most three on n vertices. Let G be a graph with $n+1$ vertices and let x be the first vertex in the elimination ordering. Assume by contradiction that x is recolored $c+1$ times in a best choice recoloring sequence for this elimination ordering. Using the induction hypothesis, all the vertices but x are colored at most c times.

If x has a single neighbor, then by Lemma 21, it is recolored at most $\frac{c}{2} + 1 \leq c$ times, a contradiction. Hence, we can assume that x has two neighbors y and z. Again, using Lemma 21, both y and z are recolored at least $c-1$ times in $\mathcal{S}$. Since G is chordal, yz is an edge of G, and we can assume without loss of generality that z is an out-neighbor of y. Let y_1 be the second out-neighbor of y (if it exists).

By the first point of Lemma 24, there are at most 34 recolorings of y in $\mathcal{S}_{|N^+[y]}$ where the subsequence starting at this point does not match the pattern yy_1zy. Moreover, by the second point of Lemma 24 applied to z, there are at most 74 triplets of consecutive colors of z which are not composed of pairwise distinct colors. Since both y and z are recolored at least $c-1$ times, there exists a subsequence $\mathcal{S}'$ of $\mathcal{S}_{N^+[y]}$ of the form $(yy_1z)^{c'}$ where $c' := \lfloor \frac{c-2}{108} \rfloor = 5$ and where three consecutive colors of z are always pairwise distinct.

For $X \in \{x, y, z, y_1\}$, let us denote by c_i^X the color of X after the i-th recoloring of X in the sequence $\mathcal{S}'$, with the convention that c_0^X is the initial color of X.

Note that all the recolorings of y are caused by y_1 in this subsequence, consequently, $c_{i+1}^{y_1} = c_i^y$ for all $i \leq c'$. Since we choose the best color for y when we recolor y from c_{i+1}^y to c_{i+2}^y, the set of colors $c_{i+1}^{y_1}, c_{i+2}^{y_1}, c_{i+1}^z, c_{i+2}^z$ are pairwise distinct. Indeed, if they were not, the algorithm could have chosen an other color for y, which would postponed the recoloring of y, a contradiction with the fact that we made a best choice for y. Using this fact, and the equality $c_{i+1}^{y_1} = c_i^y$, it follows that c_{i+2}^y is the unique color which is not in the set $\{c_i^y, c_{i+1}^y, c_{i+1}^z, c_{i+2}^z\}$, or stated differently, the colors $c_i^y, c_{i+1}^y, c_{i+1}^z, c_{i+2}^y, c_{i+2}^z$ are all pairwise disjoint. Writing this property for the index $i+1$ gives that $c_{i+1}^y, c_{i+2}^y, c_{i+2}^z, c_{i+3}^y, c_{i+3}^z$ are all pairwise disjoint. With the overlap between these two sets, it follows that we must have $\{c_i^y, c_{i+1}^z\} = \{c_{i+3}^y, c_{i+3}^z\}$.

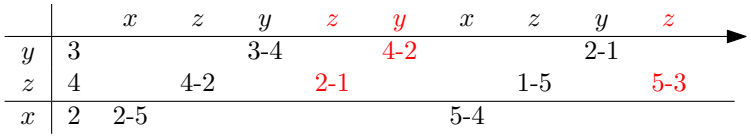


FIGURE 5.1: Example for $\mathcal{S}'_{|\{y,z\}}$ and the best choices it implies for x. Initial colors of x, y, and z are 2, 3 and 4 respectively. Recolorings are ordered from left to right. The recolorings in red are saved for x.

Moreover, since $c^z_{i+1} \neq c^z_{i+3}$ by assumption on the sequence $\mathcal{S}'$, then we must have $c^z_{i+3} = c^y_i$, and $c^z_{i+1} = c^y_{i+3}$. Hence, we can see from these conditions that the vertex y takes the colors $1, 2, 3, 4, 5$ successively (up to a permutation of colors), and similarly for y_1 and z, but with a shift of 1 and 3 respectively compared to y.

We can now use the fact that $\mathcal{S}'$ is very constrained to show that there must be a sufficient number of saved recolorings for x, which will contradict the assumption that x is recolored at least $c+1$ times. Let us first assume that there is a recoloring of x caused by the recoloring of y from c^y_i to c^y_{i+1}. This means that x is colored c^y_{i+1}, and is recolored with a color not in the set $\{c^y_i, c^y_{i+1}, c^z_i, c^z_{i+1}\}$. However, we also know that $c^y_{i+2} = c^z_i$, which implies that the recoloring of y from c^y_{i+1} to c^y_{i+2} will be saved for x. In a similar way, if the recoloring of x is caused by a recoloring of z from c^z_i to c^z_{i+1}, then the new color of x will not be in the set $\{c^z_i, c^z_{i+1}, c^y_{i+1}, c^y_{i+2}\}$. And since $c^z_i = c^y_{i+2}$, the recoloring of z from c^z_{i+1} to c^z_{i+2} will also be saved. An example of such a sequence $\mathcal{S}'$ is given in Figure 5.1.

Hence, in the sequence $\mathcal{S}'$, for every recoloring of x, there is at least one recoloring saved for x. Hence, either x is recolored at most $c'-2$ times during $\mathcal{S}'$, in which case there are at least 2 saved recolorings for x, or x is recolored at least $c'-2=3$ times during $\mathcal{S}'$, and by the argument above at least two of these recolorings cause a recoloring saved for x (the last one might be at the end of $\mathcal{S}'$). In all cases, we obtain two saved recolorings for x, which is a contradiction of the assumption that x is recolored at least $c+1$ times. Hence x is recolored at most c times and the inductive step holds.

Part II

Independent set reconfiguration

Chapter 6

Independent set reconfiguration

In the first chapter of this thesis, we described the 15-puzzle game as an example of a reconfiguration problem. Recall that in this puzzle, the player is given a board along with some blocks that can be slid on the board. The blocks are initially placed arbitrarily on the board and the goal is to reach a target configuration by sliding the blocks one after another. This problem can be easily reformulated has a single-player game on graphs. The board is the $n \times n$ grid graph for $n \geq 0$, each of the $n^2 - 1$ block is a labeled token that is placed on a vertex of the grid and there is a unique vertex of the graph with no token on it. We say that this vertex is *empty*. At each turn, the player is allowed to move a token which is on a neighbor of the empty vertex to the empty vertex. In other words, the player is allowed to *slide* a token along an edge at each turn as long as no two tokens are on the same vertex. An illustration is given if Figure 6.1

Starting from this model of the sliding block puzzle as a graph problem, one can define other games by modifying the input graph, reducing the number of tokens, or by adding more constraints on the positions that the tokens can have or go to. This is how Hearn and Demaine [56] first introduced the *sliding token* problem, which can be seen as the first formalization of the independent set reconfiguration problem. They formulate the problem as follows: the player is given a graph G and a set of tokens positioned on an independent set of G. The player is allowed to slide a token along an edge at each step, as long as the resulting configuration (set of vertices on which the tokens are) is also an independent set. The problem is the following: given an initial configuration and a particular token t of this configuration, is there a sequence of moves after which the token t can be moved? Since the seminal paper of Hearn and Demaine, the notations and names of the problems changed and this problem is today referred to as the Rigid Token problem. Let us outline the fact that the authors of [56] initially defined this problem as an equivalent formulation of a new model of computation they designed, the *Nondeterministic Constraint Logic* or *NCL* for short, to prove PSPACE-completeness of reconfiguration problems. As we will see in the next sections, this model is indeed very powerful and is at the basis of numerous PSPACE-completeness proofs for reconfiguration problems. Note that when this article was written the notion of reconfiguration problems had not been formalized yet, and the authors rather talk of puzzles.

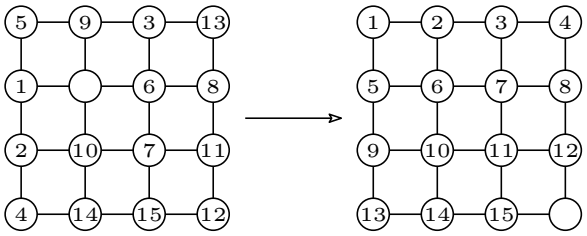


FIGURE 6.1: A representation of the 15-puzzle game on the 4×4 grid. An initial configuration is represented at the left, the target configuration is represented at the right. Each token is represented by its label, and a token can slide to a neighboring vertex if and only if it contains no token.

In this framework, a natural question arises: given a graph G, two independent sets I and J of G and a set of tokens positioned on the vertices of I, is it possible to transform I into J by sliding one token at a time along edges of G, while maintaining an independent set all along the transformation? This problem is known as the TOKEN SLIDING (TS) problem. The paternity of this definition is given to Hearn and Demaine [56] even if they only define the RIGID TOKEN problem and do not explicitly state the TOKEN SLIDING problem as it is known today.

However, *independent set reconfiguration* does not only refer to the TOKEN SLIDING problem. Indeed, there exist different rules according to which the tokens can be moved on the graph, sliding a token along an edge being just one of these. Each rule defines a particular notion of adjacency between the configurations. Hence, independent set reconfiguration refers to a whole set of problems, each one defined by its rule. Let us briefly describe here the rules that are the most studied in the literature. The formal definitions are given in the next section.

Along with TOKEN SLIDING, another well-studied rule is the TOKEN JUMPING (TJ) rule. In this version of independent set reconfiguration, first introduced by Kamiński et al. in [68], one is allowed at each step to move a token anywhere on the graph (as long as the obtained set remains independent). Note that if there exists a reconfiguration sequence between two independent sets I and J of a graph with the TS rule, then there also exists one for the TJ rule. The other direction is not true, as illustrated in Figure 6.2. Finally, the other rule that appears the most in the literature is the TOKEN ADDITION-REMOVAL (TAR) rule, first introduced by Ito et al. in [63]. In this version of the independent set reconfiguration problem, the input is a graph G, an integer $k \geq 0$ and two independent sets I and J of size at least $k+1$. At each step, one is allowed to either remove or add a token anywhere on the graph, as long as the obtained set has size at least k and is independent. As will we see in the next sections, this rule is actually equivalent to token jumping, as proven by Kamiński et al. in [67].

Let us note that these three rules can be defined in a much more general setting rather than just independent set reconfiguration. The TOKEN SLIDING and TOKEN

JUMPING problems can be defined equivalently for any optimization problem seeking for a set S that satisfies some property P (such as VERTEX COVER, DOMINATING SET, VERTEX SEPARATOR...). Informally, the question is the following: given two sets S and T satisfying property P, is it possible to move the tokens from S to T one after another (by either sliding or jumping) while always having a set that satisfies the property P?
The same can be done for TOKEN ADDITION-REMOVAL. The only difference being that the input comes with an integer k which is either a lower bound (if the corresponding optimization problem is a maximization problem) or an upper bound (if the corresponding optimization problem is a minimization problem) on the number of tokens present on the graph at all time. Without this threshold, TOKEN ADDITION-REMOVAL becomes trivial, since one only has to remove all the tokens on S one by one and then add the tokens on T (or vice-versa) to find a reconfiguration sequence between the sets S and T of the input graph.

6.1 Definitions

Let $G = (V, E)$ be a graph and let I, I' be two independent sets of G. The sets I, I' are *TJ-adjacent* if their exists $\{u, v\} \subseteq V(G)$ such that $I - I' = \{u\}$ and $I' - I = \{v\}$. If furthermore (u, v) is an edge of G then I and I' are *TS-adjacent*. Finally, I and I' are *TAR-adjacent* if either $I \subset I'$ and $|I' - I| = 1$ or $I' \subset I$ and $|I - I'| = 1$. For $\mathcal{T} \in \{TJ, TS, TAR\}$ we refer to the $\mathcal{T}$-adjacency binary relation as the *rule* $\mathcal{T}$ or simply as a *rule* when $\mathcal{T}$ is obvious from context.
Let now $\mathcal{S} = I_1, \ldots, I_t$ be a sequence of independent sets of G and $\mathcal{T} \in \{TJ, TS, TAR\}$. The sequence $\mathcal{S}$ is a $\mathcal{T}$-sequence if for every $i < s$ the independent sets I_i and I_{i+1} are $\mathcal{T}$-adjacent. We say that the sequence $\mathcal{S}$ is a $\mathcal{T}$-reconfiguration sequence from I_1 to I_s, or equivalently that $\mathcal{S}$ transforms I_1 into I_s according to rule $\mathcal{T}$. When the rule $\mathcal{T}$ is obvious from context, we simply say that it transforms I_1 into I_s.
In the independent set reconfiguration problems, an independent set is often seen as a set of tokens that lie on the vertices of the independent set. With this interpretation, a $\mathcal{T}$-reconfiguration sequence is a sequence of moves of these tokens according to rule $\mathcal{T}$: In a TJ-sequence, a token can be 'jumped' anywhere on the graph at each step. In a TS-sequence, a token can only be slid along an edge at each step. Finally, a step in a TAR-sequence consists in either removing or adding a token anywhere on the graph.
Let us now properly state the main questions of independent set reconfiguration. Let $\mathcal{T} \in \{TJ, TS\}$:

$\mathcal{T}$-REACHABILITY
Input: A graph G and two independent sets S, T of G.
Question: Does there exists a $\mathcal{T}$-sequence that transforms S into T?

Given a graph G, two independent sets S, T of G and a rule $\mathcal{T} \in \{TJ, TS\}$, the notation $S \leftrightsquigarrow_{\mathcal{T}(G)} T$ indicates that (G, I, J) is a YES instance for $\mathcal{T}$-reachability. We sometime only write $S \leftrightsquigarrow_{\mathcal{T}} T$ when the graph G is clear from context.

For the TAR rule, the problem is defined differently. Indeed, there always exists a TAR-sequence between any two independent sets of a graph, which consists in removing all the tokens on the initial independent set and then adding all the tokens on the target independent set in an arbitrary order. Hence, the input of the TAR rule comes with an integer k which is a lower bound on the number of tokens that must be present on the graph at any time:

TAR-Reachability

Input: A graph G, an integer k, and two independent sets S, T of G of size at least $k + 1$.

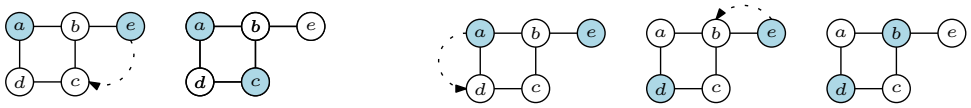
Question: Does there exists a TAR-sequence that transforms S into T such that every independent set in the sequence has size at least k?

Given a graph G, an integer k, and two independent sets S, T of G, the notation $S \leftrightsquigarrow_{TAR_k} T$ indicates that (G, k, S, T) is a YES instance for TAR-reachability. In what follows, we always assume that the TAR rule is given along with a threshold k and denote this rule as TAR_k.

Let G be a graph and I be an independent set of G, and $\mathcal{T} \in \{TJ, TS, TAR_k\}$. A vertex $v \in I$ is *frozen* (with respect to the rule $\mathcal{T}$ and the independent set I) if for any independent set I' that is $\mathcal{T}$-adjacent to I, we have $v \in I'$. A vertex $v \in I$ is *rigid* if for every independent set I' such that $I \leftrightsquigarrow_{\mathcal{T}} I'$, we have $v \in I'$. In other words, if a set of tokens is positioned on the vertices of I, then v is frozen if no sequence moving the token on v in one step exists, and v is rigid if it belongs to any independent set of a sequence starting from I. We often mix-up the vertex v and the token that lies on it and say that the token is frozen or that the token is rigid.
As for any reconfiguration problem, we can define reconfiguration graphs for independent set. Given a rule $\mathcal{T} \in \{TJ, TS\}$, a graph G and an integer k we define the *k-reconfiguration graph of G for rule $\mathcal{T}$*, and denote by $\mathcal{T}_k(G)$ the graph which vertex set is the set of all independent sets of G of size k, two independent sets being adjacent in $\mathcal{T}_k(G)$ if and only if they are $\mathcal{T}$-adjacent.
For the sake of simplicity, we will refer to the $\mathcal{T}$-Reachability as the $\mathcal{T}$ problem.

6.2 Basic results and complexity

In this section, we give some basic properties of the independent set reconfiguration problem and describe how the three rules mentioned above relate to each other.



(A) Example of token jump

(B) Example of token slide

FIGURE 6.2: A TJ-sequence from $\{a, e\}$ to $\{a, c\}$ at the left, and a TS-sequence from $\{a, e\}$ to $\{d, b\}$ at the right. Tokens are represented in blue. Note that there exists no TS-sequence from $\{a, e\}$ to $\{a, c\}$ since the vertex c has a token in is neighborhood in any TS-sequence starting from $\{a, e\}$.

Let us first begin by sketching the proof of an important result of Kamiński et al. [67] showing that the token jumping and token addition-removal rules are actually equivalent:

Theorem 22. *Let G be a graph and S, T be two independent sets of G of size $k+1$. There exists a TJ-sequence transforming S into T if and only if there exists a TAR_k-sequence transforming S into T.*

Proof. A move from a vertex x to a vertex y in a TJ-sequence can be simulated in a TAR_k-sequence by first removing the token on x and then adding the token on y. Hence if there exists a TJ-sequence, then there exists a TAR_k-sequence, and this sequence only goes through independent sets of sizes $k+1$ and k.
Let us now prove the converse part of the statement. Suppose first that there exists a TAR_k-sequence such that every independent sets have size either $k+1$ or k. Since the initial and target independents sets have size $k+1$, the sequence necessarily alternates between removal and addition of tokens. Since every two such moves can be replaced by a single token jump, there also exists a TJ-sequence. Suppose now that there exists an independent set of size $> k+1$ in the TAR_k-sequence $\mathcal{S}$. Since the target independent set has size $k+1$, there must exists two consecutive moves in the sequence where the first one is an addition and the second one is a removal. Then, it is easy to see that inverting these two moves yields a valid TAR_k-sequence transforming S into T. By applying this transformation whenever it is possible, we obtain at some point a sequence $\mathcal{S}'$ from S to T where the maximum size of an independent set in $\mathcal{S}'$ is strictly lower than the maximum size of an independent set in $\mathcal{S}$. Since this process can be applied as long as the sequence contains an independent set of size $> k+1$, we ultimately obtain a sequence containing only sets of sizes k and $k+1$, which concludes the proof. □

This result shows not only that $S \leftrightsquigarrow_{TJ} T$ if and only if $S \leftrightsquigarrow_{TAR_k} J$, but also that a TJ-sequence can be transformed in TAR_k-sequence in polynomial-time and vice-versa. Despite these two models being equivalent, it is sometimes more handy to make use of one of the formulation rather than the other, and thus both rules are still studied today (see for instance [16, 21, 71] for the token addition-removal rule and [23, 63, 67, 98] for the token jumping rule). Indeed, the TAR rule allows for more flexibility on the moves of the tokens (the operation of moving a token somewhere else

(A) An OR vertex (B) An AND vertex

FIGURE 6.3: Examples of configuration for AND/OR vertices. Edge C of the OR vertex cannot be flipped since both blue edges are directed outward. Edge C of the AND vertex can be flipped since both red edges are directed inward.

on the graph being split in two moves), but the token jumping rule is often easier to analyze since the cardinality of two TJ-adjacent independent sets is always the same. In order to keep this chapter as concise as possible, we only consider from now on the token jumping formulation of these two equivalent rules.

On the other hand, any TS-sequence is also a TJ-sequence, but it is easy to come up with examples where there exists a TJ-sequence and there exists no TS-sequence, as illustrated in Figure 6.2. These simple examples show that the problems are not equivalent. However, they are as hard as each other in the general case, both problems being PSPACE-complete. As mentioned in the introduction, one major breakthrough towards this problem is the introduction of the Nondeterministic Constraint Logic (NCL) model of computation by Hearn and Demaine in [56]. Let us briefly describe this tool.
A NCL machine is a weighted graph $G = (V, E)$ with weight function $w : E \to \mathbb{N}$ given along with a *minimum in-flow constraint function* $f : V \to \mathbb{N}$. A *valid configuration* of the machine is an orientation A of the edges, such that for each vertex $v \in V(G)$, the sum of the weights of the arcs in A pointing toward v is at least $f(v)$. A configuration A' is *adjacent* to A if it is also valid, and if it differs from A by the orientation of exactly one edge e. The operation consisting in transforming the configuration A into the configuration A' is the *flipping* of the edge e. Note that the definition itself already yields a reconfiguration problem: given two configurations A and B of an NCL machine, is it possible to transform A into B via a sequence of edge flips? In [56], Hearn and Demaine show that this problem is PSPACE-complete by a reduction from the QUANTIFIED BOOLEAN FORMULA (QBF) problem. To do so, they consider a particular type of NCL machine which they call the *AND/OR constraints graphs*. This machine contains two types of vertices:

- The OR vertices, which have minimum in-flow constraint of 2 and are adjacent to three edges of weight 2.
- The AND vertices which have minimum in-flow constraint 2 and are adjacent to two edges of weight 1 and one edge of weight 2.

Edges of weight one are often called the *red edges* and edges of weight two are often called the *blue edges*. The OR and AND vertices are illustrated in Figure 6.3. The name of the vertices are not chosen at random: an AND vertex v behaves as a logical AND, since the blue edge can be directed outward of v if and only if the two red edges point toward v. Similarly, an OR vertex behaves as a logical OR: one of the three blue edges can be directed outward of v if and only if one of the two others points toward v. By plugging these two types of vertices together, the authors of [56] then show how to obtain the universal $\forall$ and the existential $\exists$ quantifiers as AND/OR constraints graph. Building on that (and several other gadgets), they ultimately show how to construct a NCL instance starting from a QBF instance. They furthermore show by designing uncrossing gadgets that any AND/OR graph can be made planar. This leads to the following theorem:

Theorem 23. *Token Sliding is* PSPACE*-complete, even when restricted to planar graphs of maximum degree* 3*.*

Let us outline the importance of this proof in our framework: the proof of Hearn and Demaine can be considered to be the first PSPACE-hardness proof of independent set reconfiguration problems. It is still today the starting point of many reductions for reconfiguration problems, see for instance [2, 11, 58, 60, 64, 81, 82, 90] (not only independent set reconfiguration). We will describe these results with more details in the next sections.
The proof of PSPACE-completness for Token Jumping had to wait for another important result: the proof of PSPACE-completness of the Satisfiability Reconfiguration problem [50]. In this problem, we are given a Boolean formula ϕ in conjunctive normal form and two satisfying truth assignments s and t of the variables of ϕ. The question is to decide whether one can transform s into t by modifying the value of one variable at a time, while always maintaining a satisfying assignment. Ito et al. started from this problem to prove PSPACE-completness of several reconfiguration problems in [63], among which Token Jumping. Later on, Ito et al. [64] actually strengthened this result by proving the following:

Theorem 24. *Token Jumping is* PSPACE*-complete, even when restricted to planar graphs of maximum degree* 3*.*

These two theorems show how hard reconfiguration problems can be, even when restricted to very particular cases. However, these first results also raise new questions: do there exist graph classes where the Token Jumping and Token sliding problems become easy? Where is the border between easy and PSPACE-complete? And to what extent are both problems equivalent from a complexity point of view? As we will see in the next section dedicated to the study of the two rules in graph classes, token jumping and token sliding sometimes behave (somewhat surprisingly) in very different ways.

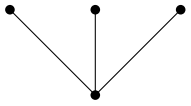
6.3 Independent set reconfiguration on graph classes

All the results described in this section are summarized in Table 6.1. In the article introducing the token jumping rule [67], Kamiński et al. show that both token jumping and token sliding reachability problems remain PSPACE-complete on perfect graphs. By the Strong Perfect Graph Theorem, perfect graphs are the graphs with no odd holes (odd-length induced cycles) nor odd antiholes (complements of odd holes). A natural question then concerns the complexity of the two rules on graph classes that forbid other induced subgraphs.

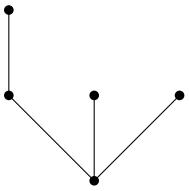
6.3.1 Excluding induced subgraphs

In the article proving the PSPACE-completeness of the two rules in perfect graphs [67], Kaminiski et al. also show that Token Jumping can be solved in linear time for even-hole-free graphs. A graph is *even hole-free* if it has no induced cycles of even length. The proof is short and elegant: Consider the starting and target independent sets S and T of an even hole-free graph G. Since S and T are independent sets, their symmetric difference induces a bipartite graph H. Since furthermore G is even hole-free, the graph H also is, and is therefore a forest. It follows that there is always a vertex $v \in T$ of degree at most one in $S\Delta T$ (up to switching the roles of S and T and considering S as the target independent set). Hence, there always exists a token that can be jumped to v (the token lying on its unique neighbor if it exists, or an arbitrary token otherwise) and the size of $S\Delta T$ can be reduced. The proof follows since such a vertex v can be found in linear time.
Note that this argument cannot work for the token sliding rule since if v is isolated in $S\Delta T$, nothing ensures that a token can be slid to v. The Token Sliding problem in even-hole-free graphs is actually PSPACE-complete, since it is PSPACE-complete on split graphs [11], which is a subclass of even-hole-free graphs. The proof of Belmonte et al. consists in a reduction from the NCL problem described in the previous section. The case of even hole-free graphs is, to the best of our knowledge, the only case where the complexity of the maximum independent set problem is still unknown for the graph class whereas the complexity for both the token jumping and token sliding rules is known. It is also an interesting case for the difference of complexity between the two rules, Token Jumping being linear-time solvable (and actually trivial for the case of split graphs) whereas Token Sliding is PSPACE-complete.

In [67], Kaminiski et al. also ask the question of the complexity of independent set reconfiguration in another well-studied class of graphs, the claw-free graphs. A *claw* is a graph on four vertices, with a vertex of degree 3 adjacent to three pairwise non-adjacent vertices of degree 1. An illustration is given in Figure 6.4a. Bonsma et al. tackle this problem in [23] and show that both Token Jumping and Token Sliding can be decided in polynomial time for connected claw-free graphs. They in



(A) A claw



(B) A fork

fact show that any token jump can be simulated by a sequence of token slides, and thus that the two decision problems are actually the same in this class. Let us briefly explain how this later proof goes. Let I and J be two independent sets of a connected claw-free graph G that differ on the position of exactly one token, which lies on u in I and on $v \neq u$ in J. Then moving the token from u to v is obviously a valid token jumping move. Let us sketch the proof that there also exists a sequence of token slides that transforms I into J. Consider a shortest $u - v$ path in G. If there are tokens on the paths, then these tokens can switch roles with the token on u: slide these tokens along the path to put a token on v, then slide the token on u on the path in order to replace the token that now lies on v. Another problem arises if there is a token on the neighborhood of the path. Since it is shortest, this path is induced. The authors of [23] use this property and the claw-freeness of G to show that the neighborhood of such a token on the path has a particular structure, always allowing for the "switching roles" argument to work as in the previous case.

Although there are many examples where the complexity of Token Jumping and Token Sliding are the same, it is less common to actually have an equivalence between the two problems. And as we will see with the case of P_4-free graphs (or cographs), the two rules sometimes behave very differently despite having similar complexity even on restricted graph classes. It is well-known that a P_4-free graph G can always be recursively constructed from single vertices, or from two P_4-free graphs G_1 and G_2 by either *joining* G_1 *to* G_2[1], or by taking G to be the disjoint union of G_1 and G_2. Kamiński et al. [67] make use of this decomposition theorem to prove that Token Sliding can be decided in polynomial time on P_4-free graphs. The proof essentially makes use of the following simple argument: the token sliding problem can always be solved separately on connected components of a graph. Hence, if G arises from a disjoint union of two graphs G_1 and G_2, one can simply solve the problems separately on G_1 and G_2. If G arises from the join of G_1 and G_2, then the problem is also simple: if there are more than two tokens on one of G_1 or G_2, say G_1, then the problem can be solved separately on G_1: indeed, by definition of the join, every vertex of G_2 sees every vertex of G_1 and thus no token can slide from G_1 to G_2 in this case. Furthermore, if there is at most one token on both subgraphs then the problem is trivial, concluding the proof.

Such arguments completely fail when trying to tackle the token jumping rule

[1] in other words, adding edges between every pair of vertices of G_1 and vertices of G_2

on P_4-free graphs, for the simple reason that tokens can jump from one connected component to one another. However, Bonsma proved in [21] that this problem can also be solved in time $O(n^2)$, through the design of a very involved dynamic programming algorithm. His algorithm can be generalized to any class of graph that is obtained by successive join and disjoint union operations starting from a collection of graphs on which Token Jumping can be decided efficiently, and on which a maximum independent set can be computed in polynomial time. Two major examples of such graph classes given by Bonsma are the claw-free graphs and the chordal graphs.
Bonamy and Bousquet [16] then completed the picture for P_4-free graphs. They showed that given a P_4-free graph G, (i) it can be decided in polynomial time whether $TJ_k(G)$ is connected and (ii) given two fixed independent sets I and J of a P_4-free graph, it can be decided in linear time whether $I \leftrightsquigarrow_{TJ} J$. Their proof makes use of a classical proof technique: they characterize a unique canonical independent set for each connected component of the reconfiguration graph that can "easily" be reached from any other independent set of the same component. Then, deciding whether $I \leftrightsquigarrow_{TJ} J$ boils down to comparing the canonical independent sets of I and J.

6.3.2 Trees, chordal graphs, bounded treewidth graphs and their relatives

A classical starting point when studying a problem on graph classes is the class of trees and its generalizations. Let us first recall that the generalization of cographs introduced by Bonsma in [21] includes chordal graphs, and thus Token Jumping can be solved in polynomial time in this class, and hence in trees.
The complexity of token sliding on trees has been settled by Demaine et al. in [35]. They show that deciding whether an independent set S can be transformed into another independent set T via token sliding can be decided in linear time. As the authors of [35] underline, linear-time is remarkable here since there exists an infinite family of instances of paths which requires $\Omega(n^2)$ slides. Hence, the decision problem is solvable in time $O(n)$, whereas finding the actual sequence requires time $\Omega(n^2)$. A large majority of proofs showing that an independent set reconfiguration reachability problem is in P consists in the design of an algorithm that outputs a sequence of polynomial length (Token Jumping on claw free-graphs [23], Token Sliding and Token Jumping on P_4-free graphs [16, 21] , Token Jumping on even-hole free graphs [67] are just a few examples among many others). It is then interesting to note that for trees, no algorithm constructing the sequence in linear time exists.
The fact that Token Sliding is PSPACE-complete on split graphs [11] implies that it is also PSPACE-complete on chordal graphs. However, Yamada and Uehara showed that that Token Sliding can be solved in polynomial time on proper interval graphs [99], and this result was later generalized by Bonamy and Bousquet for general interval graphs in [17]. Their proof is one of the few cases (along with the proof for trees mentioned previously) where the reachability problem is solved in polynomial time

Graph Class	Token Jumping	Token Sliding
Planar graphs with $\Delta = 3$	PSPACE-complete [64]	PSPACE-complete [56]
Perfect graphs	PSPACE-complete [67]	PSPACE-complete [67]
Split graphs	Linear	PSPACE-complete [11]
Even hole-free graphs	Linear [67]	PSPACE-complete [11]
Claw-free graphs	P [23]	P [23]
P4-free graphs	P [16, 21]	P [67]
Trees	Linear [35]	Linear
Chordal graphs	P [21]	PSPACE-complete [11]
Interval graphs	P [21]	P [17]
Bounded bandwidth	PSPACE-complete [98]	PSPACE-complete [98]
Bipartite graphs	NP-complete [71]	PSPACE-complete [71]

Table 6.1: Complexity of Token Jumping and Token Sliding on several graph classes.

without actually outputting a sequence between the source and target independent sets. However, Brianski et al. [30] recently showed that if two independent sets S, T of an interval graph G belong to the same connected component of $TS_k(G)$, then there always exists a TS-sequence of size $O(k \cdot n^2)$ between S and T (where k is the size of S and T). They also give a polynomial-time algorithm that outputs such a sequence.

Since the class of forests is the class of graphs of treewidth 1, it is natural to ask to ask whether a polynomial-time algorithm on trees can be generalized to a polynomial-time algorithm on bounded treewidth graphs. Wrochna answered this question by the negative in [98], by proving the following:

Theorem 25. *There exists an integer b such that Token Sliding and Token Jumping are PSPACE-complete on graphs of bandwidth at most b.*

Since the treewidth of a graph can be bounded by its bandwidth, it follows that the reachability problems defined by both rules are PSPACE-complete on bounded-treewidth graphs. We note however that the proof of Wrochna does not explicit b. To the best of our knowledge, it is only known that $b > 1$ (by the algorithm designed for trees). Hence the following question:

Question 10. *What are the largest integers b (resp. b') for which Token Jumping (resp. Token Sliding) can be solved in polynomial time on graphs of treewidth at most b (resp b')?*

A first step towards answering Question 10 would be to study the case of graphs of treewidth at most 2 (also known as the class of serie parallel graphs). Hoang and Uehara gave a partial answer to this question in [59] by designing a polynomial-time algorithm for token sliding on cactus, a subclass of serie parallel graphs. A graph is a *cactus* if it is connected and if every two simple cycles share at most one vertex. Cactus graphs are themselves a subclass of outer-planar graphs, for which the complexity of the reachabilty problem remains unknown for both rules:

Question 11. *Can* Token Jumping *and* Token Sliding *be decided in polynomial time on outerplanar graphs?*

6.3.3 A word on bipartite graphs

We conclude this section by briefly mentioning the case of bipartite graphs. In [71], Lokshanov and Mouawad study both token sliding and token jumping reachability problems on this class. They obtain the following results: Token Sliding on bipartite graphs is PSPACE-complete, whereas Token Jumping on bipartite graphs is NP-complete. This later result is rather surprising since there are few natural reconfiguration problems that belongs to NP. By "natural" we mean here that it is not specifically designed to belong to this class. Let G be a bipartite graph of size k. The proof that Token Jumping belongs to NP actually shows that a connected component of $TJ_k(G)$ has size polynomial in the order of G, therefore showing that there always exists a certificate that can be checked in polynomial time.
Let us conclude by mentionning a result of Fox-Epstein et al. who showed in [46] that Token Sliding is polynomial-time solvable on bipartite permutation graphs. The complexity of this problem on general permutation graphs remains open.

6.3.4 Our contribution

In this thesis, we study the complexity of Token Jumping and Token Sliding in graphs without a graph H as an induced subgraph, also denoted as H-free graphs. We adapt a famous proof of Alekseev [4] to the reconfiguration framework and show the following:

Theorem 6. *Token Jumping and Token Sliding on H-free graphs are* PSPACE*-complete, unless H is a path, a claw, or a subdivision of the claw.*

Consider the Token Jumping problem with the additional constraint that both source and target independent sets are maximum. Note that in this problem, any intermediate independent set in a sequence is also maximum, and thus a token can only jump to one of its neighbors. It follows that with the additional constraint of having maximum independent sets as source and target, Token Jumping and Token Sliding define the same problem. We refer to this problem as Maximum Independent Set Reconfiguration.
As stated previously, Bonsma et al. showed in [23] that both sliding and jumping

reachability problems are solvable in polynomial-time in claw-free graphs. A claw with one subdivided edge is called a *fork* (or sometime a *chair* in the literature). An illustration is given in Figure 6.4b. We pursue their work and show the following:

Theorem 26. *Maximum independent set reconfiguration is solvable in polynomial-time on fork-free graphs.*

The proofs of Theorem 26 and Theorem 6 are given in Chapter 7.

6.4 Parameterized complexity

Independent set reconfiguration problems have also been thoroughly studied from a parameterized complexity point of view in the past ten years. One of the barriers for tractability in the classical complexity setting is the existence of reconfiguration sequences of exponential length. Then, considering "small" independent sets or restricting the reachability problems to the finding of "short" sequences intuitively gives room for the design of efficient algorithm for token sliding or token jumping. Hence, the two natural parameters in the reconfiguration setting are the size k of the independent set and the length ℓ of the reconfiguration sequence. Let us also note that since the bandwidth is an upper bound on the treewidth, pathwidth, and cliquewidth, Theorem 25 shows that no FPT algorithm exists with respect to these classical parameters. This further explains why most recent work focus on the parameters k and ℓ.
The study of the parameterized complexity of reconfiguration problems was initiated by Mouawad et al. in [79]. This seminal article covers a broad range of reconfiguration problems and we only mention the results obtained by the authors of [79] concerning independent set reconfiguration in this section. By a reduction to the Independent Set problem parameterized by k, Mouawad et al. show the following:

Theorem 27. *Token Jumping is* W[1]*-hard when parameterized by* $k + \ell$.

The proof of theorem 27 being quite involved, we sketch here the proof of a weaker result due to Ito et al. [64]:

Theorem 28. *Token Jumping is* W[1]*-hard when parameterized by* k.

The proof of Theorem 28 consists in a reduction from Independent Set with parameter k. Let us briefly sketch this proof. Let (G, k) be an instance of Independent set. We show how to construct an equivalent instance $(G', k+1, S, T)$ of Token Jumping, where S and T are the source and target independent sets respectively. The graph G' consists in the disjoint union of the graph G and a biclique of size $k+1$. The source and target independent sets are the two sides of the biclique. If there exists an independent set I of size at least k in G, then one can jump k tokens from the biclique to I one by one in an arbitrary order. The last token remaining on the biclique can then jump to any vertex of T, and the k tokens on I can finally jump to

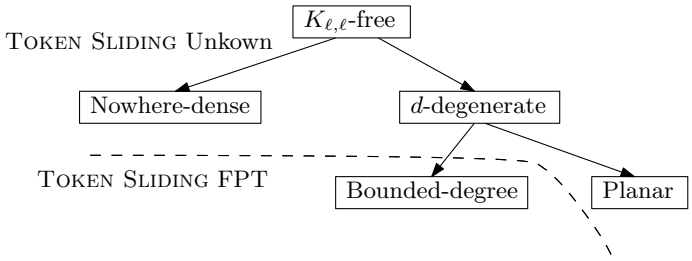


Figure 6.5: Several sparse graph classes and their inclusions. Token Jumping is FPT when parameterized by k on all these classes. The dashed line separates the classes for which we know Token Sliding to be FPT from the classes for which the complexity remains unknown (with respect to the parameter k).

the other vertices of T. Hence there exists a reconfiguration sequence from S to T in G'. Conversely, suppose that there exists a reconfiguration sequence from S to T in G', and consider the independent set I' of this sequence right after a token first jumps to a vertex of T (which exists by supposition). Since $S \cup T$ induces a biclique in G', no token of I' belongs to S, and thus there is only one token on T in I'. Hence there are k tokens on the vertices of G, and therefore G contains an independent set of size k. This concludes the proof.

Apart from these two general results for token jumping and sliding, most of the work so far focused on token jumping on sparse graph classes with parameter k. Let us show through a simple example how many tractability proofs goes for token jumping in such classes. In [64], Ito et al. show that Token Jumping is FPT when parameterized by k on bounded-degree graphs as follows: Let $G = (V, E)$ be a graph of maximum degree Δ and S, T be the source and target independent sets. Then, $N[S \cup T]$ has size $2k\Delta$ at most. If the graph induced by $V - N[S \cup T]$ has size more than $f(k, \Delta)$ for some well-chosen function f, it has an independent I set of size k (which can be computed in polynomial time). Jumping the tokens one by one from S to I and then jumping the tokens one by one from I to T yields a valid reconfiguration sequence, since I does not intersect the closed neighborhood of S and T. We can thus assume that the graph has size at most $2k\Delta + f(k, \Delta)$, which concludes the proof.
More generally, one can use the sparsity of the input graph to find such a "buffer space" I where the tokens can be jumped, or use the fact that no such set exists to bound the size of the graph. This simple strategy has been proved several time to be very efficient for Token Jumping [28, 61, 73]. Ito et al. first proved the tractability of Token Jumping on planar graphs in [61], which was then generalized to d-degenerate graphs by Mouawad et al. in [73]. Later on, Siebertz showed in [87] that Token Jumping is also fixed-parameter tractable on *nowhere-dense* graphs. A class of graphs $\mathcal{C}$ is nowhere-dense if there are functions $N : \mathbb{N} \to \mathbb{N}$ and $s : \mathbb{N} \to \mathbb{N}$ such that for all $r, m \in \mathbb{N}$ and all subsets $A \subseteq V(G)$ for $G \in \mathcal{C}$ of size $|A| \geq N(r, m)$ there is a set $S \subseteq V(G)$ of size $|S| \leq s(r)$ and a set $B \subseteq A \setminus S$ of size $|B| \geq m$ which is r-independent in $G - S$. Siebertz actually showed that Token Jumping on

Graph Class	Token Jumping	Token Sliding
$\{C_3, C_4\}$-free graphs	FPT (Chapter 8)	Open
C_4-free graphs	W[1]-hard (Chapter 8)	W[1]-hard (Chapter 8)
Bipartite graphs	Open	W[1]-hard (Chapter 8)
Bipartite C_4-free graphs	FPT (Chapter 8)	FPT (Chapter 8)
Bounded-degree graphs	FPT [64]	FPT (Chapter 9)

Table 6.2: The results obtained in this thesis (except for Token Jumping on bounded-degree graphs) on the parameterized complexity of Token Jumping and Token Sliding on several graph classes with parameter k.

r-independent sets admits a polynomial kernel for every $r \geq 1$, where an r-independent set is a set of vertices such that every two vertices in the set are at distance at least r from one another. The results on Token Jumping were ultimately generalized by Bousquet et al. in [28], who showed that it is FPT (and admits a kernel of polynomial size) on $K_{\ell,\ell}$-free graphs, a class containing both nowhere-dense graphs and d-degenerate graphs.

On the other hand, almost nothing is known concerning the tractability of Token Sliding on sparse graphs when parameterized by k. Prior to our work, it remained an open question whether this problem is fixed parameter tractable on bounded-degree graphs and bounded-treewidth graphs. The results about the parameterized complexity of token sliding and token jumping on sparse graphs are summarized in Figure 6.5.

As explained at the beginning of this section, most work on independent set reconfiguration focuses on parameters k and ℓ. To the best of our knowledge, the only other parameter that has been considered is the *modular-width* in [10]. A graph has modular-width mw if it has at most mw vertices, or if its vertex set can be partionned into mw sets, each of wich is a module and induces a graph of modular-width at most mw. The study of this parameter remains interesting since it is incomparable to the bandwidth, for which we know no FPT algorithm can exist. Belmonte et al. showed in [10] that both Token Jumping and Token Sliding are fixed-parameter tractable when parameterized by the modular width.

6.4.1 Our contribution

One of the goal of this thesis is to understand better the Token Sliding problem from a parameterized complexity point of view, and to try to draw a line between tractability and intractability. A summary of the results we obtained toward this question is given in Table 6.2. Among other results, we show the following:

Theorem 7. *Token Sliding parameterized by k is $\mathsf{W}[1]$-hard on bipartite graphs and is fixed-parameter tractable on bipartite C_4-free graphs.*

The results presented in Chapter 8 were obtained with Nicolas Bousquet, Clément Dallard, Kyle Lomer and Amer E. Mouawad. This collaboration started during the Coombinatorial Reconfiguration 2019 workshop (CoRe 2019) which was held in Aussois in May 2019. These results were accepted at the 31st International Symposium on Algorithms and Computation (ISAAC 2020) which was held online on December 2020 and were accepted for publication in Algorithmica. The full paper is available online on arXiv [9].

This result on C_4-free bipartite graphs shows in particular how to adapt the "buffer space" argument for the token sliding rule. In particular, a first step to prove the tractability of TS on bipartite C_4-free graph is to show that it is tractable on bounded-degree bipartite graphs. In Chapter 9, we generalize this result by showing the following:

Theorem 8. *Token Sliding is fixed-parameter tractable when parameterized by $k + \Delta(G)$.*

In order to do so, we introduce the notion of *galactic token sliding*, which we believe can be used as a starting point to design FPT algorithm for token sliding for other sparse graph classes.

Chapter 7

Independent set reconfiguration in H-free graphs

The results presented in this chapter were obtained with Nicolas Bousquet and Moritz Muhlenthaler. These results have not been peer-reviewed yet.

The question of finding a maximum independent set is one of the most studied optimization problem in graphs and is one of the problem we have the best understanding of. Nonetheless, it is still a very active field of research, in particular when it comes to the study of the complexity of the said problem in H-free graphs. A graph G is H-free if does not contain H as an induced subgraph, where H is a connected graph. Before we dive into the reconfiguration counterpart of the problem, let us summarize where the results stand today toward Maximum Independent Set (MIS) in H-free graphs.

Let G be a graph. A *t-subdivision* of an edge $e = (u, v)$ of G consists in deleting the edge e from G and replacing it by a path of length t with endpoints u and v and where all the intermediate vertices are new vertices (of degree exactly two). The t-subdivision of G is the graph obtained from G by subdividing every edge exactly k times. In [3] Alekseev showed that if G' is the $2t$ subdivision of graph G, then $\alpha(G') = \alpha(G) + t|E(G)|$. Informally speaking, it shows that it is as hard to find the independence number of G as it is to find the independence number of the $2t$-subdivision of G. Since a subdivision of G can have arbitrarily large girth and its two closest vertices of degree at least three arbitrarily far away from each other, it follows that excluding a graph H that contains a cycle or more that one vertex of degree at least three cannot make the problem easy. Furthemore, it is well known that MIS is NP-complete in graphs of maximum degree three. Hence, finding a maximum independent set in a H-free graph is NP-hard unless H is a path or a tree with a unique vertex of degree three. In other words, Alekseev [4] showed the following:

Theorem 29. *Maximum Independent Set in H-free graphs is NP-complete unless H is a path, the claw, or a subdivision of the claw.*

Using the representation of P_4-free graphs as cotrees, it is an easy exercise to see that a maximum independent set can be computed in linear time in this class. Hence, the smallest interesting case (for paths) left open by Theorem 29 is the case of P_5.

The problem remained open for three decades, until Lokshtanov et al. showed that (MIS) is indeed polynomial time solvable in [72]. It was then also shown to be true for P_6-free graphs by Grzesik et al. in [51], and the problem is still open for P_t-free graphs with $t \geq 7$.
Minty [76] and Sbihi [86] showed independently that MIS is solvable in polynomial-time in claw-free graphs. Twenty years later, Alekseev generalized this result to fork-free graphs [3], and this is where the results for MIS stand today. Let us however note that a tremendous amount of work exists concerning related problems on these graphs classes, such as the weighted version of the problem, the design of approximation algorithms, or the design of subexponential and quasi-polynomial time exact algorithms.

Let us now draw a parallel with the independent set reconfiguration problem. As mentionned in the previous chapter, both Token Sliding and Token Jumping were shown to be solvable in polynomial time in P_4-free graphs [16, 21, 67] and claw-free graphs [23]. However, prior to our work, no Alekseev type theorem were known for independent set reconfiguration. Hence, it might as well be true that excluding a graph containing a cycle or several high degree vertices makes the problems easy. In this chapter, we show that it is not the case and that Alekseev's partial dichotomy also holds for independent set reconfiguration:

Theorem 6. *Token Jumping and Token Sliding on H-free graphs are* PSPACE*-complete, unless H is a path, a claw, or a subdivision of the claw.*

The proof of Theorem 6 is given in Section 7.1. Since the complexity of both rules is known for claw-free graphs [23] we consider the case of fork-free graphs in Section 7.2. In particular, we show the following:

Theorem 30. *Maximum Independent Set Reconfiguration (MISR) is solvable in polynomial time in fork-free graphs.*

Our proof of Theorem 30 consist in a reduction to the claw-free case. We did not manage to adapt it to the general case, for neither Token Sliding nor Token Jumping. However, we have strong incentives to think that the following is true:

Conjecture 3. *Token Sliding and Token Jumping are polynomial time solvable on fork-free graphs.*

Although we put both rules in a single conjecture, we also think that, if exist, polynomial time algorithms for Token Jumping and Token Sliding in fork-free graphs should be radically different. We briefly discuss this question in Section 7.3, as a conclusion.

7.1 An Alekseev type theorem for reconfiguration

Let G be a graph. An odd subdivision of G is a graph obtained from G by subdividing all the edges an odd number of times. Given an odd subdivision of G, the additional

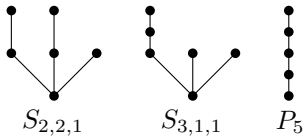


FIGURE 7.1: The three smallest graphs H for which the complexity of MISR on H-free graph remains unknown.

cost of a subdivision is equal to the sum for every edge of the original graph of its length in the subdivision minus one over two. A vertex of a subdivision is an *original vertex* if it belongs to $V(G)$ or is a *subdivided vertex* otherwise.

Let G be a graph and G' be an odd subdivision of G. Let I be an independent set of G. A *canonical representation J of I in G'* is an independent set such that:

- An original vertex is in J if and only if it is in I.
- For every edge $e \in E(G)$, exactly the length of the edge in the subdivision minus one over two vertices are in J.

One can easily prove that, if I is an independent set, we can construct such an independent set J. Actually Alekseev proved the following:

Theorem 31. *Let G be a graph and G' be an odd subdivision of G. The graph G has an independent set of size k if and only if G' has an independent set of size k plus the additional cost of G'.*

Our goal is now to adapt this statement to the framework of reconfiguration in the following way:

Theorem 32. *Let G be a graph and k be the size of a maximum independent set of G. Let G' be an odd subdivision of G and m be the additional cost of G'.*

Two independent sets I, J of size k are in the same connected component of the reconfiguration graph of (G, k) if and only if, canonical representations of I and J are in the same connected component (in both the TS and the TJ model).

Recall that MISR is PSPACE-complete in graphs of maximum degree three [64]. Hence, just as Aleskseev's Theorem 31 directly implies Theorem 29, Theorem 32 directly implies Theorem 6.

First note that by Theorem 31 the corresponding independent set is maximum in G'. And then the token jumping and the token sliding are equivalent model in that case.

The remaining of this section is devoted to prove Theorem 32. Let $p > 0$ and (u, v) be an edge of G. The set $S_{uv} = \{s_1, s_2, \ldots, s_p\} \subseteq V(G')$ is the set of subdivided vertices of the edge (u, v). By convention s_1 is adjacent to u and s_p is adjacent to v. Each vertex of S_{uv} has degree two and for every $1 < i < p$, the vertex s_i is adjacent to s_{i+1} and s_{i-1}. Note that p is even since G' is an odd subdivision of G. A *left move*

on S_{uv} is a move of a token on s_i with $i > 1$ to s_{i-1}. Let I' be an independent set of G' of size $k+m$. The *trace* of I' is the set of original vertices of I'. Let us prove the following:

Claim 2. *Let I' be an independent set of G' of size $k+m$ and (u,v) be an edge of G. If both u and v belongs to the trace of I' then $|I' \cap S_{uv}| = \frac{p-2}{2}$ and $|I' \cap S_{uv}| = \frac{p-1}{2}$ otherwise.*

Proof. Starting from I', apply any valid left move on S_{uv} as long as there exists one and let I'' denote the obtained independent set. Suppose first that $\{u,v\} \subseteq I''$: then any i such that $s_i \in I''$ is even. If not, consider the smallest j odd such that $s_j \in I''$. It satisfies $j > 1$ since $u \in I''$ and by the choice of j it follows that s_{j-1} and s_{j-2} are I''-free. But then the token on s_j can be moved to s_{j-1}, a contradiction. Furthermore, there must be a token on s_{p-2} since I'' is maximum and since s_{p-1} and s_{p-3} are free. Since no left move is possible, there must be a token on each s_i for every $2 \leq i \leq p-2$ even and thus there are exactly $\frac{p-2}{2}$ tokens on S_{uv}.
Suppose then that $|\{u,v\} \cap I''| \leq 1$. Without loss of generality we can suppose that $u \notin I''$. One can easily check that any i such that $s_i \in I''$ must be odd. Since p is even and I'' is maximum, there must be a token on s_{p-1}. As no left move is possible, there must be a token on each s_i for every $1 \leq i \leq p-1$ odd and thus there are exactly $\frac{p-1}{2}$ tokens on S_{uv}. □

The following statement is an easy consequence of Claim 2:

Claim 3. *Let I', J' be two independent sets with the same trace, then I' and J' are in the same connected component of the reconfiguration graph of $(G', k+m)$.*

Proof. Let (u,v) be an edge of G and let T denote the trace of I' and J'. If both u and v belongs to T we have $I' \cap S_{uv} = J' \cap S_{uv} = \{s_2, s_4, \ldots, s_{p-2}\}$ by Claim 2. If $|\{u,v\} \cap T| = 1$, suppose without loss of generality that $u \notin T$. Then Claim 2 ensures that $I' \cap S_{uv} = J' \cap S_{uv} = \{s_1, s_3 \ldots, s_{p-1}\}$. Finally, if $\{u,v\} \cap T = \emptyset$, Claim 2 ensures that $|T| = \frac{p-1}{2}$. By applying any possible valid left move on S_{uv} for both I' and J' we obtain independent sets I'' and J'' respectively such that $I'' \cap S_{uv} = J'' \cap S_{uv} = \{s_1, s_3, \ldots, s_{p-1}\}$. Note that the trace of I'' and J'' is still T since we only move token between subdivided vertices. It is then sufficient to repeat this process for any edge of G that satisfies $\{u,v\} \cap T = \emptyset$ to obtain a transformation between I' and J'. □

We are now ready to prove the first direction of Theorem 32:

Claim 4. *If there is a transformation from I to J then there is a transformation from a canonical representation of I to a canonical representation of J.*

Proof. Let I_1, I_2 be independent sets of G such that I_2 is obtained from I_1 by moving a token from a vertex u to a vertex v. Note that $(u,v) \in E(G)$ since I_1 and I_2 are maximum. Let I_1' be a canonical representation of I_1 and let us show that we can always

reach a canonical representation of I_2 starting from I'_1. Let $S_{uv} = \{s_1, s_2, \ldots, s_p\}$ be the subdivided vertices of the edge (u, v) in G' and suppose that there exists and edge $(v, w) \in E(G)$ with subdivided vertices $S_{vw} = \{q_1, \ldots, q_p\}$ such that $q_1 \in I'_1$. Note that $w \notin I'_1$ since I'_1 is a canonical representation of I_1 and $I_2 = I_1 + v - u$ is an independent set. It follows that $I'' = I'_1 - S_{vw} + \{q_2, q_4, \ldots q_p\}$ is a maximum independent set of G' such that I'_1 and I'' have the same trace and thus I'' can be reached from I'_1 by Claim 3. So up to applying such a transformation for each edge $(v, w) \in E(G)$ we can suppose that $N(v) \cap I'_1 = \{s_p\}$ and we can slide the token on s_p to v. Finally, it suffices to slide the token from s_i to s_{i+1} for every i such that $s_i \in I'_1$ (in decreasing order) and then slide the token from u to s_1 in order to reach a canonical representation of I_2.

□

Let us now prove the converse direction. Let I' be an independent set of G' of size $k + m$.

Claim 5. *The trace of I' contains no (non necessarily induced) path on three vertices in G.*

Proof. Suppose otherwise and let u, v, w be such a path. Let $S_{uv} = \{s_1, \ldots s_p\}$ and $S_{vw} = \{q_1, \ldots, q_p\}$ be the subdivided vertices of (u, v) and (v, w) respectively. By Claim 2 we have $I' \cap S_{uv} = \{s_2, s_4, \ldots, s_{p-2}\}$ and $I' \cap S_{vw} = \{q_2, q_4, \ldots, q_{p-2}\}$. But then we are free to move the token token on u to s_p and obtain an independent I'' such that $N(q_1) \cap I'' = \emptyset$, a contradiction since I' is maximum. □

So the trace of I' is the disjoint induced union of isolated vertices and edges. The size of the trace of I' is the number of isolated vertices plus the number of edges in it.

Claim 6. *The trace of I' has size exactly k.*

Proof. Let (u, v) be an edge of G. By Claim 2 we have $|S_{uv} \cap I'| = \frac{p}{2}$ if $\{u, v\}$ is contained in the trace of I' and $|S_{uv} \cap I'| = \frac{p-2}{2}$ otherwise. Let n_e and n_v denote respectively the number of edges and isolated vertices in the trace of I'. We have $|I'| = k + |E(G)|\frac{p}{2} = \frac{p-2}{2}n_e + \frac{p}{2}(|E(G)| - n_e) + 2n_e + n_v$ from which we obtain $n_e + n_v = k$. □

A subset of $V(G)$ is associated to I' if it contains all the isolated vertices of I' plus one endpoint of each edge of I'. The previous claims mainly ensures that the following holds:

Claim 7. *Every subset associated to I' is an independent set of G of size exactly k.*

Proof. Let A be a subset of $V(G)$ associated to I'. If two vertices $u, v \in A$ are adjacent, then (u, v) is an edge in the trace of I', a contradiction with the construction of A. The size of A follows from Claim 6. □

We can now complete the proof of Theorem 2:

Proof. Let I' and J' be canonical representations of I and J respectively and let $I_0 = I', I'_1, \ldots, I'_{s-1}, I'_s = J'$ be a valid transformation. Let us show show by induction that for every $t \leq s$, there exists an independent set $I_t \subseteq V(G)$ associated to I'_t such that $I_0, I_1, \ldots, I_s$ is a valid reconfiguration sequence from I to J. We start with $I_0 = I$, which is associated to I' by definition of a canonical representation. Note that since all the I's are maximum, each move in the sequence is a sliding move. Furthermore, if u is an isolated vertex in the trace of I'_i then the token on u cannot move since by Claim 2 there must be a token on every subdivided vertex $s_2 \in S_{uv}$ for every $(u, v) \in E(G)$. Three cases remain:

1. A token slides between two subdivided vertices. Then we set $I_{i+1} = I_i$. By induction I'_i is associated to I_i, and since I'_i and I'_{i+1} have the same trace it follows that I_i is also associated to I'_{i+1}.

2. Let (u, v) be an edge of G and suppose that a token slides from $s_p \in S_{uv}$ to v. Since I'_i is maximum we have $u \in I'_i$ by Claim 2. Furthermore for every $t \in N_G(u)$ we have $t \notin I'_i$ by Claim 5 (since otherwise t, u, v is a path of length 3 in the trace of I'_{i+1}). It follows that u is isolated in the trace of I'_i and thus that $u \in I_i$. Then we set $I_{i+1} = I_i - u + v$. The set I_{i+1} is an independent set of G by Claim 2 and it is associated to I'_{i+1} since the traces of I'_i and I'_{i+1} only differs on $\{u, v\}$.

3. Let $u \in V(G)$ and suppose that a token slides from u to a subdivided vertex $s_1 \in S_{uv}$. Suppose that $v \notin I'_i$: since there can be no token on $\{s_1, s_2\}$ there is at most $\frac{p-2}{2}$ tokens on S_{uv}, a contradiction with Claim 2. Then either I_i contains u (in which case it does not contain v by definition of an associated set) and we set $I_{i+1} = I_i - u + v$ or I_i does not contain u (in which case it must contain v since (u, v) is an edge in the trace of I'_i) and we set $I_{i+1} = I_i$. The induction hypothesis and Claim 5 ensure that I_{i+1} is an independent set of G in both cases. Furthermore, it is associated to I'_{i+1} since the traces of I'_i and I'_{i+1} only differ on $\{u, v\}$.

We obtain a sequence of independent sets $I_0, I_1, \ldots, I_s$ such that $I_0 = I$, $I_s = J$ and $|I_t \Delta I_{t+1}| \leq 1$ for every $t < s$, which concludes the proof. □

7.2 MISR in fork-free graphs

This section is dedicated to the proof of Theorem 30. Note that we can consider the token sliding rule only since the independent sets are maximum.

Let H be a claw with vertex set $\{c, x, y, z\}$ such that $N(c) = \{x, y, z\}$. The graph H is called a *c-claw*, c is the *center* of H and x, y, and z are the *leaves* of H. The *center* of a fork F is the only vertex of F of degree 3. Given a graph $G = (V, E)$ and $S \subseteq V$, the graph G' obtained by *deleting S from G* (we just say by deleting S

when G is obvious from context) is the graph $G[V - S]$. Given an independent set I, a vertex u of G is *exposed* if $|N(u) \cap I| = 1$.

Reduction rules. The proof of Theorem 30 consists in a reduction to the maximum independent set reachability problem in claw-free graphs, which can be decided in polynomial time [23]. Let G be a fork-free graph and I be a maximum independent set of G. We consider the following reduction rules:

Reduction rule 1. If $\{c, x, y, z\}$ is a c-claw where $\{x, y, z\} \subseteq I$ then delete c.
Reduction rule 2. If $\{c, x, y, z\}$ is a c-claw where $c \in I$ and $\{x, y, z\} \subseteq G - I$ then delete $\{x, y, z\}$.
Reduction rule 3. If $\{c, x, y, z\}$ is a c-claw where $\{c, x, y\} \subseteq G - I$ and $z \in I$ then delete $\{c, x, y\}$.
Reduction rule 4. If $\{c, x, y, z\}$ is a c-claw where $\{c, x, y, z\} \subseteq G - I$ then delete $\{c, x, y, z\}$.

In what follows, we will prove that the reduction rules are safe. We first apply Reduction rule 1 as long as we can, then apply Reduction rule 2 and so on. Let G be a fork-free graph and I be a maximum independent set of G. The *i-reduced graph* G_i is the graph obtained after applying exhaustively reduction rules 1 to i in that order to (G, I) with $1 \leq i \leq 4$. Clearly, I is still a maximum independent set of G_i.

Lemma 25. *Let I and J be maximum independent sets in a fork-free graph G and let G_1 be the 1-reduced graph obtained from (G, I). Then $I \leftrightsquigarrow_{TS(G_1)} J$ if $J \cap V(G) - V(G_1) \neq \emptyset$ and $I \leftrightsquigarrow_{TS(G)} J$ if and only if $I \leftrightsquigarrow_{TS(G_1)} J$ otherwise.*

Proof. If $I \leftrightsquigarrow_{TS(G_1)} J$ then indeed $I \leftrightsquigarrow_{TS(G)} J$. Let us prove the other direction. Let $\{c, x, y, z\}$ be a c-claw of G with $c \in G - I$ and $\{x, y, z\} \subseteq I$. We show that in any reconfiguration sequence starting from I, none of the tokens on $\{x, y, z\}$ can leave $N(c)$. Suppose otherwise and let I' denote the independent set just before such a move. Then there exists $\{x', y', z'\} \subseteq N(c) \cap I'$ and from I' one token from $\{x', y', z'\}$ moves to some vertex $t \notin N(c)$. But then $\{c, x, y, z, t\}$ is a fork in G, a contradiction. It follows that in any independent set I' reachable from I, $c \notin I'$ (since there are three tokens in $N(c)$). In particular if $c \in J$ then $I \not\leftrightsquigarrow_{TS(G)} J$ and since $c \notin G_1$ then clearly $I \not\leftrightsquigarrow_{TS(G_1)} J$. Otherwise if $c \notin J$ and if there exists a sequence from I to J, then for any I' in the sequence we have $c \notin I'$. But then this sequence is also a valid sequence in $G - \{c\}$ and the result follows. □

By definition of Reduction rule 1 there exists no c-claw in G_1 having three leaves in I. The following lemma shows that in fact, any remaining c-claw in G_1 has at most one leaf in I.

Lemma 26. *Let G be a fork-free graph and I be a maximum independent set of G. Let G_1 be the 1-reduced graph obtained from (G, I). Any claw in G_1 has at most one leaf in I.*

Proof. By contradiction let $\{c, x, y, z\}$ be a c-claw in G_1 such that $|\{x, y, z\} \cap I| = 2$. Suppose without loss of generality that $\{x, y\} \in I$. Since I is a maximum independent

set, z must have a neighbor t in I, and since c has at most two neighbors in I we have $t \notin N(c)$. It follows that $\{c, x, y, z, t\}$ induces a fork in G_1. □

So after applying reduction rule 1, there remains three cases to consider if there is a claw in G_1: either one leaf of the claw is in I, or the center of the claw is in I, or the whole claw is in $G - I$. We show that, in each case, we can apply one of the Reduction rules 2, 3 or 4 (and show that they are safe) until we either obtain a graph G' such that $J \nsubseteq G'$ in which case $I \nleftrightarrow_{TS(G)} J$, or we obtain a claw-free graph.
Let I be an independent set of a fork-free graph G. A set $X \subseteq V(G)$ is *locally frozen* if for all $v \in X$, $|N(v) \cap I| = 2$. Note that since Reduction rule 1 cannot be applied to G_1, we can suppose that for every $v \in G_1$, $|N(v) \cap I| \leq 2$. Given a locally frozen set X, the set $B_X := \cup_{v \in G_1} N(v) \cap I$ is the set of X-*blocking* vertices. It is obvious that given a set X of locally frozen vertices and its blocking set B_X, no token can move from B_X to X. Given a vertex u and a set $X \subseteq V(G_1)$ not containing u, the vertex $v \in G - X \cup \{u\}$ is a X-*clone of* v if $(u, v) \in E$ and if $N(v) \cap X = N(u) \cap X$.

Lemma 27. *Let I and J be maximum independent sets in a fork-free graph G and X be a set of locally frozen vertices. Suppose that for every maximum independent set I' of G reachable from I, X is also locally frozen. Then if $J \cap X \neq \emptyset$, $I \nleftrightarrow_G J$ and $I \nleftrightsquigarrow_G J$ if and only if $I \nleftrightsquigarrow_{G'} J$ where G' is the graph obtained by deleting X from G otherwise.*

Proof. By supposition, any maximum independent set I' reachable from I satisfies $I' \cap X = \emptyset$ so if $J \cap X \neq \emptyset$ then $I \nleftrightarrow_G J$. Suppose that $J \cap X = \emptyset$. If there exists a reconfiguration sequence from I to J in G, then since any independent set I' in the sequence satisfies $I' \cap X = 0$, there also exists a reconfiguration sequence from I to J in G'. □

So if there exists a claw C in G_1, it is sufficient to find a set of locally frozen vertices containing some vertices of C that satisfies the condition of Lemma 27. We can then delete the claw from G while maintaining the reachability. The following Lemma gives a simple condition to ensure that a locally frozen set of vertices remains locally frozen in any independent set I' reachable from I:

Lemma 28. *Let I be a maximum independent set of a fork-free graph G, X be a set of locally frozen vertices of G and B_X be its blocking set. Let I' be the maximum independent set obtained by sliding a token from $u \in B_X$ to a vertex v that is a X-clone of u. Then the set X is locally frozen in I' and blocked by $B_X - u + v$.*

Proof. By definition of a X-clone, $N(v) \cap X = N(u) \cap X$ and since I' is obtained from I by sliding a token from u to v, every $z \in X$ satisfies $|N(z) \cap I'| = 2$. □

Given a set of locally frozen vertices X of G, it wil be sufficient to show that at any point in a reconfiguration sequence, a token on the blocking set of X can only move from a vertex u to a X-clone of u. We need the following claim before discussing the reduction rules:

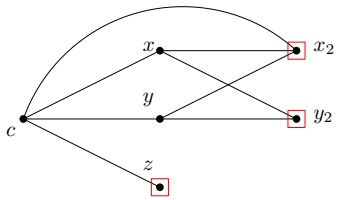


FIGURE 7.2: Irrelevant and blocking vertices of Claim 31

.

Lemma 29. *Let I and J be maximum independent sets in a fork-free graph G and let G_1 be the 1-reduced graph obtained from (G, I). Then for any vertex $v \in G_1$ and any independent set I' of G_1 reachable from I, $|N(v) \cap I'| \leq 2$.*

Proof. Since G_1 is obtained from G by applying reduction rule 1, I is a maximum independent set of G_1 and every $v \in G_1$ satisfies $|N(v) \cap I| \leq 2$. Suppose for a contradiction that there exists a vertex c and an independent set I' reachable from I such that $|N(c) \cap I'| > 2$. Let I_t be the first independent set in a sequence from I to I' that satisfies $|N(c) \cap I_t| = 3$ and I_{t-1} be the independent set that precedes it. By choice of I_{t-1}, we have $|N(c) \cap I_{t-1}| = \{x, y\}$ for some $\{x, y\} \subseteq G'$. Then I_t is obtained from I_{t-1} by sliding a token from a vertex $t \notin N(c)$ to a vertex $z \in N(c)$ and since I_{t-1} and I_t are independent sets, we have $t \notin N(x) \cup N(y)$ and $z \notin N(x) \cup N(y)$. But then $\{c, x, y, z, t\}$ induces a fork, a contradiction. □

Lemma 30. *Let I and J be maximum independent sets in a fork-free graph G and let G_2 be the 2-reduced graph obtained from (G, I). Then $I \nleftrightarrow_{TS(G)} J$ if $J \cap V(G) - V(G_2) \neq \emptyset$ and $I \leftrightsquigarrow_{TS(G)} J$ if and only if $I \leftrightsquigarrow_{TS(G_2)} J$ otherwise.*

Proof. Again, the converse direction is trivial. Let us assume that G contains a c-claw $C := \{c, y, x, z\}$ such that $c \in I$. Since I is maximum, every leaf of the claw must have another neighbor in I. No leaf has a private neighbor (regarding $\{c, x, y, z\}$)) in I since otherwise there is a fork. It follows that there must exists a vertex $u \in I$ such that $u \neq c$ and u is a common neighbor to x, y, and z. So $X := \{x, y, z\}$ is a set of locally frozen vertices blocked by $B_X = \{c, u\}$. Let us show that X and B_X satisfy the condition of Lemma 27. By symmetry it is sufficient to show that any neighbor of u in $G - X$ is a X-clone of u. First note that since we applied reduction rule 1, c and u are the only neighbors of x, y and z in I. Let $v \in N(u) - X$ be an exposed vertex. Suppose that v has a non-neighbor in $\{x, y, z\}$, say x without loss of generality. Then we can move the token on u to v, then move the token on c to x, and then move the token on v to y. But then z has no neighbor in the obtained independent set, contradicting the fact that I is maximum. □

Lemma 31. *Let I and J be maximum independent sets in a fork-free graph G and let G_3 be the 3-reduced graph obtained from (G, I). Then $I \nleftrightarrow_{TS(G)} J$ if $J \cap V(G) - V(G_3) \neq \emptyset$ and $I \leftrightsquigarrow_{TS(G)} J$ if and only if $I \leftrightsquigarrow_{TS(G_3)} J$ otherwise.*

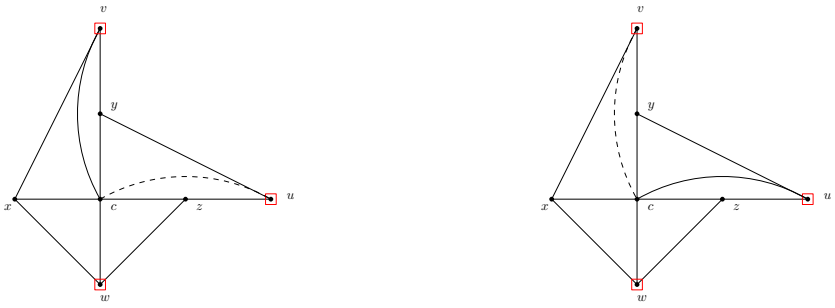
Proof. Assume that G contains a c-claw $C := \{c, y, x, z\}$ such that $z \in I$ and $\{x, y\} \in G - I$. Since I is maximum, both x and y have a neighbor in I. Suppose first that x has a neighbor $x_2 \in I$ and that y has a neighbor y_2 in I such that $x_2 \notin N(y)$ and $y_2 \notin N(x)$. Then both cx_2 and cy_2 are edges otherwise $\{c, x, y, z, x_2\}$ or $\{c, x, y, z, y_2\}$ induce forks. But then $N(c) \cap I \subseteq \{x_2, y_2, z\}$, a contradiction. It follows that x and y have a common neighbor $x_2 \in I$ (by maximality of I). At least one of $\{x, y\}$ has another neighbor in I otherwise $I - \{x_2\} \cup \{x, y\}$ is a larger independent set of G_2. By symmetry, we can assume that there exists $y_2 \in N(x) \cap (I - x_2)$. Suppose that y_2 is not in $N(y)$. Then cy_2 and cx_2 are edges since otherwise $\{c, x, y, z, y_2\}$ or $\{c, v, x, y, x_2\}$ induce forks. Then $N(c) \cap I \subseteq \{z, x_2, y_2\}$, a contradiction with Reduction Rule 1. So x and y are complete to x_2 and y_2. Furthermore, c must be a neighbor of either x_2 or y_2 or there is a fork. By symmetry we can assume that $x_2 \in N(c)$. Reduction Rule 1 implies that $y_2 \notin N(c)$. We obtain a set $X := \{x, y, z\}$ of locally frozen vertices blocked by $B_X := \{x_2, y_2, z\}$ (see Figure 7.2 for an illustration). Let us show that these sets satisfy the conditions of Lemma 27. Suppose that at any point in a reconfiguration sequence one of the token on x_2, y_2 or z moves. We have three cases to consider:

a) The token on z moves first. Let $z_2 \in N(z) - X$ be the vertex to which the token moves. Since z_2 is exposed, $z_2 \notin N(x_2) \cup N(y_2)$. Furthermore, $z_2 \notin N(x)$ (resp. $z_2 \notin N(y)$) otherwise $\{x, x_2, y_2, z_2, z\}$ (resp. $\{y, x_2, y_2, z_2, z\}$) induces a fork. It follows that $z_2 \in N(c)$ otherwise $\{c, x, y, z, z_2\}$ induces a fork. Since c is the only neighbor of z in H, we obtain that z_2 is a X-clone of z.

b) The token on y_2 moves first. Let v be the vertex on which the token moves. Then $v \notin N(x_2) \cup N(z)$. Furthermore $v \notin N(c)$ as otherwise $\{c, v, x_2, z, y_2\}$ induces a fork. If $v \notin N(x) \cup N(y)$ then the move $x_2 \to x$ is valid and y has no neighbor in the obtained independent set thus it must be that $v \in N(x) \cup N(y)$. If $v \in N(y)$ and $v \notin N(x)$ or if $v \in N(x)$ and $v \notin N(y)$ then in both cases $\{c, x, y, z, v\}$ induces a fork. It follows that $v \in N(x) \cap N(y)$ and v is a X-clone of y_2.

c) The token on x_2 moves first. Let u denote the vertex on which the token moves. We have $u \in N(x) \cup N(y)$ otherwise the move $y_2 \to y$ is valid and x has no neighbor in the obtained independent set. Suppose first $u \in N(x)$: it implies that $u \in N(y) \cup N(c)$ otherwise $\{c, x, y, z, v\}$ induces a fork. Then if $u \in N(c)$ and $u \notin N(y)$ (resp. $u \in N(y)$ and $u \notin N(c)$) then $\{c, y, y_2, z, u\}$ induces a fork of center c (resp. y). It follows that if $u \in N(x)$ then $u \in N(x) \cap N(y) \cap N(c)$. Finally suppose that $u \in N(y)$. Then $u \in N(x) \cup N(c)$ otherwise $\{c, x, y, z, v\}$ induces a fork. If $u \in N(c)$ and $u \notin N(y)$ (resp. $u \in N(y)$ and $u \notin N(c)$) then $\{c, x, y_2, z, u\}$ induces a fork of center c (resp. x). We can conclude that $u \in N(x) \cap N(y) \cap N(c)$ and u is a X-clone of v.

□



(A) Case 1 (B) Case 2

FIGURE 7.3: Irrelevant and blocking vertices of Claim 32

The last case to be considered is the case where G contains a claw that is included in $G - I$:

Lemma 32. *Let I and J be maximum independent sets in a fork-free graph G and let G_4 be the 4-reduced graph obtained from (G, I). Then $I \nleftrightarrow_{TS(G)} J$ if $J \cap V(G) - V(G_4) \neq \emptyset$ and $I \leftrightsquigarrow_{TS(G)} J$ if and only if $I \leftrightsquigarrow_{TS(G_4)} J$ otherwise.*

Proof. Let us suppose that G contains a c-claw $C := \{c, y, x, z\} \subseteq G - I$. Since I is maximum, the center c of C must have a neighbor $w \in I$. If w is a neighbor of at most one leaf of C, then we can apply reduction rule 2 and delete three vertices of C. Furthermore if $\{x, y, z\} \subseteq N(w)$ then we can apply reduction rule 3 and delete $\{x, y, z\}$. So we can suppose without loss of generality that $\{x, z\} \subseteq N(w)$ and $y \notin N(w)$ and since I is maximum, y must have a neighbor $v \neq w$ in I. This vertex must satisfy $v \in N(x) \cup N(y) \cup N(c)$ otherwise $\{c, x, y, z, v\}$ induces a fork. If $v \in N(c)$ and $v \notin N(x) \cup N(y)$ then $\{c, x, z, v\}$ induces a c-claw and reduction rule 2 applies, thus we have up to symmetry that $v \in N(x)$. It follows that $v \notin N(z)$ otherwise $\{v, x, y, z\}$ induces a v-claw and reduction rule 3 applies, and since I is maximum, z must have a neighbor $u \notin \{v, w\}$ in I. Since $N(x) \cap I = \{v, w\}$, $xu \notin E$ and we have $u \in N(c) \cup N(y)$ otherwise $\{c, x, y, z, u\}$ induces a fork. If $u \in N(c)$ and $u \notin N(y)$, then we also have $v \in N(c)$ otherwise $\{c, y, u, w, v\}$ induces a fork, but in that case c has three neighbors $\{u, v, w\}$ in I, a contradiction. It follows that $u \in N(y)$. Finally we have that either $u \in N(c)$ or $v \in N(c)$ otherwise $\{y, v, u, c, w\}$ induces a fork (note that is not possible that $\{u, v\} \in N(c)$ or c would have three neighbors in I). In both cases we obtain that $X := \{c, x, y, z\}$ is a set of locally frozen vertices blocked by $B_X := \{u, v, w\}$ (see figure 7.3 for an illustration). Let us now show that we can always apply Lemma 2 to delete X from G.

Case 1: $v \in N(c)$ $(\Leftrightarrow u \notin N(c))$. Suppose that one of the token on u, v or w moves to an exposed vertex. As for Claim 7.2, we have three cases to consider:

a) The token on v moves first. Let v_2 be the vertex on which the token moves. Note that since it is the first token on $\{u, v, w\}$ to move, we have $v_2 \notin N(u) \cup N(w)$ and $v_2 \notin N(z)$ by Lemma 29. We have $v_2 \in N(x) \cup N(y)$ since otherwise $\{v, v_2, y, x, w\}$ induces a fork. Suppose first that $v_2 \in N(x)$. Then we have

$v_2 \in N(c) \cup N(y)$ otherwise $\{c, x, y, z, v_2\}$ induces a fork. Finally if $v_2 \in N(y)$ and $v_2 \notin N(c)$ (resp. $v_2 \in N(c)$ and $v_2 \notin N(y)$) then $\{y, v_2, u, c, w\}$ induces a fork of center y (resp. c). It follows that $v_2 \in N(x) \cap N(y) \cap N(c)$. Suppose then that $v_2 \in N(y)$, then also $v_2 \in N(x) \cup N(c)$ otherwise $\{c, x, y, z, v_2\}$ induces a fork. As in the previous case it is not possible to have $v_2 \in N(x)$ and $v_2 \notin N(c)$ and if $v_2 \in N(c)$ and $v_2 \notin V(x)$ then $\{c, x, v_2, z, u\}$ induces a fork, a contradiction. It follows that v_2 is a X-clone of v.

b) The token on u moves first. Let u_2 denote the vertex on which the token moves. Clearly $u_2 \notin N(v) \cup N(w)$ and $u_2 \notin N(c) \cup N(x)$ by claim 29. We have $u_2 \in N(y) \cup N(z)$ otherwise $\{c, x, y, z, u_2\}$ induces a fork, and if $u_2 \in N(y)$ and $u_2 \notin N(z)$ (resp. $u_2 \in N(z)$ and $u_2 \notin N(y)$) then $\{c, x, y, u_2, z\}$ induces a fork. It follows that $u_2 \in N(y) \cap N(z)$ and u_2 is a X-clone of u.

c) The token on w moves first. Let w_2 be the vertex on which the token moves. Again $w_2 \notin N(u) \cup N(v)$ and $w_2 \notin N(y)$ by claim 29. Then $w_2 \in N(x) \cup N(z)$ otherwise $\{w, x, z, w_2, u\}$ induces a fork. Then we have:

- If $w_2 \in N(x)$ and $w_2 \notin N(z)$ or $w_2 \in N(z)$ and $w_2 \notin N(x)$ then $w_2 \in N(c)$ otherwise $\{c, x, w_2, y, z\}$ induces a fork in both cases.
- If $w_2 \in N(x) \cap N(c)$ and $w_2 \notin N(z)$ then $\{c, v, w_2, z, u\}$ induces a fork and if $w_2 \in N(z) \cap N(c)$ and $w_2 \notin N(x)$ then $\{c, x, y, w_2, y\}$ induces a fork.
- Finally if $w_2 \in N(x) \cap N(z)$ and $w_2 \notin N(c)$, then $\{z, c, u, w_2, v\}$ induces a fork.

It follows that $w_2 \in N(c) \cap N(x) \cap N(z)$ and w_2 is a X-clone of w.

Case 2: $u \in N(c)$ ($\Leftrightarrow v \notin N(c)$). One can easily check that up to exchanging vertices x, z and u, v Case 1 applies. □

We can now give the proof of the main theorem of this section:

Proof of Theorem 30: Let I and J be two maximum independent sets in a fork-free graph G. Let us show that we can decide whether $I \leftrightsquigarrow_{TS} J$ in polynomial time. A list of all claws of G can be found in polynomial time (by brute-force). Let G_4 be the 4-reduced graph obtained from (G, I). By Lemma 32, the answer is NO if any vertex of J has been deleted, otherwise both I and J are maximum independent sets of G_4 and $I \leftrightsquigarrow_{TS(G)} J$ if and only if $I \leftrightsquigarrow_{TS(G_4)} J$. Clearly G_4 is claw-free, I and J are maximum independent set of G_4, and by the results from [23] it can be decided in polynomial time whether $I \leftrightsquigarrow_{TS(G_4)} J$.

7.3 A short discussion on the non-maximum case

Let us conclude this chapter by discussing Conjecture 3. First, one can easily notice that when the source and target independent sets are not maximum then Token

Sliding and Token Jumping in fork free-graphs are not equivalent. Indeed, consider a claw with two tokens on the leaves: with the jumping rule, the tokens can freely jump on the leaf containing no token, whereas such a configuration is frozen with the sliding rule.
A classical approach to design efficient algorithm for independent set reachability is to find a way to reduce the size of the symmetric difference between the source and target independent sets in polynomial time. Note that the graph induced by the symmetric difference of two independent sets is a bipartite graph. A *complex* is a complete bipartite graph minus a matching. A complex is *trivial* if both side of its bipartition have size at most two. It is not hard to check that a trivial complex is either a C_4, a path, or an induced matching. In [3], Alekseev gives a complete description of the structure of connected bipartite fork free-graphs:

Theorem 33. *A connected bipartite fork-free graph is either a single vertex, a path, a cycle or a complex.*

Let G be a fork free-graph and I and J be the source and target independent set respectively. By Theorem 33, each connected component of $G[I\Delta J]$ is either a single vertex, a path, a cycle or a complex. With this first remark at hand, we can now discuss the case of Token Sliding and Token Jumping separately.

Token Sliding. Let us first show the following Lemma, which is crucial for the sliding rule:

Lemma 33. *Let S be an independent set of G and $c \in V(G)$ be a vertex that has at least three neighbors in S. Then for any independent set S' containing v we have $S \nleftrightarrow_{TS(G)} S'$.*

Proof. Suppose for a contradiction that there exists a TS-sequence from S to S' in G. In such a sequence, a token must slide to c at some point. Right before this move, there is no token on the neighborhood of c but the one that slides to c. In other words, all the tokens initially on $N(c) \cap S$ leave the neighborhood of c at some point. Consider the first token to do so. Let x be the neighbor of c on which this token lies and u be the vertex on which the token slides to. By supposition we have $u \notin N(c)$. Furthermore, there exists at least two other vertices $\{y, z\} \in N(c) \cap S$ on which there are still tokens. Since the sequence is valid, we have $\{y, z\} \notin N(u)$. Since furthermore $u \notin N(c)$, the graph induced by $\{c, x, y, z, u\}$ in G is a fork, a contradiction. □

Note that Lemma 33 is actually a generalization of Lemma 25 to the non-maximum case. Lemma 33 immediately implies that if there exists a non-trivial complex in $G[I\Delta J]$, then $I \nleftrightarrow_{TS(G)} J$. Hence we can suppose that the connected components of $G[I\Delta J]$ are either isolated vertices, paths, or cycles. Paths are easy to deal with since we can slide the vertices one by one along it to decrease the size of the symmetric difference. Hence, their remains to consider the case where $G[I\Delta J]$ contains only isolated vertices and cycles. This is exactly the problem that Bonsma et al. tackle

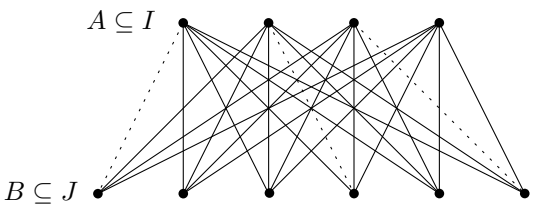


FIGURE 7.4: Example of a complex K. Plain black lines are edges and dotted black lines are non-edges. At least two tokens on A must move before a token can jump to B.

in [23] when showing that TOKEN SLIDING can be decided in polynomial-time in claw-free graphs. Hence, we believe that a reduction to the claw-free case could be a good approach to show that TOKEN SLIDING in fork-free graphs can be decided in polynomial-time.

Token Jumping. Let us now consider the jumping rule. As previously, each connected component of $G[I \Delta J]$ is either an isolated vertex, a path, a cycle or a complex. Isolated vertices are obviously not a problem with the jumping rule: any token can directly jump on such a vertex and reduce the symmetric difference. Just as for sliding, paths are also easy to deal with when considering the jumping rule. Hence, their remains to deal with the case where the connected components of $G[I \Delta J]$ are either cycles or complexes. Note that Lemma 33 does not hold for the jumping rule, and hence complexes in $G[I \Delta J]$ can be of any size. An example of such a large complex K is given if Figure 7.4. Let (A, B) be the bipartition of K such that $A \subseteq I$ and $B \subseteq J$. By definition of a complex, every vertex in A sees all but at most one vertex in B. Hence, finding a reconfiguration sequence from I to J requires to move almost all the tokens on A to vertices that are not in $N(B)$. In order to do so, we must be able to find *free* vertices, which are vertices that have no tokens on their closed neighborhood. Free vertices can be used as intermediate vertices before jumping the tokens to B. If no such vertices (or not enough of them) exist, then one can also find reconfiguration sequences that create free vertices. We believe that this can be done in polynomial time by exploiting the fact that the source and target independent sets are not maximum.

Conclusion. As seen in this section, TOKEN SLIDING and TOKEN JUMPING in fork free-graphs define different problems when the source and target independent sets are not maximum, and we believe that both of them require a specific approach. However, we believe that in both cases the structure of the symmetric difference given by Alekseev and the non-maximality of the source and target independent sets gives enough room for the design polynomial time algorithms, and hence for proving Conjecture 3.

Chapter 8

On girth and the parameterized complexity of Token Sliding and Token Jumping

The results presented in this chapter were obtained with Nicolas Bousquet, Clément Dallard, Kyle Lomer and Amer E. Mouawad. This collaboration started during the Coombinatorial Reconfiguration 2019 workshop (CoRe 2019) which was held in Aussois in May 2019. These results were accepted at the 31st International Symposium on Algorithms and Computation (ISAAC 2020) which was held online on December 2020 *and were accepted for publication in Algorithmica. The full paper is available online on arXiv [9].*

In this chapter we focus on the parameterized complexity of the Token Jumping and Token Sliding problems on graphs where some cycles with prescribed length are forbidden. Such graph classes contain bipartite graphs (odd-hole-free graphs), even-hole-free graphs and triangle-free graphs. Our main goal was to understand which cycles make the independent set reconfiguration problems hard. All the results we obtain are summarized in Table 8.1. Our main technical result consists in showing that Token Sliding is W[1]-hard paramerized by k on bipartite graphs with a reduction from Multicolored Independent Set. We were not able to adapt our reduction for Token Jumping and left it as an open question:

Graph Class	Token Jumping	Token Sliding
$\{C_3, C_4\}$-free graphs	FPT (Section 8.1.1)	Open
C_4-free graphs	W[1]-hard (Section 8.2.1)	W[1]-hard (Section 8.2.1)
Bipartite graphs	Open	W[1]-hard (Section 8.2.2)
Bipartite C_4-free graphs	FPT (Section 8.1.1)	FPT (Section 8.1.3)

Table 8.1: Parameterized complexity of Token Jumping and Token Sliding on several graph classes.

Question 12. *Is* Token Jumping FPT *parameterized by k on bipartite graphs?*

On the positive side, we prove that Token Jumping admits a quadratic kernel (i.e. an equivalent instance of size $O(k^2)$ can be found in polynomial time) for $\{C_3, C_4\}$-free graphs while it is W[1]-hard if we restrict to $\{C_4, \ldots, C_p\}$-free graphs for a fixed constant p (the same hardness result also holds for Token Sliding). Note that the fact that the problem is FPT on graphs of girth at least 5 also follows from FPT algorithms for strongly $K_{3,\ell}$-free graphs of [61], but even if a polynomial kernel can be derived from their result, the degree of our polynomial is better. We were no able to remove the C_4 condition in order to obtain a parameterized algorithm for triangle-free graphs. If an FPT algorithm exists for triangle-free graphs, it would, in particular answer Question 12.

Question 13. *Is* Token Jumping FPT *parameterized by k on triangle-free graphs?*

We then focus on Token Sliding. While FPT algorithms are (relatively) easy to design on sparse graphs for Token Jumping, they are much harder for Token Sliding. As mentionned in the previous chapter, it is still open to determine if Token Sliding is FPT on planar graphs or H-minor free graphs while they follow for instance from [28, 61] for Token Jumping. Our main positive result is that Token Sliding on bipartite C_4-free graphs (i.e. bipartite graphs of girth at least 6) admits a polynomial kernel. Our proof is in two parts, first we show that Token Sliding on bipartite graphs with bounded degree admits a polynomial kernel and then show that, if the graphs admits a vertex of large enough degree then the answer is always positive. So Token Sliding is W[1]-hard on bipartite graphs but FPT on bipartite C_4-free graphs. In our positive results, C_4-freeness really plays an important role (neighborhoods of the neighbors of a vertex x are almost disjoint). It would be interesting to know if forbidding C_4 is really important or whether it is only helpful with our proof techniques. In particular, does Token Sliding admit an FPT algorithm on bipartite C_{2p}-free graphs for some $p \geq 3$? In our hardness reduction for bipartite graphs, all (even) cycles can appear and then such a result can hold. Recall that we prove that Token Jumping admits a polynomial kernel for graphs of girth at least 6. It would be interesting to see if our result on bipartite C_4-free graphs can be extended to this class.

Question 14. *Is* Token Sliding FPT *parameterized by k on graphs of girth at least 5? Or, slightly weaker, is it* FPT *on graphs of girth at least p, for some constant p.*

Note that the fact that the girth is at least 5 is needed since Token Sliding is W[1]-hard on bipartite graphs (which have girth at least 4). Let us finally briefly discuss some cases where we forbid an infinite number of cycles. We have already discussed the case where odd cycles are forbidden. One can wonder what happens if even cycles are forbidden. It is shown in [67] that Token Jumping can be decided in polynomial time for even-hole-free graphs whereas Token Sliding is PSPACE-complete as it is PSPACE-complete on split-graphs [11]. More generally, one can

wonder what happens when we forbid all the cycles of length p mod q for every pair of integers p, q.

8.1 Positive results

8.1.1 Token Jumping on $\{C_3, C_4\}$-free graphs and bipartite C_4-free graphs

We say that a graph class $\mathcal{G}_\varepsilon$ is *ε-sparse*, for some $\varepsilon > 0$, if every induced subgraph G' of a graph $G \in \mathcal{G}_\varepsilon$ has at most $|V(G')|^{2-\varepsilon}$ edges. By extension, G is said to be ε-sparse. Given an instance $\mathcal{I} = (G, S, T, k)$ of Token Jumping, let $H = G - N_G[S \cup T]$ and J denote the graph induced by $N_G[S \cup T]$. In the remainder of this section, we show that $\mathcal{I}$ is a yes-instance whenever (at least) one of the following two conditions is true:

1. H is ε-sparse and contains more than $k(2k)^{1/\varepsilon}$ vertices, or
2. J is $\{C_3, C_4\}$-free and contains a vertex of degree at least $3k$.

Lemma 34. *Let $\mathcal{I} = (G, S, T, k)$ be an instance of* Token Jumping *and let $H = G - N_G[S \cup T]$. If H is an ε-sparse graph with more than $k(2k)^{1/\varepsilon}$ vertices then $\mathcal{I}$ is a yes-instance. Moreover, the length of the shortest reconfiguration sequence from S to T is at most $2k$.*

Proof. First, consider an ε-sparse graph H' with $n > (2k)^{1/\varepsilon}$ vertices. We claim that H' contains a vertex with degree less than $\frac{n}{k}$. Assume otherwise, i.e., suppose that the minimum degree in H' is at least $\frac{n}{k}$. Then, $|E(H')| \geq \frac{n^2}{2k}$. Moreover, since H' is ε-sparse, it holds that $|E(H')| \leq n^{2-\varepsilon}$. However, $\frac{n^2}{2k} \leq n^{2-\varepsilon}$ if and only if $n \leq (2k)^{1/\varepsilon}$, a contradiction.

Now, we shall prove, by induction on k, that H contains an independent set of size at least k. The statement holds for $k = 1$ (since H must contain at least one vertex). Now, consider the case where $k > 1$ and let z be a vertex with minimum degree in H. Following the above claim, z has degree less than $\frac{n}{k}$. Note that the graph $H' = H - N[z]$ is ϵ-sparse and contains at least $(k-1)\frac{n}{k} \geq (k-1)\frac{k(2k)^{1/\varepsilon}}{k} = (k-1)(2k)^{1/\varepsilon}$ vertices. By the induction hypothesis, H' contains an independent set X of size at least $k-1$. Thus, $X \cup \{z\}$ is an independent set in H of size at least k.

Hence, we can tranform S to T by simply jumping all the tokens in S to an independent set $X \subseteq V(G) \setminus (S \cup T)$ and then from X we jump the tokens (one by one) to T. This completes the proof. □

Lemma 35. *Let $\mathcal{I} = (G, S, T, k)$ be an instance of* Token Jumping *and let J denote the graph induced by $N_G[S \cup T]$. If J is $\{C_3, C_4\}$-free and contains a vertex v of degree at least $3k$, then $\mathcal{I}$ is a yes-instance. Moreover, the length of the shortest reconfiguration sequence from S to T is at most $2k$.*

Proof. Fix $w \in S \cup T$. First, observe that for any $u \in N(S \cup T)$, u is either ajdacent to w and no neighbor of w (otherwise J would contain a C_3), or u is adjacent to at

most one neighbor of w (otherwise J would contain a C_4). Therefore, every vertex in $N(S \cup T)$ has degree at most $2k$ and a vertex of degree at least $3k$ in J belongs necessarily to $S \cup T$. As J is C_3-free, $N_J(w)$ is an independent set. Furthermore, for any $u, v \in N_J(w)$, $u \neq v$, we have $N_J(u) \cap N_J(v) = \{w\}$, that is, w is the only common neighbor of u and v in J; otherwise, J would contain C_4. Hence, if w has at least $3k$ neighbors, then at least k of them only have w as a neighbor in $S \cup T$. Thus, we can jump the tokens on S to $N(w)$, starting with the token on w, if any. Then, we can jump the tokens on the vertices in T. Clearly, the length of such a reconfiguration sequence is at most $2k$. □

Proposition 3. *Let $\mathcal{I} = (G, S, T, k)$ be an instance of* Token Jumping*, let $H = G - N_G[S \cup T]$, and let J denote the graph induced by $N_G[S \cup T]$. If H is ε-sparse, $\varepsilon > 0$, and J is $\{C_3, C_4\}$-free then $\mathcal{I}$ admits a kernel with $\mathcal{O}(k^2 + k^{1+1/\varepsilon})$ vertices.*

Proof. If H contains more than $k(2k)^{1/\varepsilon}$ vertices then $\mathcal{I}$ is a yes-instance (by Lemma 34). If J contains a vertex of degree $3k$ or more then, again, $\mathcal{I}$ is a yes-instance (by Lemma 35). Putting it all together, we have $|S \cup T| \leq 2k$, $|N_G(S \cup T)| \leq 2k(3k-1) = 6k^2 - 2k = \mathcal{O}(k^2)$, and $|V(G) \setminus N_G[S \cup T]| \leq k(2k)^{1/\varepsilon} = \mathcal{O}(k^{1+1/\varepsilon})$. □

Theorem 34. Token Jumping *parameterized by k admits a kernel with at most $\mathcal{O}(k^2)$ vertices on $\{C_3, C_4\}$-free graphs as well as bipartite C_4-free graphs.*

Proof. Let $\mathcal{I} = (G, S, T, k)$ be an instance of Token Jumping such that G is $\{C_3, C_4\}$-free. Let $H = G - N_G[S \cup T]$ and J denote the graph induced by $N_G[S \cup T]$. Since J is $\{C_3, C_4\}$-free, Lemma 35 implies that if J contains more than $6k^2 - 2k$ vertices, then $\mathcal{I}$ is a yes-instance. Kim showed that the Ramsey number $R(3, k)$ has order of magnitude $\Omega(k^2/(\log k))$ [69]. Hence, if H contains more than $\mathcal{O}(k^2/(\log k))$ vertices, then it contains an independent set of size k and $\mathcal{I}$ is a yes-instance. Thus, G contains at most $\mathcal{O}(k^2)$ vertices. The same result holds for bipartite C_4-free graphs since they are $\{C_3, C_4\}$-free. □

8.1.2 Token Sliding on bounded-degree bipartite graphs

Unlike the case of Token Jumping, it is not known whether Token Sliding is fixed-parameter tractable (parameterized by k) on graphs of bounded degree. In this section we show that it is indeed the case for bounded-degree bipartite graphs. This result, interesting in its own right, will be crucial for proving that Token Sliding is fixed-parameter tractable on bipartite C_4-free graphs in the next section. We start with a few definitions and needed lemmas.

For $r \geq 0$, the *r-neighborhood* of a vertex $v \in V(G)$ is defined as $N_G^r[v] = \{u \mid dist_G(u, v) = r\}$. We write $B(v, r) = \{u \mid dist_G(u, v) \leq r\}$ and call it a *ball of radius r around v*; for $S \subseteq V(G)$, $B(S, r) = \bigcup_{v \in S} B(v, r)$. Given a graph G and an independent set I of G, let $R(G, I)$ be the set of rigid vertices of I (in G). Recall that $R(G, I) = \{v \mid v \in \bigcap_{I' | I \rightsquigarrow_{TS} I'} I'\}$ is the subset of tokens of I that can never move in any reconfiguration sequence starting from I. An independent set I is said to be

unlocked if $R(G, I) = \emptyset$. Given a graph G and $r \geq 1$, a set $S \subseteq V(G)$ is called an *r-independent set*, or *r-independent* for short, if $B(v, r) \cap S = \{v\}$, for all $v \in S$. Note that a 1-independent set is a standard independent set and a r-independent set, $r > 1$, is a set where the shortest path between any two vertices of the set contains at least r vertices (excluding the endpoints).

For a vertex $v \in V(G)$ and a set $S \subseteq V(G) \setminus \{v\}$, we let $D(v, S)$ denote the set of vertices in S that are closest to v. That is, $D(v, S)$ is the set of vertices in S whose distance to v is minimum. We say $D(v, S)$ is *frozen* if $|D(v, S)| \geq 2$ and it is not possible to slide a single token in $D(v, S)$ to obtain S' such that either $v \in S'$ or $|D(v, S')| = 1$. Note that, in time polynomial in $n = |V(G)|$, it can be verified whether $D(v, S)$ is frozen by simply checking, for each vertex $u \in D(v, S)$, whether u can slide to a vertex w which is closer to v (or to v itself if u is adjacent to v).

Lemma 36 ([46]). *$S \leftrightsquigarrow T$ in G if and only if $R(G, S) = R(G, T)$ and $(S \setminus R(G, S)) \leftrightsquigarrow (T \setminus R(G, S))$ in $G - N[R(G, S)]$. Moreover, if G is bipartite then $R(G, S)$ and $R(G, T)$ can be computed in time linear in $|V(G)| = n$.*

Lemma 37 ([46]). *Let $G = (L \cup R, E)$ be a bipartite graph and let S be an unlocked independent set of G. Then, in time linear in n, we can compute a reconfiguration sequence $\langle S = I_0, I_1, \ldots, I_\ell \rangle$ where $I_\ell \cap L = \emptyset$ and $\ell = |S \cap L|$.*

The next lemma was also proved in [46] but we repeat the proof here both for completeness and since we will use similar ideas in subsequent proofs.

Lemma 38 ([46]). *Let $G = (L \cup R, E)$ be a connected bipartite graph and let S be an unlocked independent set of G. Let $v \in V(G) \setminus S$ and let $D(v, S) \subseteq L$ (or symmetrically $D(v, S) \subseteq R$). Then, in time linear in $|V(G)| = n$, one can find a reconfiguration sequence $\langle S = I_0, I_1, \ldots, I_\ell \rangle$ where $v \in I_\ell$ and ℓ is at most $|S \cap L| - 1$ (or symmetrically $|S \cap R| - 1$) plus the distance between v and a token of $D(v, S)$.*

Proof. There are two cases to consider:
(1) If there is a unique token $u \in D(v, S) \subseteq S$ which is closest to v then the reconfiguration sequence is constructed by repeatedly moving the token on u to a vertex which is closer to v. Let w be any vertex in $N(u)$ where some shortest path from u to v passes through w. Since u is uniquely closest to v, it must be the case that $N(w) \cap S = \{u\}$. Hence, we construct $I_1 = (I \setminus \{u\}) \cup \{w\}$; as w is now uniquely closest to v the process can be iterated. The same strategy can be applied if $D(v, S)$ is not frozen.
(2) Assume $D(v, S)$ is frozen. Let d denote the distance from v to any vertex $u \in D(v, S)$. Without loss of generality, we can assume that $D(v, S) \subseteq L$ (the other case is symmetric). We apply Lemma 37 which guarantees (in linear time) the existence of a computable reconfiguration sequence $\langle S = I_0, I_1, \ldots, I_\ell \rangle$ where $I_\ell \cap L = \emptyset$ and $\ell = |S \cap L|$. There exists an index j, with $j < \ell = |S \cap L|$, where I_j has a unique token u which is closest to v. This follows from the fact that some tokens of $D(v, S)$ will move to be at distance $d + 1$ from v (possibly all but one) leaving a vertex u uniquely

closest to d. Therefore, we can now apply the same strategy as in the previous case. The reconfiguration sequence will be of length at most $j + d$, as needed. □

Let $\mathcal{I} = (G = (V, E), S, T, k)$ be an instance of Token Sliding where G is a bipartite graph of bounded degree Δ. We assume, without loss of generality, that G is connected; as otherwise we can solve the problem independently on each component of G (and there are at most k components containing tokens). Moreover, given Lemma 36, we can assume, without loss of generality, that S and T are unlocked. In other words, we assume that it has been verified that $R(G, S) = R(G, T)$ and $N[R(G, S)]$ has been deleted from G. We now give a slightly different version of Lemma 38 better suited for our needs.

Lemma 39. *Let G be a connected bipartite graph and let S be an unlocked independent set of G. Let v be a vertex in $V(G) \setminus S$ such that $N_G[v] \cap S = \emptyset$. Let $D(v, S) \subseteq L$ (or symmetrically $D(v, S) \subseteq R$) such that $\mathsf{dist}_G(u, v) = d$, for all $u \in D(v, S)$. Then, in time linear in $|V(G)| = n$, we can find a reconfiguration sequence $\langle S = I_0, I_1, \ldots, I_\ell \rangle$, where $I_\ell = (S \setminus \{u\}) \cup \{v\}$ for some vertex u in $D(v, S)$ and ℓ is at most $2(|S| - 1) + d$.*

Proof. Similarly to the proof of Lemma 38, there are two cases to consider:
(1) If there is a unique token $u \in D(v, S)$ which is closest to v or $D(v, S)$ is not frozen then the reconfiguration sequence obtained by repeatedly moving the token on u to a vertex which is closer to v gives us the required sequence. Since no other token is moved, we have $I_\ell = (S \setminus \{u\}) \cup \{v\}$.
(2) In the other case, we have $D(v, S) \geq 2$ and $D(v, S)$ is frozen. We assume, without loss of generality, that $D(v, S) \subseteq L$. We apply Lemma 37 which returns a reconfiguration sequence $\langle S = I_0, I_1, \ldots, I_\ell \rangle$ where $I_\ell \cap L = \emptyset$ and $\ell = |S \cap L|$. There exists an index j, with $j < \ell = |S \cap L|$, where I_j has a unique token $u \in D(v, S)$ which is closest to v. Let $\alpha = \langle I_0, I_1, \ldots, I_j \rangle$. Note that α slides exactly j distinct tokens (not including u) from L to R. We let M_α denote these tokens. Moreover, α is reversable. Hence, we let α^{-1} denote the sequence consisting of applying the slides of α in reverse order. Now, we construct a sequence β of slides that moves the token on u to v. Recall that this is a sequence of exactly d slides that repeatedly slides the same token. We denote the resulting independent set (after applying $\alpha \cdot \beta$) by I_β. We claim that $\gamma = \alpha \cdot \beta \cdot \alpha^{-1}$ is the required sequence that transforms S to $(S \setminus \{u\}) \cup \{v\}$. To see why γ is a valid reconfiguration sequence, it suffices to show that $N_G[M_\alpha] \cap N_G[v] = \emptyset$. Since $N_G[v] \cap S = \emptyset$, we know that $d \geq 2$ if both v and $D(v, S)$ are contained in L (or R) and $d \geq 3$ otherwise. If $\{v\}, D(v, S) \subseteq L$ (or $\{v\}, D(v, S) \subseteq R$) then every vertex in M_α is at distance at least three from v, as needed. Finally, if $v \in L$ and $D(v, S) \subseteq R$ (or $v \in R$ and $D(v, S) \subseteq L$) then every vertex in M_α is at distance at least four from v. □

Lemma 40. *If G is a connected graph and S and T are any two 2-independent sets of G such that $S \cup T$ is also 2-independent then $S \leftrightsquigarrow T$ in G.*

Proof. We proceed by induction on $|S\Delta T| = |(S \setminus T) \cup (T \setminus S)|$, i.e., the size of the symmetric difference between S and T. If $|S\Delta T| = 0$ then $S = T$ and there is nothing to prove. Hence, we assume that the statement is true for $|S\Delta T| = q > 0$. We compute a shortest path between all pairs of vertices (u, v) in G, where $u \in S \setminus T$ and $v \in T \setminus S$. We let (u, v) denote a pair where the distance is minimized and we fix a shortest path between u and v. There are two cases to consider:
(1) If $S \cap T = \emptyset$ then we can simply slide u to v along the shortest path and we are done. To see why, recall that both S and T are 2-independent. Hence, they are both unlocked and if there is more than one vertex in $S \setminus T$ that is closest to v then we can simply slide u into one of its neighbors, say w, that is closer to v to obtain a unique vertex which is closest to v; none of those neighbors are adjacent to a vertex in S since S is 2-independent. Now, assume that there exists a vertex x along the shortest path from w to v such that $x \in N(y)$, $y \in S$. This contradicts the choice of u since y is closer to v.
(2) If $S \cap T \neq \emptyset$, then there are two cases. When the shortest path P from u to v does not contain any vertex in $N_G[S \cap T]$, then we apply the same reasoning as above. Otherwise let $X := P \cap (S \cap T)$ be the subset of vertices of $S \cap T$ that lie on P and $Y := (N(P) \setminus P) \cap (S \cap T)$ be the subset of vertices of $S \cap T$ that do not lie on P but have at least one neighbor on P. Note that the tokens on X and Y are the only tokens that forbid us to directly move the token on u to v along P. Furthermore, each vertex in Y has at most three neighbors in P and these neighbors are consecutive in P since it is a shortest path.
We construct from X and Y the sequence $A = a_1, a_2, \ldots, a_p$: if the shortest path from u to v visits a vertex in X or visits neighbors of a vertex in Y, then we add the vertex to A (in the order in which the visits occur when traversing the path from u to v). Then we let $a_0 := u$ and $a_{p+1} := v$. We now proceed iteratively as follows: For every $i \in \{p, \ldots, 0\}$ in decreasing order we consider the token on a_i. If it belongs to Y, then we slide it to one of its neighbor on the path chosen arbitrarily (which is a valid move since $S \cap T$ is a 2-independent set) and then slide it along P to the first neighbor of a_{i+1} (these are valid moves by definition of a_i and a_{i+1}) and then slide it to a_{i+1}. If it belongs to X, then we simply slide it along P up to the first neighbor of a_{i+1} and then slide it to a_{i+1} (and these are valid moves for the same reasons). Once the token reaches a_{i+1} we obtain again a 2-independent set (since S, T, and $S \cup T$ are 2-independent set) and thus we can iterate. After the last move from u to a_1 we obtain the independent set $S' := S \setminus \{u\} \cup \{v\}$, which concludes the proof. □

Let G be a graph and let $X \subseteq V(G)$. The *interior* of X is the set of vertices in X at distance at least three from $V(G) \setminus X$ (separated by at least two vertices). We say a set X is *fat* if its interior is connected and contains a 2-independent set of size at least $2k$.

Lemma 41. *Let G be a graph of maximum degree Δ. Let $v \in V(G)$ and $r \in \mathbb{N}$. If $B(v, r)$ contains more than $2k(1 + \Delta + \Delta^2)^2$ vertices then $B(v, r)$ is fat.*

Proof. We only need to prove that the interior of $B(v,r)$, that is $B(v,r-2)$, contains a 2-independent set of size at least $2k$; as $B(v,r-2)$ is connected by construction. First, note that any graph of maximum degree Δ on more than $2k(1+\Delta+\Delta^2)$ vertices must contain a 2-independent set of size at least $2k$. So it suffices to show that $B(v,r-2)$ contains more than $2k(1+\Delta+\Delta^2)$ vertices. We divide $B(v,r)$ into layers, where $L_0=\{v\}$, $L_1=N(v)$, ..., and $L_r=N^r(v)$. Since G has maximum degree Δ, for every $i\geq 1$, layer L_i contains at most $(\Delta-1)^{i-1}\Delta$ vertices. If $B(v,r-2)$ contains more than $2k(1+\Delta+\Delta^2)$ vertices then we are done. Otherwise, L_{r-2} must contain at most $2k(1+\Delta+\Delta^2)$ vertices. Consequently, $L_{r-1}\cup L_r$ would contain at most $2k\Delta(1+\Delta+\Delta^2)+2k\Delta^2(1+\Delta+\Delta^2)=(1+\Delta+\Delta^2)(2k\Delta+2k\Delta^2)$ vertices. Therefore, $B(v,r)$ contains at most $2k(1+\Delta+\Delta^2)+(1+\Delta+\Delta^2)(2k\Delta+2k\Delta^2)=(1+\Delta+\Delta^2)(2k+2k\Delta+2k\Delta^2)$ which is equal to $2k(1+\Delta+\Delta^2)^2$ vertices, a contradiction. □

Lemma 42. *Let $\mathcal{I}=(G,S,T,k)$ be an instance of* Token Sliding *where G is a bipartite graph. If $V(G)\setminus(S\cup T)$ contains a fat set X then $\mathcal{I}$ is a yes-instance.*

Proof. First, recall that we assume that G is connected and both S and T are unlocked. Let I be a 2-independent set of size $2k$ in the interior of X (at distance at least three from any vertex outside of X). We prove that S can be transformed into $S'\subset I$. Similar arguments hold for transforming T into $T'\subset I$. Hence, the statement of the theorem follows by applying Lemma 40 on S' and T'.

We proceed by induction on $|S\Delta S'|$, i.e., the size of the symmetric difference between S and S'. If $|S\Delta S'|=0$ then $S=S'$ and we are done. Otherwise, we reduce the size of the symmetric difference as follows. Recall that initially $S\cap S'=\emptyset$; as $X\subseteq V(G)\setminus(S\cup T)$. However, the size of the intersection will increase as more tokens are moved to S'. We pick a pair (u,v) such that $u\in S\setminus S'$ and $v\in S'$ and the distance between u and v is minimized. There are two cases 2 consider:
(1) If v does not contain a token (in other words $v\in S'\setminus S$) then the shortest path from u to v does not intersect with $N_G[S'\cap S]$. We therefore invoke Lemma 39 in the graph $G-(N[S'\cap S])$. This guarantees that the token on u slides to v and every other token remains in place.
(2) Otherwise, v already contains a token (or $v\in S'\cap S$). We invoke Lemma 40 on the graph induced by the interior of X and transform $C=S'\cap S\subset I$ into another 2-independent set $C'\subseteq I$ that does not contain v; this is possible since $|C|=|C'|\leq k$. Now we can again invoke Lemma 39 similarly to the previous case. □

Theorem 35. Token Sliding *parameterized by k admits a kernel with $\mathcal{O}(k^2\Delta^5)$ vertices on bipartite graphs of maximum degree Δ. Moreover, the problem can be solved in $\mathcal{O}^\star(k^{2k}\Delta^{5k})$-time.*

Proof. Let $\mathcal{I}=(G,S,T,k)$ be an instance of Token Sliding where G is a bipartite graph of maximum degree Δ. We assume, without loss of generality, that G is connected and S and T are unlocked; for otherwise we can solve connected components independently and we can return a trivial no-instance if $R(G,S)\neq R(G,T)$

(Lemma 36). Next, from Lemmas 41 and 42, we know that each connected component of $V(G) \setminus (S \cup T)$ contains at most $\mathcal{O}(k\Delta^4)$ vertices; otherwise we can return a trivial yes-instance. Since the number of components in $V(G) \setminus (S \cup T)$ is bounded by $2k\Delta$ and $|S \cup T| \leq 2k$, we get the desired bound. To solve the problem, it suffices to construct the complete reconfiguration graph and verify if S and T belong to the same connected component. This concludes the proof. □

8.1.3 Token Sliding on bipartite C_4-free graphs

Equipped with Theorem 35, we are now ready to prove that Token Sliding admits a polynomial kernel on bipartite C_4-free graphs. Our strategy will be simple. We show that if the graph contains a vertex of large degree then we have a yes-instance. Otherwise, we invoke Theorem 35 to obtain the required kernel.

We start with a few simplifying assumptions. Let $\mathcal{I} = (G, S, T, k)$ be an instance of Token Sliding where $G = (L \cup R, E)$ is a connected bipartite C_4-free graphs. We assume that both S and T are unlocked (Lemma 36). Moreover, we assume without loss of generality, that each vertex in G can have at most one pendant neighbor. This assumption is safe (for any instance of Token Sliding) because no two tokens can occupy two pendant neighbors of a vertex; as otherwise S or T would be locked. Moreover, if a token is placed on a pendant neighbor of a vertex v then no other token can reach v. We can then safely identify pendant neighbors of a vertex.

Let $v \in V(G)$ be a vertex of degree at least $k^2 + k + 1$ in G. We let u_p denote the pendant neighbor of v (if it exists). We assume, without loss of generality, that $v \in L$. We let $N_1 = N_G(v) \setminus \{u_p\} = \{u_1, u_2, \ldots, u_q\}$, $N_2 = N^2_G[v]$, and $N_3 = N^3_G[v]$. Since G is bipartite, $N_1 \subseteq R$, $N_2 \subseteq L$, and $N_3 \subseteq R$. Moreover, since G is C_4-free, each vertex in N_2 has exactly one neighbor in N_1. Therefore, we partition N_2 into sets N_{u_1}, N_{u_2}, …, N_{u_q}, where each set N_{u_i} contains the neighbors of u_i in N_2, that is, $N(u_i) \setminus \{v\}$. We also partition N_3 into two sets M_{small} and M_{big}. Each vertex in M_{big} contains vertices connected to at least $k + 1$ sets in N_2. Note that, because of C_4-freeness, each vertex in N_3 is connected to at most one vertex of any set N_{u_i}. We let $M_{\text{small}} = N_3 \setminus M_{\text{big}}$. Each vertex in M_{small} has at most k neighbors in N_2. In other words, each vertex in M_{small} is connected to at most k sets, each one of those sets being the neighborhood of a vertex in N_1.

We now proceed in five stages. We first show how to transform S to S_1 such that $S_1 \cap B(v, 3) \subseteq N_2$. In other words, we can guarantee that all tokens in the ball of radius three around v are contained in N_2. We then tranform S_1 to S_2 such that $S_2 \cap B(v, 3) \subseteq N_1 \cup N_3$. Next, we tranform S_2 to S_3 such that $S_3 \cap B(v, 3) \subseteq N_1 \cup M_{\text{small}}$. Then, we tranform S_3 to S_4 such that $S_4 \cap B(v, 3) \subseteq N_1$ and finally to S_5 such that $S_5 \subseteq N_1$. By applying the same strategy starting from T, we obtain $T_5 \subseteq N_1$. We conclude our proof by showing that S_5 can be transformed to T_5.

Lemma 43. *Let S be an unlocked independent set of G of size k. Then, there exists S' such that $S \leftrightsquigarrow S'$ and $S' \cap B(v, 3) \subseteq N_2$.*

Proof. We invoke Lemma 37 and move all tokens in R to L (since S is unlocked). We denote the resulting set by S'. Consequently, we know that $S' \cap B(v,3) \subseteq L$. If there is no token on v then we are done; as $v \in L$, $N_1 \subseteq R$, $N_2 \subseteq L$, and $N_3 \subseteq R$. Otherwise, given that v has degree at least $k+1$, there must exist at least one path $P = v, x, y$ such that $N_G[P] \cap S' = \{v\}$. Hence, we can slide the token on v to y. This completes the proof. □

Lemma 44. *Let S be an unlocked independent set of G of size k such that $S \cap B(v,3) \subseteq N_2$. Then, there exists S' such that $S \leftrightsquigarrow S'$ and $S' \cap B(v,3) \subseteq N_1 \cup N_3$.*

Proof. Since $S \cap B(v,3) \subseteq N_2$, we simply have to invoke Lemma 37 and move all tokens in L to R. Note that no token can reach u_p in a single slide and thus every token that moves from N_2 to $N(v)$ necessarily moves to N_1. □

Lemma 45. *Let S be an unlocked independent set of G of size k such that $S \cap B(v,3) \subseteq N_1 \cup N_3$. Then, there exists S' such that $S \leftrightsquigarrow S'$ and $S' \cap B(v,3) \subseteq N_1 \cup M_{small}$.*

Proof. We make use of the fact that each vertex in M_{big} is connected to at least $k+1$ sets in N_2 and hence is connected (via a vertex in N_2) to at least $k+1$ vertices in N_1. Let w be a vertex in $S \cap M_{\text{big}}$; if $S \cap M_{\text{big}} = \emptyset$ then $S \cap B(v,3) \subseteq N_1 \cup M_{\text{small}}$ and we are done. Recall that $|S \cap N_1| + |S \cap N_3| \leq k$, no two vertices in N_3 have two common neighbors in N_2, and no two vertices in N_2 have two common neighbors in N_1. Hence, there exists at least $k+1$ vertex-disjoint path connecting v to w. At least one such path, say $P = \{w, x, y, v\}$, satisfies $N_G[P] \cap S = \{w\}$. We slide w to z and call the resulting set again S for simplicity. This process is repeated as long as there are tokens in M_{big}. We let S' denote the resulting set, i.e., where $S' \cap M_{\text{big}} = \emptyset$. □

Lemma 46. *Let S be an unlocked independent set of G of size k such that $S \cap B(v,3) \subseteq N_1 \cup M_{small}$. Then, there exists S' such that $S \leftrightsquigarrow S'$ and $S' \cap B(v,3) \subseteq N_1$.*

Proof. Since $S \cap B(v,3) \subseteq N_1 \cup M_{\text{small}}$, we know that every token not in N_1 must be in M_{small}. We let A denote the subset of M_{small} containing tokens. Note that if A is empty, then we are done. Otherwise, we know that each token in A is connected to at most k sets in N_2 (by construction) and therefore at most k vertices in N_1. We let B denote the at most k^2 subsets of N_2 that contain a vertex with a neighbor in A. We let C denote the at most k^2 vertices of N_1 whose neighborhoods are in B. We proceed in two stages. First we move all tokens in C to some vertex in $N_1 \setminus C$. To do so, we invoke Lemma 39 as follows. If there are any tokens originally in $N_1 \setminus C$, then we move them to one of their neighbors in N_2 (this is possible since no two vertices in N_2 have two common neighbors in N_1 and there are no tokens in M_{big}). We call the resulting set S''. Note that since $|C|$ is at most k^2, we have $|N_1 \setminus C| > k$. Therefore, there exists at least one vertex u in $N_1 \setminus C$ such that $N[u] \cap S = N[u] \cap S'' = \emptyset$. Consequently, we have $D(u, S'') \subseteq C$ (at distance two) and we can apply Lemma 39 to move one token from C to u and then reverse the slides of the tokens originally

in $N_1 \setminus C$. We repeat this procedure as long as there are tokens in C. In the second stage, we apply a similar procedure to move all tokens in A to some vertex in C and then from C to a vertex in $N_1 \setminus C$. This is possible because after sliding the tokens originally in $N_1 \setminus C$ to their corresponding neighbors in N_2 the vertices in A become closest to vertices in C (at distance two). □

Lemma 47. *Let S and T be two unlocked independent sets of G of size k such that $S \subseteq N_1$ and $T \subseteq N_1$. Then, $S \leftrightsquigarrow T$, all the moves are between vertices of $B(v, 2)$, and this sequence can be computed in polynomial time.*

Proof. As long as there exists $u \in S \setminus T$ and $w \in T \setminus S$ we can slide u to w as follows. Slide all tokens (except u) to one of their neighbors in N_2. Then slide u to v and then slide from v to w. Finally, reverse all the other slides from N_2 to N_1. □

Lemma 48. *Let S be an unlocked independent set of G of size k such that $S \cap B(v, 3) \subseteq N_1$. Then, there exists S' such that $S \leftrightsquigarrow S'$ and $S' \subseteq N_1$.*

Proof. We let $X = S \cap B(v, 3)$. Since $S \cap B(v, 3) \subseteq N_1$, every vertex of $S \setminus X$ is at distance at least three from X. We compute the shortest path from every vertex in $S \setminus X$ to every vertex in $N_1 \setminus X$. We let (u, w) denote a pair with the minimum distance, where $u \in N_1 \setminus X$ and $w \in S \setminus X$. If w is uniquely closest to u then there are two cases to consider:

1.1 When the shortest path from u to w does not intersect with $N[X]$ then we simply slide u to w.

1.2 Otherwise, if the shortest path P intersects with $N[X]$, then there exists a first vertex $x \in X$ such that $P \cap N_G[x] \neq \emptyset$. Therefore, we apply Lemma 47 in $G[B(v, 3)]$ to transform X into a set X' such that $u \in X'$ and $x \notin X'$. This yields a valid sequence since Lemma 47 only moves tokens within $B(v, 2)$ and vertices in $S \setminus X$ are at distance at least four from v. Then we can safely slide w to x.

Suppose now that w is not uniquely closest to u. Recall that $D(u, S \setminus X)$ is contained in $V(G) \setminus B(v, 3)$ (at distance at least three from u). We apply Lemma 37 in $G' = (L' \cup R', E) = G - (\{v\} \cup N_1 \cup N_2)$ where L' denote the part of the bipartition that contains $D(u, S \setminus X)$. This guarantees that all tokens in L' will move to R' via a single slide. Hence, there must exists a first index j, in this sequence, where the corresponding independent set I_j falls into one of the following three cases:

2.1 $|I_j \cap M_{\text{small}}| = 1$;

2.2 $|I_j \cap M_{\text{big}}| = 1$; or

2.3 $I_j \cap N_3 = \emptyset$ and there exists a token in I_j which is uniquely closest to u.

In case (2.1), we apply Lemma 46, for case (2.2) we apply Lemma 45, and finally for case (2.3) we apply either case (1.1) or case (1.2). This completes the proof. □

Lemma 49. *Let $\mathcal{I} = (G, S, T, k)$ be an instance of* Token Sliding *where G is a connected bipartite C_4-free graph and S and T are unlocked. If there exists a vertex $v \in V(G)$ of degree at least $k^2 + k + 1$ then $\mathcal{I}$ is a yes-instance.*

Proof. Recall that we can always assume that each vertex has at most one pendant neighbor as stated at the beginning of this section. Using Lemmas 43 to 48 we transform S to S' and T to T' such that $S' \subseteq N_1$ and $T' \subseteq N_1$. Then we transform S' to T' by invoking Lemma 47. □

Theorem 36. Token Sliding *parameterized by k admits a kernel with $\mathcal{O}(k^{12})$ vertices on bipartite C_4-free graphs.*

Proof. Let $\mathcal{I} = (G, S, T, k)$ be an instance of Token Sliding where G is a bipartite C_4-free graphs. We assume, without loss of generality, that G is connected and S and T are unlocked; we can solve connected components independently and we can return a trivial no-instance if $R(G, S) \neq R(G, T)$ (Lemma 36). Next, from Lemma 49, we know that each vertex has degree at most $k^2 + k$; otherwise we can return a trivial yes-instance. Finally, we invoke Theorem 35 to obtain the required kernel. □

8.2 Hardness results

8.2.1 Token Sliding and Token Jumping on C_4-free graphs

In the Grid Tiling problem we are given an integer $k \geq 0$ and k^2 sets $S_{i,j} \subseteq [m] \times [m]$, for $0 \leq i, j \leq k-1$, of cardinality n called *tiles* and we are asked whether it is possible to find an element $s^*_{i,j} \in S_{i,j}$ for every $0 \leq i, j \leq k-1$ such that $s^*_{i,j}$ and $s^*_{i,j+1}$ share the same first coordinate while $s^*_{i,j}$ and $s^*_{i+1,j}$ share the same second coordinate for each $0 \leq i, j \leq k-1$ (including modulo k). It was proven in [34] that Grid Tiling parameterized by k is W[1]-hard. We prove the next theorem via a reduction from Grid Tiling. Following the construction in [20] to give a graph G with the desired properties and extending it to a $\{C_4, \ldots, C_p\}$-free graph G' which gives a reduction to Token Sliding.

Theorem 37. *For any $p \geq 4$, Token Sliding is W[1]-hard on $\{C_4, \ldots, C_p\}$-free graphs.*

Construction of G. Given an instance of Grid Tiling, $S_{i,j} \subseteq [m] \times [m]$ $(0 \leq i, j \leq k-1)$ and an integer $p \geq 4$, we use the construction described in [20] to create a graph G with the following properties:

- **P1** - G can be partitioned into $8k^2(p+1)$ cliques $V_1, \ldots, V_{8k^2(p+1)}$ of size n with some edges between them.
- **P2** - G is $\{C_4, \ldots, C_p\}$-free.
- **P3** - The instance of Grid Tiling has a solution if and only if $\exists I \subseteq V(G)$, such that I is an independent set of size $8k^2(p+1)$

Note that as each V_i is a clique, any independent set of G can have at most one vertex in every V_i.

Construction of G'. For $k' = 8k^2(p+1)$, we construct an instance of Token Sliding $(G', S, T, k' + (3k'+1)\frac{p}{2} + \frac{p}{2})$ by extending the graph G to a new graph G'. We label the k' cliques in G arbitrarily as $V_1, \ldots, V_{k'}$. For each $1 \leq i \leq k'$ we add two vertices x_i and y_i adjacent to all vertices in V_i. These will respectively be starting and ending positions of tokens. Informally, we want to force all the tokens to be in their respective V_i at the same time to obtain an independent set in G of size k'. We do this by creating *guard paths*, which are paths on p vertices that will be alternating between starting and target positions of tokens. Note that we can assume p is even, since if p is odd we can use $p+1$ instead to create a graph which is $\{C_4, \ldots, C_p\}$-free. Let P_G be a guard path with vertices $g_1, \ldots, g_p$ and for each x_i let P_{x_i} be a guard path with vertices $x_{i1}, \ldots, x_{ip}$ such that x_i is adjacent to x_{ip} and g_p is adjacent to x_{i1}. For each y_i let P_{y_i} be a guard path with vertices $y_{i1}, \ldots, y_{ip}$ such that y_i is adjacent to y_{i1} and g_1 is adjacent to y_{ip}. Finally, for each i let P_{z_i} be a guard path between x_i and y_i with vertices $z_{i1}, \ldots, z_{ip}$ such that x_i is adjacent to z_{ip} and y_i is adjacent to z_{i1}. This completes the construction of G'. The source independent set S is the set containing all of the x_i and all of the guard path vertices with odd indices:

$$S = \bigcup_i \{x_i, x_{ij}, y_{ij}, z_{ij}, g_j \mid j \text{ is odd}\}.$$

The target independent set T consists of all of the y_i and all of the guard path vertices with even indices:

$$T = \bigcup_i \{y_i, x_{ij}, y_{ij}, z_{ij}, g_j \mid j \text{ is even}\}.$$

Lemma 50. *For any $p \geq 4$, G' is $\{C_4, \ldots, C_p\}$-free.*

Proof. By **P2**, G is $\{C_4, \ldots, C_p\}$-free. Any cycle which contains a vertex on one of the guard paths has length greater than p. Thus if a cycle of length ℓ exists for some $4 \leq \ell \leq p$ it must only have vertices in $V(G) \cup \{x_i, y_i \mid 1 \leq i \leq k'\}$ and contains at least one of x_i or y_i. Assume, without loss of generality, that it contains x_i, then the vertices adjacent to x_i in the cycle must be in V_i. As V_i is a clique the cycle contains a C_3 so is not induced. □

Lemma 51. *If there is a solution to the Grid Tiling instance then there is a reconfiguration sequence from S to T in G'.*

Proof. By **P3**, there exists an independent set I containing one vertex v_i in every V_i. This gives the following reconfiguration sequence from S to T.

1. Move each token on x_i to v_i.

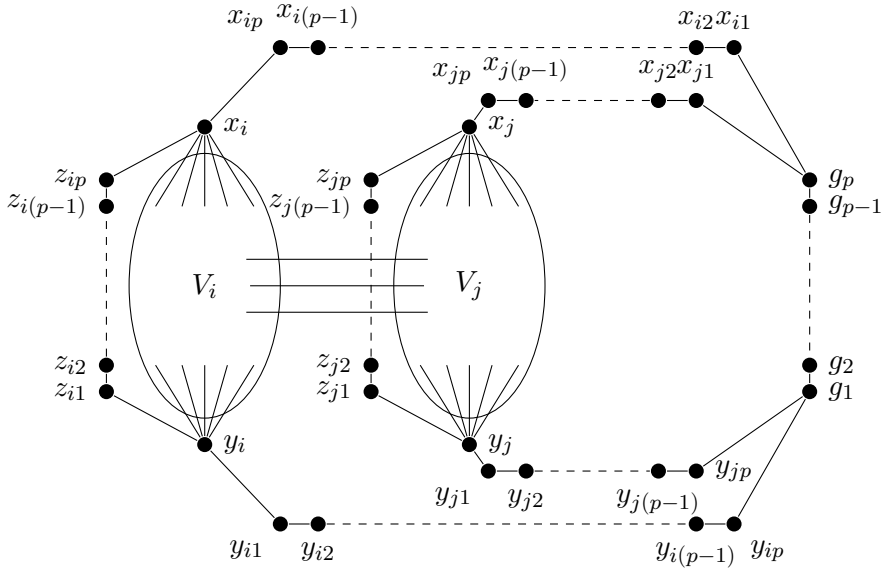


FIGURE 8.1: The construction of G' for two cliques V_i, V_j in G.

2. Move the tokens along the guard paths: for all odd j starting with the greatest j values move the token on each z_{ij} to $z_{i(j+1)}$, then move the tokens on x_{ij} to $x_{i(j+1)}$, g_j to g_{j+1} and finally y_{ij} to $y_{i(j+1)}$

3. Move each token on v_i to y_i.

This completes the proof. □

Finally let us prove the converse direction. For each i, let $W_i := \{x_i, y_i\} \cup V_i$. Let us first show that in any valid reconfiguration sequence the tokens initially on the guard paths, P_{x_i}, P_{y_i}, P_{z_i}, and P_G are stuck on their respective paths. We first need the following simple observation.

Observation 15. *Let I be an independent set of G' of size $k' + (3k'+1)\frac{p}{2}$ such that for every $i \leq k'$, $|W_i \cap I| = 1$. Then for every guard path P of G' we have $|I \cap P| = \frac{p}{2}$.*

Proof. We assume there are exactly k' tokens on $\cup_{i=1}^{k'} W_i$. Then since for any guard path P we have $|P \cap I| \leq \frac{p}{2}$, there must be exactly $\frac{p}{2}$ tokens on each of the $3k'+1$ guard paths. □

Lemma 52. *Let $I_1, I_2, \ldots, I_e$ be a valid reconfiguration sequence such that $I_1 = S$ and $I_e = T$. For every $s \leq e$ and for every $i \leq k'$, $|W_i \cap I_s| = 1$.*

Proof. By construction the statement is true for I_1. Consider the smallest integer $s \leq e$ such that I_e does not satisfy the condition of Lemma 52. By this choice of s we have $|W_j \cap I_r| = 1$ for every $r < s$ and every $j \leq k'$, hence there exists a unique $i \leq k'$ such that $|W_i \cap I_s| = 0$ or $|W_i \cap I_s| = 2$. Let us show that we obtain a contradiction in both cases:
Case 1: $|W_i \cap I_s| = 0$. Since $|W_j \cap I_s| = 1$ for every $j \neq i$, there can be no move from V_i to V_j if there is a token on V_i in I_{s-1}. So there must be a token on one of x_i, y_i

in I_{s-1} and this token must move on an adjacent guard path P. But then since I_{s-1} satisfies the condition of Observation 15, we have $|P \cap I_s| = \frac{p}{2} + 1$, a contradiction. Case 2: $|W_i \cap I_s| = 2$. If $|V_i \cap I_{s-1}| = 1$ then by construction no token can move to V_i between times $s-1$ and s. Hence we can suppose w.l.o.g that $I_{s-1} \cap W_i = \{x_i\}$ and $I_s \cap W_i = \{x_i, y_i\}$. So it must be that a token moves either from y_{i1} to y_i or from z_{i1} to y_i at time $s-1$ and then either $z_{i1} \notin I_{s-1}$ or $y_{i1} \notin I_{s-1}$. In both case, since I_{s-1} satisfies the condition of Observation 15, we obtain that there must be a token on every vertex with even index on the guard paths P_{x_i}, P_{y_i}, $P_{z,i}$ and P_G. In particular we have $\{x_{ip}, x_i\} \subseteq I_s$, a contradiction. □

Lemma 53. *If there is a reconfiguration sequence from S to T in G' then there is a solution to the* Grid Tiling *instance.*

Proof. Given the reconfiguration sequence $I_1, I_2, \ldots, I_e$ such that $I_1 = S$ and $I_e = T$ let us consider the last time $t-1$ at which a token moves from x_i for some $i \leq k'$. Such a time exists since all the tokens must move at least one time in a reconfiguration sequence from S to T. By Lemma 52 this token moves from x_i to V_i. In particular, there is no token on x_{ip} in I_{t-1}, and since I_{t-1} satisfies the condition of Observation 15, there must be a token on g_s for every s odd. This in turn implies that for every j there must be a token on y_{js} for every s odd and in particular for y_{j1}, so there cannot be a token on any y_j. Thus for every $j \leq k'$, $|V_j \cap I_t| = 1$ by Lemma 52, giving an independent set of size k' in G. By **P3**, we know that this implies a solution for the Grid Tiling instance. □

The combination of Lemmas 50, 51 and 53 give us the result of Theorem 37.

Lemma 54. *Let I be an independent set of G' of size $k' + (3k'+1)\frac{p}{2}$ then I is a maximum independent set of G'.*

Proof. First note that, by Observation 15, I has $\frac{p}{2}$ tokens on every guard path and exactly one token in every $W_i := \{x_i, y_i\} \cup V_i$. Assume I is not maximum, so there is some independent set I' of G' with $|I'| > |I|$. The maximum size of an independent set on a path of length p is $\frac{p}{2}$, so I' must have 2 tokens in some W_i which must be on x_i and y_i. However this implies that there can only be $\frac{p}{2} - 1$ tokens in I' on P_{z_i}. Thus $|I'| \leq |I|$. □

Corollary 1. *For any $p \geq 4$, Token Jumping is $\mathsf{W}[1]$-hard on $\{C_4, \ldots, C_p\}$-free graphs.*

Proof. G' is a single fully-connected component and by Lemma 54 the starting set S is a maximum set of G'. Thus the Token Sliding instance is equivalent to a Token Jumping instance and the reduction from Grid Tiling holds. □

8.2.2 Token Sliding on bipartite graphs

This section is devoted to proving the following theorem:

Theorem 38. *Token sliding on bipartite graphs is* W[1]*-hard parameterized by* k*.*

The proof of Theorem 38 consists in a reduction from Multicolored Independent Set which is known to be W[1]-hard parameterized by k (see for instance [34]). In what follows, $\mathcal{I} := (G, k, (V_1, \ldots, V_k))$ denotes an instance of Multicolored Independent Set. In Section 8.2.2.1, we detail the construction of the equivalent instance $\mathcal{I}' := (G', I_s, I_e, 4k+2)$ of Token Sliding, where G' is a bipartite graph and I_s, I_e are independent sets of size $4k+2$, and we prove that if $\mathcal{I}$ is a yes-instance then $\mathcal{I}'$ is a yes-instance. The more involved proof of the converse direction is detailed in Sections 8.2.2.2 and 8.2.2.3.

8.2.2.1 Construction of G'

In what follows, $V(G') := (\mathcal{A}, \mathcal{B})$ denotes the bipartition of G'. For every $p \in \{1, \ldots, k\}$, both $\mathcal{A}$ and $\mathcal{B}$ contain two copies of the set V_p denoted as A_{2p-1}, A_{2p} and B_{2p-1}, B_{2p} respectively, plus some additional vertices that will be described in the next subsection. Two vertices $u', v' \in V(G')$ are said to be *equivalent* and we write $u' \sim v'$ if and only if they are copies of the same vertex in G. With this definition, every vertex $u \in V_p$ has exactly four copies in G' (one in each copy of V_p). Note that the $\sim$ relation is transitive and symmetric. We also define the sets $A := \cup_{p=1}^{k} A_{2p-1} \cup A_{2p}$ and $B := \cup_{p=1}^{k} B_{2p-1} \cup B_{2p}$. For every vertex u' of $A \cup B$, the *corresponding vertex* of u' denoted as $orr(u')$ is the unique vertex $u \in V(G)$ that u' is a copy of. With these definitions at hand, we can now explain how the copies of the sets $V_1, V_2, \ldots, V_k$ are connected in G'. For every two vertices $u' \in A_i$ and $v' \in B_j$ there is an edge connecting u' to v' in G' if and only if:

1. A_i and B_j are not copies of the same subset of $V(G)$ and $(orr(u'), orr(v')) \in E(G)$, or
2. A_i and B_j are copies of the same subset of $V(G)$ and $u' \nsim v'$.

In other words, if A_i and B_j are not copies of the same subset, we connect these sets in the same way there corresponding sets are connected in G. If at the contrary A_i and B_j are copies of the same subset, then $G'[A_i \cup B_j]$ induces a complete bipartite graph minus the matching consisting of every two pairs of equivalent vertices in $A_i \cup B_j$. The connection between four copies of the same subset of $V(G)$ is illustrated in Figure 8.2. Let us explain how we make use of such a construction. The following observation follows directly from the definition of G':

Observation 16. *Let I' be an independent set of G' such that for every $p \in 1, 2, \ldots, k$ we have $I' \cap A_{2p-1} = \{u_{2p-1}\}$ and $I' \cap B_{2p-1} = \{v_{2p-1}\}$. Then the set $I := \{orr(u_1), \ldots, orr(u_k)\}$ is a multicolored independent set of G.*

Proof. For any two $i, j \in 1, 2, \ldots, k$, u_{2i-1} and v_{2j-1} are non-neighbor in G' since I' is an independent set. Furthermore, if $i \neq j$ then A_{2i-1} and B_{2j-1} are not copies of the same subset of $V(G)$ and thus $orr(u_{2i-1}) \neq orr(v_{2j-1})$, so the set

I contains k distinct vertices of G. Since $orr(u_{2j-1}) = orr(v_{2j-1})$, we have that $(orr(u_{2i-1}), orr(u_{2j-1})) \notin E(G)$ for any two $i \neq j$, and since $orr(u_{2i-1}) \in V_{2i-1}$ by construction, the set I is a multicolored independent set of G. □

Observation 16 ensures that any independent set of a reconfiguration sequence of G' having exactly one vertex in A_{2p-1} and one vertex in B_{2p-1} for every $p \in 1, 2, \dots k$ corresponds to a multicolored independent set of G. Note that up to that point, we did not make use of the sets A_{2p} and B_{2p}. The following observation explains why we need two copies of every V_p in both sides of the bipartition:

Observation 17. *Let I' be an independent set of G' and $p \in 1, 2, \dots, k$ such that $I' \cap A_{2p-1} = \{u_{2p-1}\}$, $I' \cap A_{2p} = \{u_{2p}\}$, and $u_{2p-1} \sim u_{2p}$. Then the tokens on u_{2p-1} and u_{2p} cannot move to B.*

Proof. By construction $N(u_{2p-1}) \cap B = N(u_{2p}) \cap B$ since these two vertices are equivalent. It follows that none of the two tokens on u_{2p} nor u_{2p-1} can move to B. □

If at some point in the reconfiguration sequence two tokens are positioned on equivalent vertices in A, then these tokens lock each other at their respective position in some sense. Note that by symmetry of the construction, the same observation can be made when two tokens are positioned on equivalent vertices in B. On the contrary, if two tokens on the same copies of V_p in A are positioned on two non-equivalent vertices we have the following:

Observation 18. *Let I' be an independent set of G' and $p \in 1, 2, \dots k$ such that $I' \cap A_{2p-1} = \{u_{2p-1}\}$, $I' \cap A_{2p} = \{u_{2p}\}$, and $u_{2p-1} \nsim u_{2p}$. Then $I' \cap (B_{2p-1} \cup B_{2p}) = \emptyset$.*

Proof. By construction $B_{2p-1} \cup B_{2p} \subseteq N(u_{2p-1}) \cup N(u_{2p})$ since these two vertices are not equivalent. □

This observation not only ensures that $B_{2p-1} \cup B_{2p} = \emptyset$ but also ensures that no other token but the ones positioned on u_{2p-1} and u_{2p} can move to $B_{2p-1} \cup B_{2p}$. Then, by Observations 17 and 18, either there are two tokens on equivalent vertices in $A_{2p-1} \cup A_{2p}$ and then these tokens cannot move to B (and ensures that if there is a token on $B_{2p-1} \cup B_{2p}$ it must be on an equivalent vertex), or there are two tokens on non-equivalent vertices forbidding any other token to move to $B_{2p-1} \cup B_{2p}$.

Definition of the initial and target independent sets. The initial independent set I_s consists in two sets of $2k$ vertices A_{start} and A_{end} plus two vertices s_A, e_A included in $\mathcal{A}$, and the target independent set I_e consists in two sets of $2k$ vertices B_{start} and B_{end} plus two vertices s_B, e_B included in $\mathcal{B}$. The two sets I_s and I_e are disjoint from $A \cup B$. The graph induced by $A_{start} \cup B_{end} \cup \{s_A, e_B\}$ and the graph induced by $A_{end} \cup B_{start} \cup \{s_B, e_A\}$ are complete bipartite graphs. The main goal of this section is to explain how to connect the set $A_{start} \cup B_{start}$ and the set $A_{end} \cup B_{end}$ to $A \cup B$ in order to ensure that any reconfiguration sequence transforming one into

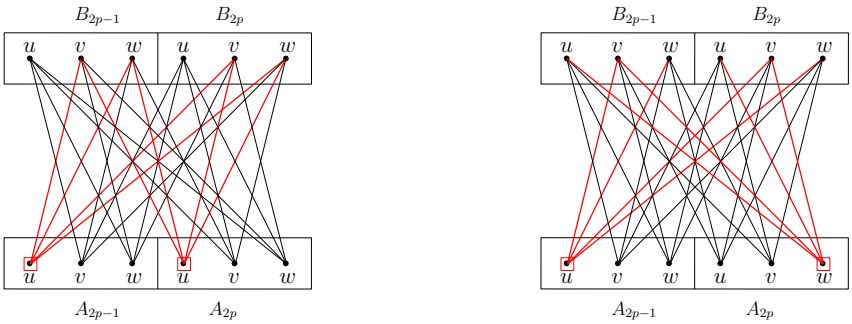


FIGURE 8.2: Connections between the four copies of V_p in $A \cup B$. Vertices with the same name are equivalent vertices. The red square represent tokens: two tokens are positioned on equivalent vertices at the left, and on non-equivalent vertices at the right.

the other enforces the $2k$ tokens starting on A_{start} and the $2k$ tokens starting on B_{start} to switch sides by going through $A \cup B$. More particularly, we will show the existence of an independent set that satisfies the condition of Observation 16 in any such reconfiguration sequence, giving a multicolored independent set of G. For $p \in 1, 2, \ldots, 2k$, we denote by $a_{s,p}$ and $b_{s,p}$ the vertices of A_{start} and B_{start} respectively and we denote by $a_{e,p}$ and $b_{e,p}$ the vertices of A_{end} and B_{end} respectively. These vertices are connected to $A \cup B$ as follows:

1. the vertices $a_{s,p}$ and $a_{e,p}$ are complete to $B - \cup_{i=1}^{p-1} B_i$, and
2. the vertices $b_{s,p}$ and $b_{e,p}$ are complete to $A - \cup_{i=1}^{p-1} A_i$.

An illustration of the full construction is given in Figure 8.3. By construction, no token starting on $A_{start} \cup \{s_A\}$ can move to $B_{end} \cup \{e_B\}$ as long as there are at least two tokens on $A_{start} \cup \{s_A\}$ (and the same goes for $B_{start} \cup \{s_B\}$ and $A_{end} \cup \{e_A\}$). Since there are initially $2k+1$ tokens on $A_{start} \cup \{s_A\}$ and since $N(s_A) \cap B = \emptyset$, the $2k$ tokens initially on A_{start} must move to B at some point in the sequence, and the same goes for B_{start} and A. The tokens initially on s_A and s_B have a special role and act as "locks": without these token, the last token remaining on A_{start} (resp. B_{start}) would be able to move directly to B_{end} without never going through B (resp. A). Let us now explain the connections to $A \cup B$.

Observation 19. *Let I' be an independent set of G' such that $\{a_{s,p}, a_{s,p+1}, \ldots, a_{s,2k}\} \subseteq I'$ for some $p < 2k$. Then the tokens on $\{a_{s,p+1}, a_{s,p+2}, \ldots, a_{s,2k}\}$ are frozen. Furthermore the token on $a_{s,p}$ cannot move to $\cup_{i=p+1}^{2k} B_p$.*

Proof. Let $q > p$ and suppose there is a token on $a_{s,q}$. This token cannot move to B_{end} nor e_B since there is a token on $a_{s,p}$ with $p < q$ and $G'[A_{start} \cup B_{end} \cup \{s_A, e_B\}]$ induces a complete bipartite graph. By construction $N(a_{s,q}) \subseteq N(a_{s,p})$ hence the token on $a_{s,q}$ cannot move to B and this token is frozen. The second statement follows from the fact that $\cup_{i=p+1}^{2k} B_p \subseteq N(a_{s,p}) \cap N(a_{s,p+1})$. □

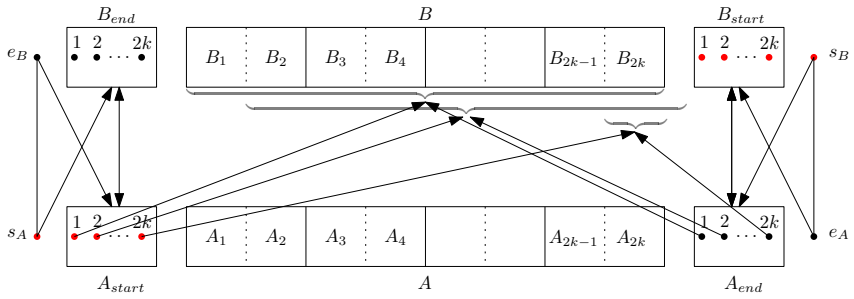


FIGURE 8.3: The constructed graph G'. Vertices in red are the vertices of I_s. An arrow between a vertex v and a subset of vertices indicates that v is complete to this subset. An arrow between a vertex v and a brace indicates that v is complete to the subsets included in the brace. A double arrow between two sets indicate these sets induce a complete bipartite graph. The connections between A and $B_{end} \cup B_{start}$ are symmetric and have been omitted for the sake of clarity.

By symmetry, the same observation can be made for tokens on B_{start}. This shows that the tokens initially on A_{start} and B_{start} must respect a strict order to move respectively to B and A: the only tokens that can initially move are the tokens on $a_{s,1}$ and $b_{s,1}$ and these have no choice but to move to B_1 and A_1 respectively. After such a move the tokens on $a_{s,2}$ and $b_{s,2}$ are free to move to B_2 and A_2 respectively, and so on. Suppose that after the first 4 moves, there is exactly one token in each of the four subset A_1, B_1, A_2 and B_2. Then it is not hard to see - but will be formally proved in the next section - that these tokens lie on equivalent vertices, corresponding to a unique vertex of G. By Observation 17 these tokens cannot move to the other side of the bipartite graph and must stay at the same position while the remaining tokens on A_{start} and B_{start} moves to $A \cup B$. With the full constructions of G', I_s and I_e at hand we can prove the direct part of the reduction:

Lemma 55. *If there is a multicolored independent set of size k in G then there exists a reconfiguration sequence transforming I_s to I_e in G'.*

Proof. Let $u_1 \in V_1, \ldots, u_k \in V_k$ be a multicolored independent set of G. For p in $1 \ldots, k$, let u'_{2p-1}, u'_{2p} (resp. v'_{2p-1}, v'_{2p}) be the copies of u_p in A (resp. B). Consider the following sequence:

1. For $p \in \{1, \ldots, k\}$ in increasing order, move the token on $a_{s,2p-1}$ to u'_{2p-1}, then move the token on $a_{s,2p}$ to u'_{2p}. Move the token on $b_{s,2p-1}$ to v'_{2p-1}, then move the token on $b_{s,2p}$ to v'_{2p}.

2. Move the token on s_A to e_B then move the token on s_B to e_A.

3. For $p \in k, \ldots, 1$ in decreasing order, move the token on u'_{2p} to $a_{e,2p}$, then move the token on u'_{2p-1} to $a_{e,2p-1}$. Move the token on v'_{2p} to $b_{e,2p}$, then move the token on v'_{2p-1} to $b_{e,2p-1}$.

□

The remainder of the section is dedicated to the converse part of the reduction. More particularly, we formally show that there is an independent set satisfying the condition of Observation 16 in any shortest reconfiguration sequence transforming I_s to I_e.

8.2.2.2 Well-organized configurations

To simplify the tracking of tokens along the transformation, we give different colors to the tokens initially on A_{start} and B_{start}. The tokens initially on A_{start} are the *blue tokens* and the tokens initially on B_{start} are the *red tokens*. We say a vertex v is *dominated* by a vertex u in G if $v \in N_G(u)$. Similarly, we say a set U is dominated by W if $U \subseteq N_G(W)$. Given a configuration C, $M_A(C)$ (resp. $M_B(C)$) is the maximum integer $p \in [\![1, 2k]\!]$ such that there is a token on A_p (resp. B_p). By convention, if there is no token on $X \in \{A, B\}$, we set $M_X(C) = 0$. A configuration C is *well-organized* if there is a token on either s_A or e_B and on either s_B or e_A and if it satisfies the following conditions:

1. For every $p \leq M_A(C)$ and every $q \leq M_B(C)$ there is exactly one token on A_p and exactly one token on B_q.
2. If $M_A(C) < 2k$ then for every $M_A(C) < p \leq 2k$ there is a token on $a_{s,p}$. If $M_B(C) < 2k$ then for every $M_B(C) < q \leq 2k$ there is a token on $b_{s,q}$.

Since both the construction and the definition of well-organized configurations are symmetric, we can always assume that $M_A(C) \leq M_B(C)$ for any well-organized configuration C. Note that the initial configuration is well-organized. We say that two configurations C and C' are *adjacent* if C can be transformed into C' by moving exactly one token.
Throughout the proof let $S := C_1, \ldots, C_N$ denote a shortest reconfiguration sequence from I_s to I_e. We say that a token *moves from a set X to a set Y at time t* and we write $(t : X \to Y)$ if there exists two set $X, Y \subseteq V(G')$ and two vertices $x \in X$, $y \in Y$ such that $C_{t+1} = C_t - \{x\} + \{y\}$. When the sets X and Y contain exactly one vertex we write $(t : x \to y)$ by abuse of notation. A move that transforms a well-organized configuration into a configuration that is not well-organized is a *bad move*. We aim to show the following:

Lemma 56. *A shortest reconfiguration sequence from I_s to I_e contains no bad move.*

With Lemma 56 at hand, the proof of the converse part of the reduction easily follows:

Lemma 57. *If there exists a reconfiguration sequence from I_s to I_e in G', then there exists a multicolored independent set in G.*

Proof. Consider a shortest reconfiguration sequence S from I_s to I_e, which exists by supposition. By Lemma 56 this sequence contains no bad moves, therefore all the

configurations of S are well-organized since the initial configuration is. Consider the configuration C just before the first token reaches $A_{end} \cup B_{end}$ (which exists since $A_{end} \cup B_{end} \subseteq I_e$). By definition of well-organized configurations there can be no token on $A_{start} \cup B_{start}$ in C and thus we have $M_A(C) = M_B(C) = 2k$. Then by Observation 16 there exists a multicolored independent set in G. □

The remainder of this section is dedicated to the proof of Lemma 56. Let us begin with a few observations about well-organized configurations, which will be useful throughout all the subsections:

Observation 20. *Let C be a well-organized configuration. For every $p \leq M_A(C)$ we have $|A_p \cap C| = |B_p \cap C| = 1$, and the unique vertex of $A_p \cap C$ and the unique vertex of $B_p \cap C$ are equivalent.*

Proof. By definition of well-organized configuration, there is exactly one token on A_p and one token on B_p for $p \leq M_A(C)$. Let u be the unique vertex of $A_p \cap C$: by construction the only vertex v of B_p that is not in $N(u)$ is the copy of u in B_p. □

Observation 21. *Let C be a well-organized configuration and $p \leq 2k$ be an odd integer such that $|A_p \cap C| = |A_{p+1} \cap C| = 1$. Then $|B_p \cap C| = |B_{p+1} \cap C| = 1$ and the four vertices in these sets are equivalent.*

Proof. Since C is well-organized, $M_B(C) \geq M_A(C)$ and there is one token on B_p and one token on B_{p+1}. Let u (resp. u') be the unique vertex of $A_p \cap C$ (resp. $B_p \cap C$) and v (resp. v') be the unique vertex of $A_{p+1} \cap C$ (resp. $B_{p+1} \cap C$). By Observation 20 we have $u \sim u'$ and $v \sim v'$. By construction the only vertex of B_{p+1} that is not in $N(u)$ is a copy of u since p is odd (B_p and B_{p+1} are copies of the same subset of $V(G)$). We obtain that $u \sim v'$, and the proof follows by the transitivity of the $\sim$ relation. □

Observation 22. *Let C be a well-organized configuration. For every $p < M_A(C)$ and every $q < M_B(C)$, the token on A_p and the token on B_q are frozen.*

Proof. Let $p < M_A(C)$ and let $\{v_p^A\} := A_p \cap C$. Since $p < M_A(C)$, there is a token on another vertex $v_{p'}^A \in A'_p$ such that $v_{p'}^A \sim v_p^A$ by Observation 21. Since these two vertices share the same neighborhood in B, the token on v_p^A cannot move to B. Furthermore, there is a token on A_q for any $q \leq p$ thus this token cannot move to $b_{s,q}$ nor $b_{e,q}$ and since $p < M_A(C)$, there is a token on A_{p+1} and the token cannot go to $b_{s,p}$ nor $b_{e,p}$. It follows that it cannot move to B_{start} nor B_{end} and that the token on v_p^A is frozen. By symmetry, the same goes for the token on B_p for $p \leq M_A(C)$. We then have to be careful about the tokens on B_q for $M_A(C) < q < M_B(C)$. Let $\{v_q^B\} := B_q \cap C$. Since $A_q \cap C = \emptyset$ for any such q, we cannot guarantee that $v_q^B \sim v_{q+1}^B$ even when A_q and A_{q+1} are copies of the same set. However, for any $p \leq M_A(C)$ the set $A_p - v_p^A$ is dominated by v_p^B and $v_p^A \notin N(v_q^B)$ for any q since C is an independent set, hence the token on B_q cannot move to A_p. Furthermore, since $b_{s,M_A(C)+1} \in C$, no token can move from B to A_p for any $p > M_A(C)$. It follows that the tokens on B_q for $M_A(C) < q < M_B(C)$ are also frozen. □

Observation 23. *Let C be any well-organized configuration reachable from C_1. Each token moves at most one time in a shortest reconfiguration sequence from C_1 to C.*

Proof. We can reach C by moving the tokens in the following order: for $p \in 1, \ldots, M_B(C)$ the token on $a_{s,p}$ moves to $B_p \cap C$ and for $p \in 1, \ldots, M_A(C)$ the token on $b_{s,p}$ moves to $A_p \cap C$. Then, if $C \cap \{s_A, e_B\} = \{e_B\}$ (resp. $C \cap \{s_B, e_A\} = \{e_A\}$), move the token from s_A to e_B (resp. from s_B to e_A). This is a shortest sequence since it contains exactly $|C \backslash C_1|$ moves, and every token moves at most one time. □

The strategy to prove Lemma 56 is as follows: we show that if there is a bad move at time t, then there exists a time $t' > t$ at which this bad move is canceled in the sense that the configuration obtained a time $t' + 1$ is, again, well-organized. Such a reconfiguration sequence contains at least $M_A(C_{t'+1}) + M_B(C_{t'+1}) + 1$ moves since at least one token moved twice, and then Observation 23 ensures that it is not a shortest sequence, contradicting our choice of S. The remainder of the proof is organized as follows. In Section 8.2.2.3 we identify, up to symmetry, three different types of bad moves and give some observations about the structure of configurations obtained after such moves. In Section 8.2.2.4 we then show how to cancel (in the sense mentioned above) bad moves of type 1, and we deal with types 2 and 3 in Section 8.2.2.5.

8.2.2.3 Bad moves

Observation 24. *Let $t \in 1, 2, \ldots, N$ be such that the configuration C_t is well-organized and C_{t+1} is not. Then one of the following holds:*

1. *$(t : A \to B)$ or $(t : B \to A)$, or*
2. *$(t : A \to B_{end})$ and $B_{start} \cap C_t \neq \emptyset$ or $(t : B \to A_{end})$ and $A_{start} \cap C_t \neq \emptyset$, or*
3. *$(t : s_A \to B_{start})$ or $(t : s_B \to A_{start})$.*

Proof. First, there can be no move from A_{start} to B_{end} at time t. Indeed if there is a token on A_{start} then there must be a token on s_A since C_t is well-organized and both of these tokens dominate all of B_{end}. By symmetry, the same goes for A_{end} and B_{start}. By construction, the only token that can move from A_{start} is the token on $a_{s,M_B(C)+1}$ which can only go to $B_{M_{B(C)+1}}$ and such a move leads to a well-organized configuration and cannot be a bad move. Conversely, the only token that can move from B is the token on $B_{M_B(C)}$ by Observation 22 and the only vertex it can reach on A_{start} is $a_{M_B(C)}$, which also leads to a well-organized configuration. By symmetry, the same goes for the moves between B_{start} and A. It follows that the only possible bad moves are the moves of condition 1, 2 and 3. □

We consider the smallest integer t such that the move between C_t and C_{t+1} is a bad move. Since C_1 is well-organized, C_t is well-organized by definition of a bad move. For brevity we set $i := M_A(C_t)$ and $j := M_B(C_t)$. Note that $i \leq j$ so there can be no move from B to A unless $i = j$, in which case the move must be from B_i

to A_i by Observation 22. By symmetry we can thus always suppose that if the first bad move is a move between A and B, then it is a move from A to B. Furthermore, we can suppose that $i < 2k$ for otherwise the configuration C_t yields a multicolored independent set of size k as shown in Section 8.2.2.2 and we are done. Using these symmetries and Observation 24 we can restrict ourselves to three cases: either the bad move is a move from A to B, or it is a move from A to B_{end}, or it is a move from s_A to B_{end}. We denote these moves as bad moves of *type 1*, *type 2*, and *type 3* respectively, and we denote the blue token making the bad move at time t as the *bad token*. Note that Observation 22 ensures that if the bad move at time t is of type 1 or 2, then the bad token is on A_i in C_t. Since $i < 2k$, C_t is well-organized, and the move at time t is the first bad move of the sequence we have:

Observation 25. *There is a red token on s_B in C_t.*

The following observations give some more information about the configurations C_t and C_{t+1} that we obtain after the first bad move, depending on its type.

Observation 26. *If the move at time t is a bad move of type 1, then $i := M_A(C_t)$ is odd. Furthermore, $(t : A_i \to B_q)$ with $q \geq i$.*

Proof. If i is even, $i \geq 2$ and A_{i-1} is a copy of A_i. Since $i \leq j$, Observation 21 ensures that there is a token on A_{i-1}, A_i and B_i on equivalent vertices, in which case the tokens on A_{i-1} and A_i cannot move to B, proving the first statement. The second statement is a direct consequence of Observation 22. □

Observation 27. *If the move at time t is a bad move of type 2 or 3, then $j := M_B(C_t) = 2k$.*

Proof. If $j < 2k$ then by definition of a well-organized configuration there are some blue tokens on A_{start} and no token can move to B_{end}. □

Finally, the two following Observations follow from the fact that C_t is well-organized:

Observation 28. *If the move at time t is a bad move of type 2, then $(t : A_i \to b_{e,i})$.*

Observation 29. *If the move at time t is a bad move of type 3, then $(t : s_A \to b_{e,p})$ with $p > i$.*

8.2.2.4 Bad moves of type 1

In this subsection, we suppose that the move at time t is a bad move of type 1. By Observation 20 we have $|A_p \cap C_t| = 1$ for every $p \leq M_A(C_t)$ and $|B_p \cap C_t| = 1$ for every $p \leq M_B(C_t)$. In this section v_p^A (resp. v_p^B) denote the only vertex of $|A_p \cap C_t|$ for $p \leq M_A(C_t)$ (resp. $|B_p \cap C_t|$ for $p \leq M_B(C_t)$. By Observation 26 we have $(t : A_i \to B_q)$ for some $q \geq i$. In the next lemma, we show that as long as no token moves from B_q after time $t+1$, the blue tokens on B and the red tokens on B_{start} at time $t+1$ remain frozen.

Lemma 58. *Let $t' \geq t+1$ such that no token has moved from B_q between C_{t+1} and $C_{t'}$. Then for any configuration between C_{t+1} and $C_{t'}$ we have:*

1. *for every $p \leq q$ there is a blue token on v_p^B.*
2. *for every $p > i$ there is a red token on $b_{s,p}$.*

Proof. First, note that C_{t+1} satisfies conditions 1 and 2 since C_t is well-organized and the move between C_t and C_{t+1} is a bad move. Suppose for a contradiction that there exists a time $t+1 < \tau < t'$ such that for every $t+1 \leq \ell \leq \tau$ the configuration C_ℓ satisfies conditions 1 and 2 and that the configuration $C_{\tau+1}$ does not. Then it must be that at time τ, either a red token moves from $b_{s,p}$ for some $p > i$ or a blue token moves from v_p^B for some $p < q$. Let us show that none of these moves is actually possible since C_τ satisfies conditions 1 and 2.
Suppose first that a red token moves from $b_{s,p}$ for $p > i$. Since C_τ satisfies condition 2 the tokens on $b_{s,x}$ for $x > i+1$ are frozen and we have $p = i+1$. By construction the red token on $b_{s,i+1}$ can only move to A_{i+1}. Furthermore, i is odd by Observation 26 and A_{i+1} is a copy of A_i. By the choice of τ there is a blue token on v_i^B and a red token on $N(v_{i+1}^A)$ since $(\tau : v_{i+1}^A \to B_q)$. It follows that A_{i+1} is fully dominated at time τ and that the red token on $b_{s,i+1}$ cannot move, a contradiction.
Suppose then that a blue token moves from v_p^B for some $p < q$. Since there are tokens on B_q and $p < q$, this token cannot move to A_{start}, and since condition 2 is satisfied by C_τ, it cannot move to A_{end}. Furthermore, since C_τ also satisfies condition 1, no blue token can move to A_x for $x \geq i+1$. We then have two sub-cases to consider :

1. $p \leq i$: let B'_p be the other copy of B_p in G. By the choice of τ we have $B_p \cap C_\tau = \{v_p^B\}$, $B_{p'} \cap C_\tau = \{v_{p'}^B\}$ and by Observation 21 we have $v_p^B \sim v_{p'}^B$ hence the token on v_p^B cannot move to A.
2. $i+1 \leq p < q$: By the choice of τ we have $B_x \cap C_\tau = \{v_x^B\}$ with $v_x^B \sim v_x^A$ for every $x \leq i$ by Observation 21. For such x, v_x^A is the only vertex that is not dominated by the blue token on A_x, and since C_t is an independent set we have $v_x^A \notin N(v_p^B)$. It follows that the blue token on v_p^B cannot move to A_x for $x \leq i$. Since C_τ satisfies condition 2 it cannot move to B_x for $x \leq i+1$, which concludes the proof.

□

So as long as there are two tokens on B_q some tokens remain frozen and cannot reach the targeted independent set. Hence one of the two tokens on B_q has to move again at some point in the reconfiguration sequence. The following Observation shows that one of the tokens on B_q necessarily moves back to A_i.

Observation 30. *There exists $t' \geq t+1$ such that $(t' : B_q \to A_i)$.*

Proof. To reach the target configuration, every token on B_{start} must move at least one time. By Lemma 58.1, the tokens on $b_{s,p}$ for $p > i$ cannot move as long as there are

two tokens on B_q. It follows that one of these token has to move at a time $t' \geq t+1$. Let $u \in B_q$ be the vertex such that $(t : v_i^A \to u)$: note that $u \notin N(v_p^A)$ for any $p < i$. Then by Lemma 58.1 there can be no move from B_q to A_p for $p < i$ and by 58.2 there can be no move from B_q to A_p for $p > i+1$ at time τ. Furthermore, Lemma 58.1 also ensures that there can be no move from B_q to $a_{s,p}$ for $p < q$, and since there are two tokens on B_q at time τ, none of them can move to $b_{s,q}$. Thus, $(\tau : B_q \to A_i)$ is the only possible move at time τ. □

In other words, the bad move at time t is in some sense "canceled" at time t'. Note that, however, it is not necessarily the red token that moves at time t': in the particular case where $q = j = i$, the blue token on B_q can move to A_i, switching role with the blue token. The next lemma shows that in-between t and t' every token has a very restricted pool of possible moves and remains locked in the closed neighborhood of the token it lies on in C_t.

Lemma 59. *Let $t' \geq t+1$ be the first time after t such that $(t' : B_q \to A_i)$. Then any configuration C_ℓ with $t+1 \leq \ell \leq t'$ satisfies the following conditions:*

1. *For every even $p < i$, there is either a red token on $b_{s,p}$ or a red token on v_p^A or a red token on $b_{e,p}$.*
2. *For every odd $p < i$, there is either a red token on $b_{s,p}$, or a red token on v_p^A, or a red token on $N(v_p^A) \cap B$, or red token on $b_{e,p}$.*
3. *For every $p > q$ there is either a blue token on $a_{s,p}$ or a blue token on B_p.*

Proof. Let us first show that C_{t+1} satisfies conditions 1 to 3. Since C_t is well-organized and since $(t : A_i \to B_q)$ there is a red token on v_p^A for every $p < i$ thus conditions 1 and 2 are satisfied. Furthermore, there is a blue token on B_p for every $q < p \leq M_B(C)$ and a blue token on $a_{s,p}$ for every $M_B(C) < p \leq 2k$ and condition 3 is satisfied by C_{t+1}. Let us now prove that these conditions are satisfied by any configuration between times t and t'. Suppose otherwise and let τ be the first time after $t+1$ such that C_τ does not satisfy one of the three conditions. Note that by Lemma 58.1 , we know that for any $t+1 \leq \ell \leq t'$ there is a blue token on v_p^B for every $p < q$ in C_ℓ and a red token on $b_{s,p}$ for every $p > i$.

1. C_τ **does not satisfy condition** 1. Since $C_{\tau-1}$ satisfies the three conditions, there exist exactly one even integer $p_0 < i$ for which condition 1 is not satisfied in C_τ.

   (a) Suppose first there is a token on b_{s,p_0} in $C_{\tau-1}$. Since there is a token on $b_{s,p}$ for every $p > i$, this token cannot move to A_{end} nor B_p for any such p, and there is no token on A_p for any $p > p_0$ in $C_{\tau-1}$. Then, since conditions 1 and 2 are satisfied by $C_{\tau-1}$, there must be a red token on $\{b_{s,p}, b_{e,p}, v_p^A\} \cup N(v_p^A) \cap B$ for every $p_0 < p \leq i$. The token on b_{s,p_0} then has to move to A_{p_0} a time $\tau - 1$, and since there is a blue token on $v_{p_0}^B$,

the only vertex it can move to is $v^A_{p_0}$. But then C_τ satisfies condition 1, a contradiction.

(b) Suppose then that there is a red token on b_{e,p_0} in $C_{\tau-1}$: since $N(b_{e,p_0}) \cap A = N(b_{s,p_0}) \cap A$, one can easily see that the only vertex this token can move to is also $v^A_{p_0}$, again leading to a contradiction.

(c) Finally suppose that there is a red token on $v^A_{p_0}$ in $C_{\tau-1}$. Then there can be no token on b_{s,p_0-1} nor on b_{e,p_0-1}. Furthermore - recall that since p_0 is even A_{p_0} and A_{p_0-1} are copies of the same set - there can be no token in $N(v^A_{p_0-1}) \cap B$ in $C_{\tau-1}$. Since condition 2 is satisfied for $p_0 - 1$, there must then be a token on $v^A_{p_0-1}$. It follows that the token on A_{p_0} can only move to b_{s,p_0} or b_{e,p_0} and condition 1 is satisfied by C_τ, a contradiction.

2. C_τ **does not satisfy condition** 2. As in case 1, there exists exactly one odd integer $p_0 < i$ for which condition 2 is not satisfied in C_τ. If there is a token on b_{s,p_0} or on b_{e,p_0} in $C_{\tau-1}$ we obtain a contradiction using the same arguments (which do not make use of the parity of p_0) than in case 1.*a* and 1.*b* respectively. Two cases remain to be considered:

(a) Suppose that there is a token on $v^A_{p_0}$ in $C_{\tau-1}$. Then there can be no token on b_{s,p_0-1} nor on b_{e,p_0-1} and since $C_{\tau-1}$ satisfies condition 1 ($p_0 - 1$ is even), there must be a token on $v^A_{p_0-1}$ in $C_{\tau-1}$. It follows that the token on A_{p_0} can either move to b_{s,p_0}, b_{e,p_0} or to B, and C_τ satisfies condition 2.

(b) Finally suppose there is a red token on $N(v^A_{p_0}) \cap B$ in $C_{\tau-1}$. Let p_1 be such that this red token is on B_{p_1}. Then by construction there can be no token on a_{s,p_1} in $C_{\tau-1}$ and since condition 3 is satisfied by $C_{\tau-1}$ there is also a blue token on B_{p_1} in $C_{\tau-1}$. It follows that there are two tokens on B_{p_1} in $C_{\tau-1}$ and that these tokens cannot move to A_{start}. Furthermore, since $i < 2k$ we have $B_{start} \cap C_\tau \neq \emptyset$, so the red token on B_{p_1} cannot move to A_{end} and must move to B at time $\tau - 1$. Let us show it can only move back to $v^A_{p_0}$. By Lemma 58.1 this token can only move to v^A_p for some $p < i$. But since $C_{\tau-1}$ satisfies condition 1 and 2, we have that for any $p \neq p_0$, there is a red token on $\{b_{s,p}, b_{e,p}, v^A_p\} \cup N(v^A_p) \cap B$. It follows that the only vertex of A this token can move to is $v^A_{p_0}$ and condition 2 is satisfied by C_τ

3. C_τ **does not satisfy condition** 3. As in the previous cases there exists exactly one integer $p_0 > q$ for which condition 3 is not satisfied in C_τ.

(a) Suppose first there is a blue token on a_{s,p_0} in $C_{\tau-1}$. If $p_0 = 2k$ this token can only move to B_{2k} and we are done. Otherwise, there can be no token on B_{p_0+1} in $C_{\tau-1}$ and since this configuration satisfies condition 3, there must then be a token on a_{s,p_0+1}. It follows that the blue token on a_{s,p_0} can only move to B_{p_0} and we obtain a contradiction.

(b) Suppose then that there is a blue token on B_{p_0} in $C_{\tau-1}$. Since there are still tokens on B_{start} this token cannot go to A_{end} nor to any A_p for $p > i$.

Furthermore by Lemma 58.1, the only vertex on A_p that is not dominated by tokens on B is v_p^A for any $p \leq i$. But since $C_{\tau-1}$ satisfies condition 1 and 2, there is a red token on $\{b_{s,p}, b_{e,p}\} \cup B$ that dominates this vertex. It follows that the blue token on B_{p_0} can only move to A_{start}. Furthermore since there is a token on B_{p_0} there can be no token on $a_{s,p}$ for $p \leq p_0$ and since $C_{\tau-1}$ satisfies condition 3 there must be a blue token on B_p for any $p \leq p_0$. It follows that the blue token on B_{p_0} in $C_{\tau-1}$ can only move to a_{s,p_0} and that condition 3 is satisfied by C_τ, which concludes the proof.

□

Furthermore, up to removing a move from the sequence we have the following:

Observation 31. *Let $t' \geq t+1$ such that no token has moved from B_q between C_{t+1} and $C_{t'}$. Then for any configuration between C_{t+1} and $C_{t'}$ there is a token on $\{s_A, e_B\}$.*

Proof. Since the move at time t is the first bad move, there is a token on $\{s_A, e_B\}$ at time t. If there is a token on e_B, there can be no token on A_{start} and by Lemma 58.3 there must be blue token on B_p for every $p \leq 2k$ so there can be no move from e_B to A_{start}. Suppose there exists $\tau > t$ such that $(\tau : s_A \to B_{e,p})$ for some p. By Lemma 59 and 58, this token cannot move to A before time $t'+1$. But then we can replace the move at time τ by $(\tau : s_A \to e_B)$: since $N(e_B) \subseteq N(b_{e,p})$ all the moves between time τ and $t'+1$ remain valid. □

Let us now consider the configurations $C_{t'}$ and $C_{t'+1}$. We know that $(t' : B_q \to A_i)$ and in particular there can be no token on $b_{s,p}$ nor $b_{e,p}$ for any $p \leq i$ in $C_{t'+1}$. Since configuration $C_{t'+1}$ satisfies condition 1 of Lemma 59 we have that for any even $p < i$ there is a red token on v_p^A, and since $v_p^A \sim v_{p-1}^A$ there cannot be any red token on $N(v_{p-1}^A) \cap B$. Then by condition 2 of Lemma 59 there is necessarily a token on v_{p-1}^A. Furthermore, by Lemma 58.2 there is a red token on $b_{s,p}$ for every $p > i$, and by Lemma 58.1 there is a blue token on B_p for every $p < q$ in $C_{t'+1}$. Condition 3 of Lemma 59 ensures that there is a blue token on $\{b_{s,p}\} \cup B_p$ for every $p > q$ in C_{t+1}. Furthermore, Observation 31 and Observation 25 ensure that there is token on $\{s_A, e_B\}$ and a token on $\{s_B, e_A\}$. Finally, there are two tokens on B_q at time t' and one of these moves to A_i, which ensures that $C_{t'+1}$ is well-organized.

In the considered shortest sequence S, the token that moves from B to A at time t' moves at least three times before we reach the well-organized configuration $C_{t'+1}$, a contradiction with the choice of S by Lemma 23.

8.2.2.5 Bad moves of type 2 and 3

The proof for bad moves of type 2 and 3 follows similar reasoning as for type 1. We first show that as long as the bad token does not move after time $t+1$, a large part of the other tokens remain frozen. We then show that the bad token has to move again after time $t+1$ and that we subsequently either obtain a well-organized configuration

or cancel a bad move. By Observation 28 and 29 we have either $(t : A_i \to b_{e,q})$ or $(t : s_A \to b_{e,q})$ for some $q \geq i$. Note that in the particular case of a bad move a type 2 we have $q = i$. By Observation 27, there is exactly one blue token on B_p for every $1 \leq p \leq 2k$ in C_t. We denote by v_p^B the only vertex of $I_t \cap B_p$ and by v_p^A the copy of v_p^B in A_p.
Let $p \leq 2k$ be odd. Recall that by construction, if $v_p^B \sim v_{p+1}^B$ then the blue tokens on $\{v_p^B, v_{p+1}^B\}$ cannot move to A, and no other token can move to $(A_p \cup A_{p+1}) - \{v_p^A, v_{p+1}^A\}$. If $v_p^B \nsim v_{p+1}^B$, no token can move $A_p \cup A_{p+1}$ except for the tokens on $\{v_p^B, v_{p+1}^B\}$. Let us first show that there necessarily exists a time $t' > t$ at which the bad token moves again:

Observation 32. *Let $t_1 > t$ be such that for any $t \leq \tau \leq \ell$, $b_{e,q} \in C_\tau$. Then for any $t \leq \tau \leq \ell$, $s_B \in C_\tau$.*

Proof. As long as there is a token on $b_{e,q}$, no token on $b_{s,p}$ for $p \geq q$ can move to A. Since by Observation 25, $s_B \in C_t$ and since $b_{s,p} \in C_t$ for $p > q$, these token are frozen as long as there is a token on $b_{e,q}$. □

Observation 33. *If the move at time t is a bad move of type 2, then there is a blue token on e_B in C_t and this token cannot move before the bad token moves again.*

In order to show that the configuration $C_{t'+1}$ is well-organized, we need a lemma similar to Lemma 59:

Lemma 60. *Let $t' \geq t+1$ be the smallest integer such that the move between $C_{t'}$ and $C_{t'+1}$ is a move of the bad token. Then any configuration C_ℓ with $t+1 \leq \ell \leq t'$ satisfies the following conditions:*

1. *If $p < q$ is even, there is either a red token on $b_{s,p}$ or a red token on v_p^A, or a red token on $b_{e,p}$.*
2. *If $p < q$ is odd, there is either a red token on $b_{s,p}$, or a red token on v_p^A, or a red token in $N(v_p^A) \cap B$, or red token on $b_{e,p}$.*
3. *For every $p \leq 2k$ there is a blue token on v_p^B.*
4. *For every $p \geq q$ there is a red token on $b_{s,p}$ and there is a red token on s_B.*

Proof. As the configuration C_t is well-organized and $(t : A_i \to B_{start})$, conditions 1 to 4 are satisfied by configuration C_{t+1}. As for the proof of Lemma 59, we suppose for a contradiction that there exist a time $t+1 < \tau < t'$ such that C_τ satisfies conditions 1 to 4 and $C_{\tau+1}$ does not. We consider the smallest such time τ.

1. **$C_{\tau+1}$ does not satisfy condition 1.** Since C_τ satisfies the four conditions, there exist exactly one even integer $p_0 < q$ for which condition 1 is not satisfied in $C_{\tau+1}$.

(a) Suppose there is a red token on b_{s,p_0} in C_τ. Since C_τ satisfies condition 4, this token cannot move to B_p for any $p > q$ nor it can move to A_{end}. Since there is a token on $b_{e,q}$, it cannot move to A_q either. So we must have $(\tau : b_{s,p_0} \to A_x)$ for some $p_0 \leq x < q$. Suppose w.l.o.g that x is even: by condition 3, there is a token on both v_x^B and v_{x-1}^B with $v_x^B \sim v_{x-1}^B$, thus we have $(\tau : b_{s,p_0} \to v_x^A)$. But then if $x \neq p_0$, condition 1 and 2 ensure that there is either a token on v_x^A, $b_{e,x}$, $b_{s,x}$ or on $N(v_x^A) \cap B$, a contradiction.

(b) Suppose there is a red token on $v_{p_0}^A$ in C_τ. Since C_τ satisfies condition 2, there is also a token on $v_{p_0-1}^A$ and by condition 3 we have $v_{p_0-1}^A \sim v_{p_0-1}^A$. So the token on $v_{p_0}^A$ can either move to b_{e,p_0} or b_{s,p_0} since any other of its neighbors in B_{end} or B_{start} dominates A_{p_0-1}, and condition 1 remains satisfied.

(c) Suppose there is a red token on b_{e,p_0} in C_τ. Since there is a token on $b_{e,q}$, the token on b_{e,p_0} cannot move to A_{end}, and it must move to A. Since the vertices b_{s,p_0} and b_{e,p_0} share the same neighborhood in A, we can apply the same arguments as for case 1.a showing that $(\tau : b_{e,p_0} \to v_{p_0}^A)$, and condition 1 remains satisfied.

2. $C_{\tau+1}$ **does not satisfy condition 2.** Since C_τ satisfies the four conditions, there exist exactly one odd integer $p_0 < q$ for which condition 2 is not satisfied in $C_{\tau+1}$. First note that the proof for case 1 do not make use at any point of the parity of p_0. Hence if there is a token on b_{s,p_0}, $v_{p_0}^A$ or on b_{e,p_0} case 1.a, 1.b and 1.c apply respectively. Only the case where there is a red token on $N(v_{p_0}^A) \cap B$ in C_τ remains to be considered. Let u denote the vertex on which the token is. Since condition 3 is satisfied by C_τ, this token cannot move to A_{start} nor A_{end}, so we have $(\tau : N(u) \to v_x^A)$ for some x. Suppose that A_x and A_{p_0} are not copies of the same set. First note that x must be odd, for otherwise there must be a token on $b_{s,x}$ or $b_{e,x}$ at time τ by condition 1 and it is not possible to move from B to A_x at time τ. Furthermore we must have $x < p$: if not, then by condition 1 there must be a token on b_{s,p_0+1} or b_{e,p_0+1} which dominates A_x since p_0 is odd and since there can be no token on $v_{p_0+1}^A$.
There can be no move from $v_{p_0}^A$ to u at any time between t and τ. Suppose otherwise and consider a time t_0 such that $t < t_0 < \tau$ at which such a move occurs: in C_{t_0} there is a token on $v_{p_0}^A$ so there cannot be any token on $b_{e,x+1}$ nor $b_{s,x+1}$ hence there is a token on $v_{x+1}^A \in N(u)$ by condition 1. It follows that, since condition 2 is satisfied for any time $t \leq \tau$, there must be a token either on b_{s,p_0} or b_{e,p_0} or on a vertex of $N(v_{p_0}^A) \cap B$ which is distinct from u. But then after the token on u moves to v_x^A at time τ condition 2 is still satisfied, a contradiction.

3. $C_{\tau+1}$ **does not satisfy condition 3.** Since C_τ satisfies condition 3, there exists a unique $p_0 \leq 2k$ such that at time τ a blue token moves from $v_{p_0}^B$. Since condition 3 is satisfied and since there is a token on $b_{e,q}$, this token cannot move

to A_{end} nor A_{start}. So we can suppose that $(\tau : v^B_{p_0} \to A_x)$ for some x odd without loss of generality. If $v^B_x \nsim v^B_{x+1}$ then there must be a red token on $b_{s,x+1}$ or $b_{e,x+1}$ by condition 1 since there can be no token on v^A_{x+1}. So it must be that $v^B_x \sim v^B_{x+1}$: but then again by condition 1 and 2 there must be either a red token on $b_{s,x}$, $b_{e,x}$ or on $N(v^A_x) \cap B$ and it follows that no blue token can move to A_x.

4. $C_{\tau+1}$ **does not satisfy condition** 4. As long as there are some tokens on B_{start}, the red token on s_B cannot move, so there exists a unique $p_0 \geq q$ such that at time τ a red token moves from b_{s,p_0}. Since there is a token on s_B this token cannot move to A_{start} and thus can only move to A. By construction it can only move to A_x for some $x \geq q$ and any such set is dominated by the token on $b_{e,q}$, a contradiction.

□

As long as the bad token does not move again after time t, condition 4 of Lemma 60 ensures that the red tokens on $B_{start} \cup \{s_B\}$ remain frozen. So there must exist a time $t' > t$ such that the bad token moves at time t'. The following observation actually show that this token moves back to the position it had in C_t:

Observation 34. *Let $t' > t$ denote the time at which the bad token moves again. We have the following:*

1. *$(t' : b_{e,q} \to v^A_i)$ if the move at time t is a bad move of type* 2.
2. *$(t' : b_{e,q} \to s_A)$ if the move at time t is a bad move of type* 3.

Proof. We prove the two statements separately:

1. **The move at time t is a bad move of type 2.** By Lemma 60.3 there is a blue token on B_p for every $p \leq 2k$ at time t' so the bad token cannot move to A_{start}. Furthermore by Observation 33 there is a blue token on e_B at time t' so it cannot move to s_A either. By Observation 28 we have $q = i$ and thus $(t' : b_{e,q} \to A_x)$ for some $x \geq i$. By Lemma 60.4 there are red tokens on $b_{s,p}$ for every $p \geq i$ so it must be that $x = i$. Finally Lemma 60.3 ensures that there is a token on v^B_i in $C_{t'}$ and since $v^A_i \sim v^B_i$ it follows that $(t' : b_{e,q} \to v^A_i)$ is the only possible move for the bad token at time t'.

2. **The move at time t is a bad move of type 3.** As for the previous case, Lemma 60.3 ensures that the bad token cannot move to A_{start}. By Observation 29 we have $q > i$ and by Lemma 60.4 there is a token on $b_{s,q}$ so the bad token cannot move to A. It follows that $(t' : b_{e,q} \to s_A)$ is the only possible move for the bad token at time t'.

□

The following Observation allows us to conclude about the bad moves of type 2:

Observation 35. *Let $t' \geq t+1$ denote the time at which the bad token moves again. If the move at time t is a bad move of type 2, then the configuration $C_{t'+1}$ is well-organized.*

Proof. By Observation 34.1, we have that $(t' : b_{e,i} \to v_i^A)$. Since $q = i$ by Observation 28, Lemma 60 ensures that there is a red token on $b_{s,p}$ for every $p \geq i$ and a blue token on B_x for every $x \leq 2k$ a time t'. It remains to show that there is a red token on A_y for every $y \leq i$ to obtain a well-organized configuration. Since there is a token on A_i in $C_{t'+1}$ there can be no token on $b_{s,p}$ nor $b_{e,p}$ for any $p \leq i$ and condition 1 of Lemma 60 then ensures that there is a red token on v_p^A for every even $p \leq i$. Since furthermore $v_p^A \sim v_{p+1}^A$ for every odd $p < i$, there can be no token on $N(v_{p+1}^A) \cap B = N(v_p^A) \cap B$ for any such p, and condition 2 of Lemma 60 ensures that there is a red token on v_p^A for every odd $p < i$. □

As for bad moves of type 1, there is a token that moves at least three times to reach the well-organized configuration $C_{t'+1}$, a contradiction with Observation 23.

It remains to check the case of bad moves of type 3. If $(t : s_A \to b_{e,q})$ we proceed as follows: we replace the move at time t by the move $(t : s_A \to e_B)$ and the move at time t' by $(t' : e_B \to s_A)$. Since $N(e_B) \subseteq N(b_{e,q})$, the moves at times $t, t+1, \ldots, t'$ remain valid. Furthermore by Observation 34.2 we obtain the same independent set at time $t'+1$. Although the modified sequence is not shorter, it does not contain any bad move of type 3: either we obtain a well-organized configuration at time $t'+1$ and we are done, or there is a bad move of type 1 or 2 between time $t+1$ and t', in which case one of the previous cases apply. It follows that the sequence contains no bad move of type 3. The proof of Lemma 56 is now straightforward:

Proof. Let S be a shortest reconfiguration sequence from I_s to I_e. In section 8.2.2.4 we showed that S contains no bad move of type 1, and we showed that S contains no bad moves of type 2 or 3 in section 8.2.2.5. Then by Observation 24 it follows that S contains no bad move. □

Acknowledgments. The authors thank the anonymous reviewer of the extended abstract of this paper, accepted in ISAAC 2020 [9], for her/his insightful comments that allowed us to improve Lemmas 35 and 34.

Chapter 9

Galactic Token Sliding

The results presented in this chapter were obtained with Nicolas Bousquet, Clément Dallard and Amer E. Mouawad. These results have not been peer-reviewed yet.

This chapter is devoted to the proof of the following theorem:

Theorem 39. *Let $\Delta \geq 3$ be a fixed constant. Token Sliding is fixed parameter tractable with respect to the parameter k on graphs of maximum degree Δ.*

In order to do so, we introduce a new model, which we call *galactic token sliding*, which we believe is of independent interest. In Chapter 6, we described a general method to obtain fixed parameter tractable algorithm for Token Jumping in sparse graphs. It basically consists in finding a buffer independent set, "far away" from the starting and ending positions of tokens, to which we can jump the tokens one by one (or use the fact that no such set exists to bound the size of the graph by a function of k). In the Token Sliding problem, the buffer space may not be used as easily as for Token Jumping since some tokens may be blocked in there initial positions (or close to it). Hence, what can we do with this extra space that we have on the graph? First, we cannot delete it, since we cannot decide *a priori* if this space will be used in a reconfiguration sequence or not. However we will see that we can merge it, under certain conditions.

Let us give a concrete example: suppose that the input graph contains a vertex v that is neither in S nor T, the starting and ending independent sets, and that is attached to a "pendant" induced path P of length $\geq 2k$. By pendant, we mean that the unique vertex of P that has a neighbor outside of P is the neighbor of v. An illustration is given in Figure 9.1. Suppose furthermore that no vertex of P is in S or T. Obviously, the path P can hold the k tokens. Furthermore we do not really "care" about the position of the tokens on the path: If a token enters P, we can always assume that it goes as far down the path as possible. If a token leaves the path, we can always assume that this token is the one in P that is closest to v. In other words, we can merge this path into a single special vertex, and simply recall that this vertex can hold all k tokens. We call such a vertex a *black-hole*.

By doing so, we of course leave the framework of classical independent set reconfiguration. We formally describe this new problem, which we call *galactic token sliding*, in the next section. The key point of the proof of Theorem 40 is then to show that,

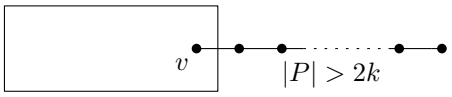


FIGURE 9.1: Illustration of a graph with a pendant path P

as we did for the path P, we can merge any geodesic path of the input graph that is long enough with respect to k and obtain an equivalent instance.

9.1 Galactic graphs

We say that a graph $G = (V, E)$ is a *galactic graph* if $V(G)$ can be partitioned into two sets $A(G)$ and $B(G)$ where the set $A(G) \subseteq V(G)$ is the set of vertices that we call *planets* and the set $B(G) \subseteq V(G)$ is the set of vertices that we call *black holes*. For a given graph G', we write $G' \prec G$ whenever $A(G') < A(G)$ or, in case of equality, $B(G') < B(G)$. In the standard TOKEN SLIDING problem, tokens are restricted to sliding along edges of a graph as long as the resulting sets remain independent. This implies that no vertex can hold more than one token and no two tokens can ever become adjacent. In a galactic graph, the rules of the game are slightly modified. When a token reaches a black hole (a special kind of vertex), the token is *absorbed* by the black hole. This implies that a black hole can hold more than one token, in fact it can hold all k tokens. Moreover, we allow tokens to be adjacent as long as one of the two vertices is a black hole (since black holes are assumed to make tokens "disappear"). On the other hand, a black hole can also "project" any of the tokens it previously absorbed onto any vertex in its neighborhood (be it a planet or a black hole). Of course, all such moves require that we remain an independent set in the galactic sense. We say that a set S is a *galactic independent set* of a galactic graph G whenever $G[S \cap A]$ is independent. To fully specify a galactic independent set S of size k containing more than one token on black holes, we use a weight function $\omega_S : V(G) \to \{0, \ldots, k\}$. Hence, $\omega_S(v) \leq 1$ whenever $v \in A(G)$, $\omega_S(v) \in \{0, \ldots, k\}$ whenever $v \in B(G)$, and $\sum_{v \in V(G)} \omega_S(v) = k$.

We are now ready to define the GALACTIC TOKEN SLIDING problem. We are given a galactic graph G, an integer k, and two galactic independent sets S and T such that $|S| = |T| = k \geq 2$ (when $k = 1$ the problem is trivial). The goal is to determine whether there exists a sequence of token slides that will transform S into T such that each intermediate set remains a galactic independent set. As for the classical TOKEN SLIDING problem, given a galactic graph G we can define a reconfiguration graph which we call the *galactic reconfiguration graph* of G. It is the graph whose vertex set is the set of all galactic independent sets of G, two vertices being adjacent if their corresponding galactic independent sets differ by exactly one token slide. We always assume the input graph G to be a connected graph, since we can deal with each component independently otherwise. Furthermore, components without tokens can be safely deleted. Given an instance (G, k, S, T) of GALACTIC TOKEN SLIDING, we say that (G, k, S, T) can be *reduced* if we can find an instance (G', k', S', T') which

is positive (a yes-instance) if and only if (G, k, S, T) is positive (a yes-instance) and $G' \prec G$.

Let G be a galactic graph. A *planetary component* is a maximal connected component of $G[A]$. A *planetary path* P, or *A-path*, composed only of vertices of A, is called *A-geodesic* if, for every x, y in P, $d_{G[A]}(x, y) = d_P(x, y)$.

We use the term *A-distance* to denote the length of a shortest path between vertices $u, v \in A$ such that all vertices of the path are also in A. Let us state a few rules that allow us to safely reduce an instance (G, k, S, T) of Galactic Token Sliding to an instance (G', k', S', T'):

- Rule 1 (adjacent black holes rule): If two black holes u and v are adjacent, we contract them into a single black hole w. If there are tokens on u or v, the merged black hole receives the union of all such tokens. In other words, $\omega_{S'}(w) = \omega_S(u) + \omega_S(v)$ and $\omega_{T'}(w) = \omega_T(u) + \omega_T(v)$. Loops and multi-edges are ignored.

- Rule 2 (dominated black hole rule): If there exists two black holes u and v such that $N(u) \subseteq N(v)$, $\omega_S(u) = 0$, and $\omega_T(u) = 0$, we delete u.

- Rule 3 (absorption rule): If there exists u, v such that u is a black hole, $v \in (N(u) \cap A) \setminus (S \cup T)$ (v is a neighboring planet that is neither in S nor T), $S \cap A \cap N(v) = \emptyset$, and $T \cap A \cap N(v) = \emptyset$, then we contract the edge uv. We say that v is absorbed by u.

- Rule 4 (path reduction rule): Let G be a galactic graph and P be a A-geodesic path of length $5k$ such that $(A \cap N[P]) \cap (S \cup T) = \emptyset$. Then, P can be contracted into a black hole.

Note that all of the above rules allow us to reduce the size of the input graph. In the remainder of this section, we prove a series of lemmas establishing the safety of the aforementioned rules. We apply the rule in order, starting from Rule 1 up to Rule 4. Every-time a rule applies, we start again from Rule 1. This way, we assume that a rule is applied exhaustively before moving on to the next one.

Lemma 61. *Let $(G = (A \cup B, E), k, S, T)$ be an instance of* Galactic Token Sliding *and let Q be any subset of $V(G)$. Let (G', k, S', T') be the instance obtained by identifying all vertices of Q into a single black hole vertex q which is adjacent to every vertex in $N_G(Q) \setminus Q$ (loops and multi-edges are ignored). If $Q \cap S \neq \emptyset$, then we set $\omega_{S'}(q) = |Q \cap S|$. Similarly, if $Q \cap T \neq \emptyset$, then we set $\omega_{T'}(q) = |Q \cap S|$. If (G, k, S, T) is a yes-instance, then (G', k, S', T') is a yes-instance.*

Proof. Assume that there exists a transformation from S to T in G. Let $\alpha := I_0 = S, I_1, \ldots, I_\ell = T$ be such a transformation. To obtain a transformation in G' we simply ignore all token slides that are restricted to $G[Q]$. Formally, we delete any I_i that is obtained from I_{i-1} by sliding a token along an edge in $G[Q]$. For every

I_i obtained from I_{i-1} by sliding a token from $N_G(Q) \setminus Q$ onto Q, we instead slide the token to q and increase the weight of q by one, i.e., $\omega_{I'_i}(q) = \omega_{I'_{i-1}}(q) + 1$. We replace every I_i obtained from I_{i-1} by sliding a token from Q to $N_G(Q) \setminus Q$ by I'_{i-1} and I'_{i-1} where one token gets projected from q onto its corresponding neighbor (and we decrease the weight of the black hole by one). All other slides in the sequence are kept as is and we obtain the desired sequence $\beta := I'_0 = S', I'_1, \ldots, I'_{\ell'} = T'$ from S' to T' in G'. □

Lemma 62. *Rule 1, the adjacent black holes rule, is safe.*

Proof. Let G be the initial galactic graph and G' be the graph obtained after contracting the two adjacent black holes u and v into a single black hole w. Let S, T be the two galactic independent sets of G and let S', T' be their counterparts in G'. If there is a transformation from S to T in G, then, by Lemma 61, there is a transformation from S' to T' in G'.

Assume now that there is a transformation from S' to T' in G'. We adapt it into a sequence in G maintaining the fact that, at each step, the weight of every vertex $s \neq u, v$ is the weight of s at the same step of the transformation in G' and $\omega(u) + \omega(v) = \omega(w)$. Note that S (resp. T) satisfies these conditions with S' (resp. T'). We perform the same sequence in G if possible, that is, if both vertices exist in G, we perform the slide (which is possible by the above condition). Now, let us explain how we simulate the moves between w and its neighbors. If a token on $s \in N(w)$ slides to w, then in G we simulate this move by sliding the corresponding token to u or v, depending on which vertex s is incident to (note that if $s \in N(u) \cap N(v)$, then the token on s can be slid to u or v). If the move corresponds to a token leaving w in G' to a vertex s, then if a vertex in $\{u, v\}$ incident to s has positive weight, we slide a token from one of these vertices to s. So we can assume, up to symmetry, that u is incident to s and $\omega(u) = 0$. Since $\omega(u) + \omega(v) = \omega(w)$ (at every step) and a token leaves w in G', we have $\omega(v) > 0$. Hence, we can move a token from v to u, and eventually move this token from u to s. □

Lemma 63. *Rule 2, the dominated black hole rule, is safe.*

Proof. Let us denote by G the original galactic graph and G' the graph where u has been deleted. Clearly, every sequence in G' is a sequence in G. We claim that every sequence in G from S to T can be adapted into a sequence where u never contains a token. Consider a reconfiguration sequence from S to T that minimizes the number of times a token enters u, and suppose for a contradiction that at least one token enters u. Let s be the last step where a token enters u and s' be the next time a token is leaving from u (note that both steps s and s' exist, since $\omega_S(u) = \omega_T(u) = 0$). Instead of moving a token to u at step s we move it to v and at step s', we move the token from v (which is possible since $N(u) \subseteq N(v)$). It still provides a sequence from S to T and the number of times a token enters u is reduced, a contradiction with the choice of the sequence. Hence, there exists sequence from S to t in G such that u never contains a token, and thus Rule 2 is safe. □

Lemma 64. *Assume that there exists a sequence between two galactic independent sets S and T of a galactic graph G. Then this sequence can be modified such that for each black hole b we have at most one token on $N(b) \cap A$ at all times.*

Proof. Consider such a reconfiguration sequence $I_0 = S, I_1, \ldots, I_\ell = T$ from S to T and suppose that there exists a black hole b such that, at some point in the sequence, $N(b) \cap A$ contains two tokens. Suppose first that $N(b) \cap A$ contains at least two vertices of S or at least two vertices of T. Let S' (resp. T') be the galactic independent set obtained by moving the token on S (resp. T) to b. By definition of black-holes this is a valid reconfiguration sequence, and thus there is a sequence transforming S into T if and only if there is one transforming S' into T', and $N(b) \cap A$ contains no vertex of the initial nor target independent set in its neighborhood anymore.

Otherwise, let I_i be the last galactic independent set in the transformation with two tokens on $N(b) \cap A$. By the choice of i, I_i is not the last independent set of the sequence. In the next galactic independent set I_{i+1} in the sequence, there is a unique token a on $N(b) \cap A$. Let $v_a \in N(b) \cap A$ be the vertex containing a in I_{i+1}. Let s be the step when a enters v_a (and does not move until I_{i+1}). We add a move in the sequence just after s consisting in sliding a from v_a to the black hole b. We then perform the same sequence of moves until we reach the galactic independent set $(I_{i+1} \setminus v_a) \cup b$. Finally, we move the token a from b to v_a, which gives the galactic independent set I_{i+1} from the original sequence. Hence, the number of steps with at least two tokens on the neighborhood of b has strictly reduced. We can repeat this argument as many times as needed on every black-hole of G, up until we obtain a sequence from S to T where no black-hole ever has two token in its neighborhood at the same step. □

Lemma 65. *Rule 3, the absorption rule, is safe.*

Proof. Let u be a black hole with a planet neighbor $v \notin S \cup T$ such that $S \cap A \cap N(v) = \emptyset$, and $T \cap A \cap N(v) = \emptyset$. We denote by G' the galactic graph where u and v are contracted into a vertex b and S' and T' be the galactic independent sets corresponding to S and T. If there is a transformation from S to T in G, then, by Lemma 61, there is a transformation from S' to T' in G'. Consider a transformation from S' to T' in G'. We claim that the transformation in G' can be changed into a transformation in G.

By Lemma 64, we can assume the existence of a sequence in G' where the number of tokens in $N(b) \cap A$ is at most one throughout the sequence. If there is a move in G' between two vertices s and t where $s, t \notin N(b)$, then the same move can be performed in G. If a token t in the sequence in G' has to move to $N(b)$ from a position distinct from b, then we first move the token t' on v (if such a token exists) to u in G (since $N_G(v) \setminus \{u\} \subseteq N_{G'}(b)$, leaving a token on v in G may result in two tokens being adjacent) before moving t. So we are left with the case where a token has to enter or leave b. If the token enters b from a neighbor w of u (in G), then we simply move the token to u (in G). So we can assume that the token enters b from a neighbor w of v (in G). In that case, we can perform the slides w to v and then v to u to put

the token on the black hole. Such a transformation is possible since there is no other token on $N(v)$ (in G', there is at most one token in $N(b)$ at all times). Similarly, if a token has to go to some vertex w of $N(v)$ from b, then there is currently no token on $N(v)$, and thus the sequence of moves u to v and v to w is possible, which completes the proof. □

As immediate consequences, the following property holds in an instance where Rules 1, 2, and 3 cannot be applied.

Corollary 2. *Each (planet) neighbor of a black hole must have at least one vertex of $S \cup T$ in its planet neighborhood.*

Proof. If a planet neighbor of a black hole has zero vertex of $S \cup T$ in its planet neighborhood, then Rule 3 can be applied and we get a contradiction. □

Lemma 66. *Rule 4, the path reduction rule, is safe.*

Proof. Let P be an A-geodesic path of length $5k$ in G such that no vertex of $A \cap N[P]$ are in the initial or target independent sets, S and T. Let G' be the graph obtained after contracting P into a single black hole b (recall that multi-edges and loops are ignored). Let S' and T' be the galactic independent sets corresponding to S and T. If there is a transformation from S to T in G, then, by Lemma 61, there is a transformatiom from S' to T' in G'. We now consider a transformatiom from S' to T' in G' and show how to adapt it in G.

By Lemma 64, we can assume the existence of a sequence in G' where the number of tokens in $N(b) \cap A$ is at most one throughout the sequence, for any black hole b. If there is a slide from a vertex u to a vertex v in G' such that $u, v \notin N[b]$, then the same slide can be applied in G. Whenever a token slides (in G') to a vertex u in $N(b)$, then we know that either u later slides to b or slides out of $N(b)$ (since we have at most one token in the neighborhood of black holes at all times). If the token does not enter b, then the same slide can be applied in G. If the token enters b, then we slide the token to a corresponding vertex in P (in G). Following that slide, two things can happen. Either this token leaves b, in which case we can easily adapt the sequence in G by sliding along the path P. In the other case, more tokens can slide into b, which mighty be the problematic case. Note, however, that P is of length $5k$ and is A-geodesic. Hence, every vertex $a \in A$ has at most three neighbors in P and any independent set of size at most k in A has at most $3k$ neighbors in P. This leaves $2k$ vertices on P which we can use to hold as many as k tokens that need to slide into b (in G'). In other words, whenever more than one token slides into b in G', we simulate this by sliding the tokens in P onto the $2k$ vertices of P that are free. Since initially $(A \cap N[P]) \cap (S \cup T) = \emptyset$, every time a token enters into a vertex $v \in N(b)$ in G', in G we can rearrange the tokens on P to guarantee that $N[v]$ contains no tokens. Finally, when a token leaves b to some vertex $v \in N(b)$ (in G'), then we rearrange the tokens on P so that a single token in P becomes closest to v. This token can safely slide from P to v. □

Corollary 3. *If Rule 4, the path reduction rule, cannot be applied, then the diameter of any planetary component is at most $O(k^2)$.*

Proof. Assume by contradiction that the diameter of any planetary component is at least $5k(k+1)$. Let P be a shortest A-path between two vertices at A-distance $5k(k+1)$. Note that P is A-geodesic. Since the size of each independent set is at most k and each planet can see at most three vertices on an A-geodesic path, there is a sub-path of P of length at least $5k$ that does not have any vertex of the independent set in its neighborhood. Hence, Rule 4 can be applied by Lemma 66, a contradiction. □

9.2 Graphs of bounded degree

We now show how the galactic reconfiguration framework can be applied to show that Token Sliding is fixed-parameter tractable (for parameter k) when restricted to graphs of bounded degree.

Note that an instance (G, k, S, T) of Token Sliding can also be considered as an instance of Galactic Token Sliding (containing only planet vertices) and S and T can be considered as two galactic independent sets of G. Then, (G, k, S, T) is a yes-instance of Token Sliding if and only if the corresponding irreducible instance (G_I, k, S, T) is a yes-instance of Galactic Token Sliding.

In the remaining of this section, we let (G, k, S, T) be an instance of Token Sliding where the maximum degree of G is $d = \Delta(G)$. Let (G_I, k, S, T) be the corresponding irreducible instance of Galactic Token Sliding. We let G'_I be the graph obtained from G_I after deleting $S \cup T$ and we let $C_1, C_2, \ldots, C_q$ denote the connected components of G'_I.

Lemma 67. *The number of connected components in G'_I is at most $2kd$.*

Proof. Observe that a planetary vertex that belongs to both to G and G_I has not been merged to any other vertex by the reduction rules, and thus has degree at most d in G'_I. It is in particular the case for the vertices in $S \cup T$. Therefore, G'_I contains at most $q \leq 2kd$ connected components $C_1, \ldots, C_q$. □

Lemma 68. *Let C be a component of G'_I. Then, $|V(C)|$ is at most $d^{O(k^2)}$*

Proof. If C does not contain any black hole, then C is a planetary component and has diameter $O(k^2)$ by Corollary 3. It follows that C contains $d^{O(k^2)}$ vertices since any vertex in C has degree at most d.

Suppose that C contains at least one black hole. Let A_C be the set of planet vertices which have at least one neighbor in $S \cup T$, i.e., the vertices that cannot be absorbed by Rule 3. Note that we have $|A_C| <= 2kd$. Since the vertices in A_C also have degree at most d, the graph obtained by deleting A_C from C contains at most $q' \leq 2kd^2$ connected components $C'_1, \ldots, C'_{q'}$. Let C' be one of these components. If C' contains a black hole b, then we claim that $C' = \{b\}$. Suppose otherwise. Then b either has a planet neighbor in C' which has degree 0 in $S \cup T$ (by definition of C') or b has

another black hole neighbor. In the former case, we get a contradiction to Corollary 2 and in the latter case we get a contradiction since Rule 1 would apply. Hence, either $C' = \{b\}$ or C' is a planetary component of diameter at most $5k$ containing at most $d^{O(k^2)}$ vertices (Corollary 3). Putting it all together, we get the desired bound. □

Theorem 40. *Token Sliding is fixed-parameter tractable when parameterized by* $k + \Delta(G)$.

Proof. Let (G, k, S, T) be an instance of Token Sliding. We first transform it to an instance of Galactic Token Sliding where all vertices are planetary vertices. We then apply all of the reduction rules exhaustively. The total number of components in G'_I is at most $2kd$ (Lemma 67) and each component contains at most $d^{O(k^2)}$ vertices (Lemma 68), as needed. □

9.2.1 On further using Galactic Token Sliding

In this chapter, we showed how to use the Galactic Token Sliding problem in order to show that Token Sliding is fixed-parameter tractable on graphs of bounded-degree. The rules designed to reduce the instances of Galactic Token Sliding essentially allow us to merge some connected subgraph of the input graph in a single vertex (black-hole) which maintains the nice properties of these subgraphs towards tokens (any number of tokens can enter, move inside, and leave these subgraphs). The reduction rules only merge neighbors and delete vertices, two operations that preserve both planarity and treewidth. Hence, we believe that Galactic Token Sliding could be a good starting point for the design of fixed-parameter tractable algorithms on planar graphs and bounded-treewidth graphs with parameter k.

Part III

Conclusion

Chapter 10

Conclusion

10.1 Graph recoloring and independent set reconfiguration

In the first part of this thesis, we investigated the question of finding linear transformations between colorings. We first tackled the case of chordal graph. Let G be a d-degenerate chordal graph of maximum degree Δ. The best known bound on the number of colors needed to obtain linear sequences prior to our work was $2d+2$, given by Theorem 16 of Bousquet and Perarnau. We showed that as long as the number of color k satisfies $k \geq d+4$, the diameter of $\mathcal{R}_k(G)$ is at most $O_\Delta(n)$. We also know from [19] that there exists an infinite family of d-degenerate chordal graphs which $(d+2)$-reconfiguration graph has diameter $\Omega(n^2)$. Therefore the case $k = d+3$ remains to be investigated:

Question 5. *Is the diameter of $\mathcal{R}_{d+3}(G)$ at most $f(\Delta(G)) \cdot n$ for any d-degenerate graph G?*

The other interesting question concerning chordal graphs is the dependency on the maximum degree of the graph. Is it possible to remove it? In other words, is the answer to the following question positive?

Question 6. *Is the diameter of $\mathcal{R}_{d+3}(G)$ at most $f(d) \cdot n$ for any d-degenerate chordal graph G?*

Our algorithm extensively makes use of the fact that the input graph has bounded degree. We believe that our technique could be used to obtain linear bound without dependency on Δ, but that it would require more than $d+3$ colors. We could, for instance, use the extra colors (above the $d+3$ threshold) in order to "isolate" high degree vertices, and then use our algorithm as a black box. However, we believe that answering Question 6 requires new ideas.

We then investigated the case of graphs of treewidth two. Again, we know from Theorem 16 that if G is a graph of treewidth two then $\mathcal{R}_6(G)$ has linear diameter, and we know from [15] that such a result is impossible for $\mathcal{R}_4(G)$. We closed the last remaining case and showed that $\mathcal{R}_5(G)$ indeed has linear diameter. Is it possible to

obtain similar results for graph of treewidth larger than two? In other words, can we answer the following question in the affirmative?

Question 8. *Does there exists a constant C such that, for every graph G of treewidth at most d, the diameter of $\mathcal{R}_k(G)$ is linear when $k \geq d + C$?*

The second part of this thesis was devoted to the independent set reconfiguration problem. We first showed that both Token Sliding and Token Jumping remain PSPACE-complete on H-free graphs unless H is a path on at least five vertices, the claw, or a subdivision of the claw. It is known from [23] that both problems are polynomial-time solvable on claw-free graphs but the complexity of Token Sliding and Token Jumping on H-free graphs remains to be investigated for the other graphs H.

We then considered independent set reconfiguration from a parameterized complexity point of view. The parameter we investigated is the size of the source and target independent sets, which we denote by k. In particular, we showed that Token Sliding is W[1]-hard on bipartite graphs, but that it admits a polynomial kernel on bipartite C_4-free graphs.
It is not hard to see that our construction proving the hardness of Token Sliding on bipartite graphs completely fails when considering Token Jumping. This leaves the following question open:

Question 12. *Is* Token Jumping FPT *parameterized by k on bipartite graphs?*

The proof of the kernel for Token Sliding on bipartite C_4-free adapts the "buffer space" technique that is regularly used for Token Jumping (and that we described in Chapter 6). This lead us to introduce a new reconfiguration problem, the Galactic Token Sliding which we used as a tool to show that Token Sliding is FPT on graphs of bounded-degree. We strongly believe that this is a powerful tool and that it could also be used to investigate the cases of planar graphs and graphs of bounded-treewidth:

Question 15. *Is it possible to adapt the reduction rules for Galatic Token Sliding to show that Token Sliding is* FPT *parameterized by k on planar graphs and bounded treewidth graphs?*

If this last question is answered in the affirmative, we could also ask whether Galactic Token Sliding can be used to design FPT algorithms for other classes of sparse graphs, on which very little is known for the Token Sliding rule.

10.2 Other problems we investigated

I also got to work on problems that are not related to reconfiguration during my Phd. I give below a brief description of two projects I worked on during the past three years and that lead to publications.

Zombies and Survivor. During a visit in Bordeaux in August 2020 I worked along with Laurine Bénéteau, Marthe Bonamy, Hoang La and Jonathan Narboni on a new variant of the famous Cops and Robbers game called *Zombies and Survivor*. In this game, zombies take the place of the cops and survivors take the place of the robber. The zombies, being of limited intelligence, have a very simple objective in each round – to move closer to a survivor. Therefore, each zombie must move along some shortest path, or geodesic, joining itself and a nearest survivor. We say that the zombies capture a survivor if one of the zombies moves onto the same vertex as a survivor. The *zombie number* of a graph G is the minimum number of zombies needed to ensure that the survivor will be eventually captured, and is denoted by $z(G)$. In [45], Fitzpatrick et al. asked the following:

Question 16. *Is* $z(G\square H) \leq z(G) + z(H)$ *for all graphs* G *and* H*?*

Where $G\square H$ is the cartesian product of G and H. We answered this question in the affirmative in a short note [8].

PACE challenge. During the years $2020 - 2021$ was held the sixth iteration of PACE, the Parameterized Algorithms and Computational Experiments Challenge, to which I participated along with Gabriel Bathie, Nicolas Bousquet, Marc Heinrich, Théo Pierron and Ulysse Prieto. PACE is a challenge at the frontier between programming and theory, where the goal for participants is to design and implement efficient algorithms to solve classical parameterized problems.
This year's problem was Cluster Editing. In this problem we are given a graph G and an integer k, and the question is whether it is possible to transform G into a graph where each connected component is a clique (sometimes called a *cluster graph*) by adding/deleting at most k edges from G.
We implemented both an exact algorithm and a heuristic for this problem. Our algorithms combine several existing techniques with new reduction rules based on the study of the symmetric difference of the neighborhoods of vertices. Both our algorithms were ranked third in the exact track and the heuristic track respectively and thus their descriptions will be published in the proceedings of the IPEC 2021 conference. The description of our exact algorithm can be found here: https://github.com/valbart/pace-2021/blob/main/solver_description.pdf and the description of the heuristic algorithm can be found here: https://github.com/GBathie/pace_2021_mu_solver/blob/main/solver_description.pdf.

Appendix A

Proof techniques for graph recoloring

A.1 Graphs with bounded maximum average degree

In this section, we sketch the proof of Theorem 17 as initially given by Bousquet and Perarnau in [29]. Let G be a graph of maximum average degree m_d and α, β be two k-colorings of G such that $k \geq m_d + 1$. A *t-partition of degree ℓ* of a graph $G = (V, E)$ is a partition of $V(G)$ into t sets $V_1, \ldots, V_t$ such that for every $i \leq t$ and every vertex $v \in V_i$, v has at most ℓ neighbors in the graph induced by $\cup_{j \geq i} V_j$. Note that a d-degenerate graph G on n vertices admits an n-partition of degree d, where each set V_i consist of a single vertex v_i such that $v_1, \ldots, v_n$ is a degeneracy ordering. Their proof relies on two key points:

1. A graph G such that $m_d(G) \leq m_d - \epsilon$ admits a $t := C \cdot log(n)$-partition of degree $m_d - 1$ where C only depends on m_d and ϵ.

2. If G admits a t-partition of degree ℓ, then G admits an independent set S such that $G - S$ admits a t-partition of degree $\ell - 1$.

Note that item 1 is a direct application of the definition of the maximum average degree: set V_1 to be all the vertices of degree at most $m_d - 1$ in G. Then repeat the process with $G - V_1$. Since $m_d(G) \leq m_d - \epsilon$, V_1 has size linear in n, and the process terminates in $O(log(n))$ steps. They make use of this slightly different peeling process as follows: First, remove the color $m_d + 1$ from the current coloring. To do so, pick a vertex v that is colored with $m_d + 1$ in V_t (if no such vertex exists, pick v in V_i for i maximum). By definition of the partition, there exists a color $a \neq m_d + 1$ that is not used in V_t. Try to recolor v with a: if not possible then there are some neighbors of v in $V_1, \ldots V_{t-1}$ colored with a: apply this procedure on these vertices with color a. Then recolor v with a. By construction, calling this procedure on a vertex in V_i does not recolor any vertex on $V_{i+1}, \ldots, V_t$. Analyzing the tree of calls of this procedure then shows that since V is partitioned into $t = Clog(n)$ subsets, each vertex calls the procedure at most $O(n^c)$ times, and thus is recolored at most $O(n^c)$ times.
Once the color $m_d + 1$ has been removed, one is free to recolor the independent set S of point 2 with $m_d + 1$ and to repeat the process with the partition of $G - S$ of degree

$m_d - 2$ with m_d colors. Note that recoloring S require at most $O(n)$ recolorings, and thus the bottleneck lies in the first part of the process. Applying this algorithm during $m_d + 1$ steps starting from any two $m_d + 1$-coloring α and β of G leads to the same $m_d + 1$ coloring γ of G (where the first independent set given by item 2 is colored with $m_d + 1$, the second independent set is colored by m_d, and so on). Hence their algorithm gives a reconfiguration sequence from α to γ and from β to γ, and thus from α to β.

A.2 Weakly chordal graphs and OAT graphs

Weakly chordal graphs A graph is *weakly chordal* if it does not contain a cycle of length at least five or the complement of a cycle of length at least five as an induced subgraph. The class of weakly chordal graphs is a natural generalization of the class of chordal graph, and was recently studied in the framework of graph recoloring by Feghali and Fiala in [43]. Unlike for chordal graphs, a k-colorable weakly chordal graph is not necessarily $k-1$-degenerate. In [43], the authors actually show that for any integer $k \geq 0$ there exists a k-colorable chordal graph G such that $\mathcal{R}_{k+1}(G)$ is not connected. However, they introduce a subclass of weakly chordal graphs which they call *compact graphs* that has the nice property that for any k-colorable compact graph G, $\mathcal{R}_{k+1}(G)$ is connected. Two non-adjacent vertices x, y of a graph G form a 2-*pair* if every induced paths $x - y$ path in G has length exactly two. Note that if $\{x, y\}$ is a 2-pair in G then $N_G(x) \cap N_G(y)$ is a separator of G. In what follows, we denote the common neighborhood $N_G(x) \cap N_G(y)$ of two vertices x, y of a graph G by $S_G(x, y)$. Given two vertices x, y of a graph we let $S_G(x, y) := N_G(x) \cap N_G(y)$.

Definition 4. *A weakly chordal graph G is compact if every subgraph H of G satisfies one of the following:*

1. *H is a clique, or*
2. *H contains a 2-pair $\{x, y\}$ such that $N_H(x) \subseteq N_H(y)$, or*
3. *H contains a 2-pair $\{x, y\}$ such that $C_x \cup S_H(x, y)$ induces a clique on at most three vertices, where C_x is the connected component of $H \setminus S_H(x, y)$ containing x.*

Let us now sketch the proof of the following theorem in [43]:

Theorem 41. *Let G be a k-colorable compact graph on n vertices. Then $\mathcal{R}_{k+1}(G)$ is connected and has diameter $O(n^2)$.*

The proof is short and make use of similar ideas as the ones used for chordal graph. Let α and β be two $k+1$ colorings of G and let us show that there exists a reconfiguration sequence from α to β of length at most $O(n^2)$. Is G is a clique, then the results holds as seen in the previous paragraph. If G satisfies condition 2 of the

definition of compact graphs, then let $\{x, y\}$ be a 2-pair that satisfies the condition. As before, we can "fold" x on y using the inclusion of the neighborhood: Since $N_G(x) \subseteq N_G(y)$, we can recolor x with the color of y and identify the two vertices. The remaining graph can be recolored by induction and the obtained sequence can be naturally extended to a sequence of G. Finally suppose that G satisfies condition 3 of the definition of compact graphs. Then $S_G(x, y)$ contains a single vertex z for otherwise condition 2 of the definition holds, and C_x contains at most one other vertex $w \neq x$. We can recolor the graph $G \setminus C_x$ by induction and extend the obtained sequence for G by recoloring vertices of C_x before z whenever z is recolored (which can be done since $C_x \cup \{z\}$ is a clique on at most k vertices and there are $k+1$ colors). It is then not hard to check that in any case the recoloring sequence contains $O(n^2)$ vertices.

We outline the fact that Theorem 41 implies that $\mathcal{R}_{k+1}(G)$ is connected and has at most quadratic diameter if G is *co-chordal*, that is G does not contain the complement of a cycle of length four or more as and induced subgraph. Showing that the class of compact graphs the class of co-chordal graph is non-trivial and we refer the interested reader to the full article for a complete proof. Let us conclude this paragraph by an observation: Feghali and Fiala showed that there exists k-colorable weakly chordal graphs G that admit frozen $k+1$-colorings, and thus that $\mathcal{R}_{k+1}(G)$ is disconnected. However, frozen colorings are trivial components in the reconfiguration graph, and nothing indicates that the size of a largest connected component of $\mathcal{R}_{k+1}(G)$ should have size more than $O(n^2)$. Hence the following question:

Question 17. *What is the maximum size of a connected component of $\mathcal{R}_{k+1}(G)$ over all possible k-colorable weakly chordal graph G on n vertices?*

OAT graphs The class of compact graphs defined by Feghali and Fiala could also be constructed in another way: starting from a clique G, a compact graph can be obtained by successively adding a vertex y to G such that the neighborhood of y is included in the neighborhood of another vertex x already in G, or by attaching a clique on at most two vertices to a single vertex z already in G.

In [12], Biedl et al. generalizes this idea to define the class of OAT graph as follows:

Definition 5. *A graph G is an OAT graph if it can be constructed from single vertex graphs with a finite sequence of the following four operations.*

1. *Taking the disjoint union of G_1 and G_2, defined as $(V_1 \cup V_2, E_1 \cup E_2)$.*
2. *Taking the join of G_1 and G_2, defined as $(V_1 \cup V_2, E_1 \cup E_2 \cup \{xy | x \in V_1, y \in V_2\}$.*
3. *Adding a vertex $u \notin V_1$ comparable to a vertex $v \in V_1$, defined as $(V_1 \cup \{u\}, E_1 \cup \{ux | x \in X\}$ where $X \subseteq N(v)$.*
4. *Attaching a complete graph $Q = (V_Q, E_Q)$ to a vertex v of G_1, defined as $V_1 \cup V_Q, E_1 \cup E_Q \cup \{qv | q \in V_Q\}$*

They show that the class of OAT graphs strictly contains the class of compact graph, but that it is disjoint from the class of weakly chordal graphs. They also show that OAT graphs contains P_4-sparse graphs, that is graphs for which any subset of five vertices induces at most one P_4. By definition this later class strictly contains the class of P_4-free graphs. The main result of Biedl et al. on recoloring OAT graphs is the following:

Theorem 42. *Let G be a k-colorable OAT graph on n vertices. Then $\mathcal{R}_{k+1}(G)$ is connected and has diameter at most $4n^2$.*

This Theorem generalizes both the result of Feghali and Fiala and a result of Bonamy and Bousquet in [15] who showed that the $k+1$-reconfiguration graph of a k-colorable P_4-free graph G is connected. However, the bound of Bonamy and Bousquet on $diam(\mathcal{R}_{k+1}(G)$ for a P_4-free graph G is better since they show that it is at most $O(\chi(G) \cdot n)$. We will not discuss the proof of Theorem 42 with as many details as for the previous results, but let us outline the fact that the definition of OAT graphs is inherently good for recoloring purposes. Let α and β be two colorings of a OAT graph G:

- If G is constructed from the disjoint union of two graphs G_1 and G_2, recolor independently G_1 and G_2 by induction
- If G is constructed from a graph G_1 by adding a vertex u comparable to a vertex $v \in G_1$, recolor u to the color of v, identify u and v and recolor by induction the obtained graph.
- If G is constructed from a graph G_1 by attaching a clique Q to a vertex $z \in G_1$, recolor G_1 by induction and extend the sequence to a sequence of G by recoloring vertices of Q whenever the color of z is modified. This is always possible since the number of available colors is strictly larger than the size of a maximum clique of G.
- If G is constructed from the join of two graphs G_1 and G_2, then the set of colors used for G_1 and G_2 are disjoint for any proper colorings of G. Then one can essentially recolor G_1 and G_2 independently as for the case of the disjoint union.

However, here again it is difficult to see how such a definition can be used to obtain linear bounds for the diameter of the reconfiguration graph. Suppose for instance that G is constructed from G_1 by adding a comparable vertex u to a vertex $v \in G_1$. The same problem as for chordal and compact graph arises: recoloring inductively the whole graph G_1 necessarily leads to recoloring each vertex a linear amount of time.

Bibliography

[1] Karen I. Aardal et al. “Models and solution techniques for frequency assignment problems”. en. In: *Annals of Operations Research* 153.1 (Sept. 2007), pp. 79–129. ISSN: 0254-5330, 1572-9338. DOI: 10.1007/s10479-007-0178-0. (Visited on 03/17/2021).

[2] Akitaya et al. “Reconfiguration of Connected Graph Partitions via Recombination”. In: 12701 (2021), pp. 61–74. DOI: 10.1007/978-3-030-75242-2_4.

[3] Vladimir E Alekseev. “Polynomial algorithm for finding the largest independent sets in graphs without forks”. en. In: *Discrete Applied Mathematics* 135.1-3 (Jan. 2004), pp. 3–16. ISSN: 0166218X. DOI: 10.1016/S0166-218X(02)00290-1.

[4] Vladimir E Alekseev. “The effect of local constraints on the complexity of determination of the graph independence number”. In: *Combinatorial-algebraic methods in applied mathematics* (1982), pp. 3–13.

[5] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. OCLC: ocn286431654. Cambridge ; New York: Cambridge University Press, 2009. ISBN: 978-0-521-42426-4.

[6] Armen S. Asratian and Carl Johan Casselgren. “Solution of Vizing’s Problem on Interchanges for the case of Graphs with Maximum Degree 4 and Related Results”. In: *Journal of Graph Theory* 82.4 (Aug. 2016), pp. 350–373. ISSN: 03649024. DOI: 10.1002/jgt.21906.

[7] Valentin Bartier, Nicolas Bousquet, and Marc Heinrich. “Recoloring graphs of treewidth 2”. In: *Discrete Mathematics* 344.12 (2021), p. 112553. ISSN: 0012-365X. DOI: https://doi.org/10.1016/j.disc.2021.112553.

[8] Valentin Bartier et al. “A note on deterministic zombies”. In: *Discrete Applied Mathematics* 301 (2021), pp. 65–68. ISSN: 0166-218X. DOI: https://doi.org/10.1016/j.dam.2021.05.001.

[9] Valentin Bartier et al. “On girth and the parameterized complexity of token sliding and token jumping”. In: *ISAAC*. 2020.

[10] Rémy Belmonte et al. “Independent Set Reconfiguration Parameterized by Modular-Width”. en. In: *Algorithmica* 82.9 (Sept. 2020), pp. 2586–2605. ISSN: 0178-4617, 1432-0541. DOI: 10.1007/s00453-020-00700-y.

[11] Rémy Belmonte et al. “Token Sliding on Split Graphs”. en. In: *Theory of Computing Systems* 65.4 (May 2021), pp. 662–686. ISSN: 1432-4350, 1433-0490. DOI: 10.1007/s00224-020-09967-8.

[12] Therese Biedl, Anna Lubiw, and Owen Merkel. "Building a larger class of graphs for efficient reconfiguration of vertex colouring". In: *arXiv:2003.01818 [cs]* (Mar. 2020). arXiv: 2003.01818.

[13] Alexandre Blanché et al. "Decremental Optimization of Dominating Sets Under the Reconfiguration Framework". In: *Combinatorial Algorithms*. Ed. by Leszek Gąsieniec, Ralf Klasing, and Tomasz Radzik. Vol. 12126. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 69–82. ISBN: 978-3-030-48965-6 978-3-030-48966-3. DOI: 10.1007/978-3-030-48966-3_6.

[14] Hans L. Bodlaender. "A Linear-Time Algorithm for Finding Tree-Decompositions of Small Treewidth". In: *SIAM Journal on Computing* 25.6 (Dec. 1996), pp. 1305–1317. ISSN: 0097-5397, 1095-7111. DOI: 10.1137/S0097539793251219.

[15] Marthe Bonamy and Nicolas Bousquet. "Recoloring graphs via tree decompositions". In: *European Journal of Combinatorics* 69 (Mar. 2018), pp. 200–213. ISSN: 01956698. DOI: 10.1016/j.ejc.2017.10.010.

[16] Marthe Bonamy and Nicolas Bousquet. "Reconfiguring Independent Sets in Cographs". In: *arXiv:1406.1433 [cs, math]* (June 2014). arXiv: 1406.1433.

[17] Marthe Bonamy and Nicolas Bousquet. "Token Sliding on Chordal Graphs". In: *Graph-Theoretic Concepts in Computer Science*. Ed. by Hans L. Bodlaender and Gerhard J. Woeginger. Vol. 10520. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, pp. 127–139. ISBN: 978-3-319-68704-9 978-3-319-68705-6. DOI: 10.1007/978-3-319-68705-6_10.

[18] Marthe Bonamy et al. *On Vizing's edge colouring question*. 2021. arXiv: 2107.07900 [math.CO].

[19] Marthe Bonamy et al. "Reconfiguration graphs for vertex colourings of chordal and chordal bipartite graphs". In: *Journal of Combinatorial Optimization* 27.1 (Jan. 2014), pp. 132–143. ISSN: 1382-6905, 1573-2886. DOI: 10.1007/s10878-012-9490-y.

[20] Édouard Bonnet et al. "Parameterized Complexity of Independent Set in H-Free Graphs". en. In: *Algorithmica* 82.8 (Aug. 2020), pp. 2360–2394. ISSN: 0178-4617, 1432-0541. DOI: 10.1007/s00453-020-00730-6.

[21] Paul Bonsma. "Independent Set Reconfiguration in Cographs and their Generalizations." en. In: *Journal of Graph Theory* 83.2 (Oct. 2016), pp. 164–195. ISSN: 03649024. DOI: 10.1002/jgt.21992. (Visited on 06/16/2021).

[22] Paul Bonsma and Luis Cereceda. "Finding Paths between graph colourings: PSPACE-completeness and superpolynomial distances". In: *Theoretical Computer Science* 410.50 (Nov. 2009), pp. 5215–5226. ISSN: 03043975. DOI: 10.1016/j.tcs.2009.08.023.

[23] Paul Bonsma, Marcin Kamiński, and Marcin Wrochna. "Reconfiguring Independent Sets in Claw-Free Graphs". In: *Algorithm Theory – SWAT 2014*. Ed. by David Hutchison et al. Vol. 8503. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2014, pp. 86–97. ISBN: 978-3-319-08403-9 978-3-319-08404-6. DOI: 10.1007/978-3-319-08404-6_8.

[24] Ludvig Borgne. "Automatic frequency assignement for cellular telephones using local search heuristicis". 1994.

[25] Prosenjit Bose and Ferran Hurtado. "Flips in planar graphs". In: *Computational Geometry* 42.1 (Jan. 2009), pp. 60–80. ISSN: 09257721. DOI: 10.1016/j.comgeo.2008.04.001.

[26] Nicolas Bousquet and Valentin Bartier. "Linear transformations between colorings in chordal graphs". In: *arXiv:1907.01863 [cs, math]* (July 2019). arXiv: 1907.01863.

[27] Nicolas Bousquet and Marc Heinrich. "A polynomial version of Cereceda's conjecture". In: *arXiv:1903.05619 [cs, math]* (Mar. 2019). arXiv: 1903.05619.

[28] Nicolas Bousquet, Arnaud Mary, and Aline Parreau. "Token Jumping in Minor-Closed Classes". In: *Fundamentals of Computation Theory*. Ed. by Ralf Klasing and Marc Zeitoun. Vol. 10472. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2017, pp. 136–149. ISBN: 978-3-662-55750-1 978-3-662-55751-8. DOI: 10.1007/978-3-662-55751-8_12.

[29] Nicolas Bousquet and Guillem Perarnau. "Fast recoloring of sparse graphs". In: *European Journal of Combinatorics* 52 (Feb. 2016), pp. 1–11. ISSN: 01956698. DOI: 10.1016/j.ejc.2015.08.001.

[30] Marcin Briański et al. "Reconfiguring Independent Sets on Interval Graphs". In: *arXiv:2105.03402 [math]* (May 2021). arXiv: 2105.03402.

[31] Luis Cereceda. "Mixing Graph Colourings". PhD thesis. London School of Economics and Political Science, 2007.

[32] Sitan Chen et al. "Improved bounds for randomly sampling colorings via linear programming". In: *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM. 2019, pp. 2216–2234.

[33] Stephen A. Cook. "The complexity of theorem-proving procedures". en. In: *Proceedings of the third annual ACM symposium on Theory of computing - STOC '71*. Shaker Heights, Ohio, United States: ACM Press, 1971, pp. 151–158. DOI: 10.1145/800157.805047.

[34] Marek Cygan et al. *Parameterized Algorithms*. 1st. Springer Publishing Company, Incorporated, 2015. ISBN: 3319212745.

[35] Erik D. Demaine et al. "Linear-time algorithm for sliding tokens on trees". en. In: *Theoretical Computer Science* 600 (Oct. 2015), pp. 132–142. ISSN: 03043975. DOI: 10.1016/j.tcs.2015.07.037.

[36] Reinhard Diestel. *Graph theory*. 3rd ed. Graduate texts in mathematics 173. OCLC: ocm61167989. Berlin ; New York: Springer, 2005. ISBN: 978-3-540-26182-7.

[37] Zdeněk Dvořák and Carl Feghali. "A Thomassen-type method for planar graph recoloring". In: *European Journal of Combinatorics* 95 (June 2021), p. 103319. ISSN: 01956698. DOI: 10.1016/j.ejc.2021.103319.

[38] Martin Dyer et al. "Randomly coloring sparse random graphs with fewer colors than the maximum degree". en. In: *Random Structures and Algorithms* 29.4 (Dec. 2006), pp. 450–465. ISSN: 10429832, 10982418. DOI: 10.1002/rsa.20129.

[39] Eduard Eiben and Carl Feghali. "Toward Cereceda's conjecture for planar graphs". In: *Journal of Graph Theory* 94.2 (June 2020), pp. 267–277. ISSN: 0364-9024, 1097-0118. DOI: 10.1002/jgt.22518.

[40] Carl Feghali. "Paths between colourings of graphs with bounded tree-width". In: *Information Processing Letters* 144 (Apr. 2019), pp. 37–38. ISSN: 00200190. DOI: 10.1016/j.ipl.2018.12.006.

[41] Carl Feghali. "Reconfiguring 10-Colourings of Planar Graphs". In: *Graphs and Combinatorics* 36.6 (Nov. 2020), pp. 1815–1818. ISSN: 0911-0119, 1435-5914. DOI: 10.1007/s00373-020-02199-0.

[42] Carl Feghali. "Reconfiguring colorings of graphs with bounded maximum average degree". In: *Journal of Combinatorial Theory, Series B* 147 (Mar. 2021), pp. 133–138. ISSN: 00958956. DOI: 10.1016/j.jctb.2020.11.001. (Visited on 03/23/2021).

[43] Carl Feghali and Jiří Fiala. "Reconfiguration graph for vertex colourings of weakly chordal graphs". In: *Discrete Mathematics* 343.3 (Mar. 2020), p. 111733. ISSN: 0012365X. DOI: 10.1016/j.disc.2019.111733.

[44] Carl Feghali, Matthew Johnson, and Daniël Paulusma. "A Reconfigurations Analogue of Brooks' Theorem and Its Consequences". In: *Journal of Graph Theory* 83.4 (Dec. 2016), pp. 340–358. ISSN: 03649024. DOI: 10.1002/jgt.22000.

[45] Shannon L Fitzpatrick et al. "A deterministic version of the game of zombies and survivors on graphs". In: *Discrete Applied Mathematics* 213 (2016), pp. 1–12.

[46] Eli Fox-Epstein et al. "Sliding Token on Bipartite Permutation Graphs". In: *Algorithms and Computation*. Vol. 9472. Berlin, Heidelberg, 2015, pp. 237–247. ISBN: 978-3-662-48970-3 978-3-662-48971-0. DOI: 10.1007/978-3-662-48971-0_21.

[47] Philippe Galinier, Michel Habib, and Christophe Paul. "Chordal graphs and their clique graphs". In: *Graph-Theoretic Concepts in Computer Science*. Ed. by Manfred Nagl. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 358–371.

[48] Philippe Galinier, Michel Habib, and Christophe Paul. "Chordal graphs and their clique graphs". en. In: *Graph-Theoretic Concepts in Computer Science*. Ed. by Gerhard Goos et al. Vol. 1017. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 358–371. ISBN: 978-3-540-60618-5 978-3-540-48487-5. DOI: 10.1007/3-540-60618-1_88.

[49] Guilherme C. M. Gomes, Sérgio H. Nogueira, and Vinicius F. dos Santos. "Some results on Vertex Separator Reconfiguration". In: *arXiv:2004.10873 [cs]* (Apr. 2020). arXiv: 2004.10873.

[50] Parikshit Gopalan et al. "The Connectivity of Boolean Satisfiability: Computational and Structural Dichotomies". en. In: *SIAM Journal on Computing* 38.6 (Jan. 2009), pp. 2330–2355. ISSN: 0097-5397, 1095-7111. DOI: 10.1137/07070440X. (Visited on 06/16/2021).

[51] Andrzej Grzesik et al. "Polynomial-time algorithm for Maximum Weight Independent Set on P_6-free graphs". In: *arXiv:1707.05491 [cs, math]* (Mar. 2020). arXiv: 1707.05491.

[52] R. Haas and K. Seyffarth. "The k-Dominating Graph". In: *Graphs and Combinatorics* 30.3 (May 2014), pp. 609–617. ISSN: 0911-0119, 1435-5914. DOI: 10.1007/s00373-013-1302-3.

[53] Michel Habib et al. "Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing". In: *Theoretical Computer Science* 234.1-2 (Mar. 2000), pp. 59–84. ISSN: 03043975. DOI: 10.1016/S0304-3975(97)00241-7.

[54] Arash Haddadan et al. "The complexity of dominating set reconfiguration". In: *Theoretical Computer Science* 651 (Oct. 2016), pp. 37–49. ISSN: 03043975. DOI: 10.1016/j.tcs.2016.08.016.

[55] Junghee Han. "Frequency reassignment problem in mobile communication networks". In: *Computers & Operations Research* 34.10 (2007), pp. 2939–2948. ISSN: 0305-0548. DOI: https://doi.org/10.1016/j.cor.2005.11.005.

[56] Robert A. Hearn and Erik D. Demaine. "PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation". en. In: *Theoretical Computer Science* 343.1-2 (Oct. 2005), pp. 72–96. ISSN: 03043975. DOI: 10.1016/j.tcs.2005.05.008. (Visited on 06/12/2021).

[57] Jan van den Heuvel. "The complexity of change". In: *Surveys in Combinatorics 2013*. Ed. by Simon Blackburn, Stefanie Gerke, and Mark Wildon. Cambridge: Cambridge University Press, 2013, pp. 127–160. ISBN: 978-1-139-50674-8. DOI: 10.1017/CBO9781139506748.005.

[58] Duc A. Hoang, Akira Suzuki, and Tsuyoshi Yagita. “Reconfiguring k-path Vertex Covers”. In: *WALCOM: Algorithms and Computation*. Ed. by M. Sohel Rahman, Kunihiko Sadakane, and Wing-Kin Sung. Vol. 12049. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 133–145. ISBN: 978-3-030-39880-4 978-3-030-39881-1. DOI: 10.1007/978-3-030-39881-1_12.

[59] Duc A. Hoang and Ryuhei Uehara. “Sliding Tokens on a Cactus”. en. In: (2016). Artwork Size: 26 pages Medium: application/pdf Publisher: Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik GmbH, Wadern/Saarbruecken, Germany, 26 pages. DOI: 10.4230/LIPICS.ISAAC.2016.37.

[60] Takehiro Ito, Marcin Kamiński, and Erik D. Demaine. “Reconfiguration of list edge-colorings in a graph”. en. In: *Discrete Applied Mathematics* 160.15 (Oct. 2012), pp. 2199–2207. ISSN: 0166218X. DOI: 10.1016/j.dam.2012.05.014.

[61] Takehiro Ito, Marcin Kamiński, and Hirotaka Ono. “Fixed-Parameter Tractability of Token Jumping on Planar Graphs”. In: *Algorithms and Computation*. Ed. by Hee-Kap Ahn and Chan-Su Shin. Vol. 8889. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2014, pp. 208–219. ISBN: 978-3-319-13074-3 978-3-319-13075-0. DOI: 10.1007/978-3-319-13075-0_17.

[62] Takehiro Ito, Hiroyuki Nooka, and Xiao Zhou. “Reconfiguration of Vertex Covers in a Graph”. In: *IEICE Transactions on Information and Systems* E99.D.3 (2016), pp. 598–606. ISSN: 0916-8532, 1745-1361. DOI: 10.1587/transinf.2015FCP0010.

[63] Takehiro Ito et al. “On the complexity of reconfiguration problems”. en. In: *Theoretical Computer Science* 412.12-14 (Mar. 2011), pp. 1054–1065. ISSN: 03043975. DOI: 10.1016/j.tcs.2010.12.005. (Visited on 06/12/2021).

[64] Takehiro Ito et al. “On the Parameterized Complexity for Token Jumping on Graphs”. In: *Theory and Applications of Models of Computation*. Ed. by David Hutchison et al. Vol. 8402. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2014, pp. 341–351. ISBN: 978-3-319-06088-0 978-3-319-06089-7. DOI: 10.1007/978-3-319-06089-7_24. (Visited on 06/16/2021).

[65] Mark Jerrum. “A very simple algorithm for estimating the number of k-colorings of a low-degree graph”. en. In: *Random Structures & Algorithms* 7.2 (Sept. 1995), pp. 157–165. ISSN: 10429832. DOI: 10.1002/rsa.3240070205. (Visited on 03/24/2021).

[66] Wm. Woolsey Johnson and William E. Story. “Notes on the "15" Puzzle”. In: *American Journal of Mathematics* 2.4 (Dec. 1879), p. 397. ISSN: 00029327. DOI: 10.2307/2369492.

[67] Marcin Kamiński, Paul Medvedev, and Martin Milanič. "Complexity of independent set reconfigurability problems". en. In: *Theoretical Computer Science* 439 (June 2012), pp. 9–15. ISSN: 03043975. DOI: 10.1016/j.tcs.2012.03.004. (Visited on 06/12/2021).

[68] Marcin Kamiński, Paul Medvedev, and Martin Milanič. "Shortest Paths between Shortest Paths and Independent Sets". In: *Combinatorial Algorithms*. Ed. by Costas S. Iliopoulos and William F. Smyth. Vol. 6460. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 56–67. ISBN: 978-3-642-19221-0 978-3-642-19222-7. DOI: 10.1007/978-3-642-19222-7_7. (Visited on 06/12/2021).

[69] Jeong Han Kim. "The Ramsey number R(3, t) has order of magnitude t2/log t". en. In: *Random Structures & Algorithms* 7.3 (Oct. 1995), pp. 173–207. ISSN: 10429832. DOI: 10.1002/rsa.3240070302.

[70] David A. Levin, Yuval Peres, and Elizabeth L. Wilmer. *Markov chains and mixing times*. American Mathematical Society, 2006.

[71] Daniel Lokshtanov and Amer E. Mouawad. "The Complexity of Independent Set Reconfiguration on Bipartite Graphs". In: *ACM Transactions on Algorithms* 15.1 (Jan. 2019), pp. 1–19. ISSN: 1549-6325, 1549-6333. DOI: 10.1145/3280825. (Visited on 06/18/2021).

[72] Daniel Lokshtanov, Martin Vatshelle, and Yngve Villanger. "Independent set in P 5-free graphs in polynomial time". In: *Proceedings of the twenty-fifth annual ACM-SIAM symposium on discrete algorithms*. SIAM. 2014, pp. 570–581.

[73] Daniel Lokshtanov et al. "Reconfiguration on sparse graphs". en. In: *Journal of Computer and System Sciences* 95 (Aug. 2018), pp. 122–131. ISSN: 00220000. DOI: 10.1016/j.jcss.2018.02.004.

[74] Anna Lubiw and Vinayak Pathak. "Flip distance between two triangulations of a point set is NP-complete". In: *Computational Geometry* 49 (Nov. 2015), pp. 17–23. ISSN: 09257721. DOI: 10.1016/j.comgeo.2014.11.001.

[75] Jessica McDonald, Bojan Mohar, and Diego Scheide. "Kempe Equivalence of Edge-Colorings in Subcubic and Subquartic Graphs". In: *Journal of Graph Theory* 70.2 (June 2012), pp. 226–239. ISSN: 03649024. DOI: 10.1002/jgt.20613.

[76] George J Minty. "On maximal independent sets of vertices in claw-free graphs". en. In: *Journal of Combinatorial Theory, Series B* 28.3 (June 1980), pp. 284–304. ISSN: 00958956. DOI: 10.1016/0095-8956(80)90074-X.

[77] Bojan Mohar. "Kempe Equivalence of Colorings". In: *Graph Theory in Paris*. Ed. by Adrian Bondy et al. Series Title: Trends in Mathematics. Basel: Birkhäuser Basel, 2007, pp. 287–297. ISBN: 978-3-7643-7228-6. DOI: 10.1007/978-3-7643-7400-6_22.

[78] Amer E. Mouawad, Naomi Nishimura, and Venkatesh Raman. "Vertex Cover Reconfiguration and Beyond". In: *Algorithms and Computation*. Ed. by Hee-Kap Ahn and Chan-Su Shin. Vol. 8889. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2014, pp. 452–463. ISBN: 978-3-319-13074-3 978-3-319-13075-0. DOI: 10.1007/978-3-319-13075-0_36.

[79] Amer E. Mouawad et al. "On the Parameterized Complexity of Reconfiguration Problems". In: *Parameterized and Exact Computation*. Ed. by David Hutchison et al. Vol. 8246. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2013, pp. 281–294. ISBN: 978-3-319-03897-1 978-3-319-03898-8. DOI: 10.1007/978-3-319-03898-8_24.

[80] Naomi Nishimura. "Introduction to Reconfiguration". In: *Algorithms* 11.4 (Apr. 2018), p. 52. ISSN: 1999-4893. DOI: 10.3390/a11040052. (Visited on 03/16/2021).

[81] Hiroki Osawa et al. "Complexity of Coloring Reconfiguration under Recolorability Constraints". en. In: (2017). Artwork Size: 12 pages Medium: application/pdf Publisher: Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik GmbH, Wadern/Saarbruecken, Germany, 12 pages. DOI: 10.4230/LIPICS.ISAAC.2017.62.

[82] Hiroki Osawa et al. "The Complexity of (List) Edge-Coloring Reconfiguration Problem". en. In: *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* E101.A.1 (2018), pp. 232–238. ISSN: 0916-8508, 1745-1337. DOI: 10.1587/transfun.E101.A.232.

[83] Donald J. Rose, R. Endre Tarjan, and George S. Lueker. "Algorithmic Aspects of Vertex Elimination on Graphs". en. In: *SIAM Journal on Computing* 5.2 (June 1976), pp. 266–283. ISSN: 0097-5397, 1095-7111. DOI: 10.1137/0205021.

[84] Jesús Salas and Alan D. Sokal. "Absence of phase transition for antiferromagnetic Potts models via the Dobrushin uniqueness theorem". en. In: *Journal of Statistical Physics* 86.3-4 (Feb. 1997), pp. 551–579. ISSN: 0022-4715, 1572-9613. DOI: 10.1007/BF02199113. (Visited on 03/24/2021).

[85] Walter J. Savitch. "Relationships between nondeterministic and deterministic tape complexities". en. In: *Journal of Computer and System Sciences* 4.2 (Apr. 1970), pp. 177–192. ISSN: 00220000. DOI: 10.1016/S0022-0000(70)80006-X.

[86] Najiba Sbihi. "Algorithme de recherche d'un stable de cardinalite maximum dans un graphe sans etoile". en. In: *Discrete Mathematics* 29.1 (1980), pp. 53–76. ISSN: 0012365X. DOI: 10.1016/0012-365X(90)90287-R.

[87] Sebastian Siebertz. "Reconfiguration on Nowhere Dense Graph Classes". In: *The Electronic Journal of Combinatorics* 25.3 (Aug. 2018), P3.24. ISSN: 1077-8926. DOI: 10.37236/7458. (Visited on 06/23/2021).

[88] Daniel D. Sleator, Robert E. Tarjan, and William P. Thurston. "Rotation distance, triangulations, and hyperbolic geometry". In: *Journal of the American Mathematical Society* 1.3 (Sept. 1988), pp. 647–647. ISSN: 0894-0347. DOI: 10.1090/S0894-0347-1988-0928904-4.

[89] Akira Suzuki, Amer E. Mouawad, and Naomi Nishimura. "Reconfiguration of Dominating Sets". In: *Computing and Combinatorics*. Ed. by Zhipeng Cai, Alex Zelikovsky, and Anu Bourgeois. Vol. 8591. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2014, pp. 405–416. ISBN: 978-3-319-08782-5 978-3-319-08783-2. DOI: 10.1007/978-3-319-08783-2_35.

[90] Asahi Takaoka. "Complexity of Hamiltonian Cycle Reconfiguration". en. In: *Algorithms* 11.9 (Sept. 2018), p. 140. ISSN: 1999-4893. DOI: 10.3390/a11090140. (Visited on 07/09/2021).

[91] C. Thomassen. "Every Planar Graph Is 5-Choosable". In: *Journal of Combinatorial Theory, Series B* 62.1 (Sept. 1994), pp. 180–181. ISSN: 00958956. DOI: 10.1006/jctb.1994.1062.

[92] Carsten Thomassen. "Decomposing a Planar Graph into an Independent Set and a 3-Degenerate Graph". In: *Journal of Combinatorial Theory, Series B* 83.2 (Nov. 2001), pp. 262–271. ISSN: 00958956. DOI: 10.1006/jctb.2001.2056.

[93] Eric Vigoda. "Improved bounds for sampling colorings". en. In: *Journal of Mathematical Physics* 41.3 (Mar. 2000), pp. 1555–1569. ISSN: 0022-2488, 1089-7658. DOI: 10.1063/1.533196. (Visited on 03/24/2021).

[94] V. G. Vizing. "The chromatic class of a multigraph". In: *Cybernetics* 1.3 (May 1965), pp. 32–41. ISSN: 0011-4235, 1573-8337. DOI: 10.1007/BF01885700.

[95] VG Vizing. *On an estimated of the chromatic class of a p-graph. Diskrete Analiz., 3: 25–30.* 1964.

[96] K. Wagner. "Bemerkungen zum Vierfarbenproblem." eng. In: *Jahresbericht der Deutschen Mathematiker-Vereinigung* 46 (1936), pp. 26–32.

[97] Douglas Brent West. *Introduction to graph theory*. 2nd ed. Upper Saddle River, N.J: Prentice Hall, 2001. ISBN: 978-0-13-014400-3.

[98] Marcin Wrochna. "Reconfiguration in bounded bandwidth and tree-depth". en. In: *Journal of Computer and System Sciences* 93 (May 2018), pp. 1–10. ISSN: 00220000. DOI: 10.1016/j.jcss.2017.11.003.

[99] Takeshi Yamada and Ryuhei Uehara. "Shortest Reconfiguration of Sliding Tokens on a Caterpillar". In: *WALCOM: Algorithms and Computation*. Ed. by Mohammad Kaykobad and Rossella Petreschi. Vol. 9627. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, pp. 236–248. ISBN: 978-3-319-30138-9 978-3-319-30139-6. DOI: 10.1007/978-3-319-30139-6_19.

Index