TU WIEN Informatics

Turbocharging Heuristics for Weak Coloring Numbers

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Logic and Computation

eingereicht von

Alexander Dobler, BSc.
Matrikelnummer 01631858

an der Fakultät für Informatik
der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Inform. Dr.rer.nat. Martin Nöllenburg
Mitwirkung: Univ.Lektor Dr.rer.nat. Manuel Sorge
Projektass.(FWF) Anaïs Villedieu

Wien, 1. Dezember 2021

Alexander Dobler	Martin Nöllenburg

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

TU WIEN Informatics

Turbocharging Heuristics for Weak Coloring Numbers

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Logic and Computation

by

Alexander Dobler, BSc.
Registration Number 01631858

to the Faculty of Informatics
at the TU Wien

Advisor: Univ.Prof. Dipl.-Inform. Dr.rer.nat. Martin Nöllenburg
Assistance: Univ.Lektor Dr.rer.nat. Manuel Sorge
Projektass.(FWF) Anaïs Villedieu

Vienna, 1st December, 2021

Alexander Dobler | Martin Nöllenburg

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Alexander Dobler, BSc.

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 1. Dezember 2021

Alexander Dobler

Acknowledgements

I want to especially thank my co-advisors Manuel Sorge and Anaïs Villedieu, and my advisor Martin Nöllenburg who supported me in many hours of fruitful discussions, who guided me through the process of writing this thesis, solving difficult tasks, and answering the main research questions. The idea for this thesis was posed by Manuel Sorge, and I am thankful that I could work on such interesting problems.

I want to express my gratitude towards my family and especially my parents, who made it possible to pursue the path that I am on right now. They supported me on every step along the way, and always had the right answers when I was needing advice.

Lastly, I want to thank my friends who made all the time of studying enjoyable, and who always had time for me when I needed some time off.

Kurzfassung

In den letzten Jahren wurden komplexere Charakterisierungen von dünnbesetzten Graphen, die *nirgendwo-dichte* Graphklassen und Graphklassen mit *beschränkter Expansion* umfassen. Beide fallen in die Kategorie der *strukturellen Sparsität* und ermöglichen in der Theorie eine Vielzahl effizienter Algorithmen.

Während es viele Charakterisierungen für diese Graphklassen gibt, kann eine in Form von *schwachen Färbungszahlen* angegeben werden. Für jeden Radius $r \in \mathbb{N}$ werden diese durch lineare Ordnungen von Knoten eines Graphen definiert. Jede Ordnung von Knoten in einem Graphen hat eine schwache r-Färbungszahl, und die schwache r-Färbungszahl eines Graphen ist die minimale schwache r-Färbungszahl der Ordnungen seiner Knoten. Die schwache r-Färbungszahl eines Graphen misst Erreichbarkeitseigenschaften an der Entfernung r, hat aber auch direkte algorithmische Anwendungen.

Obwohl es viele Forschungsartikel gibt, die sich auf obere und untere Schranken von schwachen Färbungszahlen in spezifischen Graphklassen konzentrieren, gibt es bis jetzt wenig Forschung in Bezug auf Komplexitätsergebnisse für die Berechnung schwacher r-Färbungszahlen und den Entwurf von Algorithmen zur Berechnung von Ordnungen mit kleinen schwachen Färbungszahlen. Es hat sich gezeigt, dass die Berechnung schwacher r-Färbungszahlen für $r \geq 3$ NP-vollständig ist, und daher benötigen wir effiziente Heuristiken, um gute obere Schranken für schwache r-Färbungszahlen zu berechnen.

In dieser Arbeit entwerfen wir mehrere Heuristiken zur Berechnung oberer Schranken von schwachen Färbungszahlen, die das *Turbocharging*-Framework verwenden. In der allgemeinsten Fassung kann das Framework als „Erweitern eines heuristischen Algorithmus mit einem exakten Algorithmus“ beschrieben werden, oder mit anderen Worten, „Turbocharging der Heuristik“. Mittels einiger bekannter Heuristiken entwickeln wir mehrere exakte Algorithmen, die diese Heuristiken lokal ergänzen und zusätzlich beweisen wir obere und untere Komplexitätsgrenzen dieser Algorithmen. Wir implementieren die daraus resultierenden Ansätze und führen eine umfassende experimentelle Auswertung für reale Graphinstanzen durch.

Dabei diskutieren wir Vor- und Nachteile des Turbocharging-Ansatzes, die während unserer Forschung auftraten und für zukünftige Forschungen relevant sein könnten.

Abstract

In recent years, more involved characterizations of sparse graphs were introduced, involving *nowhere dense* graph classes and graph classes of *bounded expansion*. Both fall into the category of *structural sparsity* and exhibit a wide variety of efficient algorithms in theory.

One characterization for these graph classes can be given in terms of *weak coloring numbers*. They are defined in terms of vertex orderings of a graph for a given radius $r \in \mathbb{N}$. Each ordering has a weak r-coloring number, and the weak r-coloring number of a graph is the minimum weak r-coloring number over all its vertex orderings. The weak r-coloring number of a graph measures reachability properties at distance r, but also has direct algorithmic applications.

Although there are many results focusing on upper and lower bounds for weak coloring numbers in specific graph classes, there is little research with regard to complexity results for computing weak r-coloring numbers and designing heuristics that compute vertex orderings with small weak r-coloring number. Computing weak r-coloring numbers is NP-complete for $r \geq 3$, and thus we need efficient algorithms to compute good upper bounds on weak r-coloring numbers.

In this thesis, we propose several algorithms that compute orderings of small weak coloring numbers by applying the *turbocharging* framework. In the most general form, the framework can be described as "augmenting a heuristic algorithm with an exact algorithm", or in other words, "turbocharging the heuristic". Given some known heuristics that iteratively compute an ordering of small weak coloring numbers, we propose several exact algorithms that locally augment these heuristics, and additionally, prove upper and lower complexity bounds of these algorithms. We implement the resulting approaches and provide a thorough experimental evaluation for real-world graph instances.

In the process, we discuss advantages and drawbacks of the turbocharging approach that came up during our research and might be relevant for future research.

Contents

CHAPTER 1

Introduction

We start by introducing some related background work on sparsity and structural properties of graphs (Section 1.1), weak coloring numbers (Section 1.2), and turbocharging (Section 1.3). This should serve as motivation for the subject of the thesis. We will further elaborate on the notions introduced in Sections 1.1 to 1.3 in Chapters 3 and 4. In Section 1.4 we will give the main research goals and contents of this thesis. Section 1.5 gives a brief overview on the structure of the work.

1.1 Sparsity and Structural Properties of Graphs

Complex networks such as social networks, biological networks and more have been studied intensively in recent years. We often need efficient algorithms to study computationally hard problems on such graphs. Usually, a requirement to obtain such algorithms is to identify structural properties of the underlying graph structures. The simplest example are graphs where the maximum degree is bounded by a constant; there have been established many results for this graph class, too many to even list here.

Many graph measures try to capture sparsity properties of classes of graphs. Intuitively, sparse graphs have few edges compared to their number of vertices. While this characterization is simple, it is far too vague and weak — in the sense that it is not useful for developing efficient algorithms. Thus, in recent years more involved characterizations of sparse graphs were introduced, involving *nowhere dense* graph classes [NM11] and graph classes of *bounded expansion* [NM08a; NM08b; NM08c]. Both fall into the category of *structural sparsity*. These classes are defined in terms of *bounded-depth-minors*— which are in some sense contractions of subgraphs of small diameter into a single vertex. The full definitions of these classes will be given in later parts of the thesis. Still, we want to mention that there is a relation between bounded-depth-minors and small local clusters of vertices in a graph — which could be formed by local communities in social networks. Both nowhere dense graph classes and graph classes of bounded expansion immediately

exhibit a wide variety of efficient algorithms in theory [DK09; Eic+17; GKS17; KRS19]. There are many characterizations of nowhere dense graph classes and graph classes of bounded expansion, one of them being weak coloring numbers.

1.2 Weak Coloring Numbers

Weak coloring numbers were first introduced by Kierstead and Yang [KY03], and for a radius $r \in \mathbb{N}$ they are defined in terms of vertex orderings of a graph. Each vertex ordering of a graph has a weak r-coloring number, and the weak r-coloring number of a graph is the minimum weak r-coloring number over all its vertex orderings. A full definition for weak r-coloring numbers is given in Chapter 3. The weak r-coloring number of a graph measures reachability properties at distance r, but weak r-coloring numbers and orderings of small weak r-coloring number also have direct algorithmic applications [RS20; Eic+17; PST18; RVS19; Ami+18; Dra+16; Dvo13; Dvo19; GKS17]. Zhu [Zhu09] has shown that nowhere dense graph classes and graph classes of bounded expansion can be characterized in terms of weak coloring numbers, showing a close connection between weak coloring numbers and notions of structural sparsity.

There are many papers that study weak r-coloring numbers, most of them showing upper and/or lower bounds on weak r-coloring numbers for different graph classes (see e.g. [JM21; Heu+17; HW18; Gro+18]). For a detailed list see Chapter 3.

Despite all the algorithmic applications of weak r-coloring numbers, there is little research focusing on complexity results for weak r-coloring numbers and designing algorithms that compute weak r-coloring numbers. A well-known graph measure that can be computed in linear time is the *degeneracy* of a graph [MB83]. As weak 1-coloring numbers correspond to the degeneracy plus one, they can be computed in linear time. Grohe et al. have shown that computing weak r-coloring numbers for $r \geq 3$ is NP-complete [Gro+18]. The complexity of computing weak r-coloring numbers for $r = 2$ is still unknown. Furthermore, there are no results with regard to the complexity of computing weak r-coloring numbers under the introduction of some structural properties of the input instance — that is, there are no fixed-parameter tractability/intractability results for the problem. Only one paper that we know of focuses on designing algorithms to compute upper bounds for weak r-coloring numbers: Nadara et al. [Nad+19]. They proposed several heuristics that compute vertex orderings of real-word graphs with small weak r-coloring numbers. They implemented all their algorithms and conducted experiments for different classes of real-world graphs and different radii r. As this is the only paper that focuses on computing upper bounds for weak r-coloring numbers of real-world graphs, we think that there is still a lot of room for improvement.

1.3 Turbocharging

Turbocharging is a problem-solving framework that was first applied by Hartung and Niedermeier [HN13] to the list coloring problem under the name of "parameterization by conservation". In the most general form, the framework can be described as "augmenting

a heuristic algorithm with an exact algorithm", or in other words, "turbocharging the heuristic". That is, while computing a solution for a specific problem with a heuristic, we might at some point apply an exact subroutine that optimizes the computed solution. Depending on the complexity of the exact algorithm, applying the turbocharging problem-solving framework might be fruitful for practical applications to solve hard problems. There are several papers that apply the turbocharging problem-solving framework. Gaspers et al. [Gas+19]: They apply it to augment heuristics that compute upper bound on the treewidth of a graph, while, as stated before, Hartung and Niedermeier use it for the list coloring problem. Ramaswamy and Szeider [RS21] use it to augment algorithms for bayesian network structure learning. We will give a refinement of the explanation for turbocharging and a detailed explanation of previous work in Chapter 4.

1.4 Our Contribution

The main focus of this thesis lies on designing competitive algorithms that improve upper bounds of weak r-coloring numbers by applying the turbocharging framework. Our guiding research question is: "Can the turbocharging framework be successfully applied to the domain of computing orderings of small weak r-coloring number?"

That is, we start by taking several greedy heuristics that compute orderings of small weak coloring numbers by Nadara et al. [Nad+19], and then, we identify ways to apply the turbocharging framework by defining several local exact algorithms that can be applied during the application of said heuristics. We analyze those problems from a theoretical standpoint, giving lower and upper bounds on their algorithmic complexity — also considering structural properties and parameters arising in problem inputs. These parameters (e.g. maximum degree of the input graph) will be incorporated into the problem definitions, maybe sometimes resulting in tractable algorithmic complexity bounds assuming those parameters are small (see parameterized complexity in Section 2.2).

Furthermore, we implement promising algorithmic approaches that apply the turbocharging framework. These implementations involve optimizations with proven guarantees, and heuristic optimizations based on empirical observations with respect to the considered input instances.

We then conduct a variety of experiments to validate the performance of the implemented algorithms. We compare the performance of our approaches to the performance of the heuristics of Nadara et al. [Nad+19]. Additionally, we present the results, and discuss them.

As far as we know, turbocharging has not yet been applied to computing orderings of small weak r-coloring numbers yet. Furthermore, the inspection of algorithmic and complexity properties of computing weak r-coloring numbers is still an open research field.

1.5 Structure of the Work

This thesis is divided into several chapters. We start with Chapter 2 by giving some preliminaries on linear orders, graph theory, and fixed-parameter tractability.

In Chapter 3, we introduce the concept of generalized coloring numbers, especially weak coloring numbers. We review previous research about generalized coloring numbers, and define the main problem of this thesis — computing orderings of vertices with weak r-coloring numbers at most $k \in \mathbb{N}$.

Chapter 4 gives an overview on previous research on turbocharging. Furthermore, we start to relate turbocharging to computing orderings of small weak r-coloring numbers.

The greedy heuristics of Nadara et al. [Nad+19] for computing orderings of small weak r-coloring number can be divided into two categories — heuristics that compute orderings from "left to right", and heuristics that compute orderings from "right to left". Both approaches initiate a different line of research with respect to turbocharging, which is why we also divide them into separate chapters. In this thesis, we mostly focus on turbocharging heuristics that build orderings from left to right (see Chapter 5), we give implementation details and some optimizations in Chapter 6. Finally, we initiate the research for turbocharging heuristics that build orderings from right to left in Chapter 7.

In Chapter 8, we present experimental results for all the approaches and optimizations discussed in Chapters 5 to 7. We discuss results and compare them with the results of Nadara et al. [Nad+19].

We conclude the thesis with Chapter 9, giving a brief summary of the most important results and observations.

Most chapters include a section where we discuss important insights and ideas for future research at the end.

CHAPTER 2

Preliminaries

General preliminaries. We denote by $\mathbb{N}$ the set of all natural numbers, starting at 1. Furthermore, for $n \in \mathbb{N}$, $[n]$ is the set $\{1, \ldots, n\}$.

One of the main mathematical objects studied in this thesis are linear orders L of a set S. A linear or total order L is a relation, hence $L \subseteq S \times S$, on L, which is reflexive, transitive, antisymmetric and total. For $s_1, s_2 \in S$ we write

- $s_1 \preceq_L s_2$ if and only if $(s_1, s_2) \in L$,
- $s_1 \prec_L s_2$ if and only if $(s_1, s_2) \in L$, and $s_1 \neq s_2$,
- $s_1 \succeq_L s_2$ if and only if $s_2 \preceq_L s_1$, and
- $s_1 \succ_L s_2$ if and only if $s_2 \prec_L s_1$.

We will also write linear orders as sequences of its elements, that is, $L = (s_1, \ldots, s_n)$ if and only if $s_1 \prec_L \ldots \prec_L s_n$ for linear orders L of a set $S = \{s_1, \ldots, s_n\}$. Furthermore, we say that s_1 is *to the left* (*to the right*) of s_2 if and only if $s_1 \prec_L s_2$ ($s_1 \succ_L s_2$). Given a subset $S' \subseteq S$ with $S' \neq \emptyset$, we say that an element $s \in S'$ is the *leftmost* (*rightmost*) element of S' if and only if $s \preceq s'$ ($s \succeq s'$) for all $s' \in S'$. We will also refer to linear orders as *orderings*. Linear orders induce *positions* of elements, for $s \in S$ we say that $\mathrm{pos}_L(s) = k$ if and only if $|\{s' \in S \mid s' \preceq_L s\}| = k$. For example, $\mathrm{pos}_L(s) = 1$ for the leftmost element of L. Let L be an ordering of a set S and $S' \subseteq S$, we define $L[S']$ as the ordering of the set S' such that $s \preceq_{L[S']} s'$ if and only if $s \preceq_L s'$ for all $s, s' \in S'$. That is, $L[S']$ is the *induced ordering* induced by L on S', and we say that $L[S']$ *agrees with* L on S'.

Graph theory. We only consider undirected graphs without loops. A *graph* G is a tuple $G = (V, E)$ where V is a set of *vertices* and E is a set of *edges*. Edges are of the form $\{u, v\}$ with $u, v \in V$. Thus, we have $E \subseteq \{\{u, v\} \mid u, v \in V, u \neq v\}$. A vertex $v \in V$

and edge $e \in E$ are *incident* to each other if and only if $v \in e$. Vertices $u, v \in V$ are *adjacent* to each other if and only if $\{u, v\} \in E$. A *path* $P = (v_0, \ldots, v_\ell)$ in a graph G is a non-empty sequence of vertices, such that two consecutive vertices share an incident edge. The *length* ℓ of a path is the number of vertices it contains minus one, thus, empty paths have length 0. Vertex $v \in V$ is *reachable* from $u \in V$ if there is a path $P = (v_0, \ldots, v_l)$ in G such that $v_0 = u$ and $v_l = v$. By this definition v is reachable from itself by a path of length zero. A graph G is *connected* if there exists a path between each pair of vertices in G.

Graph notation. We follow up with further graph notation that will be used frequently. Let $G = (V, E)$ be a graph, we denote by

$V(G)$	the vertex set of G;
$E(G)$	the edge set of G;
n_G	the number of vertices of G, more formally, $n_G = \lvert V(G)\rvert$;
$G[V']$	the *induced subgraph* on the vertices $V' \subseteq V$, more formally, $G[V'] = (V', \{\{v, w\} \in E \mid v, w \in V'\})$;
$G - v$	the graph obtained by removing v and its incident edges, more formally, $G - v = G[V - \{v\}]$;
$G' \subseteq G$	that G' is a *subgraph* of G, more formally, G' consists of vertices $V' \subseteq V$ and edges $E' \subseteq E$ such that for each $\{v, w\} \in E'$ we have that $v, w \in V'$;
$N_G^m(v)$	the set of vertices that are reachable from $v \in V(G)$ in G by a path of length ℓ s.t. $1 \leq \ell \leq m$;
$N_G(v)$	the set of vertices that are adjacent to v in G, more formally, $N_G(v) = N_G^1(v)$;
$d_G(v)$	the degree of v, more formally, the number of adjacent vertices. Hence, $d(v) = \lvert\{\{v, w\} \mid \{v, w\} \in E\}\rvert$;
$\mathrm{dist}_G(v, w)$	the distance between v and w in G, more formally, $\mathrm{dist}_G(v, w)$ is the length of the shortest path in G between v and w, where $v, w \in V(G)$. By convention $\mathrm{dist}_G(v, w) = \infty$, if there is no path between v and w in G;
$\Pi(G)$	the set of all linear orders of the set of vertices $V(G)$;

$\mathrm{CC}(G)$ the *connected components* of G, more formally, $(V', E') = G' \in \mathrm{CC}(G)$ if and only if

- $G' = G[V']$,
- G' is connected, and
- $\forall v \in V(G) \setminus V' : G[V' \cup \{v\}]$ is not connected.

That is, $\mathrm{CC}(G)$ consist of the maximally connected induced subgraphs of G, which form a disjoint union of G.

If it is clear from the context, we might omit explicitly mentioning the graph G in the above notations.

Vertex sets. Let G be a graph. A vertex set $S \subseteq V(G)$ is called

independent set if $\forall v, w \in S : v \neq w \Rightarrow \{v, w\} \notin E(G)$ [GJ79];

clique if $\forall v, w \in S : v \neq w \Rightarrow \{v, w\} \in E(G)$ [GJ79].

Graph parameters. Let G be a graph, we denote by

$\delta(G)$ the minimum degree of a vertex in G, more formally, $\delta(G) = \min_{v \in V(G)} d(v)$;

$\Delta(G)$ the maximum degree of a vertex in G, more formally, $\Delta(G) = \max_{v \in V(G)} d(v)$;

$\mathrm{diam}(G)$ the *diameter* of G, more formally, $\mathrm{diam}(G) = \max_{(v,w) \in V \times V} \mathrm{dist}_G(v, w)$;

$\mathrm{radius}(G)$ the *radius* of G, more formally, $\mathrm{radius}(G) = \min_{v \in V(G)} \max_{w \in V(G)} \mathrm{dist}_G(v, w)$;

$\mathrm{degeneracy}(G)$ the *degeneracy* of G [NM12], which is defined in terms of vertex orderings as

$$\mathrm{degeneracy}(G) = \min_{L \in \Pi(G)} \max_{v \in V(G)} |\{w \in N_G(v) \mid w \prec_L v\}|.$$

We call orderings L in this definition *degeneracy orderings* and define the degeneracy of a degeneracy ordering L as $\max_{v \in V(G)} |\{w \in N_G(v) \mid w \prec_L v\}|$; the degeneracy is a measure of the sparsity of a graph and can be computed in linear time [MB83].

$\mathrm{td}(G)$ the *treedepth* of G [NM12]. Formally, the treedepth of G can be defined as

$$\mathrm{td}(G) = \begin{cases} 1, & \text{if } |V(G)| = 1 \\ 1 + \min_{v \in V(G)} \mathrm{td}(G - v) & \text{if } G \text{ is connected and } |V(G)| > 1 \\ \max_{G' \in \mathrm{CC}(G)} \mathrm{td}(G') & \text{otherwise} \end{cases}$$

where $\mathrm{CC}(G)$ are the connected components of G. There are further equivalent definitions, and the treedepth is also a measure of sparsity of a graph.

Search trees. *Branching algorithms* for a specific problem often induce *search trees* with a number of *search tree nodes*. Each search tree node contains a subproblem of the original problem and its children make this subproblem more restricted by making a choice for the subproblem. Leaves of the search tree constitute solutions to the original problem of the root node.

2.1 Preliminary Problem Definitions

We define some well known problems here. Problems that are thoroughly discussed in the thesis will be introduced in later chapters.

Independent Set (IS) [GJ79]

Input: A graph $G = (V, E)$ and an integer p.
Problem: Does G have an independent set of size at least p?

3-SAT [GJ79]

Input: A set Var of variables and a set C of clauses such that

- $\mathsf{cl} \subseteq \mathsf{Var} \cup \{\neg v \mid v \in \mathsf{Var}\}$ for all $\mathsf{cl} \in C$, and
- $|\mathsf{cl}| = 3$ for all $\mathsf{cl} \in C$.

Problem: Is there a variable assignment $A : \mathsf{Var} \to \{0, 1\}$ such that

- $A(\ell) = 1$ for at least one $\ell \in \mathsf{cl}$ for each clause $\mathsf{cl} \in C$,

where $A(\neg v) = 1 - A(v)$ for each $v \in \mathsf{Var}$? Elements v and $\neg v$ for $v \in \mathsf{Var}$ are called *literals*. The required assignment A is called a *satisfying* assignment.

2.2 Parameterized Complexity

Parameterized complexity studies algorithmic problems parameterized by a specific parameter arising in the problem itself or its problem input. One of the standard textbooks is by Cygan et al. [Cyg+15], from which we will be citing the most important definitions. The interested reader is encouraged to study this book for more elaborate explanations, examples and more involved concepts and problem-solving techniques used in parameterized complexity.

Parameterized problems. Let Σ be a fixed, finite alphabet and Σ^* be the set of finite words over Σ. In standard complexity theory we are studying *decision problems* $L \subseteq \Sigma^*$. We say $x \in \Sigma^*$ is a *YES*-instance if $x \in L$, otherwise it is a no-instance. In parameterized complexity we extend problems by parameters.

Definition 2.2.1 (Parameterized Problem [Cyg+15])**.** A *parameterized problem* is a language $L \subseteq \Sigma^* \times \mathbb{N}$. For an instance $(x, k) \in \Sigma^* \times \mathbb{N}$, k is called the *parameter*.

There are many examples for such problems, we will discuss some parameterized problems specific to this thesis topic in later chapters. The size of an instance (x, k) of a parameterized problem is defined as $|x|+k$. This definition is also easily extendable to problems, where instead of one parameter, we have multiple parameters. For an instance (x, k, ℓ) to a parameterized problem with multiple parameters, say k and ℓ are two parameters, we set the instance size to be $|x|+k+\ell$ and the parameter size to $k+\ell$.

Parameterized complexity classes. The most fundamental complexity class in parameterized complexity is FPT.

Definition 2.2.2 (Fixed-Parameter Tractability [Cyg+15])**.** A parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}$ is called *fixed-parameter tractable* (FPT) if there exists an algorithm $\mathcal{A}$ (called a *fixed-parameter algorithm*), a computable function $f : \mathbb{N} \to \mathbb{N}$, and a constant c such that, given $(x, k) \in \Sigma^* \times \mathbb{N}$, the algorithm $\mathcal{A}$ correctly decides whether $(x, k) \in L$ in time bounded by $f(k) \cdot |(x, k)|^c$. The complexity class containing all fixed-parameter tractable problems is called FPT.

Another way to state fixed-parameter tractability is that it can be decided in time $f(k) \cdot |(x, k)|^{\mathcal{O}(1)}$ whether an instance (x, k) belongs to L, where $f : \mathbb{N} \to \mathbb{N}$ is a computable function. W.l.o.g. we can assume that f is non-decreasing, as otherwise we can simply set $f'(n) = \max_{i=1}^{n} f(k)$ for $n \in \mathbb{N}$ and receive another computable function f'. There is a vast landscape of literature studying problems from multiple different domains with regard to their parameterizations by a wide variety of parameters. Of course, the quality of the algorithm $\mathcal{A}$ also heavily depends on the function f. If this function grows rapidly with k, $\mathcal{A}$ will not be applicable in practice. In parameterized complexity we often strive to find parameters that are small in practice, and then design FPT-algorithms for problems parameterized by these parameters. If the function f of a designed parameterized algorithm does not grow too fast, say 2^k, and the parameter is

small in practice, say $k \leq 20$ we are left with a fast algorithm. This approach has been applied successfully throughout many works in the literature, designing fast algorithms for otherwise hard problems. Hence, in practice there are classes of instances that admit small parameters, that in turn can be used to design FPT-algorithms.

Another well-known complexity class is XP.

Definition 2.2.3 (Slice-Wise Polynomial [Cyg+15])**.** A parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}$ is called *slice-wise* polynomial (XP) if there exists an algorithm $\mathcal{A}$ and two computable functions $f, g : \mathbb{N} \to \mathbb{N}$ such that, given $(x, k) \in \Sigma^* \times \mathbb{N}$, the algorithm $\mathcal{A}$ correctly decides whether $(x, k) \in L$ in time bounded by $f(k) \cdot |(x, k)|^{g(k)}$. The complexity class containing all slice-wise polynomial problems is called XP.

For the same reason as before we can assume that f and g are non-decreasing. Notice that FPT $\subseteq$ XP holds. Furthermore, a whole hierarchy of complexity classes FPT $\subseteq$ W[1] $\subseteq$ W[2] $\subseteq \ldots \subseteq$ W[P] $\subseteq$ XP has been established. We omit the definitions of complexity classes in the W-hierarchy as they are rather complex, but they can be found in the textbook [Cyg+15]. It is widely believed that FPT $\neq$ W[1]. This leads to the question: How to show that a parameterized problem is not in FPT, i.e. is fixed-parameter intractable?

Showing fixed-parameter intractability. Similarly, as one would show NP-hardness of a problem by a polynomial reduction, parameterized complexity makes use of similar problem reductions.

Definition 2.2.4 (Parameterized Reduction [Cyg+15])**.** Let $A, B \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized problems. A *parameterized reduction* from A to B is an algorithm that, given an instance (x, k) of A, outputs an instance (x', k') of B such that

1. (x, k) is a *YES*-instance if and only if (x', k') is a *YES*-instance of B,
2. $k' \leq g(k)$ for some computable function g, and
3. the running time is $f(k) \cdot |x|^{\mathcal{O}(1)}$ for some computable function f.

Again, we can assume that f and g are non-decreasing. The new instance size $|x'|$ is now not polynomially bounded by $|x|$, but by a polynomial in $|x|$ times a function of k.

The problems that are W[c]-hard for $c \geq 1$ are all the problems that admit a parameterized reduction to a W[c]-complete problem. Here, complete refers to complete in the sense of parameterized reductions. Furthermore, parameterized reduction can be used to show fixed-parameter intractability.

Theorem 2.2.5 ([Cyg+15])**.** *If there is a parameterized reduction from A to B and B is* FPT*, then A is* FPT *as well.*

Corollary 2.2.6. *If there is a parameterized reduction from A to B and A is not* FPT*, then B is not* FPT *as well.*

This leads to the standard procedure of proving fixed parameter-intractability. If we want to show fixed parameter intractability of a problem B, we pick a problem A which is not FPT (possibly W[1]- or W[2]-hard) and prove a parameterized reduction from A to B. Of course this only works under the assumption that FPT $\neq$ W[1] (FPT $\neq$ W[2]).

CHAPTER 3

Generalized Coloring Numbers and Structural Sparsity

In this chapter, we will introduce the concept of weak and strong r-coloring numbers and present some interesting results and relations with other graph parameters. Furthermore, we will define the problems we want to solve and give known complexity results that we know of.

3.1 Weak and Strong Coloring Numbers

Let us now define the concept of weak and strong coloring numbers, which were introduced by Kierstead and Yang [KY03] in the context of coloring games and marking games on graphs.

Definition 3.1.1 (Weak Coloring Number [KY03])**.** Let G be a graph, $r \in \mathbb{N}$, L be an ordering of vertices $V(G)$, and $u, v \in V(G)$. Then u is *weakly r-reachable* from v with respect to L if $u \preceq_L v$ and there is a path P of length ℓ with $0 \leq \ell \leq r$ between u and v such that

$$\forall w \in V(P) : u \preceq_L w.$$

Let $\mathrm{Wreach}_r(G, L, v)$ be the set of vertices that are weakly r-reachable from v in G with respect to L. Furthermore, let $\mathrm{Wreach}_r^{-1}(G, L, v) = \{u \in V : v \in \mathrm{Wreach}_r(G, L, u)\}$. We say that $\mathrm{Wreach}_r(G, L, v)$ is the *weakly r-reachable set* of v w.r.t. L, and $\mathrm{Wreach}_r^{-1}(G, L, v)$ is the *weakly r-reaching set* of v w.r.t. L. The *weak r-coloring number* $\mathrm{wcol}_r(G, L)$ of L is

$$\mathrm{wcol}_r(G, L) = \max_{v \in V(G)} |\mathrm{Wreach}_r(G, L, v)|,$$

and the weak r-coloring number $\mathrm{wcol}_r(G)$ of G is

$$\mathrm{wcol}_r(G) = \min_{L \in \Pi(G)} \mathrm{wcol}_r(G, L).$$

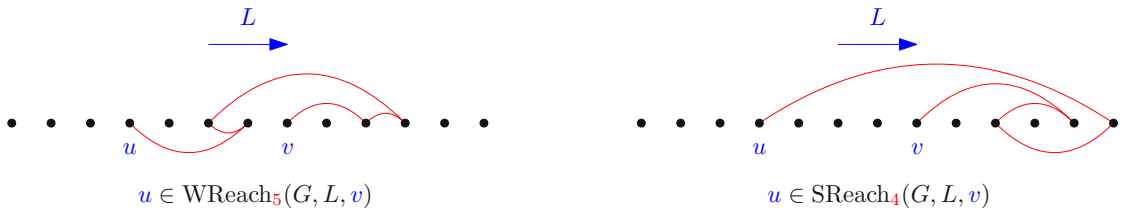


Figure 3.1: Visualization of weak and strong r-reachability.

Definition 3.1.2 (Strong Coloring Number [KY03])**.** Let G be a graph, $r \in \mathbb{N}$, L be an ordering of vertices $V(G)$, and $u, v \in V(G)$. Then u is *strongly r-reachable* from v with respect to L if $u \preceq_L v$ and there is a path P of length ℓ with $0 \leq \ell \leq r$ between u and v such that

$$\forall w \in V(P) \setminus \{u, v\} : v \preceq w.$$

Let $\mathrm{Sreach}_r(G, L, v)$, called *strongly r-reachable* set of v w.r.t. L, be the set of vertices that are strongly r-reachable from v in G with respect to L. The *strong r-coloring number* $\mathrm{scol}_r(G)$ of G is

$$\mathrm{scol}_r(G) = \min_{L \in \Pi(G)} \max_{v \in V(G)} |\mathrm{Sreach}_r(G, L, v)|$$

An example for both concepts is given in Figure 3.1, where vertex u is weakly (resp. strongly) r-reachable from v by a path of length 5 (resp. 4).

Clearly, if a vertex u is strongly r-reachable from a vertex v, then it is also weakly r-reachable. It follows that

$$\mathrm{scol}_r(G) \leq \mathrm{wcol}_r(G)$$

for all $r \in \mathbb{N}$. We also have that

$$\mathrm{wcol}_1(G) = \mathrm{scol}_1(G) = \mathrm{degeneracy}(G) + 1.$$

Furthermore, by Definition 3.1.1 and Definition 3.1.2 we also observe that

$$\mathrm{wcol}_1(G) \leq \mathrm{wcol}_2(G) \leq \ldots \leq \mathrm{wcol}_{n_G}(G)$$

and

$$\mathrm{scol}_1(G) \leq \mathrm{scol}_2(G) \leq \ldots \leq \mathrm{scol}_{n_G}(G).$$

It has been shown that strong and weak coloring numbers converge to well-known and already thoroughly studied graph parameters.

Lemma 3.1.3 ([NM12, Lemma 6.5])**.** *Let G be a graph. Then*

$$\mathrm{wcol}_{n_G}(G) = \mathrm{wcol}_\infty(G) = \mathrm{td}(G).$$

The value $\mathrm{td}(G)$ refers to the treedepth of G.

Lemma 3.1.4. *Let G be a graph. Then*

$$\mathrm{scol}_{n_G}(G) = \mathrm{scol}_\infty(G) = \mathrm{tw}(G) + 1.$$

The value $\mathrm{tw}(G)$ refers to the treewidth of G. A proof can be found in [Gro+18]. Combining Lemma 3.1.3, Lemma 3.1.4, and the observation that $\mathrm{scol}_r(G) \leq \mathrm{wcol}_r(G)$, we obtain an alternative proof for $\mathrm{tw}(G) \leq \mathrm{td}(G)$. Furthermore, Kierstead and Yang [KY03] showed that $\mathrm{wcol}_r(G) \leq \mathrm{scol}_r(G)^r$ for graphs G, meaning that there is a strong relation between weak and strong coloring numbers of graphs.

Multiple papers have studied different upper and lower bounds for weak r-coloring numbers. To give some examples we extend here a list by Joret and Micek [JM21] in Table 3.2.

3.2 Relation to Structural Sparsity

To further motivate strong and weak r-coloring numbers we want to state their relation to nowhere dense graph classes and graph classes of bounded expansion. Nowhere dense graph classes and graph classes of bounded expansion were introduced by Nesetril and Ossona de Mendez [NM08a; NM08b; NM08c; NM11; NM12] in the context of structural sparsity and are defined in terms of *bounded-depth minors*. We use the definitions of Grohe et al. [Gro+18] and Nadara et al. [Nad+19], as they are rather compact and understandable. A graph H is a minor of a graph G, if there are pairwise vertex-disjoint connected subgraphs $H_1, \ldots, H_n$ of G such that whenever $\{v_i, v_j\} \in E(H)$, there are $u_i \in V(H_i)$ and $u_j \in V(H_j)$ with $\{u_i, u_j\} \in E(G)$. We then call $(H_1, \ldots, H_n)$ a *minor model* of G. Let $r \in \mathbb{N}$. Then H is a *depth-r minor* of G if there is a minor model $(H_1, \ldots, H_n)$ of H in G such that each H_i has radius at most r. The *density* of a graph G is $|E(G)|/|V(G)|$. We are now ready to define nowhere dense graph classes and graph classes of bounded expansion.

Definition 3.2.1. A class $\mathscr{C}$ of graphs has *bounded expansion* if there exists a function $f : \mathbb{N} \to \mathbb{N}$ such that for every radius $r \in \mathbb{N}$ the density of depth-r minors in graphs from $\mathscr{C}$ is bounded by $f(r)$.

Definition 3.2.2. A class $\mathscr{C}$ of graphs is *nowhere dense* if there exists a function $t : \mathbb{N} \to \mathbb{N}$ such that for every radius $r \in \mathbb{N}$ the graphs from $\mathscr{C}$ exclude the complete graph $K_{t(r)}$ as depth-r minor.

With these definitions it is easy to see that graph classes of bounded expansion are also nowhere dense. Furthermore, it has been demonstrated that nowhere dense graph classes have nice algorithmic properties [DK09; Eic+17; GKS17; KRS19]. Interestingly, Zhu [Zhu09] proved that both classes can also be characterized by weak coloring numbers.

Theorem 3.2.3 ([Zhu09])**.** *A class $\mathscr{C}$ of graphs has bounded expansion if and only if there exists a function $f : \mathbb{N} \to \mathbb{N}$ such that $\mathrm{wcol}_r(G) \leq f(r)$ for all $r \in \mathbb{N}$ and all $G \in \mathscr{C}$.*

Theorem 3.2.4 ([Zhu09])**.** *A class $\mathscr{C}$ of graphs is nowhere dense if and only if there is a function $f : \mathbb{N} \times \mathbb{R} \to \mathbb{N}$ such that for every real $\varepsilon > 0$ and every $r \in \mathbb{N}$ and all n-vertex graphs H that are subgraphs of some $G \in \mathscr{C}$, we have $\mathrm{wcol}_r(H) \leq f(r, \varepsilon) \cdot n^{\varepsilon}$.*

Table 3.2: Upper and lower bounds on weak r-coloring numbers. The subscripts in the asymptotic notation mean that the hidden constant may depend on the parameter in the subscript. For full definitions of the graph classes we refer to the respective references.

graph class $\mathcal{C}$	bounds on $\mathrm{wcol}_r(G)$ over all graphs $G \in \mathcal{C}$	
outerplanar	$\Theta(r \log r)$	Joret and Micek [JM21]
planar	$\Omega(r^2)$ and $\mathcal{O}(r^3)$	van den Heuvel et al. [Heu+17]
	$\Omega(r^2 \log r)$	Joret and Micek [JM21]
Euler genus g	$\mathcal{O}_g(r^3)$	van den Heuvel et al. [Heu+17]
$K_{3,k}$-minor free	$\mathcal{O}_k(r^3)$	van den Heuvel and Wood [HW18]
$K_{s,k}$-minor free	$\mathcal{O}_k(r^{s+1})$	van den Heuvel and Wood [HW18]
treewidth $\leq k$	at most $\binom{r+k}{k}$	Grohe et al. [Gro+18]
simple treewidth $\leq k$	$\Omega_k(r^{k-1} \log r)$ and $\mathcal{O}(r^{k-1} \log r)$	Joret and Micek [JM21]
K_k-minor free	$\Omega_k(r^{k-2})$ and $\mathcal{O}(r^{k-1})$	Grohe et al. [Gro+18], van den Heuvel et al. [Heu+17]
max. degree $\leq \Delta$ for $\Delta \geq 4$	$\Omega\left(\left(\frac{\Delta-1}{2}\right)^r\right)$	Grohe et al. [Gro+18]
paths	at most $\lceil \log r \rceil + 2$	Joret and Micek [JM21]
paths on at least $2r-2$ vertices	at least $\lceil \log r \rceil + 1$	Joret and Micek [JM21]

Nowhere dense graph classes generalize and include many other sparse graph classes, such as graph classes of bounded degree, classes of bounded genus and classes excluding

(topological) minors. Furthermore, they admit surprisingly different but natural characterizations, meaning that being nowhere dense is a rather robust property. We have already mentioned that nowhere dense graph classes and graphs of bounded expansion have nice algorithmic properties in theory. Aforementioned reasons resulted in nowhere dense graphs and graphs of bounded expansion gaining more popularity and further motivate us to construct algorithms to compute upper bounds of weak coloring numbers.

3.3 Problem Definitions and Complexity

In this section, we define the main problems with regard to weak coloring numbers and discuss known complexity bounds. First, let us define the main decision problem WCOL(r), where we want to know if a graph has weak r-coloring number of at most k.

Weak r-coloring Number (WCOL(r))

Input: A graph $G = (V, E)$, and an integer k.
Problem: Does G have weak r-coloring number $\leq k$?

In fact, we define a problem WCOL(r) for each $r \geq 1$. As $\text{wcol}_1(G) = \text{degeneracy}(G)+1$, we know that WCOL(1) is solvable in polynomial time due to a well known greedy algorithm for the degeneracy of a graph [MB83]. It was then shown by Grohe et al. that this is not the case for $r \geq 3$ (Unless P = NP).

Theorem 3.3.1 ([Gro+18])**.** *Deciding WCOL(r) for $r \geq 3$ is NP-complete.*

Grohe et al. have shown Theorem 3.3.1 by giving a reduction from the balanced bipartite clique problem [GJ79]. Of course, we not only want to decide if $\text{wcol}_r(G) \leq k$ for a graph G, but we also want to compute an ordering L with $\text{wcol}_r(G, L) \leq k$. Therefore, we define the problem WCOLORD(r).

Weak r-coloring Number Ordering (WCOLORD(r))

Input: A graph $G = (V, E)$, and an integer k.
Problem: Compute an ordering $L \in \Pi(G)$ such that $\text{wcol}_r(G, L) \leq k$ or report that there exists no such ordering.

By the same greedy algorithm for degeneracy as mentioned above, WCOLORD(1) is solvable in polynomial time. Furthermore, by solving WCOLORD(r) we also solve WCOL(r), hence WCOLORD(r) is also NP-complete for each $r \geq 3$. To our knowledge, it is still unknown whether WCOL(r) and WCOLORD(r) are NP-complete for $r = 2$.

In this thesis, we want to solve the problem WCOLORD(r) using the technique of turbocharging (see Chapter 4). Clearly, we will not be able to give exact algorithms that work efficiently for all instances as the problem is NP-complete for $r \geq 3$ (unless P = NP). But we will exploit properties of real-world graph instances that allow us to design algorithms for finding orderings of small weak r-coloring number. Such properties will for example include parameters that we will use for parameterized algorithms.

CHAPTER 4

Turbocharging and Known Heuristics for Weak Coloring Numbers

In this chapter, we introduce the concept of turbocharging, and give some background and related research. Next, we will relate turbocharging to computing orderings with small weak r-coloring numbers. As turbocharging is closely related to greedy heuristic approaches, we will also present some already known heuristics for computing orderings of small weak r-coloring numbers.

4.1 Turbocharging

Turbocharging framework. We want to start by explaining turbocharging in a rather generic fashion, later relating it to some problems it has been applied to. But we have to note that turbocharging, as we explain it here, is adapted to the problems we are dealing with. In the literature turbocharging may simply be known as "augmenting heuristics by exact methods".

Let P be a *problem* and I be an *instance* of this problem. *Solutions* of this instance w.r.t. to the problem all have size $n \in \mathbb{N}$, but there are also incomplete solutions of size less than n. Additionally, each solution S has a quality $q(S) \in \mathbb{N}$. We also know that extending an incomplete solution S of size $m < n$ creates a new solution S' of size $k' > k$. Let us call an extension S' of S *full extension*, if the size of S' is n. It is now possible that an incomplete solution tells us something about the quality of its full extensions. If we know that $q(S') \leq k$ for any full extension S' of an incomplete solution S, we set $q(S) = k$. In other words, the value $q(S)$ is an upper bound for the quality of any full extension. Additionally, $q(S') \leq q(S)$ for any extension S' of S. Lastly, let H be an

Algorithm 4.1: Turbocharging framework

Input: Instance I of problem P and $c \in \mathbb{N}$
Output: Solution S with quality at least k or **false**

```
1 S ← ∅;
2 while size(S) < n do
3 |   S ← H(S);
4 |   if q(S) < k then
5 |   |   Try applying turbocharging algorithm TC(S, c) that changes c values of S
  |   |     such that q(S) ≥ k or return false if it fails;
6 |   end
7 end
8 return S;
```

iterative greedy heuristic that has as its input an incomplete solution S of size k and as its output an extension S' of S with size $k+1$, meaning $H(S) = S'$.

With these definitions out of the way, we are ready to explain turbocharging. A framework for turbocharging is illustrated in Algorithm 4.1. The framework takes as input an instance I of a problem P that has solutions of size n, and outputs a solution of quality at least k, or false if no such solution could be found. It is important to note that if the framework returns false, it does not mean that there does not exist a solution of quality at least k. Starting from the empty incomplete solution, Algorithm 4.1 iteratively applies the heuristic H, that creates a new (in)complete solution, whose size is larger. It might be that at some point S is an incomplete solution that cannot be extended to a solution that has quality at least k (Line 4). At this point we apply *turbocharging*; that is, we apply a *turbocharging algorithm* TC to S, that tries to "fix" S, such that $q(S) \geq k$. In other words, TC tries to find a solution to a *turbocharging problem*. The second input to TC is the *conservation parameter* c, that says that the solution S may be changed at up to c positions. It can also be the case, that a specific part of S of size c may be changed. This approach is promising if k and c are small and TC is an exact algorithm that admits a fixed-parameter runtime — meaning that TC can be computed in time $f(k, c) \cdot |I|$ for a computable function f.

We also want to mention the strong relation of turbocharging to a problem-solving technique from fixed-parameter complexity named iterative compression [DF95]. In iterative compression we add in one step an element such as a vertex to a problem. A solution that was known prior to the addition of that element is used to find a solution for the larger problem. Similarly, in turbocharging we also add one element in each step to a solution and a turbocharging algorithm might be used to "fix this solution" if it does not fit quality requirements.

Let us now present some known papers applying turbocharging and relate its studied problems to the turbocharging framework as we presented it.

Research on turbocharging. The first paper we want to present is by Hartung and Niedermeier [HN13], who study the list coloring problem w.r.t. turbocharging. Although they call it *parameterized by conservation*, their paper fits the spirit of turbocharging as we explained it here, and it is also the first paper that we know of that presents this approach. In the list coloring problem we are given a graph $G = (V, E)$, and for each vertex $v \in V$ a set of colors $L(v) \subseteq \{1, \ldots, k\}$. The task is to find a coloring $f : V \to \cup_{v \in V} L(v)$ such that $f(v) \in L(v)$ for each $v \in V$ and $f(v) \neq f(w)$ for all $\{v, w\} \in E$. Such colorings are called k-list colorings. They propose the following turbocharging problem.

Incremental Conservative k-List Coloring (IC k-List Coloring) [HN13]

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

They also show that IC k-List Coloring can be solved in $\mathcal{O}(k \cdot (k-1)^c) \cdot (|V|+|E|))$ time, which is FPT-time parameterized by k and c. It is not immediate how this turbocharging problem could be used inside a turbocharging framework, so let us explain. Incomplete solutions S for the list coloring problems could be colorings f of graphs $G[V']$ where $V' \subseteq V$ and $f(v) \in L(v)$ for all $v \in V'$. The size of S is $|V'|$, and we want to find a solution of size $|V|$. The quality of a (incomplete) solution is hidden in the constraint that $f(v) \neq f(w)$ for $\{v, w\} \in E$. Clearly, an incomplete solution cannot be extended if $f(v) = f(w)$ for some $\{v, w\} \in E$. A heuristic may color one vertex after the other until a vertex x cannot be assigned a color that none of its neighbors is assigned to. At this point the IC k-List Coloring algorithm may be applied to fix the coloring, such that it can be extended and x is also assigned a color. The conservation parameter c demonstrates how to model that an incomplete solution may be changed at up to c positions. Notice that without the conservation parameter IC k-List Coloring is very similar to an iterative compression problem; that is, if IC k-List Coloring were fixed-parameter tractable parameterized by only k, we would prove that the k-list coloring problem is fixed-parameter tractable parameterized by k; this being only a hypothetical thought, as IC k-List Coloring is not FPT parameterized by k [HN13, Theorem 2].

Another problem where turbocharging has been applied to is treewidth. We chose this problem as it is very similar to weak r-coloring numbers because it also involves vertex orderings. Gaspers et al. [Gas+19] proposed a turbocharging problem for treewidth and also used it in combination with well-known heuristics to conduct experiments on real-world graph instances. The treewidth of a graph G can be characterized by so-called *elimination orders* which are linear orders of vertices of G, and normally the treewidth of such computed orders has to be minimized. Furthermore, *partial elimination orderings*, that are linear orders of a set of vertices $V' \subseteq V$, also have a so-called width that is a lower bound to the treewidth of any extension of that partial elimination order. In this case an extension of a partial elimination order π of vertices V' is another partial elimination order π' of vertices $V'' \supseteq V'$ that agrees with π on the first $|V'|$ positions.

The quality k of a partial elimination order is its width, where lower widths symbolize "better" quality. Gaspers et al. proposed the following turbocharging problem.

IC Treewidth [Gas+19]

Input:	A graph G, a non-negative integers k and c, and a partial elimination order π of length $\ell \geq c$ and width $\leq k$.
Question:	Does there exist a partial elimination order π' of length $\ell+1$ and width $\leq k$ such that π and π' are identical on the first $\ell - c$ positions?

Given a partial elimination order π of length ℓ, we have to find a partial elimination order π' of length $\ell + 1$, and we are only allowed to change the suffix of length c. Notice that in this case the conservation parameter restricts the part of the incomplete solution we are allowed to change. Gaspers et al. proved that IC Treewidth is fixed-parameter tractable when parameterized by $c + k$. Additionally, they turbocharge some well-known heuristics for building elimination orders that iteratively place vertices from left to right into a partial elimination order π. Whenever the heuristic would place a vertex to the right of π such that the width of π would exceed k, they apply their algorithm for IC Treewidth. They then applied the turbocharged heuristics to a variety of graph instances. To summarize their results, for most of their considered instances, the turbocharged versions of the heuristics obtained orderings of better treewidth than only the heuristics themselves.

4.2 Turbocharging and Weak Coloring Numbers

We want to start our research on weak r-coloring numbers w.r.t. turbocharging. As already mentioned we will design algorithms that compute upper bounds of weak r-coloring numbers by finding orderings L of vertices V such that $\mathrm{wcol}_r(G, L)$ is as small as possible. Our method of choice is turbocharging, so let us first relate notions of turbocharging to orderings and their weak r-coloring numbers. Let us start by answering the question, how to define an incomplete solution for orderings of vertices. Clearly, the first thing that comes to mind are orderings of some subset $S \subseteq V$ of vertices of a graph $G = (V, E)$; we want to capture this in the following definition.

Definition 4.2.1. Let $G = (V, E)$ be a graph. A *subordering* L is an ordering of vertices $S \subseteq V$. We call $T = V \setminus S$ the set of *free* vertices.

Throughout the rest of the thesis, suborderings L of vertices $S \subseteq V$ will refer to incomplete solutions of the WCOLORD(r) problem. Naturally, we define the size of a subordering as $|S|$, and "full" solutions have size $|V|$. It is now clear how to define extensions. Let L be a subordering of vertices S; an extension L' of L is a subordering of vertices $S' \supsetneq S$ such that L' agrees with L on the relative order of S, that is, $L'[S] = L$. Lastly, we need to define the quality of a subordering L, that is, a lower bound on the weak r-coloring number of any full extension L' of L. From our definition of extensions,

it follows that a natural definition for the quality $q(L)$ of a subordering of vertices $S \subseteq V$ is

$$q(L) = \max_{v \in S} |\text{Wreach}_r(G[S], L, v)|,$$

as for any extension L' of L we have that for $v \in S$, $\text{Wreach}_r(G[S], L, v) \subseteq \text{Wreach}_r(G[S'], L', v)$, where L' is a subordering of vertices $S' \supsetneq S$. In Chapter 5 we will see a better lower bound than $q(L)$, under the assumption that we know that all free vertices will be placed to the right of L in a full extension of L.

4.3 Known Heuristics for Weak Coloring Numbers

As we mentioned before, a key component for turbocharging is an iterative greedy heuristic. In this thesis we will not design new heuristics for weak coloring numbers, but we will instead use some heuristics that were introduced by Nadara et al. [Nad+19], that are easy to implement, and that achieved relatively good results. These heuristics were already applied to a wide variety of graph instances from different domains, which gives us a set of benchmarking instances for our implementations, and we can also compare our results with those achieved by Nadara et al.

The heuristics that we will be using for our implementations can be classified into two types:

1. Heuristics that iteratively construct an ordering from left to right; that is the heuristic starts with the empty ordering and, in one step, places one vertex to the right of that ordering.

2. Heuristics that iteratively construct an ordering from right to left; that is the heuristic starts with the empty ordering and, in one step, places one vertex to the left of that ordering.

Let us briefly describe said heuristics, classify them into the two types, and give them identifiers.

Left-to-right heuristics.

- *Wreach-Heuristic*: Let L be a subordering of vertices $S \subseteq V$. A crucial observation is that for each free vertex v we already know the weakly r-reachable set w.r.t. S for any potential full extension of L that places all free vertices to the right of L. This is formalized in more detail in the next chapter (Definition Definition 5.1.2, Lemma 5.1.3). Furthermore, if v were to be placed directly to the right of L, then its weakly r-reachable set in any full extension where free vertices are placed to the right would correspond exactly to that said set. The Wreach-Heuristic uses this fact and makes a locally optimal choice for L by placing the free vertex that has the largest weakly r-reachable set w.r.t. S to the right of L, extending L by one vertex. Ties are broken by choosing vertices with larger degrees first.

- *Degree-Heuristic*: The Degree-Heuristic orders vertices by descending degree; that is, for a subordering L, in one step, it places the free vertex with maximum degree in G to the right of L.

Notice that the Wreach-Heuristic gives different orderings for different values of radii r, while the Degree-Heuristic does not.

Right-to-left heuristics.

- *Sreach-Heuristic*: Let L be a subordering of vertices S with free vertices T, and we know that vertices from T will be placed left of L. This heuristic is based on strongly reachable sets. A key observation is that strongly r-reachable sets for vertices in S are already fixed when considering any extension of L where free vertices will be placed to the left. Not only that, but if we know that a vertex from $v \in T$ will be placed directly to the left of L, then we also know the strongly r-reachable set of v, we call this set *potentially strongly r-reachable set*, adopting the term from Nadara et al. The Sreach-Heuristic now places the vertex $u \in T$ to the left of L that has smallest potentially strongly r-reachable set. This makes sense, as the strongly reachable sets are subsets of weakly reachable sets, and we want an ordering where the maximum size of a weakly r-reachable set is as small as possible.

- *Degree-Heuristic*: The Degree-Heuristic can also be seen as a right-to-left heuristic. That is, for a subordering L, in one step the heuristic places the vertex of minimum degree in G to the left of L.

Nadara et al. also proposed further heuristics that match the concept of right-to-left heuristics such as heuristics based on treewidth and treedepth. We do not consider them in this thesis due to time restrictions and their higher complexity when compared to the Sreach- and Degree-Heuristic. Furthermore, the heuristics we consider here, are the ones that achieved the smallest weak coloring numbers in the experiments conducted by Nadara et al.

In this thesis, we will mostly focus on left-to-right heuristics, as they are easier to work with and admit easy implementations. But we will also investigate a simple turbocharging problem that comes up for right-to-left heuristics.

We have to mention that all turbocharging approaches that we will propose will most likely be invoked multiple times during the computation of an ordering. That means, even though we achieve good running time bounds for a turbocharging problem, it may be that computing orderings of small weak r-coloring number with this approach might still be time-consuming for medium size to large size instances.

CHAPTER 5

Left-to-Right Heuristics and Turbocharging

In this chapter, we want to study turbocharging problems for left-to-right heuristics that compute weak coloring numbers. That is, we will present descriptions and complexity results for turbocharging problems that arise during the Wreach-Heuristic and the Degree-Heuristic. Unless stated otherwise, the radius r is a fixed constant throughout the chapter.

5.1 Definitions and First Observations

In this section, we will relate the turbocharging framework and its notions such as incomplete solutions and extensions to left-to-right heuristics. Let us first start by introducing some definitions and by discussing some observations that will be useful for turbocharging left-to-right heuristics. In Chapter 4 we have already discussed extensions of suborderings. But if we know that all free vertices will be placed to the right of a subordering, then we can adjust the definition for extensions accordingly.

Definition 5.1.1. Given a subordering L of vertices $S \subseteq V$, a *right extension* L' of L is a subordering of vertices S' such that

- $S' \supseteq S$,
- $L'[S] = L$, and
- $u \prec_{L'} v$ for all $u \in S, v \in S' \setminus S$.

Informally, L' is a subordering that places vertices $S' \setminus S$ to the right of the vertex set S. If $S' = V$, then we say that L' is a *full right extension*.

Let us relate right extensions to left-to-right heuristics: Given a subordering L of vertices S, in one step, a left-to-right heuristic computes a right extension L' of L with size $|S|+1$.

In the description for the Wreach-Heuristic we have already mentioned that weakly r-reachable sets w.r.t. S are fixed for any right extension L' of a subordering L which orders vertices S. We want to formalize this in the following lemma and definition.

Definition 5.1.2. Let G be a graph. Given a subordering L of vertices $S \subseteq V(G)$, a vertex $u \in V(G)$ is *weakly left r-reachable* from $v \in V(G)$ with respect to L if

- $v = u$, or
- $u \in S$ and there is a path P of length ℓ with $0 \leq \ell \leq r$ between u and v such that

$$\forall w \in V(P) \cap S : u \preceq_L w.$$

Let $\text{Wreachleft}_r(G, L, v)$ be the set of vertices that are weakly left r-reachable from v in G with respect to L. We say $\text{Wreachleft}_r(G, L, v)$ is the *weakly left r-reachable set* of v w.r.t. L. Let $\text{Wreachleft}_r^{-1}(G, L, v) = \{u \in V : v \in \text{Wreachleft}_r(G, L, u)\}$, calling this the *weakly left r-reaching set* of v.

Lemma 5.1.3. *Let L be a subordering of vertices $S \subseteq V$ with free vertices T and let L' be a right extension of L. Then*

- $\text{Wreachleft}_r(G, L, v) \subseteq \text{Wreachleft}_r(G, L', v)$ *for all $v \in T$, and*
- $\text{Wreachleft}_r(G, L, v) = \text{Wreachleft}_r(G, L', v)$ *for all $v \in S$.*

Proof. Observe that if u is weakly left r-reachable from v in L, then u is also weakly left r-reachable from v in L'. □

The above lemma gives an alternative description for the Wreach-Heuristic: Given a subordering L of vertices S, in one step, the Wreach-Heuristic places a free vertex v with maximum $|\text{Wreachleft}_r(G, L, v)|$ to the right of L, creating a right extension L' of size $|S|+1$. Due to the fact that we want to minimize weak r-coloring numbers, Lemma 5.1.3 gives a justification for this heuristic, as it places vertices with large weakly left r-reachable sets earlier, such that these sets cannot get larger anymore.

In fact, we know even more than what is stated in Lemma 5.1.3: Let L be a subordering of vertices S and free vertices T. By placing a vertex $v \in T$ to the right of L, we add v to the weakly left r-reachable sets of all vertices in $N^r_{G[T]}(v)$. This is useful for implementations for heuristics and turbocharging problems, when computing weakly left r-reachable sets of right extensions. Adding a free vertex to the right of a subordering L, we need a single breadth-first search of depth r in $G[T]$ to update weakly left r-reachable sets.

Weakly left r-reachable sets of suborderings L also give us a new lower bound for the weak r-coloring number of any full right extension of L, that can be seen as the

quality of a subordering. Namely, we know that the weak r-coloring of any full right extension is at least $\max_{v \in V} |\text{Wreachleft}_r(G, L, v)|$. When applying the turbocharging framework, we want to compute vertex orderings with weak r-coloring number of at most $k \in \mathbb{N}$. That means that we apply a left-to-right heuristic iteratively, until it made too many "bad choices", creating a subordering L such that $|\text{Wreachleft}_r(G, L, v)| > k$ for at least one vertex $v \in V$. Notice that in this case v is a free vertex. We say that vertices $v \in V$ with $|\text{Wreachleft}_r(G, L, v)| > k$ are *overfull*. Furthermore, we call such orderings *non-extendable*.

5.2 A First Natural Turbocharging Problem

Now that we have seen some observations and problems that come up in turbocharging left-to-right heuristics we want to introduce a first turbocharging problem. That means that we are given a non-extendable subordering L, and we want to fix this subordering such that it is extendable by making a limited number of modifications. Similarly to Gaspers et al. [Gas+19] who modified suffixes of partial elimination orderings, we want to change the suffix of L with size c. That is, we want to try replacing the rightmost c vertices of L by a (possibly) different set of vertices. For this, we define the problem IC-WCOL-LEFT(r) that already assumes that we have removed the last c vertices of a non-extendable subordering.

Incremental Conservative Left Weak r-coloring (IC-WCOL-LEFT(r))	
Input:	A graph $G = (V, E)$, a subordering L of vertices $S \subseteq V$, and positive integers k and c.
Problem:	Is there an extendable right extension L' of L of vertices S' with $\lvert S' \setminus S \rvert = c$.

Again, we in fact define a problem IC-WCOL-LEFT(r) for each $r \geq 1$. The motivation for this approach is that we assume that a heuristic made a "bad choice" during the last c placements of a vertex, so that replacing them by different vertices would achieve a subordering of higher quality. Assuming that k and c are small, we would like to find an FPT-algorithm for IC-WCOL-LEFT(r). But of course, the probability of fixing a non-extendable subordering grows with larger c. Still, under the assumption that we are working with a good heuristic that makes a small amount of mistakes, we would believe that we can fix a non-extendable subordering with little changes. Furthermore, we assume that the considered input graphs are sparse in the sense of structural sparsity, meaning that there are orderings of vertices that achieve small weak r-coloring numbers.

5.2.1 Complexity

The first main result of this thesis that we want to present is the NP-hardness of IC-WCOL-LEFT(r) for each $r \geq 1$, which gives first insights into the complexity of the problem.

Theorem 5.2.1. *IC-WCOL-LEFT(r) is NP-hard for each fixed $r \geq 1$.*

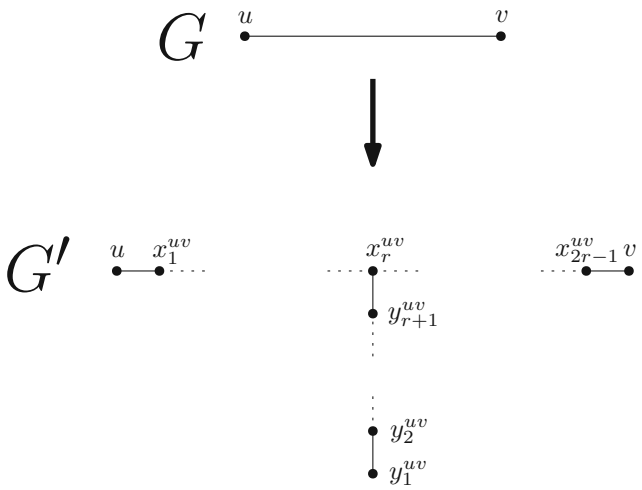


Figure 5.1: Sketch for subdivision of edge $\{u, v\}$.

Proof. We give a reduction from Independent Set which is NP-complete [GJ79]. Let (G, p) be an instance of Independent Set. We construct an instance (G', L, k, c) of IC-WCOL-LEFT(r) as follows. First, let $k = 2$ and $c = p$. Constructing G' from G proceeds in two steps.

- We subdivide each edge $\{u, v\} \in E(G)$ $2r-1$ times, introducing $2r-1$ new vertices. Let us call them $x_1^{uv}, \ldots, x_{2r-1}^{uv}$ ordered from u to v.
- For each edge $\{u, v\} \in E(G)$ we introduce a path $P = (y_1^{uv}, \ldots, y_{r+1}^{uv})$ of $r+1$ vertices connecting the end vertex y_{r+1}^{uv} to x_r^{uv}.

A sketch of the construction for a single edge $\{u, v\}$ is shown in Figure 5.1. Note that the length of the shortest path from u and v to any vertex y_i^{uv} with $1 \leq i \leq r+1$ is at least $r+1$. On the other hand, shortest paths from vertices x_i^{uv} with $1 \leq i \leq 2r-1$ to y_{r+1}^{uv} have length at most r. Lastly, we define the subordering L of vertices $S \subseteq V$ such that L is an arbitrary subordering of vertices $S = \{y_1^{uv} : \{u, v\} \in E(G)\}$. Note that $\text{Wreachleft}_r(G, L, y_i^{uv}) = \{y_1^{uv}, y_i^{uv}\}$ for $2 \leq i \leq r+1$. The remaining vertices only contain themselves in their weakly left r-reachable set. We claim the correctness of this reduction, that is, we proceed by showing that there is an independent set I of size p if and only if there is an extendable right extension L' of L of vertices S' with $|S' \setminus S| = c$.

"⇒": Assume that G has an independent set I of size $|I| = p = c$. Then any right extension L' of L of vertices $S' \supseteq S$ with $S' \setminus S = I$ is extendable and $|S' \setminus S| = c$. Observe that $|S' \setminus S| = c$ holds. Now we can look at values $|\text{Wreachleft}_r(G', L', v)|$ for vertices $v \in V(G')$.

- Clearly $\text{Wreachleft}_r(G', L', v) \subseteq \{v\}$ for $v \in V(G)$, as v can only weakly left r-reach itself in I due to the subdivision of edges. Thus, $|\text{Wreachleft}_r(G', L', v)| \leq k$ holds.

- As I is an independent set, we also have that $|\text{Wreachleft}_r(G', L', x_i^{uv})| \leq 2$ for all $\{u, v\} \in E(G)$ and $1 \leq i \leq 2r - 1$. This is true because u and v are the only potentially r-reachable vertices in I, but u and v cannot both be in I. Again $|\text{Wreachleft}_r(G', L', x_i^{uv})| \leq k$ holds.
- Finally, $\text{Wreachleft}_r(G', L', y_j^{uv}) = \{y_1^{uv}, y_j^{uv}\}$ for all $\{u, v\} \in E(G)$ and $1 \leq j \leq r+1$ holds as all paths to vertices $I \subseteq V(G)$ have length greater than r.

Summarizing this, we have

$$\forall v \in V(G') : |\text{Wreachleft}_r(G', L', v)| \leq k,$$

and we can conclude the forwards direction.

"$\Leftarrow$": Assume that we found an extendable right extension L' of L of vertices S' with $|S' \setminus S| = c$. We can observe that vertices y_i^{uv} with $2 \leq i \leq r + 1$ cannot be in $S' \setminus S$ because this would result in a too large set $\text{Wreachleft}_r(G', L', y_j^{uv})$ for some $2 \leq j \leq r+1$. Consequently, $S' \setminus S$ cannot contain any vertex x_i^{uv}, as otherwise y_{r+1}^{uv} could weakly left r-reach one of the vertices x_i^{uv} appearing left-most in L'. Putting this together we have $(S' \setminus S) \subseteq V(G)$ and define $I = (S' \setminus S)$. It remains to show that I is an independent set, meaning that we have $\{u, v\} \not\subseteq S$ for edges $\{u, v\} \in E(G)$. But this has to hold as otherwise x_r^{uv} could weakly left r-reach both u and v, meaning that

$$|\text{Wreachleft}_r(G', L', x_r^{uv})| = |\{x_r^{uv}, u, v\}| > k = 2,$$

which leads to a contradiction. Hence, the backwards direction holds as well.

Note that this reduction is polynomial under the assumption that r is a constant. Consequently, we can conclude the proof. □

We want to mention that we are not that satisfied with this reduction, as there is never an ordering of vertices of G' that achieves a weak r-coloring of 2, and this is easily detectable — we will never be able to place one of the vertices y_i^{uv} for $i \geq 2$ and $\{u, v\} \in E$ in any right extension of L. It would be nice to have a reduction that also reflects the hardness of the entire problem, including right extensions of larger size.

Nonetheless, we can copy the proof for Theorem 5.2.1, and show that IC-WCOL-LEFT(r) is W[1]-hard when parameterized by k and c, meaning that we cannot expect an FPT-algorithm for IC-WCOL-LEFT(r) parameterized by k and c unless FPT = W[1].

Theorem 5.2.2. *IC-WCOL-LEFT(r) parameterized by c is W[1]-hard for each fixed $r \geq 1$ and $k \geq 2$. In particular, the problem is W[1]-hard when parameterized by $k + c$ for each $r \geq 1$.*

Proof sketch. We observe that this is achieved by the same reduction as in the proof for Theorem 5.2.1 by adding further vertices. Namely, for $k > 2$ we add $k - 2$ vertices for each edge $\{u, v\}$ in the original graph and make them adjacent to y_2^{uv}. This will result in an offset for the size of weakly left r-reachable sets of vertices x_r^{uv}. Note that Independent Set is W[1]-hard parameterized by the size p of the independent set [DF95]. The parameter p of Independent Set is transformed to parameter $c = p$. For fixed k, the integer $k + c$ only depends on p, and we have a parameterized reduction. □

As we now know that k and c are not enough for designing an FPT-algorithm, we look for other parameters that often come up in graph theory. A well-known parameter is the maximum degree Δ of a graph. We want to present an FPT-algorithm that makes use of this parameter, showing fixed-parameter tractability of IC-WCOL-LEFT(r) parameterized by r, c and the maximum degree of a graph. Although the maximum degree of a graph is not always small in real-world graph instances, there are still instances where it might be small, and our presented algorithm might be useful. Furthermore, this results gives first insights for parameterizations by other parameters, as the maximum degree is an upper bound to a variety of other graph parameters.

Theorem 5.2.3. *IC-WCOL-LEFT(r) is solvable in time $\mathcal{O}((2c)^c \cdot \Delta^{3r \cdot c} \cdot n_G^{\mathcal{O}(1)})$. In particular, IC-WCOL-LEFT(r) is fixed-parameter tractable when parameterized by $c + r + \Delta$.*

Proof. We provide a search tree algorithm that runs in FPT-time and results in a *NO*-instance, a *YES*-instance, or an instance where an exponential algorithm will run in a time bounded by c, r, and Δ. Let (G, L, k, c) be an instance of IC-WCOL-LEFT(r) such that L is a subordering of vertices $S \subseteq V$ and G has maximum degree Δ. It follows immediately that $|N_r^G(v)| \leq \Delta^r$ and $|N_{2r}^G(v)| \leq \Delta^{2r}$. Notice that this also holds for every subgraph G' of G. We have to place c vertices of $V \setminus S$ to the right of L. Let us call these c positions right of L *slots* and let a slot be *free* if no vertex is placed into the slot. Let the set of slots be $[c]$. In a particular node in the search tree we will have that some slots are occupied by a vertex from $V \setminus S$ and some slots are free. Notice that the sets $\text{Wreachleft}_r^{-1}(G, L', v)$ cannot change for each $v \in S$ in any right extension L' of L by Lemma 5.1.3. Furthermore, $\text{Wreachleft}_r(G, L', v) = \text{Wreachleft}_r(G, L, v)$ for any right extension of L' of L and $v \in S$. Moreover, $\text{Wreachleft}_r(G, L', v) \supseteq \text{Wreachleft}_r(G, L, v)$ for any right extension of L' of L and $v \in V \setminus S$. As new paths resulting in the increase of a weakly left r-reachable set will never go through S we can forget about S completely during the algorithm, only tracking sizes of weakly left r-reachable sets of vertices $v \in V \setminus S$ w.r.t. L. Hence, let G' be the graph obtained from G by removing all vertices in S and let $f(v) = |\text{Wreachleft}_r(G, L, v)|$ for each $v \in V \setminus S$. To formalize all of this, we define the following more general problem.

IC-Left-Branch

Input: A graph G, a function $f : V(G) \to \mathbb{N} \cup \{0\}$, integers k and r, and an injective assignment $g : X \to V(G)$ with $X \subseteq [c]$.

Problem: Find an injective assignment $g' : [c] \to V(G)$ such that

- $g(s) = g'(s)$ for all slots $s \in X$ and
- $f(v) + \text{Wreachleft}_r(G, L, v) \leq k$ for each $v \in V(G)$,

where L is the subordering induced by the function g'. To be more precise, L is a subordering of the vertices $g'([c])$ with $u \preceq_L v$ if and only if $g^{-1}(u) \leq g^{-1}(v)$ for $u, v \in g'([c])$.
Otherwise, state that there is no such assignment.

In each node of the search tree we are given an instance of IC-Left-Branch that we want to solve. The problem instance of IC-Left-Branch corresponding to the root node of the search tree will have $g = \emptyset$ as every slot is free. Problem inputs f, k and r are defined as above, and the input graph is G'. We define Y as the set of vertices that are already placed into a slot, so $Y = g(X)$. The set of free slots is $[c] \setminus X$. In what follows, G refers to the graph corresponding to the problem IC-Left-Branch. We continue by providing the following branching rule that decides on placing a vertex into a free slot — creating a new assignment g' by extending the domain of g by one new slot.

Branching rule. We are given an instance (G, f, k, r, g, X) of IC-Left-Branch. Let $v \in V \setminus Y$ be any vertex with $N^G_{2r}(v) \cap Y = \emptyset$ and $\forall w \in N^G_r(v) : f(w) < k$.

(1) Try all possibilities of placing a vertex $u \in N^G_{2r}(v) \setminus N^G_r(v)$ into any free slot, resulting in $|N_{2r}(v) \setminus N^G_r(v)| \cdot \ell$ branches where ℓ is the number of free slots inside the current search tree node.

(2) Try all possibilities of placing a vertex $u \in (\{v\} \cup N^G_r(v))$ into the leftmost free slot, that is, the slot $\min\{s \in [c] \setminus X\}$, resulting in $|N^G_r(v)|+1$ branches.

Correctness of the branching rule. We argue that if there is valid a solution g^*, that is, an injective assignment satisfying the requirements of IC-Left-Branch that extends the current branch, then there is a (possibly different) solution g' that extends the current branch and can be found in a branch obtained by branching rules (1) and (2). Let L^* and L' be the suborderings corresponding to the assignments g^* and g' respectively. Let $S^* = g^*([c])$ and $S' = g'([c])$. We consider two cases.

- Case 1: S^* contains a vertex $u \in N^G_{2r}(v) \setminus N^G_r(v)$. Observe that g^* can be found in some branch resulting from (1).
- Case 2: S^* does not contain any vertex from $N^G_{2r}(v) \setminus N^G_r(v)$.

– Case 2.1: S^* contains vertices from $\{v\} \cup N_r^G(v)$. Then inspect the leftmost vertex $u \in S^* \cap (\{v\} \cup N_r^G(v))$ w.r.t. L^*. We construct another solution L' that places u in the first free slot w.r.t. to the current branch by rotating vertices between the first free slot and u inclusively one step to the right; that is, we replace $(v_1, \ldots, v_m, u)$ by $(u, v_1, \ldots, v_m)$, where v_1 is the vertex placed at the first free slot w.r.t. the current branch and $(v_1, \ldots, v_m, u)$ is the consecutive part of the subordering L^* between the first free slot and u, inclusively. This is another solution because shortest paths from u to vertices $v_1, \ldots, v_m$ have length at least $r+1$ as S^* does not contain vertices from $N_{2r}^G(v) \setminus N_r^G(v)$, and we chose the leftmost u.

– Case 2.2: S^* does not contain any vertex from $\{v\} \cup N_r^{G'}(v)$. We can construct another solution L' that places v in the first free slot s w.r.t. the current branch by shifting every vertex in L^* starting from s one step to the right and placing v at slot s; that is, we replace $(v_1, \ldots, v_m)$ by $(v, v_1, \ldots, v_{m-1})$ where v_1 is the vertex placed at slot s and $(v_1, \ldots, v_m)$ is the consecutive part of L^* from slot s. The vertex v_m simply is removed from the ordering. This is another solution because $f(u) < k$ for each $u \in N_r^{G'}(v)$, so that v being in weakly left r-reachable sets of vertices u does not contradict $f(u) + |\text{Wreachleft}_r(G, L', u)| \leq k$.

Branching is not applicable anymore. Assume that the branching rule is not applicable anymore. There are two cases, either there are no free slots or there are free slots. If there are no free slots, we check if we have a valid solution for the original instance of the problem IC-WCOL-LEFT(r) by checking the value $f(v) + |\text{Wreachleft}_r(G, L, v)|$ for every $v \in V(G)$ which can be done in polynomial time.

In the second case there still are free slots, so we have work to do by placing vertices into slots.

Bounded number of vertices. We argue that in the second case the vertices that still can be placed into a free slot such that we obtain a valid solution is bounded by a function of r, Δ, and c. A simple exponential algorithm that tries to place these vertices into the remaining free slots in any order will run in FPT-time parameterized by r, Δ, and c. As the branching rule is not applicable anymore, for every vertex $v \in V \setminus (S \cup Y)$ we have that either $N_{2r}^G(v) \cap Y \neq \emptyset$ or $\exists w \in N_r^G(v) : f(w) \geq k$. We consider two cases:

- Case 1: $N_{2r}^G(v) \cap Y \neq \emptyset$. As we have c free slots in the beginning of the algorithm, $|Y|$ is at most c. Each vertex $u \in Y$ can contribute to at most $|N_{2r}^G(u)| \leq \Delta^{2r}$ vertices v with $N_{2r}^G(v) \cap Y \neq \emptyset$. Hence, we have at most $c \cdot \Delta^{2r}$ vertices v of this kind.

- Case 2: $N_{2r}^G(v) \cap Y = \emptyset$ and $\exists w \in N_r^G(v) : f(w) \geq k$. It follows that $w \notin Y$. We would only be able to place v into a free slot if we also place w into an earlier slot. So we consider two cases:

  – Case 2.1: $\exists x \in N_r^G(w) : f(x) \geq k$. As $N_r^G(w) \subseteq N_{2r}^G(v)$ we have that $x \notin Y$. In this case we will not be able to place neither w nor x into a free slot such

that we obtain a valid solution. Thus, we also cannot place v, and we do not have to count v at all as there is no solution that will place v into a free slot. Notice that this case is detectable in polynomial time.

– Case 2.2: $\neg\exists x \in N_r^{G'}(w) : f(x) \geq k$. As the branching rule is not applicable anymore we must have that $N_{2r}^{G}(w) \cap Y \neq \emptyset$. Thus, w is one of the $c \cdot \Delta^{2r}$ vertices from case 1. Furthermore, w can only contribute to Δ^r vertices v. Hence, we can only have $c \cdot \Delta^{3r}$ vertices v.

Putting everything together we have at most $c \cdot (\Delta^{3r} + \Delta^{2r}) \leq 2c \cdot \Delta^{3r}$ vertices that still can be placed into a free slot. For the remaining part of the algorithm we can simply apply a brute-force algorithm, that tries putting any of the vertices into the remaining free slots of the current node of the branching tree.

Time complexity. Without loss of generality we assume that the branching rule can always be applied $0 \leq i \leq c$ times. Then the branching part of the algorithm runs in time

$$\begin{aligned}\mathcal{O}(n_G^{\mathcal{O}(1)} \cdot \prod_{j=0}^{i-1}[(\Delta^{2r} - \Delta^r)(c-j) + \Delta^r]) &\subseteq \mathcal{O}(n_G^{\mathcal{O}(1)} \cdot \prod_{j=0}^{i-1}[\Delta^{2r}(c-j)]) \\ &\subseteq \mathcal{O}(n_G^{\mathcal{O}(1)} \cdot c^i \cdot \Delta^{2r \cdot i}).\end{aligned}$$

For each node of the search tree where the branching rule cannot be applied anymore the remaining part runs in time

$$\mathcal{O}(n_G^{\mathcal{O}(1)} \cdot \prod_{i}^{c-1}[2c \cdot \Delta^{3r}]) = \mathcal{O}(n_G^{\mathcal{O}(1)} \cdot (2c)^{c-i} \cdot \Delta^{3r \cdot (c-i)}).$$

Putting everything together the whole algorithm will run in $\mathcal{O}((2c)^c \cdot \Delta^{3r \cdot c} \cdot n_G^{\mathcal{O}(1)})$, meaning that the problem is fixed parameter tractable parameterized by $c + r + \Delta$. Certainly, there exists a better bound, but we do not expect there to be a feasible asymptotic runtime applying this branching rule. □

We do not implement the algorithm contained in the above proof, as the implementation would be rather complicated, and we do not see a big runtime improvement compared to the following XP-algorithm of Proposition 5.2.4. Furthermore, the polynomial factor of the FPT-algorithm is also rather high if we want to update weakly left r-reachable sets during the branching algorithm to achieve lower bounds for weakly left r-reachable sets, as we do not always place vertices at the end of a subordering. Thus, Theorem 5.2.3 exclusively serves as theoretical evidence. As an open question we propose looking at other parameters that are always smaller than the maximum degree of a graph such as the h-index [ES12].

A much simpler algorithm for IC-WCOL-LEFT(r) can be found by realizing that IC-WCOL-LEFT(r) is in XP.

Proposition 5.2.4. *IC-WCOL-LEFT(r) parameterized by c is in XP.*

Proof. We can simply try placing any of the vertices from $V \setminus S$ into the next free slot. As there are c free slots the overall algorithm runs in $\mathcal{O}(|V \setminus S|^c \cdot n_G^{\mathcal{O}(1)}) \subseteq \mathcal{O}(n_G^c \cdot n_G^{\mathcal{O}(1)})$ time. □

We include the algorithm acquired by Proposition 5.2.4 in our implementations. Furthermore, we call the turbocharging approach that applies IC-WCOL-LEFT(r) implementing this algorithm TC-LastC. We will see more on the implementation details of this approach in Chapter 6.

Furthermore, our implementations also include an approach that only changes the order of the last c vertices of a non-extendable subordering, we call this approach TC-LastC-reorder. Clearly, the runtime of a trivial implementation is bounded by $\mathcal{O}(c! \cdot n_G^{\mathcal{O}(1)})$. Note that TC-LastC-reorder is more of a heuristic approach — it could be that there is an extendable subordering of size $|S|$, but TC-LastC-reorder does not find it, even if we let $c = |S|$.

5.3 A More Local Approach

Due to the relatively bad results achieved by the TC-LastC approach for turbocharging left-to-right heuristics we continue designing new turbocharging problems. For this, we want to point out a big weakness of only considering the last c positions of a subordering during turbocharging. Namely, a heuristic could have made mistakes rather early in the subordering, while the caused problem in the form of an overfull vertex in a non-extendable subordering, is detected at a much later point in time. We would have to make the parameter c very big to fix this problem using the TC-LastC approach, leading to infeasible runtimes. Motivated by this we want to present two different turbocharging problems that do not consider the last c vertices of a non-extendable subordering but rather a specific part that we deem to cause the aforementioned problem.

To find such problematic parts we consider the first indication for problems in a non-extendable subordering — overfull vertices, that is, vertices whose weakly left r-reachable sets exceed size k. Let L be a non-extendable subordering of vertices S and U be the set of overfull vertices. Assuming that we can fix the problem by only considering local neighborhoods of vertices in U, we can look at either the r-neighbourhoods of vertices in U, or the weakly left r-reachable sets of vertices in U. These two different local neighborhoods lead to two new turbocharging problems. But as in both of them we want to replace a subset of vertices of S by different vertices, we can combine both approaches into the single theoretical problem IC-Replace(r).

IC-Replace(r)

Input: A graph G, a subordering L of vertices $S \subseteq V$, a positive integer k, and a set of vertices $X \subseteq S$.

Problem: Compute an extendable subordering L' of vertices S' such that

- $(S \setminus X) \subseteq S'$,
- $|S| = |S'|$, and
- $\text{pos}_{L'}(v) = \text{pos}_L(v)$ for all $v \in (S \setminus X)$.

Otherwise, report that there exists no such subordering.

Informally, IC-Replace(r) asks, for a subordering L of vertices S, to replace the vertices $X \subseteq S$ by (possibly) different vertices. It is clear that we can solve IC-WCOL-LEFT(r) with IC-Replace(r) by letting X be the rightmost c vertices of L. Due to this, we directly obtain some hardness results.

Corollary 5.3.1. *IC-Replace(r) is NP-hard for each $r \geq 1$ and is W[1]-hard when parameterized by $|X|$ for each fixed $r \geq 1$ and $k \geq 2$.*

Proof sketch. This result follows immediately from Theorem 5.2.1 and Theorem 5.2.2 by a reduction from IC-WCOL-LEFT(r) to IC-Replace(r). □

But as for IC-WCOL-LEFT(r) there is a trivial XP-algorithm for IC-Replace(r) that we will be using as a starting point for our implementations.

Proposition 5.3.2. *IC-Replace(r) parameterized by $|X|$ is in XP.*

Proof sketch. We can simply try placing any of the vertices from $V \setminus (S \setminus X)$ into the next free position that is held by a vertex from X. As there are $|X|$ free positions the overall algorithm runs in $\mathcal{O}(|V \setminus (S \setminus X)|^{|X|} \cdot n_G^{\mathcal{O}(1)}) \subseteq \mathcal{O}(n_G^{|X|} \cdot n_G^{\mathcal{O}(1)})$ time. □

Some implementation details for the XP-algorithm sketched above that allow efficient updating of weakly left r-reachable sets during placement of a vertex into a free position will follow in Chapter 6.

We want to describe two turbocharging approaches that make direct use of IC-Replace(r). Let L be a non-extendable subordering of vertices S with overfull vertices U, and let c be a positive integer. We propose two different invocations of IC-Replace(r) that try to make L extendable again.

- TC-RNeigh: Let X be a random subset of $S \cap (\bigcup_{v \in U} [N_r^G(v) \cup \{v\}])$ of size $\min(c, |S \cap (\bigcup_{v \in U} [N_r^G(v) \cup \{v\}])|)$. Try to fix L by applying IC-Replace(r) with this set X.
- TC-Wreach: Let X be a random subset of $S \cap (\bigcup_{v \in U} \text{Wreachleft}_r(G, L, v))$ of size $\min(c, |S \cap (\bigcup_{v \in U} \text{Wreachleft}_r(G, L, v))|)$. Try to fix L by applying IC-Replace(r) with this set X.

As we fixed c, we only obtain sets X of size at most c for both turbocharging problems. Hence, we can bound the running time of both approaches. The motivation for both approaches is that we want to fix problematic parts of the subordering L for each overfull vertex by considering the two kinds of local neighborhoods — r-neighbourhoods and weakly left r-reachable sets.

As for the TC-LastC approach we also include an implementation that only changes the order of vertices in X for vertices X defined as in TC-RNeigh and TC-Wreach. We call these approaches TC-RNeigh-reorder and TC-Wreach-reorder; again, running times of trivial implementations for both approaches are bounded by $\mathcal{O}(c! \cdot n_G^{\mathcal{O}(1)})$.

Note that in contrast to TC-LastC, for all approaches TC-RNeigh, TC-RNeigh-reorder, TC-Wreach, and TC-Wreach-reorder there is a possibility that they do not find an extendable subordering of size $|S|$ if there exists such a subordering, even if we let $c = |V|$.

5.4 Turbocharging by Iterative Swapping

During their research regarding weak coloring numbers, Nadara et al. [Nad+19] not only proposed a variety of heuristics for computing orderings of small weak r-coloring numbers, but also proposed a local search that takes an ordering and tries to modify it until its weak r-coloring number decreases. We want to briefly describe this local search algorithm and propose a turbocharging approach based on it. We do this as the local search by Nadara et al. performs exceptionally well and improves weak r-coloring numbers of orderings computed by heuristics for a lot of instances by a significant amount. Furthermore, we have to mention that the following turbocharging approach does not fit the spirit of turbocharging, as we are not using an exact method. Rather, we are augmenting a heuristic with another heuristic.

Let L be an ordering of vertices; the local search by Nadara et al. makes use of the following two rules.

Rule 1: Let $v \in V$ be any vertex such that the value $|\mathrm{Wreach}_r(G, L, v)|$ is maximal. Swap v with a random vertex earlier in the ordering. This should decrease the size of the weakly r-reachable set of v. If the swap did not decrease the weakly r-reachable set of v, we still apply this swap.

Rule 2: If we sort V by $|\mathrm{Wreach}_r(G, L, v)|$, choose $v \in V$ randomly among the 10 largest vertices w.r.t. this order; let $k_1 = |\mathrm{Wreach}_r(G, L, v)|$ and choose k_2 randomly such that $\max(1, k_1 - 3) \leq k_2 \leq k_1 - 1$. Swap v with its left neighbor until $|\mathrm{Wreach}_r(G, L, v)| = k_2$.

During one run of the local search, in the first step Rule 1 is applied a number of times or until there is no improvement in the weak r-coloring number of L for a specific amount of applications of Rule 1. Then Rule 2 is applied a number of times to the best obtained ordering in the first step. The ordering with the smallest weak r-coloring number that was found during one of both steps is returned.

Algorithm 5.1: Turbocharging with iterative swapping

Input: A graph $G = (V, E)$ and a non-extendable subordering L with overfull vertices U

Output: An extendable subordering

```
while L is non-extendable do
    Apply TC-Rule 2 until L is extendable or the size of U did not decrease for
     the last 50 applications of TC-Rule 2;
    Apply TC-Rule 1 until L is extendable or at most 20 times;
end
return L
```

Based on this local search algorithm we propose a similar local search algorithm that fixes a non-extendable subordering L. Let L be a non-extendable subordering of vertices S with overfull vertices U. We again make use of two rules.

TC-Rule 1: Let $v \in U$ be a random overfull vertex. Swap v with a random vertex earlier in the subordering. If $v \notin S$, then v will replace a random vertex $w \in S$, while w will be removed from the subordering, making it a free vertex.

TC-Rule 2: If we sort V by $|\text{Wreachleft}_r(G, L, v)|$, choose $v \in V$ randomly among the 10 largest vertices w.r.t. this order; let $k_1 = |\text{Wreachleft}_r(G, L, v)|$ and choose k_2 randomly such that $\max(1, k_1 - 3) \leq k_2 \leq k_1 - 1$. Swap v with its left neighbor until $|\text{Wreachleft}_r(G, L, v)| = k_2$. If $v \notin S$ then the first swap will make the rightmost vertex of L free and place v at the rightmost position instead.

The applications of TC-Rule 1 and TC-Rule 2 during the local search are illustrated in Algorithm 5.1. TC-Rule 2 does a more local search by small changes to L. But if the application of TC-Rule 2 does not make any progress, then TC-Rule 1 is applied a number of times such that we make bigger changes to L. The values 50 and 20 were chosen as they empirically produced good results. We call the turbocharging approach based on Algorithm 5.1 TC-Iterative-Swap. Again, we want to emphasize that this approach does not fit the scheme of turbocharging as we defined it, notice that Algorithm 5.1 cannot return a negative result but can only time out, and that we do not have a conservation parameter c.

5.5 Turbocharging by Merging

The last turbocharging approach for a left-to-right heuristic we want to present is based on "merging" a set of vertices into a fixed subordering L. That is, we might assume that the relative order of a vertex set S_1 ordered by L is already correct, but we want to extend it by another vertex set S_2 of vertices whose relative order we might or might not know. To formalize this we introduce the problems WCOL-Merge-Ordered(r) and WCOL-Merge(r), and discuss their complexity. Later we will describe how we apply

WCOL-Merge(r) to non-extendable suborderings, introducing another turbocharging approach.

WCOL-Merge-Ordered(r)

Input: A graph G, an integer k, three disjoint sets S_1, S_2, and T such that $V(G) = S_1 \cup S_2 \cup T$, and two suborderings L_1 and L_2 such that L_1 is a subordering of the vertex set S_1, and L_2 is a subordering of the vertex set S_2.

Problem: Compute an extendable subordering L of vertices $S_1 \cup S_2$ such that

- $L[S_1] = L_1$, and
- $L[S_2] = L_2$.

Otherwise, report that there is no such subordering.

Informally, WCOL-Merge-Ordered(r) asks to find an extendable subordering of vertices $S_1 \cup S_2$ that agrees with L_1 on S_1 and that agrees with L_2 on S_2.

WCOL-Merge(r)

Input: A graph G, an integer k, three disjoint sets S_1, S_2 and T such that $V(G) = S_1 \cup S_2 \cup T$, and a subordering L_1 of the vertex set S_1.

Problem: Compute an extendable subordering L of vertices $S_1 \cup S_2$ such that

- $L[S_1] = L_1$.

Otherwise, report that there is no such subordering.

Informally, WCOL-Merge(r) asks to find an extendable subordering of vertices $S_1 \cup S_2$ that agrees with L_1 on S_1.

First, we want to investigate whether WCOL-Merge-Ordered(r) admits a polynomial time algorithm. We already know that this is not the case for WCOL-Merge(r) because by setting $S_1 = T = \emptyset$ we could solve WCOLORD(r). In Theorem 5.5.1 we show that we cannot expect a polynomial-time algorithm for WCOL-Merge-Ordered(r) (unless P = NP).

Theorem 5.5.1. *WCOL-Merge-Ordered(r) is NP-complete for $r = 2$ and $k = 6$.*

To prove Theorem 5.5.1, we first introduce the problem Restricted 2,3-SAT and show its NP-hardness. The idea for this problem and its NP-hardness proof stem from a post on `https://cs.stackexchange.com/` [17].

Restricted 2,3-SAT

Input: A set Var of variables and a set C of clauses such that

- $\mathsf{cl} \subseteq \mathsf{Var} \cup \{\neg v \mid v \in \mathsf{Var}\}$ for all $\mathsf{cl} \in C$,
- $|\mathsf{cl}| = 3$ or $|\mathsf{cl}| = 2$ for all $\mathsf{cl} \in C$, and
- each literal $\ell \in \mathsf{Var} \cup \{\neg v \mid v \in \mathsf{Var}\}$ appears in at most two clauses.

Problem: Is there a variable assignment $A : \mathsf{Var} \to \{0, 1\}$ such that

- $A(\ell) = 1$ for at least one $\ell \in \mathsf{cl}$ for each clause $\mathsf{cl} \in C$,

where $A(\neg v) = 1 - A(v)$ for each $v \in \mathsf{Var}$? We call such an assignment A *satisfying assignment.*

Comparing to 3-SAT, Restricted 2,3-SAT also allows clauses of size 2, but each literal can appear in at most two clauses. This does not change its NP-hardness.

Lemma 5.5.2. *Restricted 2,3-SAT is NP-hard.*

Proof sketch. We give a reduction from 3-SAT which is NP-complete [GJ79]. Let (Var, C) be an instance of 3-SAT. We construct an instance of Restricted 2,3-SAT as follows. Let $v \in \mathsf{Var}$. For each occurrence of v or $\neg v$ inside a clause cl we introduce a new variable v_i and replace v by v_i and $\neg v$ by $\neg v_i$ in cl, respectively. Assume that we have introduced k instances $v_1, \ldots, v_k$ of v. We create k clauses $\{v_1, \neg v_2\}, \{v_2, \neg v_3\}, \ldots, \{v_k, \neg v_1\}$ and add them to C, creating a new clause set C'. This set of clauses forces an equivalence of variables $v_1, \ldots, v_k$, hence $A(v_1) = \cdots = A(v_k)$ must hold in every satisfying assignment A. We have that there is a satisfying assignment A of the given 3-SAT-instance if and only if there is a satisfying assignment A for the constructed Restricted 2,3-SAT-instance. Clearly, in the newly constructed instance every literal appears at most twice — once in an old clause $\mathsf{cl} \in C$ of size three and once in a new clause of size two that forces its equivalence with other variables. □

We are now ready to prove Theorem 5.5.1.

Proof of Theorem 5.5.1. Obviously, WCOL-Merge-Ordered(r) is in NP. We show its NP-hardness by a reduction from Restricted 2,3-SAT which is NP-hard by Lemma 5.5.2. Let (Var, C) be an instance of Restricted 2,3-SAT, where $\mathsf{Var} = \{v_1, \ldots, v_n\}$ is a set of variables and $C = \{\mathsf{cl}_1, \ldots, \mathsf{cl}_m\}$ is a set of clauses of sizes two and three. Without loss of generality let cl_i with $1 \leq i \leq p$ be the clauses of size three and cl_j with $p < j \leq m$ be the clauses of size two. We write $\neg\mathsf{Var}$ for $\{\neg v \mid v \in \mathsf{Var}\}$. We construct an instance $(G, k, S_1, S_2, T, L_1, L_2)$ of WCOL-Merge-Ordered(2) as follows. Let $G = (V, E)$ where

- $V = C \cup \mathsf{Var} \cup \neg\mathsf{Var} \cup \{x, y_1, \ldots, y_6\} \cup \{z_{j,1}, z_{j,2} \mid p < j \leq m\}$, and

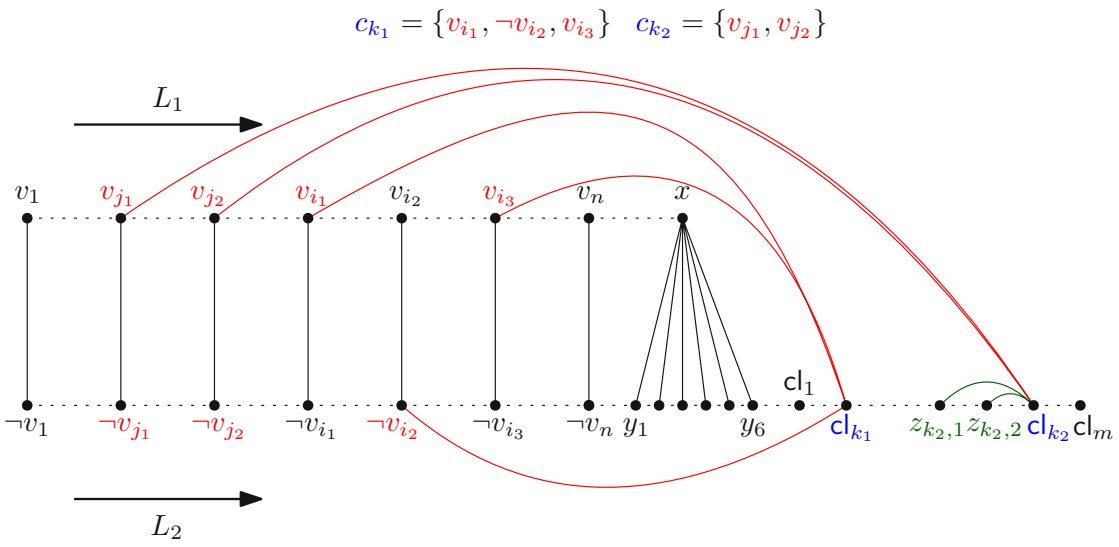


Figure 5.2: Sketch of the reduction for the proof of Theorem 5.5.1.

- $E = \{\{v, \neg v\} \mid v \in \mathsf{Var}\} \cup \{\{x, y_i\} \mid 1 \leq i \leq 6\} \cup \{\{\mathsf{cl}, \ell\} \mid \mathsf{cl} \in C, \ell \in \mathsf{cl}\} \cup \{\{\mathsf{cl}_j, z_{j,1}\}, \{\mathsf{cl}_j, z_{j,2}\} \mid p < j \leq m\}$.

Furthermore, let $S_1 = \mathsf{Var} \cup \{x\}$, $S_2 = \neg\mathsf{Var} \cup \{y_1, \ldots, y_6\} \cup C$, and $T = \emptyset$. Lastly we define the two suborderings L_1 and L_2 as

$$L_1 = (v_1, \ldots, v_n, x),$$

and

$$L_2 = (\neg v_1, \ldots, \neg v_n, y_1, \ldots, y_6, \mathsf{cl}_1, \ldots, \mathsf{cl}_p, z_{p+1,1}, z_{p+1,2}, \mathsf{cl}_{p+1}, \ldots, z_{m,1}, z_{m,2}, \mathsf{cl}_m).$$

An instantiation for this reduction is depicted in Figure 5.2, where it is shown in more detail for a clause cl_{k_1} of size three and cl_{k_2} of size two. We show that there is an extendable subordering L of vertices $V(G)$ such that

- $L[S_1] = L_1$, and
- $L[S_2] = L_2$

if and only if there is an assignment $A : \mathsf{Var} \to \{0, 1\}$ with

- $A(\ell) = 1$ for at least one $\ell \in \mathsf{cl}$ for each clause $\mathsf{cl} \in C$.

We argue correctness, that is, we show that there is an extendable subordering L of vertices $S_1 \cup S_2$ such that $L[S_1] = L_1$ and $L[S_2] = L_2$ if and only if there is a satisfying assignment A.

"⇒": Assume that there is the aforementioned subordering L. Notice that $v_i \prec_L \mathsf{cl}_j$ for all $1 \leq i \leq n$ and $1 \leq j \leq m$ because x has to be left of y_6 and thus also cl_1 as

otherwise $|\text{Wreachleft}_2(G, L, x)| > 6$. We construct an assignment $A : \mathsf{Var} \to \{0, 1\}$ as follows. Let $A(v) = 1$ if $v \prec_L \neg v$ and $A(v) = 0$ otherwise. We claim that $A(\ell) = 1$ for at least one $\ell \in \mathsf{cl}$ for each clause $\mathsf{cl} \in C$. Assume to the contrary that this is not true, hence, there exists a clause cl such that $A(\ell) = 0$ for all $\ell \in \mathsf{cl}$. Then by the construction of A we have that

- $v_i \prec_L \neg v_i$ if $\ell = \neg v_i$ for some $i \in [n]$ and
- $\neg v_i \prec_L v_i$ if $\ell = v_i$ for some $i \in [n]$.

As all v_i and $\neg v_i$ are left of cl w.r.t. L we have $|\text{Wreachleft}_2(G, L, \mathsf{cl})| = 7 > 6$ because $v, \neg v \in \text{Wreachleft}_2(G, L, \mathsf{cl})$ if $v \in \mathsf{cl}$ or $\neg v \in \mathsf{cl}$. It does not matter if $|\mathsf{cl}| = 2$ or $|\mathsf{cl}| = 3$ as we have "offset" the weakly left 2-reachable sets of clauses cl of size two by two. Hence, we have a contradiction.

"$\Leftarrow$" Assume that we have an assignment $A : \mathsf{Var} \to \{0, 1\}$ such that $A(\ell) = 1$ for at least one $\ell \in \mathsf{cl}$ for each $\mathsf{cl} \in C$. Let us define L as

$$L = (\ell_1^1, \ell_1^2, \ldots, \ell_n^1, \ell_n^2, x, y_1, \ldots, y_6, \mathsf{cl}_1, \ldots, \mathsf{cl}_p, z_{p+1,1}, z_{p+1,2}, \mathsf{cl}_{p+1}, \ldots, z_{m,1}, z_{m,2}, \mathsf{cl}_m)),$$

and

- $\ell_i^1 = v_i$ if $A(v_i) = 1$ and $\ell_i^1 = \neg v_i$ otherwise, and
- $\ell_i^2 = v_i$ if $A(v_i) = 0$ and $\ell_i^2 = \neg v_i$ otherwise.

It is evident that

- $L[S_1] = L_1$, and
- $L[S_2] = L_2$.

We only have to check cardinalities of weakly left 2-reachable sets.

- We have that $|\text{Wreachleft}_2(G, L, \ell_i^j)| \leq 6$ for all $1 \leq i \leq n$ and $j = 1, 2$ because each literal appears in at most two clauses. Hence, ℓ_i^j can at most weakly 2-reach itself, $\ell_i^{j'}$, and four other literals that it appears with inside a clause. Here, j' is 1 if $j = 2$ and 2 otherwise. The vertex ℓ_i^j can only weakly 2-reach other literals as all other vertices are right to it w.r.t. L.
- It is also the case that $|\text{Wreachleft}_2(G, L, \mathsf{cl}_i)| \leq 6$ for each $1 \leq i \leq m$ because for some variable v_j where $v_j \in \mathsf{cl}_i$ or $\neg v_j \in \mathsf{cl}_i$ not both v_j and $\neg v_j$ are weakly 2-reachable from cl_i. This is true because $A(\ell) = 1$ for some $\ell \in \mathsf{cl}_i$. So each clause of size 3 can weakly 2-reach at most 5 literals and each clause of size two can weakly 2-reach at most 3 literals. Furthermore, clauses c_i of size 2 also weakly 2-reach 2 vertices $z_{i,1}$ and $z_{i,2}$.
- It is clear that $|\text{Wreachleft}_2(G, L, z_i^j)| \leq 5$ for $p < i \leq m$ and $j = 1, 2$.

- Furthermore, $|\text{Wreachleft}_2(G, L, x)| = 1$ and $|\text{Wreachleft}_2(G, L, y_i)| = 2$ for all $1 \leq i \leq 6$.

As all cardinalities do not exceed 6 we are done with the backwards direction.

This reduction is polynomial, so we can conclude the proof. □

An interesting fact about the proof and the contained reduction of Theorem 5.5.1 is that it could give us insights about another problem — if we could "force" a specific order of vertices for an ordering without increasing weakly r-reachable sets too much this could lead to a similar reduction to prove NP-completeness of WCOLORD(r) for the missing case of $r = 2$.

As we now know that both WCOL-Merge(r) and WCOL-Merge-Ordered(r) are NP-complete, we want to focus on finding fixed-parameter algorithms for WCOL-Merge(r), as it is more powerful than WCOL-Merge-Ordered(r) — in the sense that it covers more possibilities of merging S_2 into S_1. In Theorem 5.5.3 we show that WCOL-Merge(r) is fixed-parameter tractable parameterized by $k+|S_2|$, which supports its relevance in designing a turbocharging approach based on it.

Theorem 5.5.3. *WCOL-Merge(r) is solvable in time $\mathcal{O}(|S_2|! \cdot k^{|S_2|} \cdot n_G^{\mathcal{O}(1)})$. In particular, WCOL-Merge(r) is fixed-parameter tractable when parameterized by $k + |S_2|$.*

For the proof of Theorem 5.5.3 we need definitions for two operations that we will use throughout the proof. The algorithm that proves fixed-parameter tractability of WCOL-Merge(r) will apply these operations during a search tree algorithm.

Definition 5.5.4. Let G be a graph, $L = (s_1, \ldots, s_n)$ be a subordering of vertices $S \subseteq V(G)$, and $v \in V(G) \setminus S$. We denote by placeafter(L, s_i, v) the subordering of vertices $S \cup \{v\}$ that is obtained by placing v directly after s_i. To be precise, placeafter(L, s_i, v) is defined as

$$\text{placeafter}(L, s_i, v) = (s_1, \ldots, s_i, v, s_{i+1}, \ldots, s_n).$$

Equivalently, placebefore(L, s_i, v) is defined as

$$\text{placebefore}(L, s_i, v) = (s_1, \ldots, s_{i-1}, v, s_i, \ldots, s_n).$$

This leads to the definitions of breakpoints, which will be crucial for the proof of Theorem 5.5.3.

Definition 5.5.5. Let G be a graph, $L = (s_1, \ldots, s_n)$ be a subordering of vertices $S \subseteq V(G)$, and $v \in V(G) \setminus S$. A vertex $s \in S$ is called *breakpoint* of v if

$$\text{Wreachleft}_r(G, \text{placebefore}(L, s, v), v) \neq \text{Wreachleft}_r(G, \text{placeafter}(L, s, v), v).$$

Let $\text{bp}(G, L, v) \subseteq S$ be the set of breakpoints of v.

We also notice another useful property of breakpoints.

Observation 5.5.6. *We have that* $s \notin \mathrm{bp}(G, L, v)$ *if and only if*

$$\mathrm{Wreachleft}_r(G, \mathrm{placebefore}(L, s, v), u) = \mathrm{Wreachleft}_r(G, \mathrm{placeafter}(L, s, v), u)$$

for all $u \in V(G)$.

This observation will be very helpful throughout the proof of Theorem 5.5.3.

If we want to merge a vertex v into a subordering L of vertices S, creating a new subordering L', then the breakpoints $\mathrm{bp}(L, v)$ of v are exactly the candidates for elements of the weakly left r-reachable set of v w.r.t. L'. We want to formalize this in the following claim.

Lemma 5.5.7. *Let* G *be a graph,* $L = (s_1, \ldots, s_n)$ *be a subordering of vertices* $S \subseteq V(G)$*, and* $v \in V(G) \setminus S$*. Furthermore, let* L' *be a subordering of vertices* $S \cup \{v\}$ *such that* $L'[S] = L$*. Then*

$$\mathrm{Wreachleft}_r(G, L', v) \setminus \{v\} = \{s \in \mathrm{bp}(G, L, v) \mid s \preceq_{L'} v\}.$$

Proof. Let $X = \{s \in \mathrm{bp}(G, L, v) \mid s \preceq_{L'} v\}$. We prove both inclusions of $\mathrm{Wreachleft}_r(G, L', v) \setminus \{v\} = X$.

Assume that $s \in (\mathrm{Wreachleft}_r(G, L', v) \setminus \{v\})$. As s is weakly left r-reachable from v, there is a path $P = (v, u_1, \ldots, u_\ell, s)$ of length of at most r that does not go left of s w.r.t. to L'. Consider the same path P in $\mathrm{placeafter}(L, s, v)$. Clearly, s is also weakly left r-reachable in this ordering because of the same path. Contrary to that, s cannot be weakly left r-reachable from v in $\mathrm{placebefore}(L, s, v)$ because v is left of s in that ordering. Hence, $s \in \mathrm{bp}(G, L, v)$ and $\mathrm{Wreachleft}_r(G, L', v) \setminus \{v\} \subseteq X$.

Assume that $s \in X$. Then s must be weakly left r-reachable from v w.r.t. $\mathrm{placeafter}(L, s, v)$ through a path P of length at most r. But s is also weakly left r-reachable from v w.r.t. L' through the same path P. Hence, $X \subseteq \mathrm{Wreachleft}_r(G, L', v) \setminus \{v\}$ also holds. □

We are ready to prove Theorem 5.5.3.

Proof of Theorem 5.5.3. We give a recursive search tree algorithm that solves WCOL-Merge(r) in time $\mathcal{O}(|S_2|! \cdot k^{|S_2|} \cdot n_G^{\mathcal{O}(1)})$. This algorithm is given in Algorithm 5.2 as a recursive function. Given an instance (G, L_1, S_1, S_2, S_3) of WCOL-Merge(r), the given function can be invoked as Recursive-merge(S_1, S_2, S_3, L_1), and will either return an extendable subordering L' of vertices $S_1 \cup S_2$ such that $L'[S_1] = L_1$, or it will return false if no such subordering exists. In a recursive call, the algorithm places one vertex $v \in S_2$ into L and thus decreases the cardinality of S_2 and increases the cardinality of S_1. Furthermore, in each recursive call the following new calls are created for each vertex $v \in S_2$.

(1) For each vertex $s \in \mathrm{bp}(G[S_1 \cup S_3 \cup \{v\}], L, v)$ consider $L' = \mathrm{placeafter}(L, s, v)$. If $|\mathrm{Wreachleft}_r(G[S_1 \cup S_3 \cup \{v\}], L', v)| \leq k$, then create a new recursive call $(S_1 \cup \{v\}, S_2 \setminus \{v\}, S_3, L')$.

Algorithm 5.2: Recursive FPT-algorithm for WCOL-Merge(r)

```
RECURSIVE-MERGE (S_1, S_2, S_3, L)
if |S_2|= 0 ∧ ∀v ∈ V(G) : |Wreachleft_r(G, L, v)|≤ k then
    return L
end
for v ∈ S_2 do
    for s ∈ bp(G[S_1 ∪ S_3 ∪ {v}], L, v) do
        L' ← placebefore(L, s, v);
        if |Wreachleft_r(G[S_1 ∪ S_3 ∪ {v}], L', v)|≤ k then
            answer←RECURSIVE-MERGE (S_1 ∪ {v}, S_2 \ {v}, S_3, L') ;
            if answer≠false then
                return answer
            end
        end
    end
    s ←rightmost vertex of S_1 w.r.t. L;
    L' ← placeafter(L, s, v);
    if |Wreachleft_r(G[S_1 ∪ S_3 ∪ {v}], L', v)|≤ k then
        answer←RECURSIVE-MERGE (S_1 ∪ {v}, S_2 \ {v}, S_3, L') ;
        if answer≠false then
            return answer
        end
    end
end
return false
```

(2) Let L' be the subordering obtained from L by placing v at the right end. If $|\text{Wreachleft}_r(G[S_1 \cup S_3 \cup \{v\}], L', v)| \leq k$, then create a new recursive call $(S_1 \cup \{v\}, S_2 \setminus \{v\}, S_3, L')$.

Notice that we are only considering the induced subgraph $G[S_1 \cup S_3 \cup \{v\}]$, as we do not know where the remaining vertices of S_2 will be placed. We want to show that in this way only k new recursive calls are created for each vertex $v \in S_2$ and that the algorithm finds a solution (subordering L') if there exists one.

Correctness. Assume that we are in a recursive call (S_1, S_2, S_3, L) of the algorithm and that there exists an extendable subordering L' of vertices $S_1 \cup S_2$ such that $L'[S_1] = L_1$. We show that a (possibly different) subordering L^* of $S_1 \cup S_2$ with these properties can be found in one of the generated new recursive calls. Consider the rightmost vertex $v \in S_2$ w.r.t. L'. There are two possibilities.

- There is a breakpoint of v (w.r.t. L) right of (w.r.t. L') v, that is $\text{bp}(G[S_1 \cup S_3 \cup \{v\}], L, v) \cap \{s \in L \mid v \prec_{L'} s\} \neq \emptyset$. Then consider the leftmost breakpoint

$s \in \mathrm{bp}(G[S_1 \cup S_3 \cup \{v\}], L, v)$ that is right of v w.r.t. L'. We shift v left of s, creating a new subordering L^* where the weakly left r-reachable sets of all vertices do not change because of Observation 5.5.6; that is, we replace $(v, u_1, \ldots, u_\ell, s)$ by $(u_1, \ldots, u_\ell, v, s)$, where $(v, u_1, \ldots, u_\ell, s)$ is the consecutive part of L' between v and s. Weakly left r-reachable sets cannot change because none of $u_1, \ldots, u_\ell$ are in $\mathrm{bp}(L, v)$, and v is the rightmost vertex of S_2 w.r.t. L'. Clearly, L^* can be found in a recursive call generated from (1) in Line 8 because we try all breakpoints for every vertex $v \in S_2$.

- There is no breakpoint of v (w.r.t. L) right of (w.r.t. L') v, that is, $\mathrm{bp}(G[S_1 \cup S_3 \cup \{v\}], L, v) \cap \{s \in L \mid v \prec_{L'} s\} = \emptyset$. Then we can shift v to the end of L', creating a new subordering L^* where weakly left r-reachable sets do not change; that is, we replace $(v, u_1, \ldots, u_\ell)$ by $(u_1, \ldots, u_\ell, v)$, where $(v, u_1, \ldots, u_\ell)$ is the consecutive part of L' right of v. Again, weakly left r-reachable sets cannot change because none of $u_1, \ldots, u_\ell$ are in $\mathrm{bp}(L, v)$, and v is the rightmost vertex of S_2 w.r.t. L'. The subordering L^* can be found in a recursion call generated from (2) in Line 17.

Notice that none of the if-statements in Line 7 and Line 16 of Algorithm 5.2 contradict the correctness because the cardinality of the weakly left r-reachable set of v can only increase in subsequent recursive calls.

Runtime. We show that in each recursion call at most $k \cdot |S_2|$ new recursion calls are created. As the size of S_2 decreases by one in each of these new calls, the stated runtime follows. But this is a direct consequence of Lemma 5.5.7, as if a vertex v is placed to the right of one of its breakpoints $s \in \mathrm{bp}(G[S_1 \cup S_3 \cup \{v\}], L, v)$, then s will be in the weakly left r-reachable set of v. As we do not recurse if the size of the weakly left r-reachable set of v exceeds k (if-statements in Line 7 and Line 16), we generate at most k recursive calls for each vertex $v \in S_2$. The rest of the algorithm can be implemented in polynomial time, resulting in an overall runtime bounded by $\mathcal{O}(|S_2|! \cdot k^{|S_2|} \cdot n_G^{\mathcal{O}(1)})$. □

Algorithm 5.2 is held rather simple for demonstration purposes, but there are further optimizations that can be implemented and do not affect the overall time complexity:

- Consider the vertex v in Algorithm 5.2. We only have to iterate over the k leftmost breakpoints of v by Lemma 5.5.7, which can be easily done by keeping track of weakly left r-reaching sets of v. Let s be a breakpoint of v and let $L' = \mathrm{placeafter}(L, s, v)$. The leftmost $s' \in \mathrm{Wreachleft}_r^{-1}(G[S_1 \cup S_3 \cup \{v\}], L', v)$ that is not v is the next possible breakpoint of v.

- We do not need to recurse if the size of some set $\mathrm{Wreachleft}_r(G[S_1 \cup S_3], L, v)$ exceeds k for some $v \in S_1 \cup S_3$, as these sets can only increase in subsequent recursion calls.

By taking a closer look at the proof of Theorem 5.5.3, we can see that we have implicitly proven that WCOL-Merge-Ordered(r) is solvable in time $\mathcal{O}(k^{|S_2|} \cdot n_G^{\mathcal{O}(1)})$,

as in WCOL-Merge-Ordered(r) we do not have to guess which vertex is rightmost during the recursive algorithm. This means that we could drop the outer loop of Algorithm 5.2, letting v be the rightmost vertex of S_2 w.r.t. L_2.

Turbocharging approach. Applying the fixed-parameter algorithm for WCOL-Merge(r), we want to introduce another turbocharging approach that we call TC-Merge. Let L be a non-extendable subordering of vertices S with overfull vertices U. Similar to TC-Wreach, we want to look at weakly left r-reachable sets of all vertices in U. But instead of replacing them by other vertices, we now want to resolve problems in L immediately by defining a merging problem where we want to merge these vertices into a subordering. Let c be a positive integer and let X be a random subset of $\bigcup_{v\in U} \text{Wreachleft}_r(G, L, v)$ of size $\min(c, |\bigcup_{v\in U} \text{Wreachleft}_r(G, L, v)|)$. If the size of X is less than c, we randomly add further vertices from $V(G)$ to X, until the size of X is c. We try to fix L by defining a WCOL-Merge(r) problem that is solved by Algorithm 5.2. Namely, we set $S_1 = S \setminus X$, $S_2 = X$, $T = V \setminus (S \cup X)$, and $L_1 = L[S_1]$. Letting c be our conservation parameter we obtain a turbocharging approach with a fixed-parameter tractable turbocharging problem parameterized by k and c. Notice that the size of the subordering L might increase but never decrease during one application of the turbocharging problem; but that is not a problem. Due to the addition of random vertices to X if its size is less than c, we achieve a turbocharging approach that always finds an extendable subordering if there exists one by letting c be large enough.

5.6 Discussion

We want to briefly summarize this chapter and give pointers for future research. We have started by relating the turbocharging framework to left-to-right heuristics, later proposing different turbocharging approaches. In the first approach, called TC-LastC, we tried replacing the suffix of a subordering of size c by different vertices. We have shown that the underlying problem is NP-hard and W[1]-hard when parameterized by c and the weak r-coloring number k, mentioning that we are not satisfied with the proofs as they do not reflect the hardness of the whole problem of computing an ordering of vertices with small weak r-coloring number. Next, we showed that by considering another parameter, the maximum degree of a graph, we can design a fixed-parameter algorithm for the problem — although we are certain that there can be achieved a better worst-case time complexity for a fixed parameter algorithm. There are still a variety of graph parameters left to investigate that could lead to fixed-parameter tractability of the problem. We are particularly interested if the problem is fixed-parameter-tractable when parameterized by the h-index [ES12], the suffix size c and the radius r and the weak r-coloring number k.

We continued by proposing two turbocharging approaches that tried to exploit local neighborhoods of problematic parts of a subordering, and called these two approaches TC-Wreach and TC-RNeigh, and again showed their W[1]-hardness when parameterized by c and the weak r-coloring number k, and NP-hardness for the underlying problems.

Inspired by a local search algorithm of Nadara et al. [Nad+19], we proposed a turbocharging algorithm that is similar to this local search, as both the local search and our approach iteratively apply swaps to a (sub)ordering to decrease sizes of weakly (left) r-reachable sets of vertices.

The last approach we proposed is based on "merging" a set S_2 of vertices into an already fixed subordering. For this we have proposed the problem WCOL-Merge(r) and have shown its fixed-parameter tractability parameterized by the weak r-coloring number k and the size of the set S_2. We are wondering whether there is an algorithm that solves WCOL-Merge(r) with a better running time bound, in particular, if the running time can be reduced to $\mathcal{O}(k^{|S_2|}n_G^{\mathcal{O}(1)})$ or if there is an algorithm that achieves a running time of $\mathcal{O}(\alpha^{p(k,|S_2|)} \cdot n_G^{\mathcal{O}(1)})$ for some constant α and a polynomial p. Furthermore, it is still unclear to us how other graph parameters such as the h-index, the maximum degree, or decomposition parameters such as treewidth influence the runtime complexity of the problems w.r.t. fixed-parameter tractability when parameterizing by these parameters. Moreover, we were not able to prove or contradict W[1]-hardness of WCOL-Merge(r) when only parameterizing by the size of S_2. Additionally, there are certainly different ways to apply WCOL-Merge(r) to a turbocharging approach, such as merging a random subset of vertices.

We are also certain that we did not exhaust turbocharging approaches for turbocharging left-to-right heuristics, so there are still directions to take.

In the next chapter, we will see how we implemented our proposed turbocharging approaches and some interesting ways to optimize implementations for these approaches and the underlying heuristics.

CHAPTER 6

Optimizations and Implementation Details

In this chapter, we want to present further optimizations that we will be using for our implementations of turbocharging approaches for left-to-right heuristics. Furthermore, we will present some interesting implementation details. We start by presenting an observation that leads to an optimization for the heuristic part of our algorithms. Then, we will explain our implementations for all approaches that turbocharge left-to-right heuristics in more detail. Next, we will show how some operations on suborderings that are often used in our implementations can be optimized with respect to their runtime. We will continue by introducing further optimizations based on connected components of free vertices and on an adjacency list data structure for a graph G that considers the subordering L of vertices $S \subseteq V(G)$. At the end of the chapter we will present two lower bounds for weak coloring numbers for full right extensions of a subordering, one of which can be also used as a new lower bound for weak coloring numbers of graphs. We apply these lower bounds during heuristics and in our search tree algorithms for turbocharging approaches to decrease their explored search spaces.

6.1 Placing Full Vertices Next

During the application of a left-to-right heuristic we iteratively extend a subordering by one vertex to the right. When applying turbocharging, we want to use this heuristic to find an ordering of vertices that has weak r-coloring number at most k. At some point this heuristic might have created a subordering L s.t. $|\text{Wreachleft}_r(G, L, v)| = k$ for a free vertex v. It makes sense to place this vertex immediately to the right of L, such that its weakly left r-reachable set cannot increase anymore. In this section, we want to show that this is always correct; that is, if there is a full right extension of L whose weak r-coloring number is at most k, then there is another full right extension of L whose

weak r-coloring number is at most k and v is placed immediately to the right of L. For this we need to show the following simple lemma first.

Lemma 6.1.1. *Consider an extendable subordering L of vertices S with free vertices T. Assume that there is a full right extension L' of L with*

$$\forall v \in V : |\text{Wreach}_r(G, L', v)| \leq k.$$

For each vertex $u \in T$, let $X_{L'}(u) = \{v \in T \mid v \prec_{L'} u\}$ be the vertices that are placed between S and u w.r.t. L'. If $\text{Wreach}_r(G, L', u) \cap X_{L'}(u) = \emptyset$, then each path of length at most r between u and a vertex in $X_{L'}(u)$ goes through S.

Proof. Assume to the contrary that there is a path P of length at most r between u and a vertex in $X_{L'}(u)$ that does not go through S. Consider the leftmost (w.r.t. L') vertex x on this path. Since there is a path from u to x of length at most r that does not go left of x, x has to be in $\text{Wreach}_r(G, L', u) \cap X_{L'}(u)$, yielding a contraction. □

Proposition 6.1.2. *Consider an extendable subordering L of vertices S with free vertices T. Assume that there is a full right extension L' of L with*

$$\forall v \in V : |\text{Wreach}_r(G, L', v)| \leq k.$$

If for $u \in T$, $|\text{Wreachleft}_r(G, L, u)| = k$, then there is another full right extension $\overline{L}$ of L where u is the leftmost vertex of T w.r.t. $\overline{L}$ and

$$\forall v \in V : |\text{Wreach}_r(G, \overline{L}, v)| \leq k.$$

Proof. We construct $\overline{L}$ from L'. Assume $L' = (s_1, \ldots, s_n, t_1, \ldots, t_m)$ with $S = \{s_1, \ldots, s_n\}$ and $T = \{t_1, \ldots, t_m\}$; furthermore, assume $u = t_i$ with $1 \leq i \leq m$. We set $\overline{L} = (s_1, \ldots, s_n, u, t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_m)$. The ordering $\overline{L}$ has the required properties because of Lemma 6.1.1: Assuming that there exists a vertex v such that $|\text{Wreachleft}_r(G, \overline{L}, v)| > |\text{Wreachleft}_r(G, L, v)|$, we will reach a contradiction; by shifting u left, we only increase weakly r-reachable sets of vertices $t_j, j \neq i$, by adding u to their weakly r-reachable sets. If u is weakly r-reachable from a vertex t_j with respect to $\overline{L}$, and it was not weakly r-reachable with respect to L', then this new path goes through a vertex in $X_{L'}(u)$ as defined in Lemma 6.1.1 as it cannot go left of u. Whenever $|\text{Wreachleft}_r(G, L, u)| = k$, then $X_{L'}(u) \cap \text{Wreach}_r(G, L', u) = \emptyset$. Thus, every path of length at most r from u to a vertex in $X_{L'}(u)$ must go through S. But since all vertices in S are placed left of u with respect to $\overline{L}$, u cannot be reached from t_j, leading to a contradiction. □

Proposition 6.1.2 enables us to slightly adapt left-to-right heuristics during turbocharging. That is, if a left-to-right heuristic created a subordering L of vertices S with free vertices T such that $|\text{Wreachleft}_r(G, L, v)| = k$ for some $v \in T$, in the next step of the heuristic we immediately place v to the right of L. Notice that this only changes the implementation of the Degree-Heuristic, but not the implementation of the Wreach-Heuristic, as the Wreach-Heuristic already places free vertices with a maximum size weakly left r-reachable sets next.

6.2 Implementation Details of Turbocharging Approaches

In this section, we want to present some interesting implementation details of our turbocharging approaches for left-to-right heuristics. Namely, we will present some data structures, branching strategies for search tree algorithms and runtime optimizations for TC-LastC, TC-RNeigh, TC-Wreach, TC-Iterative-Swap, and TC-Merge. Throughout the section, L refers to a subordering of vertices $S \subseteq V$ with free vertices $T \subseteq V$, where V is the vertex set of a graph G.

In all implementations we keep track of L using an array A of size $|V|$. We set $A[\text{pos}_L(v)] = v$ for $v \in S$. Furthermore, we always keep track of $G[T]$ using a hash set for T and an array of hash sets as adjacency list for the edges of $G[T]$. If a vertex v is removed from T, then we remove v from the hash set and remove all incident edges of v in $G[T]$ in the adjacency list. If a vertex v is added to T, then we add v to the hash set and add all edges $E(G) \cap (\{\{u, v\} \mid u \in T\})$ to the adjacency list.

We continue by explaining implementation details for all turbocharging approaches individually.

TC-LastC. For the TC-LastC approach we implement the XP-algorithm for IC-WCOL-LEFT(r) as outlined in Proposition 5.2.4. Given a subordering L of vertices S, we have to extend L to the right by c vertices, that means that we have c positions to fill. We are implementing a search tree algorithm that fills these positions from left to right recursively. Meaning, in a search tree node we try all possibilities of placing a free vertex into position i and recurse into search tree nodes that try placing the remaining free vertices into position $i + 1$, and so on, until all c positions are filled.

If after a placement of a vertex we obtain a non-extendable subordering, we can cut off this branch of the search tree, as weakly left r-reachable sets of vertices can only increase in this branch.

Since we keep track of $G[T]$ with T being the free vertices in a search tree node, updating weakly left r-reaching sets on placement/removal can be realized by a simple depth-r breadth-first-search in $G[T]$.

Furthermore, we also try placing free vertices T into a free position i in a specific order inside a search tree node: Let L be the non-extendable subordering that triggered turbocharging and let v be the rightmost vertex of L. We then try placing $u_1 \in T$ before $u_2 \in T$ into the free slot i if $\text{dist}_G(u_1, v) < \text{dist}_G(u_2, v)$. We compute $\text{dist}_G(u, v)$ for all u, v with Johnson's Algorithm [Cor+09] for sparse graphs.

The implementation for TC-LastC-reorder is the same with some small modifications — during the search tree algorithm, we do not try placing all free vertices into the next free position, but only the vertices that were present in the suffix of size c in L.

TC-RNeigh and TC-Wreach. In both approaches we are given a non-extendable subordering L of vertices S and a specific set $X \subseteq S$ of size c, and we want to replace vertices in X by (possibly) different vertices such that L becomes extendable. This is captured by the problem IC-Replace(r) from Section 5.3.

The first modification to the basic algorithm of both approaches we make is applying IC-Replace(r) multiple times, until we find an extendable subordering, where each time when we apply IC-Replace(r) we use a different set X. If we do not find an extendable subordering after the 10th application of IC-Replace(r), we report that turbocharging was not successful.

In the algorithm for IC-Replace(r) we first remove vertices X from L, keeping the positions $p_1 < p_2 < \cdots < p_c$ they were placed in unoccupied for now. We then apply a search tree algorithm that tries to fill these positions with free vertices from left to right. That means, in a search tree node we try all possibilities of placing a free vertex into position p_i and recurse into search tree nodes that try placing the remaining free vertices into position p_{i+1}, and so on, until all c positions are filled.

Every time when we place a free vertex into a position, or when we remove it again, we have to update some weakly left r-reaching sets of vertices of the subordering. But notice that these placements and removals of vertices are similar to rotations of vertices as defined in Section 6.3, so we only update a subset of weakly left r-reaching sets similar as in Section 6.3; we omit the details here.

Furthermore, computing weakly left r-reaching for a vertex $v \in S$ requires a depth-r breadth-first-search in G, where we have to make sure not to go left of v w.r.t. L. We cannot use $G[T]$ here, as the positions we are placing vertices into, are not to the right of the subordering.

If we place a vertex into position p_i, obtaining a subordering L_i, then we know that weakly left r-reachable sets of vertices that are placed in positions $p > p_i$ can only increase. Hence, if the weakly left r-reachable set of a vertex that is placed at such a position exceeds k, we can cut off the current branch. Weakly left r-reachable sets of free vertices $v \in T$ could decrease by placing them into a position p_j with $j > i$, but $\mathrm{Wreachleft}_r(G, L_i, v) \cap (\{v\} \cup \{u \mid p \leq p_i, \mathrm{pos}_{L_i}(u) = p\})$ is already fixed. We use this intersection for lower-bounding the weakly left r-reachable set of v, and cut off branches if its size exceeds k.

The implementations for TC-Wreach-reorder and TC-RNeigh-reorder are the same with some small modifications — in search tree nodes, we again only try placing free vertices from X, instead of vertices from $X \cup T$.

TC-Iterative-Swap. In TC-Iterative-Swap we try to fix a non-extendable subordering by iteratively applying rules that swap vertices in the subordering.

Most of the details for this approach can be found in Section 5.4 and Algorithm 5.1. There is only one optimization missing that is based on Lemma 6.3.3 of the next section. Namely, during swaps of vertices we do not have to update all weakly left r-reaching sets as done by Nadara et al. [Nad+19] in their local search implementation, but we only update weakly left r-reaching sets for some vertices.

Furthermore, notice that if we swap two neighboring vertices u and v in L with $u \prec_L v$ during TC-Rule 1, and $u \notin \mathrm{Wreachleft}_r(G, L, v)$, then we do not have to update any weakly left r-reaching sets.

TC-Merge. In TC-Merge we try to merge a set of vertices into an already fixed subordering. These vertices consist of weakly left r-reachable sets of overfull vertices in a non-extendable subordering. The main algorithm for this approach is already given in Algorithm 5.2. We now present further runtime optimizations that we have implemented.

Let L be the non-extendable subordering of vertices S that caused us to apply the turbocharging algorithm. Recall that we have a subordering L_1 of vertices $S_1 = S \setminus X$ and want to merge $S_2 = X$ into L_1; furthermore, we have free vertices $S_3 = V(G) \setminus (S_1 \cup S_2)$. This is captured in the problem WCOL-Merge(r) defined in Section 5.5.

As for TC-RNeigh and TC-Wreach, the first modification we make is applying WCOL-Merge(r) multiple times with different randomly selected different sets X as defined in Section 5.5, until we obtain an extendable subordering. Again, if we do not find an extendable subordering after the 10th application of WCOL-Merge(r), we report that turbocharging was not successful.

We now discuss the implementation of WCOL-Merge(r). As we do not know where we will place vertices S_2 into L_1, we need a data structure that allows such placements efficiently. We use an array A of size $|S_2|\cdot(|S_1|+1)+|S_1|$, such that for a vertex $v \in S_1$ we have that $A[i \cdot |S_2|+i] = v$ if $\text{pos}_{L_1}(v) = i$. Then, before and after each vertex $v \in S_1$, we have exactly $|S_2|$ free indices where we can place the vertices of S_2. If we want to place a vertex $w \in S_2$ between two vertices, then we have to move at most $|S_2|-1$ vertices of S_2 to a different index to make place for w, and a placement takes $\mathcal{O}(|S_2|)$ time. We only use this data structure during the turbocharging algorithm and not during the heuristic. We do not use other data structures such as linked lists that allow for these placements in constant time because our implementation is much simpler, and the time-consuming part of the algorithm is updating weakly left r-reaching sets — making the added time complexity of $\mathcal{O}(|S_2|)$ irrelevant.

Assume that we are in a search tree node of Algorithm 5.2, we have a subordering L of vertices S_1, and want to merge S_2 into L. Updating a weakly left r-reaching set for a vertex v is done by a depth-r breadth-first-search in G that does not go left of v w.r.t. the current subordering L. We can cut off branches of the search tree if $|\text{Wreachleft}_r(G[S_1 \cup S_3 \cup \{v\}], L, v)| > k$ for some $v \in S_1 \cup S_3$ because the sizes of those sets can only increase in subsequent search tree nodes. But we also know subsets of weakly left r-reaching sets of vertices S_2 that are already fixed. Namely, for all $v \in N^r_{G[\{u\}\cup S_3]}$ with $u \in S_2$, u will always be in the weakly left r-reachable set of v if u is placed somewhere into the ordering L. We use this to increase the lower bound for sizes of weakly left r-reachable sets of vertices in S_3.

6.3 Swapping and Rotations

In our implementations for turbocharging problems (see Section 6.2) we make use of two operations on suborderings. We want to look at these two operations — swaps and rotations — and their effect on weakly left r-reachable(reaching) sets. We will show that only a subset of weakly left r-reaching sets will change. Some of our observations are helpful for optimizing runtimes of turbocharging algorithms, as in our implementations

we keep track of all weakly left r-reachable sets and weakly left r-reaching sets for a subordering. Hence, when we swap or rotate vertices we only need to update a subset of the weakly left r-reaching sets.

Let us first define the operations formally.

Definition 6.3.1. Let L be a subordering of vertices $S \subseteq V$ with $u, v \in S$ s.t. $u \prec_L v$. By $\text{swap}_L(u, v)$ we denote the subordering that is obtained by *swapping* u and v, that is, if

$$L = (x_1, \ldots, x_i, u, x_j, \ldots, x_k, v, x_\ell, \ldots, x_n),$$

then

$$\text{swap}_L(u, v) = (x_1, \ldots, x_i, v, x_j, \ldots, x_k, u, x_\ell, \ldots, x_n).$$

Definition 6.3.2. Let L be a subordering of vertices $S \subseteq V$ with $u, v \in S$ s.t. $u \prec_L v$. By $\text{rotate}(u, v)$ we denote the subordering that is obtained by *rotating* the consecutive part of L between u and v inclusive by one step to the right, that is, if

$$L = (x_1, \ldots, x_i, u, x_j, \ldots, x_k, v, x_\ell, \ldots, x_n),$$

then

$$\text{rotate}(u, v) = (x_1, \ldots, x_i, v, u, x_j, \ldots, x_k, x_\ell, \ldots, x_n).$$

To keep track of a subordering L, we are always computing weakly left r-reaching sets of vertices, from which we can derive weakly left r-reachable sets. That is, if during some operation a weakly left r-reaching set of a vertex v changes, then we need a single breadth-first search of depth r to determine which vertices $U \subseteq V$ can now weakly left r-reach v; then we can simply add v to the weakly left r-reachable sets of all $u \in U$. In the following lemma we show that during a swap, only some weakly left r-reaching sets need to be updated.

Lemma 6.3.3. *Let L be a subordering of vertices $S \subseteq V$ with $u, v \in S$ s.t. $u \prec_L v$, and $L_s = \text{swap}_L(u, v)$. Let $B(u, v) = \{x \in S : u \prec_L x \prec_L v\}$ and let $w \in V(G)$. If $\text{Wreachleft}_r^{-1}(G, L, w) \neq \text{Wreachleft}_r^{-1}(G, L_s, w)$ then*

- *$w = u$ and $(B(u, v) \cup \{v\}) \cap \text{Wreachleft}_r^{-1}(G, L, u) \neq \emptyset$ or*
- *$w = v$ and $(B(u, v) \cup \{u\}) \cap \text{Wreachleft}_r(G, L, v) \neq \emptyset$ or*
- *$w \in B(u, v)$ and $w \in (\text{Wreachleft}_r(G, L, v) \cup \text{Wreachleft}_r^{-1}(G, L, u))$.*

Proof. We first notice that by swapping u and v, $\text{Wreachleft}_r^{-1}(G, L, w)$ cannot change if $w \notin (\{u, v\} \cup B(u, v))$. Indeed, paths in G with w as endpoint and that do not go left of w are the same w.r.t. L and L_s. We consider the three remaining cases:

1. $w = u$: In this case $\text{Wreachleft}_r^{-1}(G, L, w) \supsetneq \text{Wreachleft}_r^{-1}(G, L_s, w)$ because paths in G of length at most r that do not go left of w w.r.t. L_s also do not go left of w w.r.t. L. Thus, there exists a vertex x that weakly left r-reaches w in L but not in L_s. Consider a path P from w to x that results in $x \in \text{Wreachleft}_r^{-1}(G, L, w)$.

This path goes through $(B(w,v) \cup \{v\})$ as otherwise $x \in \text{Wreachleft}_r^{-1}(G, L_s, w)$ and hence $(B(w,v) \cup \{v\}) \cap \text{Wreachleft}_r^{-1}(G, L, w) \neq \emptyset$.

2. $w = v$: In this case $\text{Wreachleft}_r^{-1}(G, L, w) \subsetneq \text{Wreachleft}_r^{-1}(G, L_s, w)$ because paths in G of length at most r that do not go left of w w.r.t. L also do not go left of w w.r.t. L_s. Thus, there exists a vertex x that weakly left r-reaches w in L_s but not in L. Consider a path P from w to x that results in $x \in \text{Wreachleft}_r^{-1}(G, L_s, w)$. Again, this path goes through $(B(u,w) \cup \{u\})$. The vertex $y \in V(P) \cap (B(u,w) \cup \{u\})$ that is leftmost w.r.t. L must be in $\text{Wreachleft}_r(G, L, w)$, and hence $(B(u,w) \cup \{u\}) \cap \text{Wreachleft}_r(G, L, w) \neq \emptyset$.

3. $w \in B(u,v)$: As $\text{Wreachleft}_r^{-1}(G, L, w) \neq \text{Wreachleft}^{-1}(G, L_s, w)$, there exists a vertex x s.t. $x \in \text{Wreachleft}_r^{-1}(G, L, w)$ and $x \notin \text{Wreachleft}_r^{-1}(G, L_s, w)$, or there exists a vertex x s.t. $x \notin \text{Wreachleft}_r^{-1}(G, L, w)$ and $x \in \text{Wreachleft}_r^{-1}(G, L_s, w)$. We argue both cases:

   3.1 There is a vertex x s.t. $x \in \text{Wreachleft}_r^{-1}(G, L, w)$ and $x \notin \text{Wreachleft}_r^{-1}(G, L_s, w)$. Consider a path P from w to x that results in $x \in \text{Wreachleft}_r^{-1}(G, L, w)$. This path goes through v as otherwise $x \in \text{Wreachleft}_r^{-1}(G, L_s, w)$. Thus, $w \in \text{Wreachleft}_r(v)$ and $w \in (\text{Wreachleft}_r(G, L, v) \cup \text{Wreachleft}_r^{-1}(G, L, u))$.

   3.2 There is a vertex x s.t. $x \notin \text{Wreachleft}_r^{-1}(G, L, w)$ and $x \in \text{Wreachleft}_r^{-1}(G, L_s, w)$. Consider a path P from w to x that results in $x \in \text{Wreachleft}_r^{-1}(G, L_s, w)$. Similar to before, this path goes through u, and we have $w \in \text{Wreachleft}_r^{-1}(G, L, u)$. □

In a special setting no weakly left r-reaching sets and thus no weakly left r-reachable sets change at all.

Corollary 6.3.4. *Let L be a subordering of vertices $S \subseteq V$ with $u, v \in S$ s.t. $u \prec_L v$, and $L_s = \text{swap}_L(u,v)$. If $\text{Wreachleft}_r^{-1}(G, L, u) \cap (\{v\} \cup B(u,v)) = \emptyset$ and $\text{Wreachleft}_r(G, L, v) \cap (\{u\} \cup B(u,v)) = \emptyset$ then*

$$\forall w \in V : \text{Wreachleft}_r(G, L, w) = \text{Wreachleft}_r(G, L_s, w).$$

As we can model rotations by series of swaps, we can also find which weakly left r-reaching sets change during a rotation.

Corollary 6.3.5. *Let L be a subordering of vertices $S \subseteq V$ with $u, v \in S$ s.t. $u \prec_L v$, and $L_r = \text{rotate}_L(u,v)$. If $\text{Wreachleft}_r^{-1}(G, L, w) \neq \text{Wreachleft}_r(G, L_r, w)$ then*

- *$w = v$ and $\text{Wreachleft}_r(G, L, v) \cap (B(u,v) \cap \{u\}) \neq \emptyset$, or*
- *$w \in B(u,v) \cup \{u\}$ and $w \in \text{Wreachleft}_r(G, L, v)$.*

Proof sketch. The rotation is equivalent to multiple swaps of neighboring (w.r.t. L) vertices, where v is swapped with its left neighbor until v is at the required position. The statement follows immediately. □

In our implementations we also use more general swaps and rotations that include free vertices. Swaps of that kind replace a vertex $u \in S$ with v, making u free, and rotations place u between two vertices in L, making the rightmost vertex of L free. To model this, these swaps and rotations can be realized by placing u directly to the right of L updating weakly left r-reachable sets w.r.t. u, doing a swap or rotation,applying Lemma 6.3.3 or Corollary 6.3.5, removing the new rightmost vertex from the ordering again, and updating weakly left r-reachable sets w.r.t. the removed vertex. Thus, during these more general swaps and rotations we also only have to update a subset of weakly left r-reaching sets.

6.4 Considering Connected Components

In this section, we discuss exploiting connected components of the graph induced by free vertices of a subordering, to guide heuristics to make more local consecutive choices. Consider a subordering L of vertices S with free vertices T. Let $u, v \in T$ such that u and v reside in two different connected components of $G[T]$. Then $u \notin \text{Wreachleft}_r(G, L', v)$ and $v \notin \text{Wreachleft}_r(G, L', u)$ for any right extension L' of L. This observation can be utilized in the following way. Let L be a subordering of vertices S with free vertices T and let $v \in S$ be the rightmost vertex w.r.t. L. Assume that there exists a vertex $w \in T$ that is adjacent to v and let G_w be the connected component of $G[T]$ such that $w \in V(G_w)$. We can force the heuristic to place a vertex from $V(G_w)$ next because $V(G_w)$ is independent of $T \setminus V(G_w)$ w.r.t. weakly left r-reachable sets in any right extension of L. This pushes the heuristic to make "bad" choices earlier that would have appeared later if we chose the correct vertex w, and that the constructed subordering will be better — in the sense that adjacent vertices w.r.t. L will be in the same connected component w.r.t. the graph induced by free vertices during their placement. We leave as on open question, which vertex w adjacent to v to choose. If v does not contain an adjacent vertex $w \in T$, we can choose any vertex $w \in T$ and define G_w as above. Again, we can force the heuristic to choose a vertex from $V(G_w)$ next.

Additionally, we implement an optimization for the branching algorithm of the TC-LastC approach to reduce its search space that is based on this observation. Namely, for a subordering L with free vertices T, we do not have to consider all possibilities of placing vertices in the set T next; we can restrict the possibilities to $V(G_w)$ where G_w is defined as above.

We implement an optional program parameter that enables both optimizations — guiding left-to-right heuristics and reducing the search space of the TC-LastC approach. In all implementations the vertex w is chosen randomly.

6.5 Ordered Adjacency List

An optimization that applies to all approaches except TC-LastC and TC-LastC-reorder concerns the computation of weakly left r-reaching sets of vertices that are not rightmost w.r.t. a subordering L. Consider the following straight forward way of computing weakly

left r-reaching sets. Let L be a subordering of vertices S with free vertices T, $v \in S$ and let $i = \text{pos}_L(v)$. By setting $\text{pos}_L(u) = \infty$ for all $u \in T$, we can compute the weakly left r-reaching set of v by a depth-r breadth-first-search in G that does not go left of v w.r.t. L. Assume that this implementation uses a simple adjacency list for the graph G. If we encounter a vertex $w \in V$ with $\text{pos}_L(w) \geq v$ during the breadth-first-search, we must iterate over all vertices in the set $N^G(w)$, even though only $N^G(w) \cap \{u \in V(G) \mid v \preceq_L u\}$ is significant.

Instead, we can speed up this computation using adjacency lists, where we impose an order on the vertices $N^G(w)$ for a vertex $w \in V(G)$ that corresponds with the order L. Namely, our implementations include an adjacency list for G, that is an array of sorted sets; an entry of the array corresponds to the vertex set $N^G(w)$ for a vertex $w \in V(G)$; the vertices $N^G(w)$ are sorted w.r.t. L, that is, a vertex u_1 comes before u_2 in the sorted set for w if and only if $\text{pos}_L(u_1) < \text{pos}_L(u_2)$. Note that we again assume $\text{pos}_L(u) = \infty$ for free vertices $u \in T$ w.r.t. the subordering L. Ties between free vertices for a sorted set are broken arbitrarily by assigning numbers to vertices. We can now optimize the breadth-first-search that does not go left of v w.r.t. L using this new adjacency list: If we encounter a vertex $w \in V$ with $\text{pos}_L(w) \geq v$ during the breadth-first-search, we can iterate the sorted set corresponding to $N^G(w)$ from the end to the beginning until we encounter a vertex that is left of v w.r.t. L. This allows us to only iterate over the vertex set $N^G(w) \cap \{u \in V(G) \mid v \preceq_L u\}$, as we wanted.

In this approach, updating the adjacency list requires non-negligible time whenever a position of a vertex changes. If the position of a vertex $w \in V(G)$ changes, we have to update all sorted sets for neighbors $N^G(w)$ of w. As we use sorted sets this takes $\sum_{u \in N^G(w)} \log|N^G(u)|$ time. We discovered that in some cases (see Chapter 8) not using this ordered adjacency list might be an improvement. Due to this problem, we do not always use the ordered adjacency list in our implementations, but rather add an optional program parameter to enable this feature.

6.6 Lower Bounds

Lower bounds are crucial for the design of search tree algorithms as they might reduce the search space drastically by recognizing parts of solutions that cannot be extended to meet some quality criteria. In our case, we consider a subordering L of vertices $S \subseteq V(G)$. Lower bounds can be helpful for all our turbocharging approaches, as they recognize suborderings that are not extendable anymore, during the heuristic and/or during some of our turbocharging approaches. If said lower bound exceeds our target value k for turbocharging, we know that there is no full right extension L' of L such that $\text{wcol}_r(G, L') \leq k$. The simplest lower bound for the weak r-coloring number of any full right extension is

$$\max_{v \in V(G)} |\text{Wreachleft}_r(G, L, v)|.$$

In this section we explore tighter bounds. We will introduce a lower bound that is similar to degeneracy, but also considers weakly left r-reachable sets. We will then adjust this

lower bound such that it also considers the radii r by contracting some parts of a graph into single vertices.

6.6.1 Lower Bound Based on f-degeneracy

From Section 3.1 we know that degeneracy$(G) + 1 \leq \text{wcol}_r(G)$ for all graphs G and $r \geq 1$. Considering a subordering L of vertices $S \subseteq V(G)$, we want to define a variant of degeneracy for the graph induced by the free vertices T that considers the subordering L, that can be computed in polynomial time, and that is a lower bound for the weak r-coloring number of any full right extension of L. Note that the degeneracy of $G[T]$ is also a lower bound, but we will also consider weakly left r-reachable sets induced by vertices from S. We first define the general notion of f-degeneracy.

Definition 6.6.1. Let G be a graph and $f : V(G) \to \mathbb{N}$ be a function. We define the f-degeneracy of G as

$$f\text{-degeneracy}(G) = \min_{L \in \Pi(G)} \max_{v \in V(G)} \{f(v) + |\{v' \in N_G(v) \mid v' \prec_L v\}|\}$$

The f-degeneracy of a graph G and a function f is similar to the degeneracy of G, but the cardinality of adjacent vertices left of a vertex v are offset by the value $f(v)$. In the following lemma we show that f-degeneracy can be used as a lower bound for weak r-coloring numbers of right extensions by setting $f(v) = |\text{Wreachleft}_r(G, L, v)|$ for a subordering L.

Lemma 6.6.2. *Let G be a graph, L be a subordering of vertices S with free vertices T, and let $f(v) = |\text{Wreachleft}_r(G, L, v)|$ for $v \in T$; let L' be any full right extension of L. Then*

$$f\text{-degeneracy}(G[T]) \leq \max_{v \in T} |\text{Wreachleft}_r(G, L, v)|.$$

Proof. Consider the subordering $L_T = L[T]$. Then

$$|\text{Wreachleft}_r(G, L, v)| + |\{v' \in N_{G[T]}(v) \mid v' \prec_{L_T} v\}| \leq |\text{Wreach}_r(G, L', v)|$$

for each $v \in T$ because $\{v' \in N_{G[T]}(v) \mid v' \prec_{L_T} v\} \subseteq \text{Wreach}_r(G, L', v)$. The claim follows immediately. □

An interesting question is, in which cases f-degeneracy$(G[T])$ is a tight bound. That is, let L' be a full right extension of L such that $\max_{v \in T} |\text{Wreachleft}_r(G, L, v)|$ is minimal. We would like to know when f-degeneracy$(G[T]) = \max_{v \in T} |\text{Wreachleft}_r(G, L', v)|$. Of course that is the case for $r = 1$. We propose an open question: For which graph classes $G[T]$, functions f and radii r does this also holds?

We also want to mention that the f-degeneracy is a "better" lower bound than weakly left r-reachable sets, as

$$\max_{v \in T} |\text{Wreachleft}_r(G, L, v)| \leq f\text{-degeneracy}(G)$$

Algorithm 6.1: f-degeneracy

Input: Graph $G = (V, E)$ and function $f : V \to \mathbb{N}$
Output: f-degeneracy(G)

```
answer ← 0;
while V ≠ ∅ do
    select vertex v from V that has minimum d_G(v) + f(v);
    answer ← max(answer, d_G(v) + f(v));
    for w ∈ N_G(v) do
        remove edge {v, w} from G;
    end
    V ← V \ {v};
end
return answer;
```

for suborderings L with free vertices T.

For a lower bound to be useful during search tree algorithms, we need to be able to compute it efficiently. The next lemma shows that this is possible for the f-degeneracy.

Lemma 6.6.3. *Let $G = (V, E)$ be a graph and $f : V \to \mathbb{N}$ be a function. Then f-degeneracy(G) can be computed in time $\Theta((|V|+|E|) \cdot \log|V|)$.*

Proof. The algorithm that computes the f-degeneracy is depicted in Algorithm 6.1. It repeatedly removes the vertex v of minimum $d_G(v) + f(v)$ from G. The answer is the maximum over all $d_G(v) + f(v)$ for all removed vertices v at the time of their removal. Notice that this algorithm is similar to the standard algorithm that computes the degeneracy of a graph (see [MB83]). We now argue correctness and runtime.

Correctness. Let v_i be the i-th vertex of V that is removed during Algorithm 6.1. Furthermore, let $G_0 = G$ and $G_i = G_{i-1} - v_i$ for $1 \leq i \leq |V|$. Notice that G_{i-1} contains the same edges as the variable G during the removal of vertex v_i. Consider the ordering L of vertices V that achieves minimum $\max_{v \in T}\{f(v) + |\{v' \in N_G(v) \mid v' \prec_{L_T} v\}|\}$. Let L_{G_i} be the ordering of vertices $V(G_i)$ such that $u_1 \preceq_{L_{G_i}} u_2$ if and only if $u_1 \preceq_{L_T} u_2$ for $u_1, u_2 \in V(G_i)$. Considering the rightmost vertex v w.r.t. L_{G_i}, we have that

$$\begin{aligned} d_{G_{i-1}}(v_i) + f(v_i) &\leq |\{v' \in N_{G_i}(v) \mid v' \preceq_{L_{G_i}} v\}| + f(v) \\ &\leq |\{v' \in N_G(v) \mid v' \preceq_L v\}| + f(v) \\ &\leq f\text{-degeneracy}(G), \end{aligned}$$

as we always choose the vertex v_i with minimum $d_{G_{i-1}}(v_i) + f(v_i)$. This is true for each $v_i, 1 \leq i \leq |V|$, hence our algorithm returns at most f-degeneracy(G).

Let L' be the ordering of vertices V such that $v_i \preceq v_j$ iff $i \geq j$. Clearly, Algorithm 6.1 returns $\max_{v \in V}\{f(v) + |\{v' \in N_G(v) \mid v' \prec_{L'} v\}|\}$, thus it returns at least f-degeneracy(G). Hence, Algorithm 6.1 returns f-degeneracy(G).

Runtime. We can keep track of values $d_G(v) + f(v)$ using an ordered set. Then Line 3 runs in $\mathcal{O}(\log|V|)$ steps. Furthermore, we remove each edge in $E(G_T)$ at most once in Line 6, and update the values in our ordered set. Edge removal can be implemented in constant time using an array of linked lists as adjacency list. The outer loop runs $|V|$ times. This leads to a runtime of

$$\Theta((|V|+|E|)\log|V|+|E|) = \Theta((|V|+|E|)\log|V|). \qquad \square$$

We observe that $\min_{v\in V(G')}\{f(v) + d_{G'}(v)\} \leq f\text{-degeneracy}(G)$ for any subgraph G' of G. This will be useful for the next lower bound that we propose.

6.6.2 Contracting Subgraphs of Diameter $\lfloor\frac{r-1}{2}\rfloor$

By setting $f(v) = |\text{Wreachleft}_r(G, L, v)|$, the value f-degeneracy$(G[T])$ does not consider the radii r besides for the function f. This makes f-degeneracy impractical as a lower bound for larger values of r, as weak r-coloring numbers grow for larger values of r. We want to introduce an adaptation of f-degeneracy, that is inspired by the MMD+ algorithm of Bodlaender and Koster [BK11] and considers the radius r. The MMD+ algorithm computes a lower bound for treewidth, that combines degeneracy computation with arbitrary edge contractions. In our case we cannot use arbitrary edge contractions, but as we will see below we can contract subgraphs of diameter $\lfloor\frac{r-1}{2}\rfloor$ into a single vertex.

Let us first recall minors and minor models from Section 3.2: A graph H is a *minor* of a graph G, if there are pairwise vertex-disjoint connected subgraphs $H_1, \ldots, H_n$ of G such that whenever $\{v_i, v_j\} \in E(H)$, there are $u_i \in V(H_i)$ and $u_j \in V(H_j)$ with $\{u_i, u_j\} \in E(G)$. We then call $(H_1, \ldots, H_n)$ a *minor model* of H in G, and let $\phi(v_i) = H_i$.

Let us now formulate the key lemma for this new lower bound.

Lemma 6.6.4. *Let $G = (V, E)$ be a graph and L be a subordering of vertices $S \subseteq V$ with free vertices T. Furthermore, let H be a minor of $G[T]$ such that for its minor model $(H_1, \ldots, H_n)$ each $H_i, i \in \{1, \ldots, n\}$, has diameter at most $\lfloor\frac{r-1}{2}\rfloor$; lastly, define $f(v) = \max_{u\in\phi(v)}|\text{Wreachleft}_r(G, L, u)|$ for each $v \in V(H)$. Then*

$$f\text{-degeneracy}(H) \leq \max_{v\in T}|\text{Wreach}_r(G, L', v)|$$

for each full right extension L' of L.

Proof. Let L' be any full right extension of L. We construct an ordering L_H of the vertex set $V(H)$. For each $v \in V(H)$, let $\alpha(v) \in V(\phi(v))$ be the leftmost vertex w.r.t. L'. We define L_H such that $v_1 \prec_{L_H} v_2$ if and only if $\alpha(v_1) \prec_{L'} \alpha(v_2)$. We claim that

$$f(v) + |\{v' \in N_H(v) \mid v' \prec_{L_H} v\}| \leq \max_{u\in V(\phi(v))}|\text{Wreach}_r(G, L', u)|$$

for all $v \in V(H)$. We start with an arbitrary vertex $v \in V(H)$. Let $\beta(v) \in V(\phi(v))$ be any vertex with $|\text{Wreachleft}_r(G, L, \beta(v))| = f(v)$. Consider an arbitrary vertex $v' \in N_H(v)$ with $v' \preceq_{L_H} v$. There is a path P_1 of length at most $\lfloor\frac{r-1}{2}\rfloor + 1$ from $\beta(v)$ to some vertex in

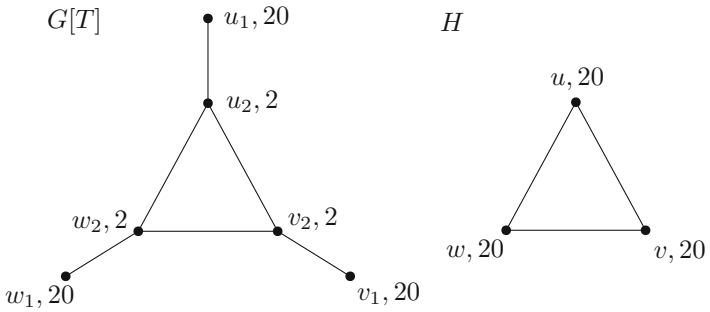


Figure 6.1: Counterexample for contracting subgraphs of diameter more than $\lfloor \frac{r-1}{2} \rfloor$. The integer values besides the node labels represent $|\text{Wreachleft}(G, L, v)|$ for $v \in V(G[T])$ and $f(v)$ for $v \in V(H)$, respectively.

$u \in V(\phi(v'))$ because $\phi(v)$ has diameter at most $\lfloor \frac{r-1}{2} \rfloor$. Furthermore, there is a path P_2 from u to $\alpha(v')$ of length at most $\lfloor \frac{r-1}{2} \rfloor$. The path P that is a concatenation of P_1 and P_2 is a path of length at most r that goes from $\beta(v)$ to $\alpha(v')$ and that does not go left of $\alpha(v')$ w.r.t. L', meaning $\alpha(v') \in \text{Wreach}_r(G, L', \beta(v))$. As $\alpha(v')$ is different for each $v' \in \{v' \in N_H(v) \mid v' \prec_{L_H} v\}$ and $\alpha(v') \in T$, we have that

$$\begin{aligned} f(v) + |\{v' \in N_H(v) \mid v' \prec_{L_T} v\}| &= f(\beta(v)) + |\{v' \in N_H(V) \mid v' \prec_{L_T} v\}| \\ &\leq f(\beta(v)) + |\text{Wreach}_r(G, L', \beta(v)) \cap T| \\ &\leq \max_{u \in V(\phi(v))} |\text{Wreach}_r(G, L', u)|, \end{aligned}$$

which concludes the proof. □

To complement the relevance of Lemma 6.6.4, we want to show that we really only can contract subgraphs of diameter not larger than $\lfloor \frac{r-1}{2} \rfloor$ with the following example.

Example 6.6.5. In this example we assume that $r = 2$. Consider the graphs $G[T]$ and H from Figure 6.1. The graph $G[T]$ represents the subgraph induced by vertices T, which are free vertices w.r.t. a subordering L. Integer labels to the side of node labels represent the value $|\text{Wreachleft}_r(G, L, x)|$ for $x \in T$ (those exact values are obtainable for a subordering L and a graph G). The graph H is a minor of $G[T]$ with minor model $(G[\{u_1, u_2\}], G[\{v_1, v_2\}], G[\{w_1, w_2\}])$ where each subgraph has diameter one, and the integer values represent $f(x) = \max_{y \in V(\phi(x))} |\text{Wreachleft}_r(G, L, y)|$ for $x \in V(H)$. Notice that f-degeneracy$(H) = 22$. But ordering the vertices from $G[T]$ s.t. $u_1 \prec v_1 \prec w_1 \prec u_2 \prec v_2 \prec w_2$ for a full right extension L' of L we have

$$|\text{Wreach}_r(G, L', v)| \leq 20$$

for all $v \in V(G[T])$. This means that f-degeneracy(H) is no longer a lower bound and for $r = 2$ we cannot contract subgraphs of diameter one.

We present Algorithm 6.2 that computes a lower bound for the weak r-coloring number of any full right extension of a subordering L, and makes direct use of Lemma 6.6.4.

Algorithm 6.2: WCOL-MMD+

Input: A graph $G = (V, E)$, a subordering L of vertices $S \subseteq V$ with free vertices T

Output: A lower bound for the weak r-coloring number of any full right extension of L

```
1  answer ← 0;
2  H ← G[T];
3  f(v) = |Wreachleft_r(G, L, v)| for v ∈ T;
4  while V(H) ≠ ∅ do
5  |  select v from V(H) that has minimum d_G(v) + f(v);
6  |  answer← max(answer, d_H(v) + f(v));
7  |  if ∃w ∈ N_H(v) : diam(G[V(φ(v)) ∪ V(φ(w))]) ≤ ⌊(r−1)/2⌋ then
   |  |  // Strategy can be used for selection of w
8  |  |  Contract v and w in H, creating a new vertex u with
   |  |    f(u) = max(f(v), f(w));
9  |  else
10 |  |  H ← H − v
11 |  end
12 end
13 return answer;
```

The values $\phi(v)$ for $v \in V(H)$ refer to the contracted subgraphs with respect to the minor H of $G[T]$. Comparing Algorithm 6.2 to Algorithm 6.1, instead of removing the vertex v of minimum $d_H(v) + f(v)$, we first check if there is a vertex w that we can merge onto v. This is possible if $G[V(\phi(v)) \cup V(\phi(w))]$ has diameter at most $\lfloor \frac{r-1}{2} \rfloor$. Contracting v and w yields a new vertex u that has $f(u) = \max(f(v), f(w))$. We call this algorithm WCOL-MMD+ inspired by the MMD+ algorithm by Bodlaender and Koster [BK11]. WCOL-MMD+ returns a lower bound for the weak r-coloring number of any full right extension because of Lemma 6.6.4, as we only contract subgraphs of diameter at most $\lfloor \frac{r-1}{2} \rfloor$, and we set $f(v) = \max_{u \in \phi(v)} |\text{Wreachleft}_r(G, L, u)|$.

We can use different strategies for the selection of w. For our implementations we select a vertex w that has minimum $f(w) + d_H(w)$, to potentially increase values $f(v) + d_H(w)$ in H for vertices $v \in V(H)$. This strategy is also similar to a strategy for the MMD+ algorithm mentioned by Bodlaender and Koster.

The WCOL-MMD+ algorithm has high time complexity, as there are no fast algorithms for computing the diameter in Line 7 of Algorithm 6.2. The best algorithms we know of are Floyd Warshall's Algorithm [Cor+09] that runs in $\mathcal{O}(|V|^3)$ time and Johnson's Algorithm [Cor+09] for sparse graphs that runs in $\mathcal{O}(|V|^2 \cdot \log|V| + |V| \cdot |E|)$. There is also strong evidence that the diameter cannot be computed in subquadratic time as this would otherwise refute the orthogonal vectors conjecture [AWW16]. This conjecture is implied by the Strong Exponential Time Hypothesis (SETH) which is widely believed, further supporting this evidence. That means that we cannot really

use WCOL-MMD+ during search tree algorithms as it will be invoked a lot of times. To tackle this problem, we adapt WCOL-MMD+ by upper-bounding the diameter of contracted subgraphs. We use the simplest upper bound — the number of vertices minus one. This upper bound is easy to compute and implement, and we did not find tighter upper bounds that work better during our research. We replace the if-statement in Line 7 by $\exists w \in N_H(v) : |V(\phi(v))|+|V(\phi(w))|-1 \leq \lfloor \frac{r-1}{2} \rfloor$, calling the newly obtained algorithm WCOL-UB-MMD+. WCOL-UB-MMD+ will be used as lower bound during branching in turbocharging algorithms and during heuristics, as it performs better than only the f-degeneracy in most cases.

Finally, we want to mention that the algorithms WCOL-MMD+ and WCOL-UB-MMD+ can also be used for computing lower bounds for weak r-coloring of graphs G, by letting the initial subordering L of vertices be empty. In that case, $T = V(G), S = \emptyset$ and $|\text{Wreachleft}_r(G, L, v)| = 1$ for all $v \in V(G)$.

In our implementations we add an optional program parameter that enables lower bounding by WCOL-UB-MMD+. For a subordering L of vertices S, let k_{LB} be the lower bound computed by WCOL-UB-MMD+. We use $\max(k_{\text{LB}}, \max_{v \in S} |\text{Wreachleft}_r(G, L, v)|)$ as lower bound for the weak r-coloring number of any right extension of L during heuristics and for the turbocharging approaches TC-LastC(-reorder), TC-Iterative-Swap, and TC-Merge. We do not see a simple way to use it for TC-RNeigh-(reorder) and TC-Wreach-(reorder) because of the implementation details we chose for those approaches: During the placement of a vertex in the search tree algorithm, the size of the weakly left r-reachable set of this vertex might decrease, which prohibits us from using free vertices in the lower bound.

Additionally, note that complexity results for our turbocharging problems are based on the lower bound $\max_{v \in V} |\text{Wreachleft}_r(G, L, v)|$, because all problems pose the question of finding a subordering that is extendable with respect to this lower bound. When applying another lower bound such as WCOL-MMD+, the complexity results may change, but we strongly believe they do not.

Furthermore, we have to adapt some turbocharging approaches when applying this lower bound. Notice that if the lower bound based on the WCOL-UB-MMD+ reports that a subordering L is not extendable, it could be that there is no overfull vertex. But the problem we want to solve in TC-Merge is almost entirely based on weakly left r-reachable sets of overfull vertices. If the lower bound WCOL-UB-MMD+ reports non-extensibility during the heuristic, we tackle this problem, by instead considering weakly left r-reachable sets of the last vertex that was placed by the heuristic for merging. Similarly, we adapt TC-Rule 1 of TC-Iterative-Swap such that it selects a vertex v for swapping such that v is chosen exactly as in TC-Rule 2.

Furthermore, we add another optional program parameter that disables lower bounding by weakly left r-reachable sets of free vertices. That is, for a subordering L of free vertices S, we apply $\max_{v \in S} |\text{Wreachleft}_r(G, L, v)|$ as the lower bound for the weak r-coloring number of any full right extension L' of L if this program parameter is enabled.

6.7 Discussion

In this chapter, we have given a more detailed look on our implementations for approaches that we use to turbocharge left-to-right heuristics. We have explained details of search tree algorithms, data structures that we use, and given some optimization methods — some of which required formal proofs. As most of our approaches have a heuristic character, there is still additional fine-tuning that can be done. We are also certain that there are many possibilities for optimizations that need to be investigated such as lowering the search space of search tree algorithms.

Additionally, we have proposed two lower bounds for weak r-coloring numbers of full right extensions of a subordering. We are curious whether these lower bounds can be made even tighter, and if they give new insights for lower bounds regarding weak r-coloring numbers of graphs. Another interesting question is, under which conditions our provided lower bounds for weak r-coloring numbers are tight. Based on the difference between good lower bounds and computed upper bounds for weak r-coloring numbers, we could also evaluate the performance of algorithms for computing weak r-coloring numbers.

CHAPTER 7

Right-to-Left Heuristics and Turbocharging

In this chapter, we want to start investigating turbocharging for right-to-left heuristics for weak coloring numbers. We will proceed as in Chapter 5 by first giving some simple observations for suborderings if we know that remaining vertices will be placed left of that subordering. Then we will define a simple turbocharging problem and prove some complexity results. At the end of the chapter we will give a lower bound for the weak r-coloring number of an ordering that is obtained by placing vertices left of a subordering.

7.1 Definitions and First Observations

For right-to-left heuristics we know that, given a subordering L of vertices S, the remaining free vertices T will be placed to the left of L. Motivated by this, we define the following notion.

Definition 7.1.1. Given a subordering L of vertices $S \subseteq V$, a *left extension* of L is a subordering L' of vertices S' such that

- $S' \supseteq S$,
- $L'[S] = L$, and
- $u \succ_{L'} v$ for all $u \in S, v \in S' \setminus S$.

Informally, S' is a subordering that places vertices $S' \setminus S$ to the left of vertices in S. If $S' = V$, then we say that L' is a *full left extension*.

Thus, given a subordering L, in one step, a right-to-left heuristic will generate a left extension of L by adding one vertex to the left. We might want to ask, which

parts of weakly r-reachable sets of vertices are already fixed with respect to any full left extension L' of L. For this, we define the following notions, which will lead to an answer of the stated question in Lemma 7.1.3.

Definition 7.1.2. Let G be a graph. Given a subordering L of vertices $S \subseteq V(G)$ with free vertices T, a vertex $u \in V(G)$ is *weakly right r-reachable* from $v \in S$ with respect to L if

- $u \in \text{Wreach}_r(G[S], L, v)$, or
- there exists a path P of length at most r from v to u in G such that $V(P) \cap T = \{u\}$.

Let $\text{Wreachright}_r(G, L, v)$, the *weakly right r-reachable* set of v w.r.t. L, be the set of vertices that are weakly right r-reachable from v in G with respect to L. Let $\text{Wreachright}_r^{-1}(G, L, v) = \{u \in V : v \in \text{Wreachright}_r(G, L, u)\}$, calling this the set of *weakly right r-reaching set* of u.

Note that $\text{Wreachright}_r(G, L, v)$ is only defined for $v \in S$. Furthermore, notice that $\text{Sreach}_r(G, L', v) \subseteq \text{Wreachright}_r(G, L, v)$ for any full left extension L' of L, where $\text{Sreach}_r(G, L', v)$ is the strongly r-reachable set of v w.r.t. L' (see Section 3.1). We now show that weakly right r-reachable sets can only increase for left extensions.

Lemma 7.1.3. *Let L be a subordering of vertices $S \subseteq V$ with free vertices T and let L' be a left extension of L. Then*

- $\text{Wreachright}_r(G, L, v) \subseteq \text{Wreachright}_r(G, L', v)$ *for all $v \in S$.*

Proof. Let $v \in S$ and let $u \in \text{Wreachright}_r(G, L, v)$. We show that $u \in \text{Wreachright}_r(G, L', v)$. There are two cases per Definition 7.1.2:

- Case 1: $u \in \text{Wreach}_r(G[S], L, v)$. We know that $G[S'] \supset G[S]$ and $L'[S] = L$, hence $u \in \text{Wreach}_r(G[S'], L', v)$.
- Case 2: There exists a path of length at most r from v to u in G such that $V(P) \cap T = \{u\}$. Let $T' = V(G) \setminus S'$ be the free vertices with respect to L'. If $u \in T'$, then $V(P) \cap T' = \{u\}$, and $u \in \text{Wreachright}_r(G, L', v)$ because of the same path P. If $u \in S'$, then notice that $u \in \text{Wreach}(G[S'], L', v)$ because of P, as this path certainly does not go left of u w.r.t. L' as L' is a left extension of L.

In both cases $u \in \text{Wreachright}_r(G, L', v)$, which concludes the proof. □

This leads us to non-extensibility and quality of a subordering when considering right-to-left heuristics and turbocharging problems for right-to-left heuristics. In this chapter, the quality of a subordering is defined as $\max_{v \in S} |\text{Wreachright}_r(G, L, v)|$, because this certainly is a lower bound for the weak r-coloring of any full left extension L' of L. Furthermore, we say that a subordering is *non-extendable* if $\max_{v \in S} |\text{Wreachright}_r(G, L, v)| > k$. Turbocharging problems will be designed such that they take a non-extendable subordering L and make it extendable.

Definition 7.1.2 and Lemma 7.1.3 also enable us to give an alternative definition for the Sreach-Heuristic proposed by Nadara et al. [Nad+19]. Namely, given a subordering L of vertices S with free vertices T, we define the set of *potentially strongly r-reachable* vertices for $v \in T$ as $\text{Wreachright}_r(G, L', v)$, where L' is obtained from L by placing v to the left of L. These vertices are called potentially strongly r-reachable as $\text{Sreach}_r(G, \overline{L}, v) \subseteq \text{Wreachright}_r(G, L', v)$ for any full left extension $\overline{L}$ of L'. The Sreach-Heuristic places the vertex $v \in T$ with smallest potentially strongly r-reachable set left of L. Notice that for $r = 1$ this results in an exact algorithm for computing $\text{wcol}_1(G) = \text{degeneracy}(G) + 1$ [MB83].

We continue by presenting a simple turbocharging problem and a turbocharging approach that is based on in.

7.2 Turbocharging Problem and its Complexity

Motivated by the turbocharging problem for treewidth introduced by Gaspers et al. [Gas+19] that replaces a suffix of size c for an ordering, we present a similar turbocharging problem that replaces the prefix of a subordering of size c.

Incremental Conservative Right Weak r-coloring (IC-WCOL-RIGHT(r))	
Input:	A graph $G = (V, E)$, a subordering L of vertices $S \subseteq V$, and positive integers k and c.
Question:	Is there an extendable left extension L' of L of vertices S' with $\lvert S' \setminus S \rvert = c$?

This problem is similar to IC-WCOL-LEFT(r), but instead of extending to the right, we extend the subordering to the left.

An interesting observation is that by our definition of weakly right r-reachable sets, including potentially strongly r-reachable sets, the problem is solvable in polynomial time for $r = 1$ by greedily taking the vertex with smallest potentially strongly r-reachable set (see [MB83] for the algorithm idea). In Theorems 7.2.1 and 7.2.2 we show two hardness results for IC-WCOL-RIGHT(r) for $r \geq 3$.

Theorem 7.2.1. *IC-WCOL-RIGHT(r) is NP-hard for each fixed $r \geq 3$.*

Proof. We give a reduction from Independent Set which is NP-complete [GJ79]. Let (G, p) be an instance of Independent Set. We construct an instance (G', L, k, c) of IC-WCOL-RIGHT(r) as follows. First, let $k = 2r$ and $c = 2p$. We obtain G' from G by adding the following vertices and edges.

- We subdivide each edge $\{u, v\} \in E(G)$ $2r-3$ times, introducing $2r-3$ new vertices $x_1^{uv}, \ldots, x_{2r-3}^{uv}$, ordered following the path from u to v.
- For each vertex $u \in V$, we introduce $2r = k$ vertices $u', y_1^u, \ldots, y_{2r-1}^u$. We add edges such that vertices $u', y_1^u, \ldots, y_{2r-1}^u$ form a clique. Furthermore, we add the edge $\{u, u'\}$.

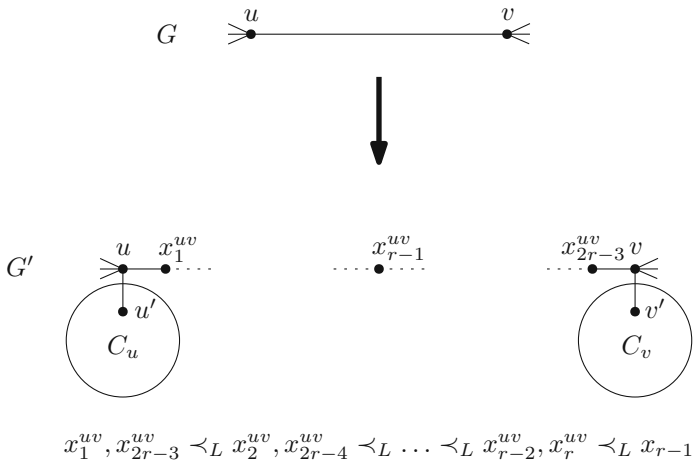


Figure 7.1: Sketch for subdivision of edge $\{u, v\}$.

It is worth mentioning that x_{r-1}^{uv} lies exactly in the middle of the shortest path from u to v and that the shortest paths from x_{r-1}^{uv} to u' and v' have length r. Let L be any subordering of the vertex set S, where

$$S = \{x_i^{uv} \mid \{u, v\} \in E(G), 1 \leq i \leq 2r-3\} \cup \{y_i^u \mid u \in V(G), 1 \leq i \leq 2r-1\},$$

and that satisfies

$$\forall \{u, v\} \in E(G) \forall i, j \in [2r-3] : |i-(r-1)| > |j-(r-1)| \Rightarrow x_i^{uv} \prec_L x_j^{uv}.$$

Notice that such a subordering exists and that this condition enforces that all vertices x_i^{uv} with $1 \leq i \leq 2r-3$ are in the weakly right r-reachable set of x_{r-1}^{uv} w.r.t. L. A sketch for this construction for an edge $\{u, v\} \in E(G)$ is shown in Figure 7.1. The circles labelled C_u and C_v denote the cliques $G'[\{u', y_1^u, \ldots, y_{2r-1}^u\}]$ and $G'[\{v', y_1^v, \ldots, y_{2r-1}^v\}]$ of size k for vertices u and v.

Correctness. We continue by arguing correctness of the reduction. That is, we show that G contains an independent set of size p if and only if there exists an extendable left extension L' of L on vertices S' such that $|S' \setminus S| = 2p = c$. We argue forwards and backwards direction of the stated equivalence.

"⇒": Let I be an independent set of size p in G. We obtain L' from L by adding the vertices $I \cup \{u' \mid u \in I\}$ to the left such that $u' \prec_{L'} u$ for all $u \in I$, otherwise ordered arbitrarily. It remains to show that the weakly right r-reachable sets of vertices in L' do not exceed size k.

- We have $|\text{Wreachright}_r(G', L', u)| = |\{u, u'\}| \leq k$ for all $u \in I$. All other vertices that are not right of u w.r.t. L' have distance more than r to u because of the subdivision of edges. Notice that $2r-2 > r$ holds for $r \geq 3$, where $2r-2$ is the length of the shortest path from u to v in G' for $\{u, v\} \in E(G)$.

- $|\text{Wreachright}_r(G', L', u')| = |\{u'\}| \leq k$ for all $u \in I$. Again, all other vertices that are not right of u' w.r.t. L' have distance more than r to u' because of the subdivision of edges.
- $\text{Wreachright}_r(G', L', y_i^u) \subseteq \{u', y_1^u, \ldots, y_{2r-1}\}$ for all $u \in V(G)$ and $1 \leq i \leq 2r-1$ because if u' is placed in L', then u is also placed right of u', and u' separates all y_i^u with $1 \leq i \leq 2r-1$ from the rest of G'.
- Notice that for all $\{u, v\} \in E(G)$ and $1 \leq i \leq 2r-3$

  $$\text{Wreachright}_r(G', L', x_i^{uv}) \subseteq \text{Wreachright}_r(G', L', x_{r-1}^{uv}),$$

  as x_{r-1}^{uv} is the rightmost vertex w.r.t. L' considering the subdivision between u and v, and u and v separate the vertices introduced during the subdivision from the rest of the graph. Furthermore,

  $$\text{Wreachright}_r(G', L', x_{r-1}^{uv}) \subseteq \{u, v, u', v', x_1^{uv}, \ldots, x_{2r-3}^{uv}\}.$$

  It remains to argue that u' and v' cannot be both in $\text{Wreachright}_r(G', L', x_{r-1}^{uv})$. But this is clearly true, as I is an independent set. If u' and v' both were in $\text{Wreachright}_r(G', L', x_{r-1}^{uv})$, then u and v must have been placed in L', contradicting that I is an independent set.

No weakly right r-reachable set exceeds size $k = 2r$, so we can conclude this direction.

"$\Leftarrow$": Let L' be an extendable left extension of L of vertices S' such that $|S' \setminus S| = c = 2p$. We observe that, if $u' \in S'$ for a vertex $u \in V(G)$, then $u \in S'$ must hold, because otherwise $|\text{Wreachright}_r(G', L', y_i^u)| = |\{u, u' y_1^u, \ldots, y_{2r-1}^u\}| > 2r$ for the rightmost $y_i^u, i \in \{1, \ldots, 2r-1\}$ w.r.t. L'. Furthermore, if $\{u, u'\} \subseteq S'$ for $u \in V(G)$ then $u' \prec_{L'} u$ must hold because of the same reason. As $u' \in S'$ implies $u \in S'$, we have that $|S' \cap V(G)| \geq p$. We let $I \subseteq |S' \cap V(G)|$ be any set of size p, which clearly exists. We claim that I is an independent set in G by contradiction. Assume that there exists an edge $\{u, v\} \in E(G)$ with $\{u, v\} \subseteq I \subseteq S'$. If $u' \in S'$, we have seen that $u' \prec_{L'} u$ holds. Hence, $u' \in \text{Wreachright}_r(G', L', x_{r-1}^{uv})$ follows because of the construction of L. If $u' \notin S'$, then $u' \in \text{Wreachright}_r(G', L', x_{r-1}^{uv})$ because of the construction of L and the definition of weakly right r-reachable sets. Equivalently, we argue $v' \in \text{Wreachright}_r(G', L', x_{r-1}^{uv})$. We conclude $|\text{Wreachright}_r(G', L', x_{r-1}^{uv})| = |\{u, v, u', v', x_1^{uv}, \ldots, x_{2r-3}^{uv}\}| > 2r = k$, which clearly is a contradiction to the assumption that L' is extendable. □

We now see why the proof fails for $r = 2$: The subdivision of edges $\{u, v\}$ creates a single vertex for each edge. This means that the shortest path between u and v will have length $2 = r$. We cannot bound the weakly right r-reachable sets of u or v if $u \in I$ or $v \in I$.

The given reduction immediately gives another result with respect to fixed-parameter tractability.

Theorem 7.2.2. *IC-WCOL-RIGHT(r) is W[1]-hard when parameterized by $k + c$ for each $r \geq 3$.*

Proof sketch. In Theorem 7.2.1 we gave a polynomial reduction from Independent Set to IC-WCOL-RIGHT(r) for each $r \geq 3$. This reduction can be seen as parameterized reduction, where the parameter p indicating the size of the independent set is transformed to $k = 2r$ and $c = 2p$. The claim follows from W[1]-hardness of independent set when parameterized by p [DF95], and by the assumption that r is a constant. □

We can still find a simple algorithm that solves IC-WCOL-RIGHT(r) for small graphs and small c efficiently based on the following result.

Proposition 7.2.3. *IC-WCOL-RIGHT(r) parameterized by c is in XP.*

Proof. We can simply try placing any of the vertices from the free vertices T into the next free position left L. As there are c free position the overall algorithm runs in $\mathcal{O}(|V \setminus S|^c \cdot n_G^{\mathcal{O}(1)}) \subseteq \mathcal{O}(n_G^c \cdot n_G^{\mathcal{O}(1)})$ time. □

We include an implementation of the algorithm based on Proposition 7.2.3 that we use for turbocharging the Sreach-Heuristic and the Degree-Heuristic. We call the underlying turbocharging approach *TC-LastC-RL*. As we have to keep track of all potentially strongly r-reachable sets of free vertices for a subordering in our implementation, the polynomial overhead of the algorithm is worse than the overhead for IC-WCOL-LEFT(r). We will see how this affects the performance of TC-LastC-RL in the experimental part.

7.3 A Lower Bound

We have already seen that $\max_{v \in S} |\text{Wreachright}_r(G, L, v)|$ is a lower bound for the weak r-coloring number of any full left extension L' of L. We want to propose an even tighter lower bound that is based on the degeneracy of a graph that is formed by the free vertices or L.

Proposition 7.3.1. *Let L be a subordering of vertices $S \subseteq V(G)$ with free vertices $T \subseteq V(G)$. Let $G' = (T, E')$ where $E' = E(G[T]) \cup E_T$. The set E_T is defined such that $\{u, v\} \in E_T$ if and only if*

- *$u, v \in T$ with $u \neq v$, and*
- *there exists a path P in G of length at most r such that $V(P) \cap T = \{u, v\}$.*

Then

$$\text{degeneracy}(G') + 1 \leq \text{wcol}_r(G, L')$$

for any full left extension L' of L.

Proof. Consider any full left extension L' of L. Let $M(v) = \{u \mid \{u, v\} \in E(G'), u \prec_{L'} v\}$. Observe that it is enough to show that $|M(v)|+1 \leq |\text{Wreach}_r(G, L', v)|$ for all $v \in T$ because $\text{degeneracy}(G) = \max_{v \in V(G)} |M(v)|$. Let $u \in M(v)$. If $\{u, v\} \in E(G)$ then clearly $u \in \text{Wreach}_r(G, L', v)$. If there exists a path P in G of length at most r such that $V(P) \cap T = \{u, v\}$ then this path does not go left of u w.r.t. L', hence $u \in \text{Wreach}_r(G, L', v)$. The plus one comes from $v \notin M(v)$ but $v \in \text{Wreach}_r(G, L', v)$. □

Notice that the lower bound in Proposition 7.3.1 can be computed efficiently, as G' can be found efficiently when keeping track of potentially strongly r-reachable sets, and the degeneracy can be computed in linear time [MB83]. This is why we implement a lower bound for TC-LastC-RL that is $\max_{v \in S} |\text{Wreachright}_r(G, L, v)| + \text{degeneracy}(G')$, where G' is constructed as in Proposition 7.3.1. We added an optional program parameter enabling this lower bound for the heuristic and turbocharging part to see how it affects the performance of the turbocharging approach.

7.4 Discussion

In this chapter, we have initiated research with regard to turbocharging for right to left heuristics for weak coloring numbers. We have defined a simple turbocharging approach, TC-LastC-RL, that is based on the problem IC-WCOL-RIGHT(r). The defined problem has two parameters: k, which is an upper bound on the weak coloring number we want to achieve, and c, which defines the length of the prefix of the subordering we want to replace. We have given a reduction that proves NP-hardness and W[1]-hardness of the problem when parameterized by $k+c$ for $r \geq 3$. Even though this reduction only works for $r \geq 3$, it shows that we cannot expect an algorithm that runs in $f(k, c) \cdot |G|^{\mathcal{O}(1)}$ for all r. Follow-up research discussions have shown that there possibly is a reduction from the Independent Set on regular graphs problem which is W[1]-hard when parameterized by the size of the independent set [Cyg+15] to IC-WCOL-RIGHT(r). This reduction would show NP-hardness and W[1]-hardness when parameterized only by c of IC-WCOL-RIGHT(r) for $r \geq 2$. Hence, it would include the case where $r = 2$, but provide a weaker result with regard to fixed-parameter intractability.

CHAPTER 8

Experimental Evaluation

In this chapter, we will be evaluating turbocharging approaches. We will compare weak coloring numbers of computed orderings by heuristics and the corresponding turbocharged heuristics. We apply a novel framework for evaluating turbocharged heuristics, that iteratively tries to improve the computed ordering for one graph instance. Furthermore, we will also compare turbocharging approaches to each other.

Moreover, we will also evaluate how some optimizations described in Chapter 6 affect the performance of turbocharged heuristics with respect to the weak coloring numbers of the computed orderings.

8.1 Hard- and Software

All experiments were performed on a cluster of 16 nodes provided by the Algorithms and Complexity Group at the Vienna University of Technology. Each node is equipped with two Intel Xeon E5-2640 v4, 2.40GHz 10-core processors and 160 GB ram. The sizes of inputs and outputs are relatively small, so network communication is also small, and we never exceed the provided RAM.

All implementations were done in C++17, and made use of the Boost library [21a] with version 1.77.0. The source code is published online [Dob21].

8.2 Test Data

Our data set consists of the same graphs as used by Nadara et al. [Nad+19]. This enables us to use weak coloring numbers of orderings computed by Nadara et al. as a baseline. Furthermore, we can compare for a heuristic, the improvement achieved by the local search of Nadara et al. to the improvement achieved by our turbocharging approaches.

The graphs consist of real-world data, the PACE 2016 Feedback Vertex Set problems, random planar graphs, and random graphs with bounded expansion. For a detailed

Table 8.1: Overview of graph sizes for the different classes.

	$\lvert V(G)\rvert$				$\lvert E(G)\rvert$			
group	min	med	avg	max	min	med	avg	max
small	34	115	223	620	62	612	521	930
medium	235	1302	1448	4941	1017	3032	3343	8581
big	1224	7610	7963	16264	10445	21000	19518	47594
huge	3656	27775	34599	77360	48130	186940	237300	546487

explanation and references for all input graphs we refer to [Nad+19], where, additionally, a reference is provided for downloading all instances.

Nadara et al. classified the graphs into four classes based on the number of edges — small, medium, big, and huge. A detailed overview of graph sizes for the different classes is shown in Table 8.1. As graphs from the class "big" already have over 10K edges we only consider graphs from the small and medium classes, as most of our exact turbocharging approaches would time out for small conservation parameters c on graphs from the big and large classes.

8.3 The Evaluation Framework

When applying a turbocharged version of a heuristic to an input graph, we have to set a target weak r-coloring number k, such that the turbocharged heuristic tries to compute an ordering of vertices with weak r-coloring number at most k. As we do not know k in advance, we propose a novel (to our knowledge) evaluation framework, that iteratively applies the turbocharged heuristic to decrease weak r-coloring numbers of computed orderings. In the process, we also adapt the conservation parameter c. The evaluation framework is given in Algorithm 8.1. The input is a graph G, an integer r, a heuristic H that computes orderings of vertices $V(G)$, and a turbocharged version $TC\text{-}H$ of the heuristic H. An initial ordering of vertices is computed by the heuristic and assigned to L. Next, a timer that runs for t seconds is started. When the timer expires, the ordering L is returned and the program is terminated; note that L can still change after the timer is started. In the two nested loops, we try to decrease the weak r-coloring number of the computed ordering L by applying the turbocharged version of H, where we iteratively decrease the target weak r-coloring number by one. If we are unsuccessful with a conservation parameter c for a target weak r-coloring number, we increase c by one, until we find the desired ordering. Note that the inner loop does not make sense for turbocharging approaches without a conservation parameter such as TC-Iterative-Swap, so we drop it for these approaches.

During our experiments we applied all combinations of heuristics and the turbocharged versions to each graph. Furthermore, we computed orderings for radii ranging from 2 to 5, motivated by Nadara et al. who used the same values. The timeout t for all

Algorithm 8.1: Evaluation framework for turbocharging a heuristic

Input: A graph $G = (V, E)$, an integer r, a heuristic H, and a turbocharged heuristic $TC\text{-}H$
Output: An ordering L of vertices V

```
1  L ← ordering of vertices V computed by the heuristic H;
2  k ← wcol_r(G, L);
3  Start a timer; after t seconds abort the program, and return the current value
     of L
4  while true do
5  |  c ← 1;
6  |  while true do
7  |  |  Try to compute an ordering of vertices V with weak r-coloring number
   |  |    k − 1 using TC-H with conservation parameter c;
8  |  |  If successful, assign this ordering to L, set k ← wcol_r(G, L), and break;
9  |  |  Otherwise, set c ← c + 1;
10 |  end
11 end
```

experiments consisting of a combination of a heuristic, a turbocharging approach with possible optimizations, and a radius r is 300 seconds.

Quality ratio. We evaluate the quality of an approach the same way as is done by Nadara et al.: For each graph and each radius we take note of the smallest weak r-coloring number of an ordering of vertices of this graph that was computed by one of the approaches of Nadara et al. Note that these approaches also include a local search that iteratively tries to reduce the weak r-coloring number of an ordering, and that achieved significant improvements of weak coloring numbers. Then, we take the average ratio of the computed weak r-coloring number of the ordering computed by the turbocharged heuristic to the aforementioned upper bound minus one. We provide this value in percent broken down by radius and by the class of graphs (small and medium). Note that negative values mean that the approach achieves lower weak coloring numbers when compared to the best weak coloring numbers achieved by Nadara et al. when we average over the instances over which the average is taken. Positive values mean that the averaged weak coloring numbers are higher. Furthermore, we also provide the average relative reduction of the weak r-coloring number when comparing the heuristic to the turbocharged heuristic; that is, the relative reduction is $1 - \frac{\text{coloring number turbocharge}}{\text{coloring number heuristic}}$. We choose this evaluation approach to see how our turbocharged heuristics fare against the algorithms given by Nadara et al.

We will also see how the baseline weak r-coloring number of the ordering computed by a heuristic affects the performance of a turbocharging approach.

All plots are created with Plotnine [21c] which is a python library based on ggplot

[21b].

8.4 Results and Analysis

In this section, we will present experimental results and discuss unexpected outcomes. For each turbocharging approach we will present its quality without any optimizations. By "without optimizations" we mean without optimizations for which we added optional program parameters, that is, ordered adjacency (Section 6.5), and optimizations based on connected components (Section 6.4), and lower bounds (Sections 6.6 and 7.3). We will then describe how some optimizations influence the quality of the turbocharging approach. Note that we will not be presenting every combination of optimizations for each turbocharging approach and each heuristic, as this would exceed a reasonable length for an experimental part. Rather, we will focus on optimizations that significantly improve the performance of a turbocharging approach, or where an optimization results in unexpected changes of the performance.

In Table 8.2 we present the results by Nadara et al. that are relevant to us. The gray columns show the average quality ratios minus one of the heuristics (resp. heuristics after applying their local search to the computed orderings) and the gray columns show relative improvements. We only selected the results for the Wreach-, Degree-, and Sreach-Heuristic, as they are subject to our turbocharging approaches; furthermore, we only present results for the small and medium data set. We chose exactly these three heuristics because they fared best compared to other heuristics in the work by Nadara et al. To quote Nadara et al. [Nad+19]: "In summary, on our data sets, the greedy approaches of Sections 3.5.1 and 3.5.2 produce the best results and have competitive running times. If one looks for something faster, then the simple sort-by-degrees heuristic is consistently the fastest and produces good results." The greedy approaches from Sections 3.5.1 and 3.5.2 they refer to are the Wreach-Heuristic and the Sreach-Heuristic. Note that those heuristics being the best is not contradicted by values in Table 8.2 in the gray columns being positive, as these are averaged values, and while one heuristic might have produced the best upper bound for one instance, another heuristic might have computed the best upper bound for another instance. As these results are subject to the same evaluation scheme we use, the numbers can be directly compared to ones achieved by the different turbocharging approaches in this thesis.

We start by presenting results for turbocharging approaches for left-to-right heuristics — TC-LastC, TC-RNeigh, TC-Wreach, TC-Iterative-Swap, and TC-Merge.

8.4.1 TC-LastC

Without optimizations. In Table 8.3 we present the performance of the TC-LastC approach without any optimizations. Only for the medium data set and the Wreach-Heuristic we achieve a slightly better relative improvement of weak r-coloring numbers than the relative improvement of the local search by Nadara et al. It is also evident that the relative improvement achieved for the Degree-Heuristic is significantly higher than for

Table 8.2: Results obtained by Nadara et al. for their heuristics and their heuristics after the local search (LS). White columns depict average quality ratios minus one and gray columns depict relative improvements when comparing with the underlying heuristic without turbocharging and without local search.

tests	r	Wreach		Degree		Sreach	
small	2	8.3%		26.7%		15.5%	
	3	10%		27.6%		10.7%	
	4	17.7%		26.9%		7.5%	
	5	22.6%		29%		8.4%	
medium	2	8.5%		28.5%		19.2%	
	3	11.6%		27.1%		10.4%	
	4	8.9%		28.5%		5.8%	
	5	16.3%		29.4%		4%	
tests	r	Wreach LS		Degree LS		Sreach LS	
small	2	6.9%	6.7%	5.3%	16.2%	6.3%	7.3%
	3	5.3%		6.5%		4.1%	
	4	9.6%		6.6%		3.2%	
	5	13.6%		10%		5.6%	
medium	2	7%	2.8%	2.4%	17.1%	1.2%	9.9%
	3	11%		6.2%		1.1%	
	4	10.8%		8.7%		0.6%	
	5	16.1%		11.8%		2.1%	

the Wreach-Heuristic, although this is partly due to the fact that the Degree-Heuristic achieved worse results than the Wreach-Heuristic before turbocharging.

In Figure 8.4 we compare the performance of TC-LastC to the best weak coloring numbers achieved by Nadara et al. based on the weak coloring numbers of the underlying heuristics without turbocharging and without local search. That is, for each red data point, its x-value corresponds to the weak coloring number of the Wreach-Heuristic (resp. Degree-Heuristic), and its y-value corresponds to the weak coloring number achieved by the turbocharged variant of the Wreach-Heuristic (resp. Degree-Heuristic). The y-values of blue points correspond to the best weak coloring number achieved by approaches of Nadara et al. Both plots contain data points for all graphs from the data sets small and medium, and radii ranging from 2 to 5. Note that the x- and y-axis are scaled logarithmically to spread data points across the plot. We also added linear trend lines, which clearly show that for TC-LastC combined with both heuristics we are still off the achievable weak coloring numbers.

Table 8.3: Results for TC-LastC without optimizations. White columns depict average quality ratios minus one and gray columns depict relative improvements when comparing with the underlying heuristic without turbocharging and without local search.

tests	r	Wreach TC-LastC		Degree TC-LastC	
small	2	6.2%	6.3%	7.3%	14.0%
	3	7.3%		9.2%	
	4	11.3%		11.2%	
	5	15.8%		16.8%	
medium	2	5.6%	3.2%	6.3%	13.7%
	3	9.2%		9.6%	
	4	11.5%		15.5%	
	5	16.8%		20.1%	

(a) Wreach-Heuristic (b) Degree-Heuristic

Figure 8.4: Comparing performances of TC-LastC to the best weak coloring numbers achieved by Nadara et al. based on the weak coloring numbers of the underlying heuristics without turbocharging and without local search.

TC-LastC-reorder. Table 8.5 shows results for TC-LastC-reorder. We added a comparison to the results of TC-LastC in the brackets. We see that, while TC-LastC improved the performance for the Wreach-Heuristic, it decreased the performance for the Degree-Heuristic. Our explanation is that the Wreach-Heuristic is superior to the Degree-Heuristic — in the sense that it is enough to reorder some parts of the ordering generated by the Wreach-Heuristic to decrease its weak coloring number, whereas for the Degree-Heuristic we have to replace parts of an ordering completely.

Table 8.5: Results for TC-LastC-reorder without optimizations. White columns depict average quality ratios minus one and gray columns depict relative improvements when comparing with the underlying heuristic without turbocharging and without local search. Values in brackets correspond to comparisons with TC-LastC without optimizations.

tests	r	Wreach TC-LastC reorder		Degree TC-LastC reorder	
small	2	7.2% (+0.9)	7.5% (+1.3)	11.9% (+4.5)	13.1% (-0.9)
	3	4.6% (-2.6)		9.3% (+0.1)	
	4	8.7% (-2.6)		13.4% (+2.2)	
	5	13.4% (-2.4)		16.1% (-0.7)	
medium	2	4.3% (-1.3)	4.6% (+1.4)	10.7% (+4.4)	11.2% (-2.5)
	3	8.6% (-0.6)		12.8% (+3.2)	
	4	6.8% (-4.7)		19.0% (+3.5)	
	5	17.9% (+1.1)		24.0% (+3.8)	

Table 8.6: Results for TC-LastC with the optimization based on connected components. White columns depict average quality ratios minus one and gray columns depict relative improvements when comparing with the underlying heuristic without turbocharging and without local search. Values in brackets correspond to comparisons with TC-LastC without optimizations.

tests	r	Wreach TC-LastC CC		Degree TC-LastC CC	
small	2	6.0% (-0.2)	6.8% (+0.6)	8.3% (+1.0)	15.6% (+1.6)
	3	5.6% (-1.7)		5.7% (-3.5)	
	4	10.6% (-0.7)		8.4% (-2.8)	
	5	15.7% (-0.1)		13.2% (-3.6)	
medium	2	4.8% (-0.9)	3.6% (+0.3)	5.2% (-1.1)	15.2% (+1.4)
	3	9.2% (+0.1)		7.9% (-1.8)	
	4	11.0% (-0.5)		13.3% (-2.3)	
	5	16.3% (-0.5)		17.1% (-3.1)	

None of the optimizations increased the performance of TC-LastC-reorder, so we do not present further results for this approach.

Considering connected components. In Table 8.6 we can see results for TC-LastC when applying the optimizations based on connected components (CC) from Section 6.4. We observe that this optimization increases the performance of both approaches by a small amount, independent of the radii.

Table 8.7: Results for TC-LastC with the optimization based on lower bounds. White columns depict average quality ratios minus one and gray columns depict relative improvements when comparing with the underlying heuristic without turbocharging and without local search. Values in brackets correspond to comparisons with TC-LastC without optimizations.

tests	r	Wreach TC-LastC LB		Degree TC-LastC LB	
small	2	4.6% (-1.6)	6.9% (+0.7)	7.5% (+0.1)	14.1% (+0.0)
	3	7.0% (-0.3)		8.6% (-0.6)	
	4	10.8% (-0.5)		12.4% (+1.1)	
	5	15.3% (-0.5)		16.0% (-0.8)	
medium	2	6.2% (+0.5)	3.2% (-0.0)	6.6% (+0.2)	13.8% (+0.1)
	3	8.6% (-0.6)		9.2% (-0.4)	
	4	11.4% (-0.1)		15.4% (-0.1)	
	5	16.9% (+0.1)		21.1% (+0.9)	

Lower bound. Table 8.7 shows results for TC-LastC when applying the WCOL-UB-MMD+ lower bound from Section 6.6 to both the turbocharging part and the heuristic part of the approach. To our surprise we only see small improvements in weak coloring numbers for the small data set and the Wreach-Heuristic. That means that either the lower bound is not tight enough, or the benefit of the lower bound only slightly exceeds drawbacks coming from its computational cost.

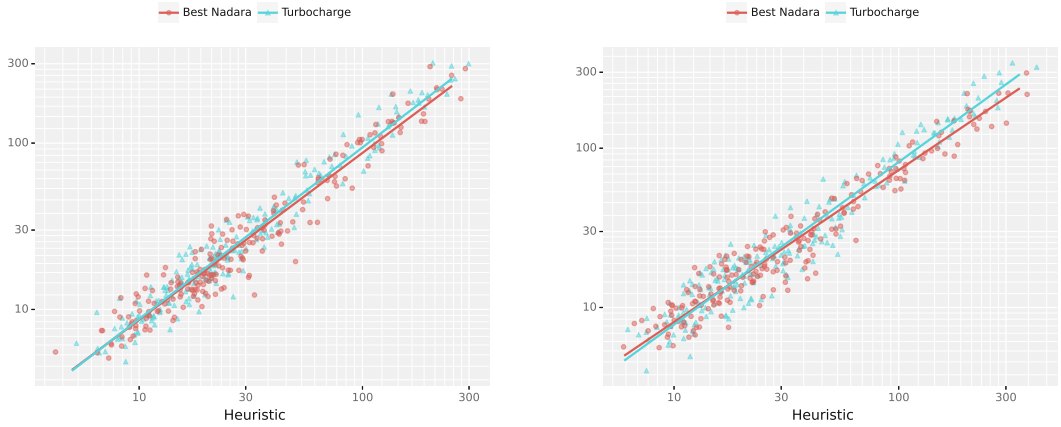
8.4.2 TC-RNeigh(-reorder) and TC-Wreach(-reorder)

We continue by presenting results for TC-RNeigh(-reorder) and TC-Wreach(-reorder). As tables would otherwise exceed the space limit, for a set of optimizations, we only present results for one of the approaches TC-RNeigh and TC-Wreach. During inspection of the results for both approaches we noticed that TC-Wreach is always slightly superior to TC-RNeigh with respect to weak coloring numbers of computed orders when considering no, or any optimization. Furthermore, TC-RNeigh-reorder is superior to TC-Wreach-reorder. That led us to only present a subset of results for TC-RNeigh and TC-Wreach-reorder.

TC-Wreach without optimizations. In Table 8.8 we present results for TC-Wreach without optimizations for the Wreach- and Degree-Heuristics. We can see that this is the first approach where the relative improvement compared to the local search by Nadara et al. is significantly higher, and that we come close to the best weak coloring numbers achieved by Nadara et al. Again, the relative improvement for the Degree-Heuristic is significantly higher than the relative improvement for the Wreach-Heuristic, based on the fact that the Wreach-Heuristic achieves better weak coloring numbers before turbocharging. The quality ratios for the turbocharged versions of both heuristics

Table 8.8: Results for TC-Wreach without optimizations. White columns depict average quality ratios minus one and gray columns depict relative improvements when comparing with the underlying heuristic without turbocharging and without local search.

tests	r	Wreach	TC-Wreach	Degree	TC-Wreach
small	2	1.8%	11.7%	2.6%	20.0%
	3	0.5%		2.0%	
	4	3.5%		2.1%	
	5	6.2%		3.9%	
medium	2	3.2%	6.9%	1.5%	18.2%
	3	2.9%		4.3%	
	4	5.2%		7.1%	
	5	11.3%		12.1%	



(a) Wreach-Heuristic (b) Degree-Heuristic

Figure 8.9: Comparing performances of TC-Wreach to the best weak coloring numbers achieved by Nadara et al. based on the weak coloring numbers of the underlying heuristics without turbocharging and without local search.

are similar, meaning that this turbocharging approach also works for slightly "worse" heuristics.

Figure 8.9 shows a plot that compares weak coloring numbers of the TC-Wreach approach to the best weak coloring numbers achieved by Nadara et al. The data points underlie the same principle as in Figure 8.4. We notice that the turbocharged versions of the heuristics perform worse, when the weak coloring number computed by the heuristic is already high. This is shown by the blue line, that is above the red line for higher x-values.

Table 8.10: Results for TC-RNeigh-reorder without optimizations. White columns depict average quality ratios minus one and gray columns depict relative improvements when comparing with the underlying heuristic without turbocharging and without local search. Values in brackets correspond to comparisons with TC-Wreach without optimizations.

tests	r	Wreach TC-RNeigh-reorder		Degree TC-RNeigh-reorder	
small	2	3.5% (+1.7)	12.5% (+0.8)	3.3% (+0.8)	20.4% (+0.4)
	3	-0.5% (-1.0)		0.6% (-1.4)	
	4	1.9% (-1.6)		1.5% (-0.6)	
	5	4.1% (-2.0)		3.5% (-0.4)	
medium	2	0.1% (-3.2)	8.3% (+1.4)	3.5% (+2.0)	18.1% (-0.1)
	3	2.4% (-0.5)		4.0% (-0.3)	
	4	3.8% (-1.3)		6.8% (-0.3)	
	5	9.9% (-1.3)		10.9% (-1.2)	

TC-Wreach and optimizations. The optimization based on connected components from Section 6.4 slightly decreased the performance of TC-Wreach for all radii and both classes of instances (small and medium), so we do not present the results here.

Applying the ordered adjacency list from Section 6.5 does not influence the quality ratios for TC-Wreach in any significant way, so we also refrain from presenting results for this optimization. Note that we also explained why we are not applying the lower bound WCOL-UB-MMD+ from Section 6.6 to TC-Wreach. As expected, the lower bound $\max_{v \in S} |\text{Wreachleft}_r(G, L, v)|$ for a subordering of vertices S from Section 6.6 where we ignore weakly left r-reachable sets of free vertices, only decreased the performance of TC-Wreach.

We continue presenting results for TC-RNeigh-reorder.

TC-RNeigh-reorder without optimizations In Table 8.10 we show results for TC-RNeigh-reorder without optimizations when compared with TC-Wreach without optimizations. Surprisingly, for this rather straight-forward adaptation of TC-RNeigh we achieve decent results for both left-to-right heuristics. We even achieve a quality ratio of below one for small graphs with radii 3 and the Wreach-Heuristic. Overall, this approach achieves slightly better results than TC-Wreach.

TC-RNeigh-reorder and optimizations Again, the optimization based on connected components from Section 6.4 only slightly decreased the performance of TC-Wreach for all radii and both classes of instances (small and medium), so we do not present the results here.

As for TC-Wreach, the optimization based on ordered adjacency lists from Section 6.5 did not influence the results when comparing with TC-RNeigh-reorder without optimizations. For all the approaches where ordered adjacency did not influence the results, we

Table 8.11: Results for TC-Iterative-Swap without optimizations. White columns depict average quality ratios minus one and gray columns depict relative improvements when comparing with the underlying heuristic without turbocharging and without local search.

tests	r	Wreach TC-Iterative-Swap		Degree TC-Iterative-Swap	
small	2	3.5%	9.8%	1.9%	18.4%
	3	1.2%		2.3%	
	4	5.4%		4.1%	
	5	9.8%		7.4%	
medium	2	2.9%	4.4%	3.7%	15.3%
	3	8.5%		7.4%	
	4	9.4%		11.3%	
	5	15.6%		16.7%	

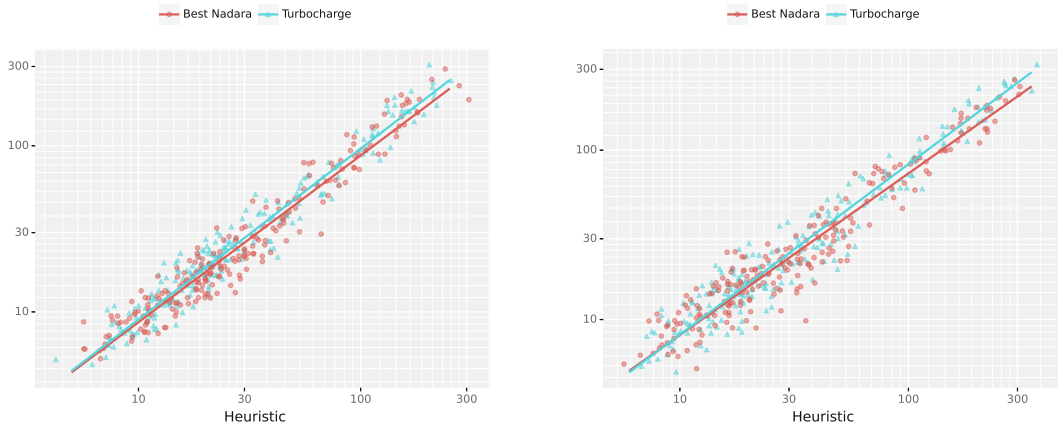
think that comes from our timeout of 300 seconds being rather high. For smaller timeouts, we saw the expected trade off: Namely, for small radii (2,3) the performance decreased, and for larger radii (4,5) the performance increased. This comes from the fact that updating ordered adjacency lists takes some time. But for larger radii this computation cost is exceeded by the decrease in runtime of computing weakly left r-reaching sets, as weakly left r-reaching sets are generally larger for larger radii r.

As stated for TC-Wreach and due to the same reason, we do not provide results for TC-RNeigh-reorder applying the WCOL-UB-MMD+ lower bound from Section 6.6. The lower bound $\max_{v \in S} |\text{Wreachleft}_r(G, L, v)|$ where we ignore weakly left r-reachable sets of free vertices only decreased the performance of TC-RNeigh-reorder.

8.4.3 TC-Iterative-Swap

Without optimizations. In Table 8.11 we show results for TC-Iterative-Swap without optimizations for both left-to-right heuristics. Even though TC-Iterative-Swap is very similar to the local search algorithm of Nadara et al., it still achieves higher relative improvements for all classes of graphs, all radii, and both heuristics. We think this could be due to two reasons. First, turbocharging is mostly applied to suborderings that do not include all vertices. That is, if we find that a subordering L is not extendable during the heuristic part of turbocharging, then L mostly consists of a subset of vertices $V(G)$. There are fewer possibilities to swap vertices in L than there would be possibilities if L consisted of all vertices $V(G)$. Hence, the probability of fixing this non-extendable ordering for fewer vertices is higher.

Second, our implementations for swapping vertices and subsequent updates of weakly left r-reachable sets are further optimized with respect to runtime when comparing to the implementation of Nadara et al. (see Section 6.3), resulting in a larger search space that can be explored during the turbocharging algorithm.



(a) Wreach-Heuristic (b) Degree-Heuristic

Figure 8.12: Comparing performances of TC-Iterative-Swap to the best weak coloring numbers achieved by Nadara et al. based on the weak coloring numbers of the underlying heuristics without turbocharging and without local search.

Our results give first empirical evidence for these hypotheses. Furthermore, they suggest that it is worth investigating if local search algorithms can be adapted for turbocharging.

In Figure 8.12 we compare weak coloring numbers achieved by TC-Iterative-Swap to the best weak coloring numbers achieved by Nadara et al. A description for how data points in the plot are drawn is given in Section 8.4.1. We see a similar distribution of data points as for TC-Wreach indicated by the linear trend line. For smaller weak coloring numbers of orderings computed by the heuristic, TC-Iterative-Swap achieves similar weak coloring numbers as the best weak coloring numbers of Nadara et al. Yet, for larger weak coloring numbers, TC-Iterative-Swap achieves larger weak coloring numbers. We have to mention that the best weak coloring numbers that were achieved by Nadara et al. for larger radii (where weak coloring numbers are larger) stem from the Sreach-Heuristic, that is not a left-to-right heuristic, and thus we cannot apply any of the turbocharging approaches for left-to-right heuristics. Thus, this could have influenced the distribution of data points and the trend lines seen in Figures 8.4, 8.9, 8.12 and 8.15.

TC-Iterative-Swap and Optimizations As for other turbocharging approaches, the optimization based on connected components from Section 6.4 and the optimization based on an ordered adjacency list from Section 6.5 did not influence the results for TC-Iterative-Swap in any significant way. We will provide reasons for this in Section 8.4.5.

In Table 8.13 we present results for TC-Iterative-Swap when applying the WCOL-UB-MMD+ lower bound from Section 6.6. We do not see any significant changes in quality ratios and relative improvements. As TC-Iterative-Swap is designed to decrease weakly

Table 8.13: Results for TC-Iterative-Swap with the WCOL-UB-MMD+ lower bound (LB). White columns depict average quality ratios minus one and gray columns depict relative improvements when comparing with the underlying heuristic without turbocharging and without local search. Values in brackets correspond to comparisons with TC-Iterative-Swap without optimizations.

tests	r	Wreach TC-Iterative-Swap LB		Degree TC-Iterative-Swap LB	
small	2	3.2% (-0.3)	10.1% (+0.3)	1.3% (-0.7)	18.8% (+0.3)
	3	1.4% (+0.2)		1.6% (-0.8)	
	4	5.4% (+0.0)		3.5% (-0.5)	
	5	8.5% (-1.3)		7.5% (+0.1)	
medium	2	3.1% (+0.3)	4.3% (-0.1)	3.7% (+0.0)	15.2% (-0.1)
	3	8.5% (+0.0)		7.2% (-0.2)	
	4	9.4% (+0.0)		12.8% (+1.4)	
	5	16.0% (+0.4)		16.8% (+0.1)	

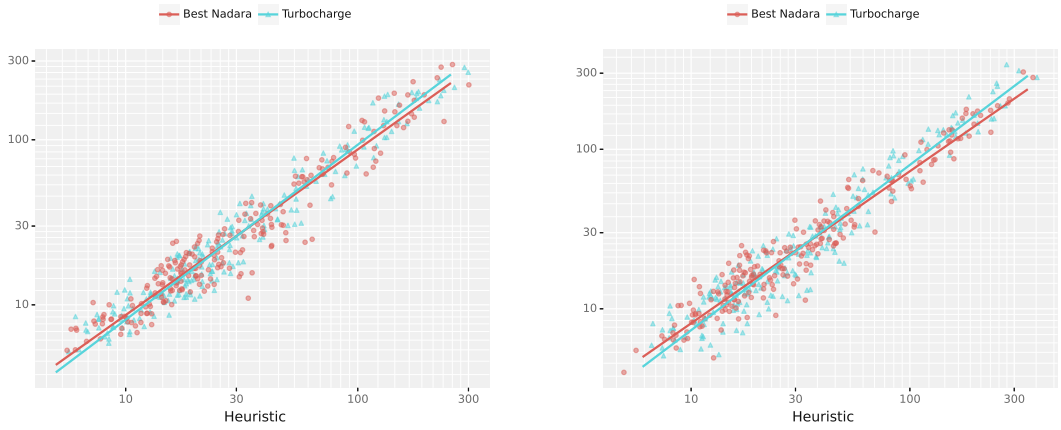
Table 8.14: Results for TC-Merge without optimizations. White columns depict average quality ratios minus one and gray columns depict relative improvements when comparing with the underlying heuristic without turbocharging and without local search.

tests	r	Wreach TC-Merge		Degree TC-Merge	
small	2	0.6%	14.6%	-2.4%	23.4%
	3	-2.3%		-2.3%	
	4	-0.2%		-1.1%	
	5	1.3%		-0.2%	
medium	2	-2.4%	11.3%	-1.3%	21.4%
	3	-0.2%		-0.7%	
	4	-0.5%		2.6%	
	5	4.6%		6.2%	

left r-reachable sets, it could have difficulties dealing with non-extensibility based on the WCOL-UB-MMD+ lower bound. So this result was expected.

8.4.4 TC-Merge

We continue by presenting results for TC-Merge in Table 8.14. As we reach a quality ratio of below one for most instance classes and radii it is clear that this turbocharging approach performs best when compared with other turbocharging approaches for left-to-right heuristics. The only types of instances where we do not reach weak coloring numbers of Nadara et al. are graphs from the medium class with radius 5. It is also interesting that



(a) Wreach-Heuristic (b) Degree-Heuristic

Figure 8.15: Comparing performances of TC-Merge to the best weak coloring numbers achieved by Nadara et al. based on the weak coloring numbers of the underlying heuristics without turbocharging and without local search.

while the Degree-Heuristic generally computes orderings of higher weak coloring number than the Wreach-Heuristic, the turbocharged version of the Degree-Heuristic computes orderings of similar or even lower weak coloring numbers than the turbocharged version of the Wreach-Heuristic. We do not see an obvious reason for that.

In Figure 8.15 we show a comparison of the weak coloring numbers achieved by TC-Merge to the best weak coloring numbers of Nadara et al. based on the weak coloring number of orderings produced by the underlying heuristics without turbocharging and without local search. We spot an expected difference of the two linear trend lines — while the trend line for the turbocharged heuristic is below the trend line of the best achieved weak coloring numbers by Nadara et al. for small x-values, it is above that line for larger x-values. We argue that this is based on the fixed-parameter tractability of the underlying problem of TC-Merge, where the target weak coloring number k is part of that parameterization, resulting in a rapid increase of computational complexity for larger values k. Furthermore, we have to mention that this slightly skews the evaluation based on quality ratio. Namely, whenever we come below the best weak coloring number of Nadara et al. for graphs that have small weak coloring numbers, this will have a larger impact on the average quality ratio. Still, we can say for certain that this turbocharging approach is effective when we expect the weak r-coloring number of the inspected graph instance to be small.

TC-Merge and optimizations. Both the optimization based on connected components from Section 6.4, and the optimization based on ordered adjacency lists from Section 6.5 did not influence the performance of TC-Merge in any significant way. We only present results for TC-Merge when applying the WCOL-UB-MMD+ lower bound

Table 8.16: Results for TC-Merge with the optimization based on the WCOL-UB-MMD+ lower bound (LB). White columns depict average quality ratios minus one and gray columns depict relative improvements when comparing with the underlying heuristic without turbocharging and without local search. Values in brackets correspond to comparisons with TC-Merge without optimizations.

tests	r	Wreach TC-Merge LB		Degree TC-Merge LB	
small	2	4.0% (+3.3)	13.2% (-1.4)	1.9% (+4.3)	21.2% (-2.2)
	3	-1.8% (+0.5)		1.5% (+3.9)	
	4	0.5% (+0.7)		-0.3% (+0.7)	
	5	3.3% (+2.1)		2.7% (+2.9)	
medium	2	-0.9% (+1.6)	10.0% (-1.4)	4.0% (+5.3)	19.2% (-2.3)
	3	2.1% (+2.3)		2.4% (+3.1)	
	4	1.0% (+1.6)		5.1% (+2.5)	
	5	6.3% (+1.7)		10.4% (+4.2)	

from Section 6.6 in Table 8.16. We observe that the relative improvement decreased for all considered classes of graphs and both heuristics. This suggests that it is not appropriate to apply this lower bound for TC-Merge, because TC-Merge is almost entirely based on weakly left r-reachable sets of overfull vertices; that is, TC-Merge defines a turbocharging problem based on weakly left r-reachable sets of overfull vertices. Still, we proposed an adaptation of TC-Merge that enables us to apply WCOL-UB-MMD+ that is explained in Section 6.6. But WCOL-UB-MMD+ can report that an ordering cannot be extended based on a more complex algorithm where parts of the input graph are contracted into a single vertex, even if there are no overfull vertices — not giving any feedback on where this "problem" stems from. TC-Merge is not designed to tackle this problem, as it requires some indication of the problem, in the form of overfull vertices that enable us to focus on a specific search space.

8.4.5 Discussing Optimizations for Left-to-Right Heuristics

In experimental results for most of the left-to-right turbocharging approaches we have seen that most of the optimizations from Chapter 6 where we added an optional program parameter did not influence the performance of those approaches or even had a negative effect. We want to discuss possible reasons in this section.

Connected components. We have seen that the optimization based on connected components from Section 6.4 only increased the performance of the TC-LastC approach. One reason is that this optimization is especially important for TC-LastC, as we could decrease the search space of only this approach based on this optimization. But this

does not explain why some turbocharging approaches even perform worse with this optimization, and we do not have an obvious explanation for that.

It could be argued though that this optimization is not fully worked out yet. Recall that the optimization can roughly be described as, for a subordering L of vertices S with free vertices T, selecting a connected component of $G[T]$ and placing $V(G[T])$ to the right of L as a contiguous part. We left open how to select which connected component of $G[T]$ to choose, thus, it might be that our implementation sometimes chooses a suboptimal or even disadvantageous connected component. This leaves room for further research, and we are interested if a policy that enforces how to choose a connected component of $G[T]$ can increase the effectiveness of this optimization for turbocharging approaches different from TC-LastC.

Ordered adjacency list. The ordered adjacency list from Section 6.5 did not influence the performance of any left-to-right-heuristic where it has been applied to in any significant way. We have already touched on the reasons for this. Namely, our timeout of 300 seconds for all experiments is rather high, which results in the optimization based on ordered adjacency lists not influencing the explored combinations of conservation parameters c and target weak coloring numbers k that can be explored for a specific instance enough. For a graph instance, we think that a turbocharging approach and the same approach applying the ordered adjacency list will time out during the same combination of target weak coloring number k and conservation parameter c, because the runtime increases exponentially for increasing c.

Nonetheless, we have seen the influence of the optimization based on ordered adjacency lists for smaller timeouts: For smaller radii, the performance of turbocharging with this optimization decreased, and for larger radii it increased. The reason is that updating the ordered adjacency list takes some time if the position of a vertex in the subordering changes. But for larger radii this time is exceeded by the time that is saved by applying the ordered adjacency list for computing weakly reaching sets.

Lower bounds. Firstly, the lower bound $\max_{v \in S} |\text{Wreachleft}_r(G, L, v)|$ for a subordering L of vertices S from Section 6.6, where we ignore the weakly left r-reachable sets of free vertices, did not improve the performance of any approach.

Secondly, the lower bound WCOL-UB-MMD+ only increased the performance of the Wreach-Heuristic with TC-LastC and the small graph class. We think that this is because the TC-LastC approach is a more general and less heuristic approach than all the other turbocharging approaches that we have seen: Assume that for any other approach the lower bound WCOL-UB-MMD+ reports that the current subordering is non-extendable, and we have to turbocharge. Then this could be because of a part of the graph where weakly left r-reachable sets are still relatively small when compared to the maximum size of a weakly left r-reachable set, and particularly, we have no overfull vertices. The "more heuristic" turbocharging approaches are all designed such that they examine a specific search space based on the weakly r-reachable sets of overfull vertices. This "indication of non-extensibility" in the form of overfull vertices is normally

given by the basic lower bound $\max_{v \in V} |\text{Wreach}_r(G, L, v)|$. Now that the lower bound WCOL-UB-MMD+ reported non-extensibility, the turbocharging approach is lacking this indication, and might consider a less suitable search space. Instead, for TC-LastC the search space is not based on any lower bound, but only based on the assumption that we can fix non-extensibility by replacing some vertices of the suffix of the subordering.

As we will see in Section 8.5, another problem is that the WCOL-UB-MMD+ lower bound is not that tight, and will rarely find that there is no full right extension of desired weak r-coloring number soon enough. So the trade-off or running time and tightness seems to be unfavorable.

We want to further expand on why some unexpected behavior might occur for lower bounds and our specific turbocharging framework. First, let us mention that all lower bounds ℓ we consider for a subordering L are lower bounds for the weak r-coloring number of any full right extension of L. Thus, ℓ is a lower bound for the weak r-coloring number of an ordering computed by the considered left-to-right heuristic. The value ℓ might not be a lower bound in another context: Assume that we computed a lower bound $\ell \in \mathbb{N}$ for the weak r-coloring number of any full right extension L' of a subordering L during some "heuristic" turbocharging algorithm (that is TC-Wreach, TC-RNeigh, TC-Iterative-Swap, or TC-Merge). Furthermore, assume that this turbocharging algorithm was successful in fixing the non-extendable subordering such that L is still the left part of the subordering. The heuristic part of the turbocharging framework might now place some vertices until our subordering is non-extendable again. The turbocharging algorithm is now applied again but may compute an extendable subordering where the lower bound for the weak r-coloring number of any full right extension is even less than ℓ. This is because the turbocharging algorithm might change parts of the subordering that are in the beginning (to the left) w.r.t. that subordering. Hence, ℓ is not a lower bound when considering the combination of turbocharging algorithm and heuristic.

To fix this issue we propose two different solutions to tackle this problem for future research: Either lower bounds could be adapted in some way to "also be lower bounds" for applications of the turbocharging problem in the future, or the turbocharging framework may be redesigned in some way, such that this problem does not occur anymore. The stated problem should also give some intuition why tighter lower bounds may perform worse than weaker lower bounds when applied during the turbocharging framework: As a lower bound for the weak r-coloring number of any full right extension might not be a lower bound for the combination of the heuristic and turbocharging, it could give some false indication of non-extensibility. It could be that a lower bound reports that a subordering is non-extendable, but a later application of a turbocharging algorithm might still fix this problem. Note that this is why we even performed experiments for the rather weak lower bound that is $\max_{v \in S} |\text{Wreachleft}_r(G, L, v)|$ for a subordering L of vertices S.

The main takeaway from this discussion is that we have to be careful when designing lower bounds in the turbocharging framework that we apply for weak coloring numbers. It might be that we have to tailor a turbocharging approach to a lower bound and also the other way around, there being an intricate interplay. We can relate this to TC-Merge,

Table 8.17: Results for TC-LastC-RL without optimizations. White columns depict average quality ratios minus one and gray columns depict relative improvements when comparing with the underlying heuristic without turbocharging and without local search.

tests	r	Sreach TC-LastC-RL		Degree TC-LastC-RL	
small	2	2.2%	13.1%	8.5%	16.9%
	3	-1.4%		7.5%	
	4	-4.0%		6.5%	
	5	-3.8%		9.0%	
medium	2	8.6%	9.0%	15.0%	11.9%
	3	3.7%		13.0%	
	4	0.6%		16.9%	
	5	-1.1%		17.9%	

which is based on merging weakly left r-reachable sets of overfull vertices. These overfull vertices are in turn signified by the lower bound $\max_{v \in V} |\text{Wreachleft}_r(G, L, v)|$ exceeding size k for a subordering L.

8.4.6 TC-LastC-RL

We continue presenting results for TC-LastC-RL which is the only approach for turbocharging right-to-left heuristic that we investigated.

Without lower bound. In Table 8.17 we see results for TC-LastC-RL for the Sreach- and Degree-Heuristic. Although the relative improvements for the turbocharged version of the Degree-Heuristic are slightly higher, the quality ratios for the turbocharged version of the Sreach-Heuristic are significantly better. This could imply that TC-LastC-RL struggles to turbocharge slightly worse heuristics such as the Degree-Heuristic. Furthermore, we see that for the Sreach-Heuristic the quality ratios are better for larger radii. The reason for this could be that the Sreach-Heuristic performs well for larger radii even before turbocharging. We also notice that for the medium graph class the quality ratios get worse — the reason being that the implementation for turbocharging right-to-left heuristic is slightly more computationally expensive than for left-to-right heuristics.

In Figure 8.18 we compare the weak coloring numbers of TC-LastC-RL to the best weak coloring numbers achieved by Nadara et al. based on the weak coloring numbers of the heuristics without turbocharging and without local search. The trend line suggests that, even though the Sreach-Heuristic with TC-LastC performs well for larger radii, it is still a bit worse for larger weak coloring numbers.

(a) Sreach-Heuristic (b) Degree-Heuristic

Figure 8.18: Comparing performances of TC-LastC-RL to the best weak coloring numbers achieved by Nadara et al. based on the weak coloring numbers of the underlying heuristics without turbocharging and without local search.

Table 8.19: Results for TC-LastC with the optimization based on the lower bound (LB) from Section 7.3. White columns depict average quality ratios minus one and gray columns depict relative improvements when comparing with the underlying heuristic without turbocharging and without local search. Values in brackets correspond to comparisons with TC-LastC-RL without optimizations.

tests	r	Sreach TC-LastC-RL LB		Degree TC-LastC-RL LB	
small	2	2.3% (+0.2)	13.1% (+0.1)	8.7% (+0.2)	16.8% (-0.1)
	3	-1.6% (-0.2)		7.4% (-0.1)	
	4	-3.8% (+0.2)		7.0% (+0.6)	
	5	-4.3% (-0.5)		9.1% (+0.1)	
medium	2	8.7% (+0.1)	8.8% (-0.2)	15.3% (+0.3)	11.7% (-0.2)
	3	4.5% (+0.8)		13.4% (+0.5)	
	4	0.9% (+0.3)		16.0% (-0.8)	
	5	-1.3% (-0.2)		19.3% (+1.4)	

With lower bound. Table 8.19 shows results for TC-LastC-RL when applying the lower bound from Section 7.3. We are again surprised that this lower bound results in no noticeable improvements of quality ratios for both heuristics. We have to assume that this lower bound is not tight enough to make a difference.

8.4.7 Improvements on Upper Bound

We want to conclude this section by presenting, if and for how many graph instances we were able to improve best known upper bounds for weak coloring numbers. For this analysis we compare coloring numbers of all the approaches we presented until now and break them down by left-to-right and right-to-left heuristic. We will also give the average ratio when comparing the best weak coloring number achieved by Nadara et al. and the best weak coloring number achieved by any of our approaches.

Improvements on upper bounds. There are a total of 220 instances in our data set when considering graphs and radii ranging from 2 to 5. Out of these instances the turbocharged left-to-right heuristics were able to improve upper bounds for 120 instances, the turbocharged right-to-left heuristics for 84 instances. Overall, we were able to compute tighter upper bounds for 137 instances.

Ratios. The average ratio when taking the best weak coloring numbers achieved by left-to-right heuristics is 0.955, for right-to-left heuristics it is 1.001. When considering the best weak coloring number achieved by any of our approaches, the average ratio is 0.939. As 0.939 is lower than 0.955 by a considerable amount, this means that both approaches (left-to-right and right-to-left) achieve good results on different classes of instances. By simply looking at quality ratios for the respective turbocharging approaches, we observe that left-to-right turbocharging approaches work better for smaller radii and right-to-left turbocharging approaches for larger radii.

8.5 Comparing Lower and Upper Bounds

In Section 6.6 we have presented two lower bounds, WCOL-UB-MMD+ and WCOL-MMD+, that are able to compute lower bounds for weak r-coloring of graphs. Furthermore, in Section 8.4 we have seen that the optimization based on the WCOL-UB-MMD+ lower bound rarely improved the performance of any turbocharging approach, which led us to believe that this lower bound is not tight enough. That is why we want to compare current best upper bounds for our investigated instances to both lower bounds. Note that it was not the main goal of this thesis to optimize and design good lower bounds for weak coloring numbers of graphs. Rather, they emerged as a byproduct during our research. As results for lower bounds are easy to test and represent, we still want to devote a small part of the experimental evaluation to lower bounds. For the best upper bounds, we compare the weak coloring numbers computed by any of our algorithms and any of the algorithms of Nadara et al., and take the minimum. In Table 8.20 we present the average ratio of a lower bound to the best known upper bound broken down by radius and graph class. As a baseline we add the average ratio for $\mathrm{wcol}_1(G)$, which clearly is a lower bound for $\mathrm{wcol}_r(G)$ with $r \geq 1$. Ratios being the same for $r = 2$ is not an error, as for $r = 2$, $\lfloor \frac{r-1}{2} \rfloor$ is zero, which means that we cannot contract any vertices in the WCOL-UB-MMD+ and WCOL-MMD+ lower bounds. Under the

Table 8.20: Average ratios of lower bounds to best known upper bounds.

tests	r	$\text{wcol}_1(G)$	WCOL-UB-MMD+	WCOL-MMD+
small	2	0.522	0.522	0.522
	3	0.456	0.472	0.478
	4	0.416	0.435	0.442
	5	0.389	0.415	0.423
medium	2	0.451	0.451	0.451
	3	0.372	0.381	0.383
	4	0.329	0.339	0.342
	5	0.301	0.314	0.322

assumption that our upper bounds nearly represent the exact weak coloring numbers, we can see that our believes of the lower bounds not being tight enough got confirmed, as both WCOL-UB-MMD+ and WCOL-MMD+ are only slightly better than $\text{wcol}_1(G)$. Additionally, the ratio gets worse for larger radii. A positive observation is that the ratios for WCOL-UB-MMD+ and WCOL-MMD+ are not that far apart, meaning that the upper bound on diameters of subgraphs we use for WCOL-UB-MMD+ does not influence the tightness of the lower bound to an extensive amount. To summarize, even though our proposed lower bounds are not as tight as we wanted them to be, they still are an improvement to the trivial lower bound that is $\text{wcol}_1(G)$.

8.6 Discussion

In this chapter, we have presented experimental evaluations for proposed turbocharging approaches and lower bounds. We want to give a brief summary and highlight some results that are especially worth mentioning.

We have started by explaining our test setup, input data, and our novel (to our knowledge) evaluation framework in Sections 8.1 to 8.3. Then we have presented results for all turbocharging approaches. For turbocharging approaches for left-to-right heuristics, TC-LastC performed the worst, followed by TC-Iterative-Swap, TC-Wreach, and TC-RNeigh. Interestingly, even though TC-Iterative-Swap is very similar to the local search of Nadara et al., the computed weak coloring numbers are significantly better. Furthermore, TC-Wreach-reorder, TC-RNeigh-reorder, and TC-LastC-reorder, performed better than their non-reordered counterparts. TC-Merge performed the best for both considered heuristics and was able to improve upper bounds for several instances, predominantly for instance where the weak r-coloring number is small. For each turbocharging approach we have also applied several optimizations and in Section 8.4.5 we have explained why some of them did not perform as expected.

Even though TC-LastC-RL is a very simple turbocharging approach for right-to-left heuristics, it was still able to compute orderings of small weak coloring numbers and

improved upper bounds on weak coloring numbers for several instances, mostly for instances with larger radii. Its performance is close to TC-Merge, but while TC-Merge works well for instances of smaller radii, TC-LastC-RL works well for instances of larger radii. Overall, we were able to improve upper bounds on weak coloring numbers for 137 out of the 220 considered instances by roughly 10% in average.

In Section 8.5 we performed experiments for the WCOL-UB-MMD+ and WCOL-MMD+ lower bounds for weak coloring numbers of graphs. We compared these lower bounds to best known upper bounds and have seen that there is still a large gap of around factor two between them.

CHAPTER 9

Conclusion

At the end of most chapters we have already given brief summaries, open questions, and pointers for future research. We give here a broad summary of the thesis and the answer to the guiding research question, which asks if we can successfully apply the turbocharging framework to the domain of computing vertex orderings with small weak r-coloring numbers. Furthermore, we want to give a bigger picture on possible future research directions, leaving the details to the aforementioned discussions at the end of each chapter.

We have studied and designed algorithms to compute vertex orderings of real-world graphs with small weak coloring numbers. Specifically, we applied a distinct variant of the turbocharging framework to already established heuristics for computing orderings of small weak r-coloring number. We divided the research into two parts, designing turbocharging algorithms for left-to-right heuristics and for right-to-left heuristics. For left-to-right heuristics, we proposed several turbocharging approaches, some of which have heuristic aspects to them. As a by-product of this research we also proposed a novel lower bound for weak r-coloring number of graphs that is based on contracting subgraphs with diameter at most $\lfloor \frac{r-1}{2} \rfloor$ into single vertices.

We proposed one turbocharging approach for turbocharging right-to-left heuristics, but believe there is room for future research.

In our experimental evaluation we have seen that the performance of the different turbocharging approaches varied significantly. The turbocharging approach for left-to-right heuristics that merges a subset of vertices into an already fixed subordering, and the simple turbocharging approach for right-to-left heuristics that replaces a prefix of an ordering of a subset of the vertices of a graph by different vertices performed the best for most of the considered input instances.

With our approaches, we were able to improve upper bounds for 137 of the 220 considered instances. The average ratio of weak coloring numbers achieved by any of our approaches and the previous best upper bounds is 0.939. Hence, we conclude that for a

class of real-world graphs turbocharging is a useful tool to compute orderings of small weak r-coloring number.

Future research. As the main conclusion of the thesis is that turbocharging is as a fruitful problem-solving framework, it would be interesting to see if and how it can be applied to different problem domains, as there are a lot of problems that are still untouched with regard to turbocharging.

Furthermore, the term "turbocharging" is to this date understood in different ways, as "augmenting heuristics with exact algorithms" is a very vague description. In this thesis, we adapted the turbocharging framework to our problem domain which is computing linear orders, which at some points turned out to also pose some problems, for example with regard to the design of lower bounds (see Section 8.4.5). We would be interested to see whether there is a unified description of turbocharging that can immediately be applied to a wide variety of problem domains such as ordering problems or assignment problems.

We have given several turbocharging problems and provided complexity results, most of which are hardness results — we are interested if there are further parameterizations by structural properties of graph instances such as the h-index [ES12] that make the problems tractable in a fixed-parameter sense.

Additionally, we have only initiated research for turbocharging with regard to right-to-left heuristics. As the simple turbocharging approach for right-to-left heuristics provided very good results, further research in this direction might be successful in providing even better algorithms for computing vertex orderings of real world graphs with small weak r-coloring numbers.

Nadara et al. [Nad+19] are as far as we know the only other researchers that focused on algorithms for computing vertex orderings of real-world graphs with small weak r-coloring numbers. We wonder, how other algorithmic techniques would perform in the domain of computing upper bounds for weak coloring numbers. Finally, there are still no tight bounds for the weak r-coloring number of specific graph classes such as planar graphs or even for small real-world graphs.

List of Figures

List of Tables

List of Algorithms

Bibliography

[AWW16] Amir Abboud, Virginia Vassilevska Williams, and Joshua R. Wang. „Approximation and Fixed Parameter Subquadratic Algorithms for Radius and Diameter in Sparse Graphs“. In: *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016*. SIAM, 2016, pp. 377–391. DOI: 10.1137/1.9781611974331.ch28.

[Ami+18] Saeed Akhoondian Amiri et al. „Distributed Domination on Graph Classes of Bounded Expansion“. In: *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures, SPAA 2018*. ACM, 2018, pp. 143–151. DOI: 10.1145/3210377.3210383.

[BK11] Hans L. Bodlaender and Arie M. C. A. Koster. „Treewidth computations II. Lower bounds“. In: *Inf. Comput.* 209.7 (2011), pp. 1103–1119. DOI: 10.1016/j.ic.2011.04.003.

[21a] *Boost C++ Libraries*. 2021. URL: https://www.boost.org/ (visited on 10/08/2021).

[Cor+09] Thomas H. Cormen et al. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. URL: http://mitpress.mit.edu/books/introduction-algorithms.

[Cyg+15] Marek Cygan et al. *Parameterized Algorithms*. Springer, 2015. DOI: 10.1007/978-3-319-21275-3.

[DK09] Anuj Dawar and Stephan Kreutzer. „Domination Problems in Nowhere-Dense Classes“. In: *Proceedings of the IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2009*. Vol. 4. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2009, pp. 157–168. DOI: 10.4230/LIPIcs.FSTTCS.2009.2315.

[Dob21] Alexander Dobler. „Turbocharging Heuristics for Weak Coloring Numbers: Source Code“. In: (2021). DOI: 10.5281/zenodo.5732923.

[DF95] Rodney G. Downey and Michael R. Fellows. „Fixed-Parameter Tractability and Completeness II: On Completeness for W[1]“. In: *Theor. Comput. Sci.* 141.1&2 (1995), pp. 109–131. DOI: 10.1016/0304-3975(94)00097-3.

[Dra+16] Pål Grønås Drange et al. „Kernelization and Sparseness: the Case of Dominating Set“. In: *Proceedings of the 33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016*. Vol. 47. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, 31:1–31:14. DOI: 10.4230/LIPIcs.STACS.2016.31.

[Dvo13] Zdenek Dvorák. „Constant-factor approximation of the domination number in sparse graphs“. In: *Eur. J. Comb.* 34.5 (2013), pp. 833–840. DOI: 10.1016/j.ejc.2012.12.004.

[Dvo19] Zdenek Dvorák. „On distance r-dominating and 2r-independent sets in sparse graphs“. In: *J. Graph Theory* 91.2 (2019), pp. 162–173. DOI: 10.1002/jgt.22426.

[Eic+17] Kord Eickmeyer et al. „Neighborhood Complexity and Kernelization for Nowhere Dense Classes of Graphs“. In: *Proceedings of the 44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*. Vol. 80. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 63:1–63:14. DOI: 10.4230/LIPIcs.ICALP.2017.63.

[ES12] David Eppstein and Emma S. Spiro. „The h-Index of a Graph and its Application to Dynamic Subgraph Statistics“. In: *J. Graph Algorithms Appl.* 16.2 (2012), pp. 543–567. DOI: 10.7155/jgaa.00273.

[GJ79] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[Gas+19] Serge Gaspers et al. „Turbocharging Treewidth Heuristics“. In: *Algorithmica* 81.2 (2019), pp. 439–475. DOI: 10.1007/s00453-018-0499-1.

[21b] *ggplot*. 2021. URL: https://ggplot2.tidyverse.org/reference/ggplot.html (visited on 10/11/2021).

[GKS17] Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. „Deciding First-Order Properties of Nowhere Dense Graphs“. In: *J. ACM* 64.3 (2017), 17:1–17:32. DOI: 10.1145/3051095.

[Gro+18] Martin Grohe et al. „Coloring and Covering Nowhere Dense Graphs“. In: *SIAM J. Discret. Math.* 4 (2018), pp. 2467–2481. DOI: 10.1137/18M1168753.

[HN13] Sepp Hartung and Rolf Niedermeier. „Incremental list coloring of graphs, parameterized by conservation“. In: *Theor. Comput. Sci.* 494 (2013), pp. 86–98. DOI: 10.1016/j.tcs.2012.12.049.

[HW18] Jan van den Heuvel and David R. Wood. „Improper colourings inspired by Hadwiger's conjecture“. In: *J. London Math. Soc.* 98.1 (2018), pp. 129–148. DOI: 10.1112/jlms.12127.

[Heu+17] Jan van den Heuvel et al. „On the generalised colouring numbers of graphs that exclude a fixed minor“. In: *Eur. J. Comb.* 66 (2017), pp. 129–144. DOI: 10.1016/j.ejc.2017.06.019.

[JM21] Gwenaël Joret and Piotr Micek. *Improved bounds for weak coloring numbers.* 2021. arXiv: 2102.10061 [math.CO].

[KY03] Hal A. Kierstead and Daqing Yang. „Orderings on Graphs and Game Coloring Number“. In: *Order* 20.3 (2003), pp. 255–264. DOI: 10.1023/B:ORDE.0000026489.93166.cb.

[KRS19] Stephan Kreutzer, Roman Rabinovich, and Sebastian Siebertz. „Polynomial Kernels and Wideness Properties of Nowhere Dense Graph Classes“. In: *ACM Trans. Algorithms* 15.2 (2019), 24:1–24:19. DOI: 10.1145/3274652.

[MB83] David W. Matula and Leland L. Beck. „Smallest-Last Ordering and clustering and Graph Coloring Algorithms“. In: *J. ACM* 30.3 (1983), pp. 417–427. DOI: 10.1145/2402.322385.

[Nad+19] Wojciech Nadara et al. „Empirical Evaluation of Approximation Algorithms for Generalized Graph Coloring and Uniform Quasi-wideness“. In: *ACM J. Exp. Algorithmics* 24.1 (2019), 2.6:1–2.6:34. DOI: 10.1145/3368630.

[NM08a] Jaroslav Nesetril and Patrice Ossona de Mendez. „Grad and classes with bounded expansion I. Decompositions“. In: *Eur. J. Comb.* 29.3 (2008), pp. 760–776. DOI: 10.1016/j.ejc.2006.07.013.

[NM08b] Jaroslav Nesetril and Patrice Ossona de Mendez. „Grad and classes with bounded expansion II. Algorithmic aspects“. In: *Eur. J. Comb.* 29.3 (2008), pp. 777–791. DOI: 10.1016/j.ejc.2006.07.014.

[NM08c] Jaroslav Nesetril and Patrice Ossona de Mendez. „Grad and classes with bounded expansion III. Restricted graph homomorphism dualities“. In: *Eur. J. Comb.* 29.4 (2008), pp. 1012–1024. DOI: 10.1016/j.ejc.2007.11.019.

[NM11] Jaroslav Nesetril and Patrice Ossona de Mendez. „On nowhere dense graphs“. In: *Eur. J. Comb.* 32.4 (2011), pp. 600–617. DOI: 10.1016/j.ejc.2011.01.006.

[NM12] Jaroslav Nesetril and Patrice Ossona de Mendez. *Sparsity - Graphs, Structures, and Algorithms.* Algorithms and combinatorics. Springer, 2012. DOI: 10.1007/978-3-642-27875-4.

[PST18] Michal Pilipczuk, Sebastian Siebertz, and Szymon Torunczyk. „On the number of types in sparse graphs“. In: *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018.* ACM, 2018, pp. 799–808. DOI: 10.1145/3209108.3209178.

[21c] *Plotnine.* 2021. URL: https://plotnine.readthedocs.io/en/stable/ (visited on 10/11/2021).

[RS21] Vaidyanathan Peruvemba Ramaswamy and Stefan Szeider. „Turbocharging Treewidth-Bounded Bayesian Network Structure Learning“. In: *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021*. AAAI Press, 2021, pp. 3895–3903. URL: https://ojs.aaai.org/index.php/AAAI/article/view/16508.

[RS20] Felix Reidl and Blair D. Sullivan. „A color-avoiding approach to subgraph counting in bounded expansion classes“. In: *CoRR* abs/2001.05236 (2020). arXiv: 2001.05236.

[RVS19] Felix Reidl, Fernando Sánchez Villaamil, and Konstantinos S. Stavropoulos. „Characterising bounded expansion by neighbourhood complexity“. In: *Eur. J. Comb.* 75 (2019), pp. 152–168. DOI: 10.1016/j.ejc.2018.08.001.

[17] *Restricted 2,3-SAT problem*. 2017. URL: https://cs.stackexchange.com/questions/90760/variant-of-3-sat-is-np-complete (visited on 08/09/2021).

[Zhu09] Xuding Zhu. „Colouring graphs with bounded generalized colouring number“. In: *Discret. Math.* 309.18 (2009), pp. 5562–5568. DOI: 10.1016/j.disc.2008.03.024.