TU WIEN Informatics

Twin-Width Heuristiken Turbocharged mit SAT

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Damian Jäger, BSc
Matrikelnummer 11776843

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Mag.rer.nat. Dr.rer.nat. Stefan Szeider
Mitwirkung: Univ.Ass. Mathis Teva Rocton, MSc

Wien, 20. November 2024

Damian Jäger	Stefan Szeider

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

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

TU WIEN Informatics

Turbocharging Twin-Width Heuristics with SAT

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Damian Jäger, BSc
Registration Number 11776843

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Mag.rer.nat. Dr.rer.nat. Stefan Szeider
Assistance: Univ.Ass. Mathis Teva Rocton, MSc

Vienna, November 20, 2024

Damian Jäger	Stefan Szeider

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

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Erklärung zur Verfassung der Arbeit

Damian Jäger, 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, 20. November 2024

Damian Jäger

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Kurzfassung

Twin-Width ist eine neuartige Graphinvariante, die in mancher Hinsicht Tree Width und Rank Width ähnelt. Sie misst in gewisser Weise, wie weit ein Graph davon entfernt ist, ein Cograph zu sein. Dafür werden iterativ zwei Knoten kontrahiert und die Differenz ihrer Nachbarschaften wird mit roten Kanten aufgezeichnet. Liegt ein Zertifikat über begrenzte Twin-Width eines Graphen vor, werden darauf viele NP-schwere Probleme handhabbar. Das Finden solch eines Zertifikats ist jedoch auch NP-schwer.

In dieser Arbeit verbesseren wir Greedy-Heuristiken für Obergrenzen für Twin-Width durch Turbocharging. Dabei wird ein exakter Algorithmus, der Turbocharging Algorithmus, auf ein Unterproblem angewendet, wodurch der Heuristik geholfen wird, wenn die Qualität der Lösung zu niedrig wird. Wir entwickeln SAT-Encodings für den Turbocharging Algorithmus und wenden diese auf zwei Greedy-Heuristiken an.

In unseren Experimenten ist unsere Methode in der Lage, die Obergrenzen der Grundheuristiken für Graphen verschiedenster Größen zu verbessern. Wir vergleichen sie auch mit einer randomisierten Methode von Berthe et al., um einen Vergleich zum State of the Art herzustellen. Unsere Heuristiken sind in der Lage, diesen randomisierten Algorithmus besonders bei großen Graphen signifikant zu übertreffen. Auf kleinen Graphen liefert der randomisierten Ansatz jedoch bessere Ergebnisse.

Heuristiken für Obergrenzen für Twin-Width sind wertvoll sowohl für exakte Algorithmen, als auch zum Finden von Zertifikaten über begrenzte Twin-Width für Graphen, für die exakte Algorithmen zu langsam sind. Besonders für den zweiten Fall eignet sich unser Ansatz, da er Ergebnisse von Greedy-Heuristiken signifikant verbessern kann. Das ist besonders für größere Graphen der Fall.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abstract

Twin-width is a novel graph invariant, in some ways similar to tree width and rank width. In a way, it measures how far a graph is from being a cograph. This is done by iteratively contracting two vertices and keeping track of the edges that separate them from having the same neighborhood using red edges. Given a certificate for bounded twin-width of a graph, many NP-hard problems become tractable on it. However, determining the twin-width and obtaining a certificate is an NP-hard problem itself.

In this thesis we improve greedy upper bound heuristics for twin-width using turbocharging. This involves using an exact approach for a subproblem, the turbocharging algorithm, to aid the heuristic if the solution quality gets too low at some point. We develop SAT encodings for the turbocharging algorithm and apply them to two greedy heuristics.

In our experiments, our turbocharged approach was able to improve the upper bounds of the base heuristics across different instance sizes. We further compare it to a randomized approach by Berthe et al. Particularly for large instances, our turbocharged heuristic outperformed this approach significantly, while the randomized algorithm delivered better results on small instances.

Upper bound heuristics for twin-width are valuable for both exact algorithms and to obtain certificates for bounded twin-width for graphs which are too large for exact algorithm to handle. Especially for the second use case, our approach is well suited, as it manages to improve upon solutions obtained by greedy heuristics significantly. This is particularly the case for larger graphs.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Contents

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 1

Introduction

1.1 Motivation

Twin-width is a width parameter for graphs introduced by Bonnet et al. [15]. A graph has bounded twin-width if it is possible to iteratively select two vertices such that the difference of their neighborhood is bounded until only a singleton remains. During this process, the errors are being tracked using *red edges* and it is required that the degree in red edges is bounded throughout. Twin-width is an active research topic [10, 11, 12, 13, 14, 15, 24, 25] and was the topic of the 8th Parameterized Algorithms and Computational Experiments Challenge (PACE 2023) [6].

What makes twin-width an interesting property is that Bonnet et al. [15] show that given a certificate that a graph has twin-width d, first-order model checking is FPT when parameterized by d and the size of the formula. Using first-order model checking, many NP-hard problems can be solved. They therefore can also be solved in FPT time given a certificate for bounded twin-width. Other width measures, such as treewidth, have similar properties. Twin-width, however, stands out by being bounded for many classes of graphs. This is discussed in Section 2.1.3 in more detail.

Much like for treewidth, determining the twin-width of a graph is a hard problem. Since twin-width is a novel problem, there exist limited knowledge on how to compute it. There do, however, exist some exact and heuristic approaches. This thesis is concerned with improving heuristics for computing upper bounds for twin-width.

1.2 Aim of the Work

The aim of this work is to design, implement and evaluate a *turbocharged* heuristic for twin-width. Turbocharging combines a heuristic that incrementally builds a solution with an exact approach, the turbocharging algorithm, that recomputes previous steps if a

certain quality goal is not reached (see Section 3.3). For this, heuristics need be selected, the algorithmic problem for turbocharging twin-width heuristics needs to be analyzed, and an algorithm to compute it needs to be developed.

1.2.1 Heuristics

To implement a turbocharged heuristic, we need to select at least one heuristic to use. The aim is to find heuristics that provide a reasonable trade-off between run time and quality of results. This balance should be fitting for the context of the turbocharged algorithm.

1.2.2 SAT Encodings

As a turbocharging algorithm we aim to find good SAT encodings. These should have a reasonable size and be as efficient as possible. The encodings should yield exact and correct solutions.

1.3 Methodology

To achieve the aim of the thesis, the following methods are used:

1. **Literature review**: The literature review covers relevant theoretical background and the state of the art for computing twin-width. This includes exact and heuristic approaches. Further, existing turbocharging approaches are covered.
2. **Design**: This entails selecting one or more heuristics to turbocharge as well as finding SAT encodings for computing solutions for the problem involved in turbocharging twin-width heuristics.
3. **Analysis**: The complexity of the encoding and the underlying algorithmic problem is analyzed.
4. **Implementation**: The turbocharging algorithm and heuristics are implemented and combined into a single heuristic.
5. **Experiments**: To evaluate the resulting heuristic, experiments are designed and executed. The aim compare the quality of the turbocharged heuristic to the heuristic without turbocharging in terms of running time and solution quality. For this, a good set of test instances has to be selected.

1.4 Structure of the Thesis

Chapter 2 of this thesis covers the theoretical *Background* required for this thesis. This entails the definition of twin-width and its properties as well as the concept of

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

turbocharging algorithms. The *State of the Art* of computing twin-width both exactly heuristically is discussed in Chapter 3, together with a related turbocharging approach. Next, in Chapter 4 the *Design* of our turbocharged twin-width heuristic is discussed. This encompasses a theoretical analysis of the underlying algorithmic problem, choosing heuristics, and designing SAT encodings. The *Implementation* of this heuristic is covered in Chapter 5. In Chapter 6 we discuss the evaluation of our approach using *Experiments*. Finally, the *Conclusion* in Chapter 7 gives a summary of the thesis and its results.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 2

Background

In this chapter, relevant theoretical background for this thesis is covered. First, Section 2.1 discusses the concept of twin-width and its relevancy. Then, turbocharging heuristic algorithms is covered in Section 2.2. Finally, Section 2.3 discusses SAT solvers.

2.1 Twin-Width

Twin-width is a graph invariant recently introduced by Bonnet et al. [15]. In a way, twin-width measures the distance of a graph to being a *cograph*. A cograph is a graph where it is possible to iteratively select two vertices u and v with the same neighborhood (excluding u and v), also called *twins*, and contracting them, i.e. deleting one of them, until only a singleton remains. Intuitively, a general graph has bounded twin-width if each contraction consists of near twins. To keep track of past errors, *red edges* are used. The maximum *red degree*, which is the number of red edges incident to a vertex, needs to stay bounded for twin-width to be bounded. An formal definition of twin-width is given in Section 2.1.1.

Bounded twin-width generalizes several classes of graphs. Given a certificate for bounded twin-width many hard problems become FPT. This is discussed in more detail in Section 2.1.2 and 2.1.3.

2.1.1 Definitions

In the following, relevant definitions from Bonnet et al. [15] are given. For a graph G we denote its set of vertices by $V(G)$ and its set of edges by $E(G)$. We denote the neighborhood of a vertex v by $N_G(v)$. A *trigraph* $G = (V, E, R)$ is a graph with two sets of edges. It has vertices $V(G)$, (black) edges $E(G)$ and red edges $R(G)$. The sets $E(G)$ and $R(G)$ are disjoint. A trigraph G is a *d-trigraph* if the graph $(V(G), R(G))$ has degree at most d.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

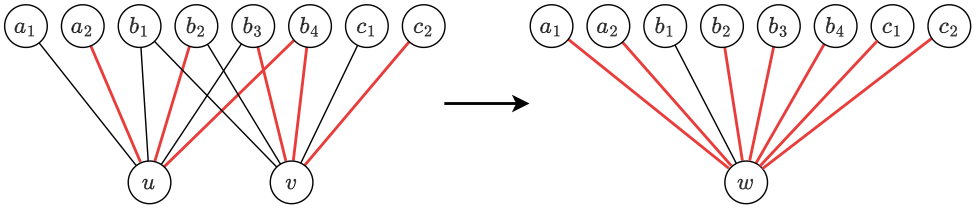

Figure 2.1: Contraction of two vertices u, v into w

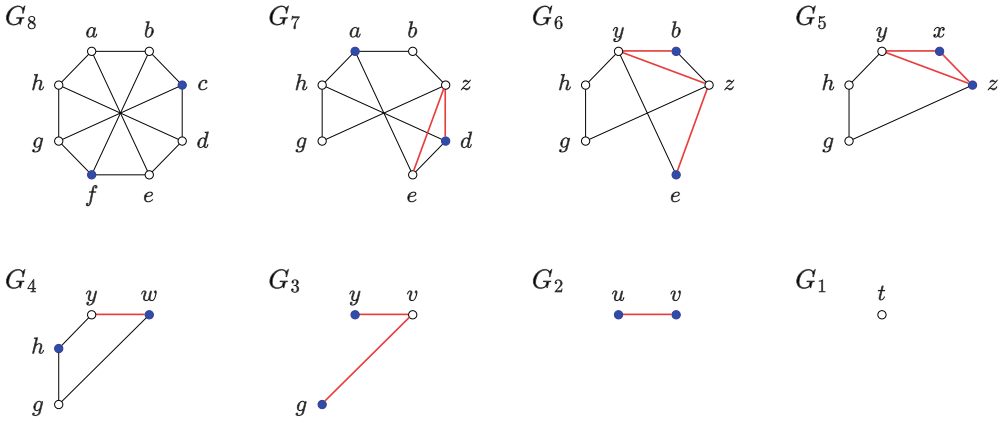

Figure 2.2: A 2-sequence for the Wagner graph with vertices to be contracted highlighted (after [24])

Given a trigraph G and two vertices u and v, a new trigraph $G' = (V', E', R')$ can be obtained through a *contraction* as follows: The vertices u and v are contracted into a new vertex w. Therefore, V' is defined as $(V(G) \setminus \{u, v\}) \cup \{w\}$. The edge sets E' and R' are obtained by removing edges incident to u or v from $E(G)$ and $R(G)$ respectively and adding new edges incident to w. For every $x \in V'$: if $xu \in E(G)$ and $xv \in E(G)$ then $xw \in E'$. Else if $xu \in E(G) \cup R(G)$ or $xv \in E(G) \cup R(G)$ then $xw \in R'$. Otherwise $xw \notin E'$ and $xw \notin R'$. Intuitively this means that red edges stay red throughout the contraction and red edges are added for vertices incident to only one of u and v, tracking the distance from them being twins. Note that u and v may be incident but are not required to be. An example of a contraction can be seen in Figure 2.1. A contraction is called a *d-contraction* if both G and G' are d-trigraphs.

A *contraction sequence* of a graph G is a sequence of trigraphs $G_n, G_{n-1}, ..., G_1$ where each G_i can be obtained from G_{i+1} through a contraction and $G_n = (V(G), E(G), \emptyset)$. Then, G_1 consists of just a single vertex. Such a sequence is called a *d-sequence* if each G_i is a d-trigraph. Figure 2.2 depicts a 2-sequence for the Wagner graph.

The twin-width $tww(G)$ of a graph G is the minimum number d such that G admits a d-sequence.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

2.1.2 Efficient Algorithms Given Bounded Twin-Width

Bonnet et al. [15] show that first-order model checking is FPT on graphs with bounded twin-width when parameterized by twin-width and formula size given that a d-sequence is provided as an input. More precisely, they prove that given a graph G with n vertices, a d-sequence $G = G_n, G_{n-1}, ..., G_1 = K_1$ and a first-order formula φ, $G \models \varphi$ can be decided in time $f(|\varphi|, d) \times n$ for a computable function f. This general result is theoretically interesting, in the runtime of the resulting algorithm, however, f contains a tower of stacked exponents whose height depends on the size of φ.

In another paper, Bonnet et al. [11] present problem-specific algorithms with more practical complexity. In particular, they present an algorithm that solves k-Independent Set in $O(k^2 d^{2k} n)$ time given a d-sequence. It works by following the d-sequence. The connected subgraphs of red graphs $R_i = (V(G_i), R(G_i))$ are solved to obtain partial solutions. As R_n has no edges, this is trivial. After each contraction, the partial solutions are updated by merging partial solutions of sets that were disconnected before. Similar algorithms are possible for r-Scattered Set, k-Clique and k-Dominating Set with runtime $2^{O(k)}n$, and for the weighted version of k-Independent Set, Subgraph Isomorphism, and Induced Subgraph Isomorphism with runtime $2^{O(k \log k)}n$ given a $O(1)$-sequence.

2.1.3 Graphs with Bounded Twin-Width

It turns out that a variety of classes of graphs have bounded twin-width. Bonnet et al. [11] list the following such classes:

- Bounded clique-width/rank-width, and more generally, boolean-width graphs,
- every hereditary proper subclass of permutation graphs,
- posets of bounded antichain size (seen as digraphs), unit interval graphs,
- K_t-minor free graphs,
- map graphs,
- subgraphs of d-dimensional grids,
- K_t-free unit d-dimensional ball graphs,
- $\Omega(log\ n)$-subdivisions of all the n-vertex graphs,
- cubic expanders defined by iterative random 2-lifts from K_4,
- strong products of two bounded twin-width classes one of which has also bounded degree,
- any subgraph closure of a $K_{t,t}$-free bounded twin-width class, and

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

- any first-order interpretation of a bounded twin-width class. [11]

Bonnet et al. [15] note that clique-width and rank-width extend treewidth to dense graphs. Twin-width, however, is additionally bounded for simple graphs such as unit interval graphs. This large range of classes of graphs with bounded twin-width combined with the simplicity of the concept makes twin-width a highly interesting invariant.

Further, Jacob and Pilipczuk [20] and Hliněný and Jedelský [19] prove bounded twin-width and give bounds for the following classes:

- For a graph G with treewidth $tw(G)$ the twin-width is bounded by $tww(G) \leq 3 \times 2^{tw(G)-1}$.
- For a planar graph G with branchwidth $bw(G) \geq 2$ the twin-width is bounded by $tww(G) \leq max(4 \times bw(G), \frac{9}{2} \times bw(G) - 3)$.
- For a universal bipartite graph $(X, 2^X, E)$ with $|X| = n$ the twin-width is $n - log_2(n) + O(1)$.
- For a planar graph G the twin-width is bounded by $tww(G) \leq 8$.
- For a planar bipartite graph G the twin-width is bounded by $tww(G) \leq 6$.
- For a map graph G the twin-width is bounded by $tww(G) \leq 38$.

2.1.4 Complexity of Computing Twin-Width

Computing the twin-width d and a d-sequence of a graph is a hard problem. Bonnet et al. [15] show that, for some classes of graphs, structural properties can be used to achieve efficient algorithms. For example, for graphs of boolean-width at most k they show that a $(2^{k+1} - 1)$-sequence can be computed in linear time. For a d-dimensional grid of side-length n an algorithm to compute a $3d$-sequence exists. Such algorithms, however, are not generally applicable. In fact, Bergé et al. [7] show that deciding whether a graph has twin-width at most 4 is NP-complete.

2.2 Turbocharging Heuristics

Turbocharging is a technique aimed at improving the quality of greedy heuristics. Generally, the basic idea is the follows: For a given problem, an upper bound k for a relevant property of the desired solution is selected. An example for such a property is the twin-width of a graph. Then, a greedy heuristic computes a solution iteratively as long as the intermediate solution does not violate the bound k. If k is exceeded at a certain step, it is called the *moment of regret*. At this point, a *turbocharging algorithm* is applied. Its goal is to recompute the last c steps in such a way that the newly computed intermediate

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

solution can be extended by one additional step without violating the bound k. Then, the greedy heuristic continues generating a solution [16, 17].

This approach has been applied to various different problems. Examples include Downey et al. [17], who use it on a dynamic version of Dominating Set, Gaspers et al [18] use turbocharging for treewidth, and Abu-Khzam et al. [3] use it on Dominating Set. In many cases, turbocharging lead to improved solutions without sacrificing an unreasonable amount of runtime.

2.3 SAT Solvers

The Boolean satisfiability problem (SAT) is a well known NP-complete decision problem in computer science. SAT solvers accept a propositional formula φ as an input and attempt to evaluate whether there exists a variable assignment such that φ is satisfied. φ is usually represented in conjunctive normal form (CNF) [23]. In practice, applications can view SAT solvers as a black box. They submit a formula encoding the problem they aim to solve and await the result without further interaction [4].

2.3.1 DPLL Algorithm

A commonly used algorithm in modern SAT solvers is the Davis-Putnam-Logemann-Loveland (DPLL) algorithm. It is a sound and complete algorithm, and, therefore, completes in finite time and finds a solution if and only if the input formula φ is satisfiable [4].

DPLL attempts to find a variable assignment that satisfies φ using a branching search with backtracking. It is outlined in Algorithm 2.1. The DPLL algorithm iteratively selects an unassigned variable and assigns a value to it. This is done in first step of the outer loop in $decideNextBranch$. If no unassigned variable exists, this means that a variable assignment satisfying the formula has been found and the algorithm returns that φ is satisfiable. If this is not the case, the algorithm deduces variable assignments based on the decision made. This is done using Boolean Constraint Propagation (BCP). BCP usually applies the *unit clause rule* repeatedly: If a clauses has an unassigned literal and the remaining literals are false, the unassigned literal is implied to be true. However, if a variable is implied to be both true and false at the same time, a *conflict* occurs. If no conflict occurred, the outer loop continues and a new variable is assigned [23].

In the case of a conflict, *backtracking* is performed. Backtracking reverses decisions and implications that lead to the conflict. If every single decision has to be undone, *blevel* is 0 and the algorithm returns that the formula is unsatisfiable. *Chronological backtracking* reverses the most recent decisions until a variable is reached for which the opposite assignment as not been attempted. This approach was used in the original DPLL algorithm [23].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Algorithm 2.1: Basic structure of the DPLL algorithm (after Prasad et al. [23])

```
1  while True do
2      if decideNextBranch() == False then
3          return SAT;
4      else
5          while deduce() == CONFLICT do
6              blevel = analyzeConflict();
7              if blevel == 0 then
8                  return UNSAT;
9              end
10             backtrack(blevel);
11         end
12     end
13 end
```

2.3.2 Implication Graphs

When solving a SAT instance, an implication graph, which is a directed acyclic graph, holds the relationships of implications of variables assignments. Variable assignments are represented by vertices, and edges incident to a vertex indicate the reasons for this assignment. A vertex without incoming arcs is called a decision vertex. Positive and negative variables are assigned 1 and 0, respectively. If a variable is assigned both 0 and 1, a conflict occurs and it is called a *conflicting variable* [26]. Figure 2.3 depicts an implication graph with a conflict.

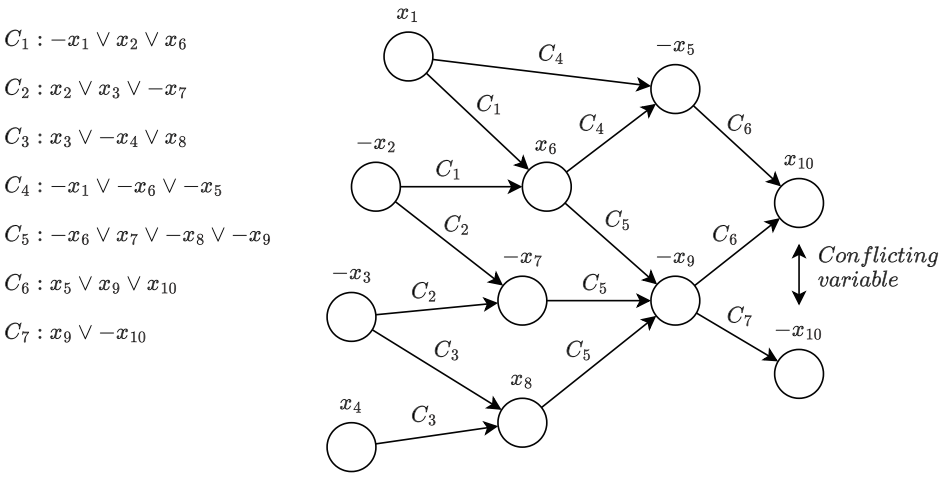

Figure 2.3: Implication graph with conflict after Prasad et al. [23]

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

2.3.3 Conflict-Driven Clause Learning

Modern SAT solvers use conflict learning, or Conflict-Driven Clause Learning (CDCL), and conflict-driven backtracking instead of chronological backtracking. For this, conflict analysis techniques are utilized. They aim to analyze the reason for a conflict [4, 23].

While chronological backtracking works well for random instances, instances of real world problem can have structure that it can not exploit. For instances with some structure, it can be valuable to analyze the root of conflicts in more detail, as the information learned from this may be useful for other parts of the search space. For this, the implication graph is used, enabling *non-chronological backtracking*, or *conflict directed backjumping*. This way, multiple levels of decisions can be undone. Additionally, through conflict analysis, new clauses are added, or *learned*. These clauses are *conflict clauses* and contain information on the reason of the conflict. They prevent the search from making the same mistake again [26].

When a conflict occurs, the implication graph can be partitioned into a *conflict side* containing the conflicting variables, and the *reason side*. This bipartition, or *cut*, can be used to obtain conflict clauses. To find a conflict clause given a cut, the vertices on the reason side have to be examined. If they have at least one edge to the conflict side, they are part of the reason of the conflict. The conflict clause is then constructed from these decision variables. It is usually possible to select different cuts, leading to different learning schemes [26].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 3

State of the Art

As twin-width is a novel concept, the number of approaches for computing it is still limited. The first exact approach was introduced by Schidler and Szeider [24]. Recently, computing twin-width was the subject of the 8th Parameterized Algorithms and Computational Experiments Challenge (PACE 2023). This attracted interest from the research community and resulted in many of the approaches discussed in this chapter.

PACE 2023 featured an exact track in which submitted algorithms had a time budget of 30 minutes per instance to find optimal solutions. The heuristic track required algorithms to complete within 5 minutes but did not require solutions to be optimal [6]. The time constraints on the heuristic track forced participants to focus on speed. In fact, Bonnet and Duron [9] note that a greedy heuristic that executes the contraction leading to the minimum red degree in the next step in the given time frame likely would have won the competition. Given the quadratic time complexity of this approach and the size of the instances, this was practically not possible.

This chapter covers the state of the art of existing exact and heuristic approaches for computing twin-width. Section 3.1 covers SAT encodings as well as the three best performing exact approaches from PACE 2023 *Hydra Prime*, *GUTHMI* and *Touiouidth*. In Section 3.2 the three best heuristic approaches from PACE 2023 are discussed, namely *GUTHM*, *Zygosity* and *Red Alert*. Finally, Section 3.3 covers an example of a turbocharging algorithm.

3.1 Exact Approaches

3.1.1 SAT

Schidler and Szeider [24] propose two SAT encodings to exactly compute twin-width. The encodings give a CNF formula $F(G, d)$ that is satisfiable for a graph G exactly if

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

G has twin-width at most d. By varying d, the exact twin-width can be obtained. To reduce the search space for d, upper and lower bounds are computed.

Preprocessing

Schidler and Szeider [24] propose a graph decomposition that gives a collection $prime(G)$ of induced subgraphs of G such that $tww(G) = max_{H \in prime(G)} tww(H)$. Depending on the input graph, this can lead to smaller graphs to be encoded to SAT leading to less computational effort.

A nonempty set of vertices $M \subseteq V(G)$ is called a *module* if for every pair $u, v \in M$ of vertices the following holds: For any $w \in V(G) \backslash M$ either $uw \in E(G)$ and $vw \in E(G)$ or $uw \notin E(G)$ and $vw \notin E(G)$. If $M = V(G)$ or $|M| = 1$, M is *trivial* and M is *maximal* if it is not strictly contained in any nontrivial module. In a *prime* graph all maximal modules are trivial. The vertices of graph G can be partitioned into a unique P_{max} of maximal modules $M_1, ..., M_s$. From this, the quotient graph G/P_{max} can be constructed: each M_i is a vertex and edges between two maximal modules M_i and M_j are inserted if and only if for all pairs $x_i \in M_i$, $x_j \in M_j$ it holds that $x_i x_j \in E(G)$. A graph isomorphic to G/P_{max} can be obtained by selecting a representative vertex $x_i \in M_i$ for each maximal module M_i and taking the subgraph of G induced by the set $\{x_1, ..., x_s\}$.

$prime(G)$ is then recursively defined as follows:

1. G is disconnected: $prime(G) = \{prime(C) \mid C \in \text{connected components of } G\}$
2. $\overline{G}$ is disconnected: $prime(G) = \{prime(\overline{C}) \mid C \in \text{connected components of } \overline{G}\}$
3. G and $\overline{G}$ are connected: $prime(G) = \{G/P_{max}\} \cup \{prime(G[M]) \mid \text{nontrivial } M \in P_{max}\}$

Elimination Sequences for Twin-Width

For better suitability for formulating SAT encodings, Schidler and Szeider [24] give an alternative definition of twin-width using *d-elimination sequences*. For a graph G this definition uses a *contraction tree* T such that $V(G) = V(T)$ with a root vertex r_T. The *elimination ordering* $\prec$ is a linear ordering such that $u \prec v$ if v is the parent of u in T for $u, v \in V(T)$. The parent of a vertex v_i in T is denoted as p_i and per definition $v_i \prec p_i$. A *twin-width tree decomposition* $(T, \prec)$ is a pair of a contraction tree and an elimination ordering.

$V(G) = \{v_1, ..., v_n\}$ with $v_1 \prec ... \prec v_n$ gives a sequence of graphs $H_0, ...H_{n-1}$ such that $V(H_i) = \{v_i, ..., v_n\}$. $E(H_0)$ is the empty set and for subsequent H_i the edge set is defined recursively:

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

$$E(H_i) = \{uv \in E(H_{i-1}) \mid u, v \in V(H_i)\} \tag{3.1}$$

$$\cup \{up_i \mid v_i u \in E(H_{i-1})\} \tag{3.2}$$

$$\cup \{up_i \mid v_i u \in E(G), p_i u \notin E(G), u \in V(H_i)\} \tag{3.3}$$

$$\cup \{up_i \mid v_i u \notin E(G), p_i u \in E(G), u \in V(H_i)\} \tag{3.4}$$

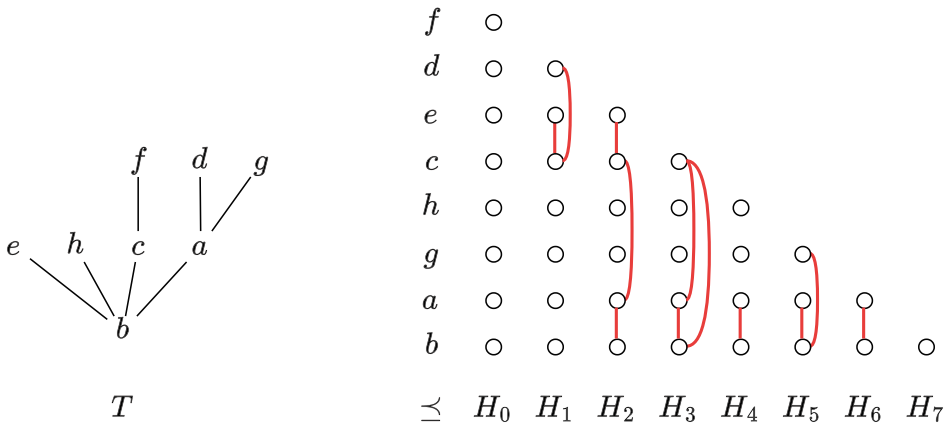

Figure 3.1: T and $H_0, ..., H_{n-1}$ derived from a 2-elimination sequence for the Wagner graph. The sequence is equivalent to the red graphs of the 2-sequence from Figure 2.2. (after [24])

Intuitively, Subset (3.1) states that edges between unchanged vertices remain from H_{i-1} to H_i for vertices which have not been eliminated. Subset (3.2) adds edges between the parent p_i of the newly eliminated vertex v_i to the neighbors of v_i in H_{i-1}, effectively transferring edges from v_i to p_i. Subsets (3.3) and (3.4) ensure that the symmetric difference $N_G(v_i) \triangle N_G(p_i)$ of the neighborhoods of v_i and p_i in G is connected to p_i in H_i as long as the vertices are still part of H_i. Here, $N_G(v_i)$ denotes the neighborhood of v_i. Effectively, the sequence $H_0, ..., H_{n-1}$ is equivalent to the red graphs $(V(G_i), R(G_i))$ in a contraction sequence. In each step, v_i and p_i are equivalent to the pair that is contracted, but instead of a contraction, an elimination is executed. An example for an elimination sequence is depicted in Figure 3.1.

Such a sequence $H_0, ..., H_{n-1}$ defined by $(T, \prec)$ is called an *elimination sequence* for G and a d-elimination sequence if for an integer d the maximum degree of all H_i is at most d. The *width* of a twin-width tree decomposition $(T, \prec)$ is defined as the smallest d for which it admits a d-elimination sequence. Schidler and Szeider [24] show that a graph G has twin-width $\leq d$ if and only if there exists a d-elimination sequence $(T, \prec)$ with width $\leq d$.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Name	Range	Meaning
$a_{i,j}$	$1 \leq i < j \leq n$	$v_i v_j \in E_k$ for some k
$o_{i,j}$	$1 \leq i < j \leq n$	$v_i \prec v_j$
$p_{i,j}$	$1 \leq i < j \leq n$	$p_i = v_j$
$r_{i,j,k}$	$1 \leq i, j \leq n$ and $j < k \leq n$	$v_j v_k \in E(H_{\varphi \prec v_i})$ after eliminating v_i

Table 3.1: Variables of the relative encoding (after [24])

When converting a contraction sequence into an elimination sequence, for each contraction of vertices i and j, either one could be selected to be eliminated. However, it is useful to use the following rule: Eliminate vertex v_j if $i > j$, else eliminate v_i. The consequence of this is that vertex v_n is never eliminated and v_n is the root r_T of the contraction tree.

Relative Encoding

The relative encoding encodes the order of vertices in the elimination ordering relatively for each pair of vertices. For this, the variable $o_{i,j}$ indicates that $v_i \prec v_j$. Table 3.1 gives a summary of all variables used in this encoding. To add transitivity, the clause

$$\neg o^*_{i,j} \vee \neg o^*_{j,k} \vee o^*_{i,k}$$

is added for mutually distinct i, j and k. The notation $o^*_{i,j}$ is an abbreviation for $o_{i,j}$ if $i < j$ and $\neg o_{j,i}$ if $j > i$.

To encode the elimination tree T, the variable $p_{i,j}$ is introduced which is true if and only if $p_i = v_j$. To ensure that every node - except the root node v_n, which is never eliminated - has exactly one parent, for each $i < n$ the at-least-one constraint

$$\bigvee_{i<j} p_{i,j}$$

for all $i < n$ and the at-most-one constraint

$$\neg p_{i,j} \vee \neg p_{i,k}$$

for all mutually distinct i, j, k are added. Finally, for each vertex v_i and its parent v_j, $v_i \prec v_j$ has to hold, which is encoded in the following clause for all $i < j$:

$$\neg p_{i,j} \vee o^*_{i,j}.$$

This gives an encoding for T and $\prec$. The encoding of the sequence of graphs $H_0, ..., H_{n-1}$ remains. For this, the variables $r_{i,j,k}$ with $j < k$ and the auxiliary variables $a_{i,j}$ with $i < j$ are used. $r_{i,j,k}$ is true if and only if $v_j v_k \in E(H_{\varphi \prec (v_i)})$ after v_i has been eliminated.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Here, the mapping $\varphi_{\prec}(v_i)$ is the index of v_i in $\prec$. $a_{i,j}$ encodes whether the edge v_iv_j exists in any H_k, i.e. $a_{i,j}$ is true if and only if there exists a k such that $v_iv_j \in E(H_k)$. To encode r, the definition of $E(H_i)$ given in (3.1) - (3.4) is used. Subset (3.1) of $E(H_i)$ is enforced by the clause

$$\neg o^*_{i,j} \vee \neg o^*_{j,k} \vee \neg o^*_{j,m} \vee \neg r^*_{i,k,m} \vee r^*_{j,k,m}$$

for mutually distinct i, j, k, m with $k < m$. Here, r^* is defined analogously to o^*, the same holds for a^* later. Next, Subset (3.2) is encoded by the following clause for mutually distinct i, j, k with $i < j$:

$$\neg p_{i,j} \vee \neg o^*_{i,k} \vee \neg a^*_{i,k} \vee r^*_{i,j,k}.$$

To complete the semantics of r, Subsets (3.3) and (3.4) have to be encoded as well. For this, for each $v_k \in (N_G(v_i) \triangle N_G(v_j)) \backslash \{v_i, v_j\}$ with $i < j$, the following clause is added:

$$\neg p_{i,j} \vee \neg o^*_{i,k} \vee r^*_{i,j,k}.$$

Finally, the semantics of a are encoded for mutually distinct i, j and k in the clause

$$\neg o^*_{i,j} \vee \neg o^*_{i,k} \vee \neg r^*_{i,j,k} \vee a^*_{j,k}.$$

The only remaining part of the encoding is the upper bound d. This is achieved by limiting the number of true occurrences of $r_{i,j,k}$ for each vertex v_j in each graph H_i. Therefore, $\{\ r^*_{i,j,k} \mid 1 \leq j, k \leq n\ \}$ is limited to at most d true values for each $1 \leq i \leq n$, ensuring the number of neighbors of v_j in each H_i is at most d. Schidler and Szeider [24] use *totalizer* constraints for this.

The resulting formula $F(G, d)$ contains $O(n^4)$ clauses due to the clauses required to encode Subset (3.1). Due to its construction following the definition of elimination sequences, $F(G, d)$ is satisfiable if and only if G has twin-width at most d.

Absolute Encoding

The most significant difference of the absolute encoding to the relative encoding, is that the position of vertices in the elimination ordering is encoded absolutely. For this, the variable $o_{i,j}$ is replaced by $o'_{i,j}$ which is true if and only if $\varphi_{\prec}(v_j) = i$, i.e. when vertex v_j is on the ith position in the elimination ordering. Table 3.2 gives an overview of all variables used in this encoding. This section uses the source code [1] to fill in parts of the encoding not described in detail by Schidler and Szeider [24].

Further, the meaning of indices i, j and k change for the variables r and p. In the absolute encoding, they refer to the position in the elimination ordering $\varphi_{\prec}(v_i)$, $\varphi_{\prec}(v_j)$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Name	Range	Meaning
$o'_{i,j}$	$1 \leq i, j \leq n$	$\varphi_\prec(v_j) = i$
$e_{i,j}$	$1 \leq i < j \leq n$	$\exists uv \in E(G) : i = \varphi_\prec(u) \wedge j = \varphi_\prec(v)$
$ep_{i,k}$	$1 \leq i, k \leq n$	$\varphi_\prec(v_j) = i \wedge jk \in E(G)$
$p_{i,j}$	$1 \leq i < j \leq n - d$	$p_m = v_n : i = \varphi_\prec(m) \wedge j = \varphi_\prec(n)$
$r_{i,j,k}$	$1 \leq i < j < k \leq n - d$	$\exists uv \in E(H_i) : j = \varphi_\prec(u) \wedge k = \varphi_\prec(v)$

Table 3.2: Variables of the absolute encoding

and $\varphi_\prec(v_k)$ instead of the indices of the vertices v_i, v_j and v_k, respectively. Thus, $r_{i,j,k}$ is true if and only if there exists an edge $uv \in E(H_i)$ with $j = \varphi_\prec(u)$ and $k = \varphi_\prec(v)$.

Since the value of $\varphi_\prec(v_i)$ is not known beforehand, it has to be encoded that an edge between $\varphi_\prec(v_i)$ and $\varphi_\prec(v_j)$ exists if i and j are adjacent in G. For this, the variable $e_{i,j}$ is used, where i and j represent the position in the ordering. For the encoding, an additional variable $ep_{i,k}$ is used, which is true exactly if the vertex at position i in the elimination ordering is adjacent to v_k in G. This is ensured with the following clauses for $i < k \leq n$ and $j \leq n$ with $j \neq k$:

$$\begin{array}{ll} \neg o'_{i,j} \vee ep_{i,k} & \text{if } jk \in E(G) \\ \neg o'_{i,j} \vee \neg ep_{i,k} & \text{otherwise.} \end{array}$$

Additionally, the clauses

$$\neg ep_{i,j} \vee o'_{i,x_1} \vee \ldots \vee o'_{i,x_m} \text{ where } x := N_G(v_j) \text{ and } m := |x|$$

reduce the search space. Then, the semantics of $e_{i,j}$ can be encoded using the clauses

$$\begin{array}{l} \neg o'_{i,j} \vee \neg ep_{k,j} \vee e_{i,j} \\ \neg o'_{i,j} \vee ep_{k,j} \vee \neg e_{i,j}. \end{array}$$

The order of eliminations is encoded by the variable o'. Since v_n is never eliminated, the clause $o'_{n,n}$ is added as a fact. To specify that there has to be exactly one vertex at each position in the ordering, at-most one and at-least one constraints are added. For each i at least one and at most one $o'_{i,j}$ must be true.

Similarly to the relative encoding, the parent relationship is encoded using the variable $p_{i,j}$. Using at-least one and at-most one constraints it is ensured that for all $1 \leq i \leq n - d$ exactly one $p_{i,j}$ is true.

Since the indices of $r_{i,j,k}$ refer the the position in the elimination ordering, Subset (3.1) of $E(H_i)$ for $i > 1$ can be succinctly encoded by the clauses

$$\neg r_{i-1,j,k} \vee r_{i,j,k}.$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

To encode Subset (3.2) of $E(H_i)$, for all $1 < i \leq n - d$ and $j < k$ the clauses

$$\neg p_{i,j} \vee \neg r_{i-1,i,k} \vee r_{i,j,k}$$
$$\neg p_{i,k} \vee \neg r_{i-1,i,j} \vee r_{i,j,k}$$

are added. Subset (3.3) and Subset (3.4) of $E(H_i)$ are encoded using the following clauses for all $i \leq n - d$ and $j < k$:

$$\neg p_{i,j} \vee \neg e_{i,k} \vee e_{j,k} \vee r_{i,j,k}$$
$$\neg p_{i,j} \vee \neg e_{j,k} \vee e_{i,k} \vee r_{i,j,k}$$
$$\neg p_{i,k} \vee \neg e_{i,j} \vee e_{j,k} \vee r_{i,j,k}$$
$$\neg p_{i,k} \vee \neg e_{j,k} \vee e_{i,k} \vee r_{i,j,k}.$$

Lastly, the maximal red degree is encoding in the same way as in the relative encoding.

The main advantage of the absolute encoding is its smaller number of variables. The succinct encoding of Subset (3.1) removes the need for $O(n^4)$ clauses, leaving the need for just $O(n^3)$ clauses. Further, the range for i, j, k for $r_{i,j,k}$ is reduced to $i < j < k$. Lastly, the last d trigraphs of the sequence $H_1, ..., H_n$ do not need to be taken into account, as graphs with at most d vertices cannot have twin-width greater than d. These factors lead to significantly less variables and clauses compared to the relative encoding.

A major disadvantage of this encoding, however, is that it is not possible to succinctly encode the fact that the index of a vertex is always smaller than the index of its parent. This leads to symmetries in the encoding. Therefore, the search space is increased.

Comparison

The difference in size of the encodings is expectedly significant. For example, Paley-73 is a graph with 73 vertices, 1314 edges and twin-width 36. The relative and absolute encodings required 30 million and 2.5 million clauses, and 2.5 million and 0.3 million variables, respectively. This is, however, not directly reflected in their performance. When computing the exact twin-width by solving $F(G, tww(G))$ or the last unsatisfiable case $F(G, tww(G) - 1)$, the relative encoding outperforms the absolute encoding by an order of magnitude.

However, Schidler and Szeider [24] find that the performance of the absolute encoding quickly improves for with an increasing $i = 1, 2, ...$ for solving $F(G, tww(G)+i)$. Therefore, they note that it could be useful to compute upper bounds for larger graphs, as its smaller size allows it to be used for graphs the relative encoding can not handle.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

3.1.2 Hydra Prime

Hydra Prime is an exact solver for twin-width proposed by Mizutani et al. [22]. It uses modular decomposition for preprocessing based on the approach described in Section 3.1.1 and then processes each prime graph separately. Hydra Prime contains various algorithms. The `PrimeTreeSolver`, a linear-time exact algorithm for trees without twins, is used for prime graphs that are trees. Otherwise, various upper- and lower-bound algorithms are employed. They are alternated until the bounds match and an optimal solution is found. At first, faster algorithms are used, before progressively switching to slower ones. The following algorithms are used:

Exact algorithms

- `PrimeTreeSolver`: Described above.
- `BranchSolver`: A brute-force solver with caching and reduction rules
- `DirectSolver`: Based on the relative encoding described in Section 3.1.1

Lower-bound algorithms

- `LBGreedy`: A greedy algorithm removing the vertex v of graph G such that the value of $|\triangle(u, v)|$ is minimal at each step. Returns the maximum minimal value of all steps.
- `LBCore`: Uses a SAT solver to find $max_{S \subseteq V(G)} min_{u,v \in S, u \neq v} |\triangle_{G[S]}(u, v)|$.
- `LBSample`: Computes the exact twin-width or lower bound of a connected induced subgraph G' of G found using a random walk.
- `LBSeparate`: Works like `LBSample` but uses the hydra decomposition, a novel approach introduced by Mizutani et al. [22], to find the induced subgraph.

Upper-bound algorithms

- `UBGreedy`: A greedy algorithm minimizing the red degree of the newly created vertex at each step.
- `UBLocalSearch`: Makes small changes to an elimination order to find a better solution.
- `UBSeparate`: Uses the hydra decomposition to refine the solution.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

3.1.3 GUTHMI

Leonhardt et al. [21] proposed GUTHMI, a branch-and-bound algorithm for determining twin-width. GUTHMI first contracts all twins. For determining branching, a similar scoring method to GUTHM (see Section 3.2.1) is used. Lower bounds are obtained by sampling subgraphs. This works, as the twin-width of a graph G can not be lower than the twin-width of any induced subgraph G' of G.

3.1.4 Touiouidth

Touiouidth is a branch-and-bound algorithm proposed by Berthe et al. [8]. It tracks *forbidden contractions* to avoid contractions in branches rooted in sibling nodes that would lead to symmetries. Further, Berthe et al. [8] use the reduction rule

Reduction Rule 1. *If there exist two vertices $u, v \in V(G)$ such that*

- $N_b(u) \subseteq N_b(v)$, *and*
- $(N_r(v) \cup N_b(v)) \subseteq (N_r(u) \cup \{u, v\})$,

then contract u and v.

at each search tree node. Here, $N_b(v)$ and $N_r(v)$ are the open black and red neighborhoods of vertex v, respectively.

Lower bound Heuristics

Similarly to GUTHMI (see Section 3.1.3), induced subgraphs are used to obtain lower bounds. Touiouidth constructs $|V(G)|$ induced subgraphs G_i. At first, each G_i contains a single distinct vertex. Then, new vertices are added to each G_i iteratively. A new vertex v has to be connected to G_i and has to maximize the minimal the symmetric difference of neighborhoods of v in G with vertices in G_i. Each time a vertex is added to any G_i, the twin-width of the new induced subgraph is calculated and the lower bound is updated if it is tighter than the best one found thus far.

Upper bound heuristics

Touiouidth uses two heuristics to computer upper bounds. `Heuristic1` is a greedy heuristic. It iteratively selects two vertices u, v such that $\triangle(u, v)$ is minimal and contracts them. `Heuristic2` relies on random contractions and can be used several time to find better solutions. It takes a graph G and an upper bound T as input. It then constructs a contraction sequence by contracting a random pair of vertices such that the red trigraph does not exceed degree $T - 1$. If a complete contraction sequence is found, it can be used as the upper bound for another iteration.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

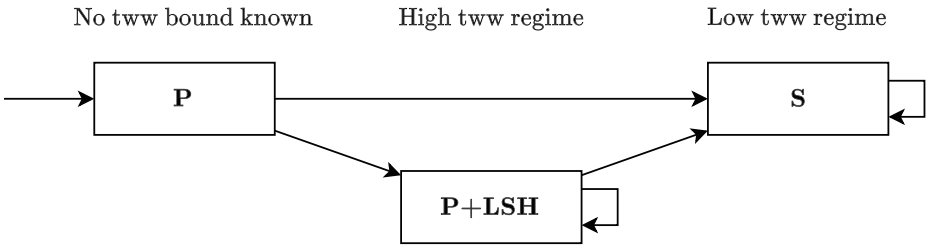

Figure 3.2: Heuristic strategies used by GUTHM (after [21])

3.2 Heuristic Approaches

3.2.1 GUTHM

GUTHM is a greedy heuristic proposed by Leonhardt et al. [21]. It consists of the three strategies **P**, **S** and **P+LSH**. For disconnected graphs, each connected component is handled in isolation. After obtaining trivial contraction sequences for each component using **P**, the component with the highest bound for the twin-width is processed.

GUTHM starts by obtaining an initial upper bound for the twin-width using **P**. Depending on the the expected twin-width, the strategy is changed. If it is expected to be high, **P+LSH** is used repeatedly. If the twin-width is expected to be low or **P+LSH** has found it to be low, **S** is used repeatedly until the heuristic terminates. Figure 3.2 depicts the sequence of the strategies. The algorithm can only move from left to right.

P: Priority based solver

P, the priority based solver, selects a vertex u with minimal red degree and uses the scoring function $score(u, v)$ to select a vertex v from the 2-neighborhood of u. For this, they propose an adjusted two-level BFS approach to speed up the calculation. In $score(u, v)$, the sets R_{new} and R_{rem} represent the sets of red edges added and removed by the contraction of u and v, respectively.

$$score(u, v) = \sum_{(v,x)\in R_{new}} (redDeg(x) + 1) - \sum_{(v,y)\in R_{rem}} redDeg(y)$$

S: Sweeping based solver

The sweeping based solver **S** works by selecting a threshold for the twin-width and executing contractions that do not lead to the threshold being exceeded. This is done in multiple rounds. At the start of each round, the threshold is established and continuously tweaked to balance execution speed and accuracy.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

P+LSH: Priority with support for locality sensitive hashing

The final approach **P+LSH** attempts to improve the solution obtained by **P** for d-sequences with high d. For this, global information is collected using MinHashing. Additionally, local information is used in a similar way to **P**. This, however, does not handle finding near-twins with high red degree. MinHashing enables finding near-twins in large graphs.

3.2.2 Zygosity

Zygosity is a randomized greedy heuristic proposed by Arrighi et al. [5]. It uses two preprocessing steps: At first, it contracts all twins. Then, trees are contracted in a manner described by Bonnet et al. [15]: Either two leaves which share a parent, or a leaf and its parent are contracted.

The contraction sequence is constructed by evaluating a number of random contractions at each step. This number depends on the remaining time as well as how long it takes to consider one contraction on average. The best contraction is then evaluated by two measures:

- **Red degree**: When contracting two vertices, the red degree of some vertices may increase. For the newly created node, the red degree is considered increased if it is greater than both contracted vertices. The measure considered is the highest red degree of any vertex with an increased red degree due to the contraction. This is the intuitively most important measure to keep low to obtain a d-sequence with a low value of d.

- **Intersection size**: The Intersection size refers to the number of shared neighbors of the contracted vertices. If the contracted vertices are adjacent this contributes one to the intersection size. The intersection size is also equal to the number of edges removed by the contraction. Intuitively, removing more edges leads to fewer problems later on.

The red degree is the primary measure used by Zygosity. The intersection size is only used to break ties when multiple candidates have the same red degree. However, if two contractions being compared do not increase the red degree of the current partial contraction sequence, only the intersection size is considered.

Zygosity does not pick potential contraction pairs completely at random. Instead, a single vertex is selected randomly. From there, a random walk of either one or two steps is performed. The length of the walk is selected randomly but biased based on the density of the input graph. On dense graphs longer walks are favored.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

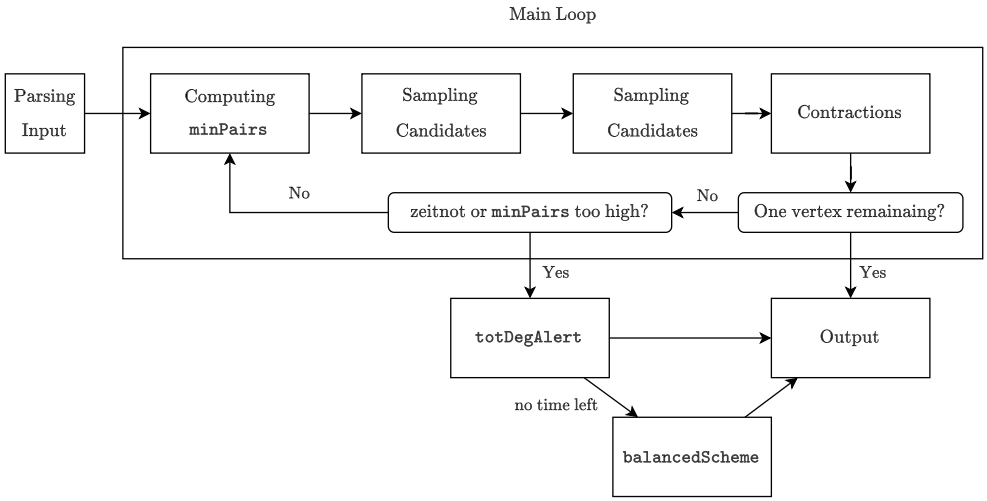

Figure 3.3: Overview of the Red Alert heuristic (after [9])

3.2.3 Red Alert

Red Alert is a heuristic presented by Bonnet and Duron [9] which works by sampling candidates to contract. In each round, about 10000 pairs of vertices are sampled and at least `minPairs` of them are contracted. If the remaining time is not sufficient to complete this computation, the faster subroutines `totDegAlert` and `balancedScheme` are used to obtain a solution in time. Figure 3.3 gives an overview of the entire Red Alert heuristic. In the main loop `minPairs` is estimated and *candidate pairs* are sampled. Then at least `minPairs` are selected from the candidate pairs based on a cost function and the selected pairs are contracted. If the remaining time is too low, increasingly cruder heuristics are used.

Computing `minPairs`

In each round of the main loop at least `minPairs` have to be chosen from the sampled candidate pairs. The value of `minPairs` is chosen based on the time the last iteration took t and the remaining time T. While iterations of the main loop do not all take the same amount of time, T/t still gives a reasonable estimate for how many more iterations are possible within the time budget. For a graph with k vertices in each iteration $\frac{k}{T/t}$ pairs should be contracted, therefore `minPairs` $= tk/T$. `minPairs` also affects sampling: For a small `minPairs` the sampling size is increased to make more use of the time and decreased if `minPairs` gets larger.

Sampling

The selection of candidates depends on whether the input graph is sparse or dense. For sparse graphs, the candidate distribution is biased toward selecting vertices that are close

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

to each other. At first, a vertex v is picked uniformly at random. Then either a neighbor of v in the input graph or a second neighbor of v in the input graph is chosen uniformly. Either case is chosen with a probability 1/2. This way, half of the pairs have distance of 2 which naturally leads to contractions decreasing the red degree. For dense graphs, candidates are sampled uniformly as most pairs of vertices have a distance of 2.

Filtering candidates

To evaluate candidates, a cost function $f(u, v) = (r, p, e)$ is used where u and v are vertices of the current trigraph G. Let G' be the trigraph obtained by contracting u and v, and w be the newly created vertex. Then r, p and e are defined by Bonnet and Duron [9] as follows:

- $r \in \{0, 1\}$ is 0 iff the maximum red degree of G' is smaller than that of G.
- p is the maximum red degree among vertices of G' in the closed red neighborhood of w.
- e is the total number of red edges in G'. [9]

When comparing two pairs of vertices, p is only considered if r is equal for both, and e only if p is equal as well, giving r the highest priority followed by p and e, respectively. The pair with a lower value is preferred. It is not possible to contract two pairs sharing a vertex and the value f for each pair can change with each contraction. This is a problem since `minPairs` candidate pairs need to contracted. Bonnet and Duron [9] solve this problem by using a min-heap according to f to store candidate pairs. Contractions are then executed as follows: First, the minimal candidate c is popped from the heap. If one of its vertices does not exist anymore, the next candidate is selected. Then, $f(c)$ is computed and compared to the last time it was evaluated. If $f(c)$ has not gotten worse, c is contracted. Otherwise, c is inserted into the heap again, but with the updated value $f(c)$.

Fast subroutines

As Red Alert is designed for the format of PACE 2023, it includes the faster subroutines `totDegAlert` and `balancedScheme` to finish within the given time budget if the remaining time gets low or `minPairs` gets too large. It is possible that `minPairs` increases steadily. This is usually caused by the trigraph getting increasingly more dense. Given that the black degree of vertices can not get higher and the input graph was rather sparse in comparison, the degree of a vertex is now a good approximation for the red degree. `totDegAlert` builds on this observation and greedily contracts candidates with the smallest total degree. If even `totDegAlert` is too slow to terminate in time, `balancedScheme` is used to output a valid solution before the time budget is used up. It splits the remaining vertices into buckets which are contracted into a single vertex each. The remaining contraction sequence is arbitrary.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

3.3 Turbocharging Treewidth Heuristics

Gaspers et al. [18] apply the approach of turbocharging heuristics to treewidth. Treewidth is a graph invariant, much like twin-width. They use two different heuristics and show that the turbocharging algorithm is FPT. Here, only one heuristic is described to illustrate the approach.

3.3.1 Definitions

A *tree decomposition* $\tau = (T, \chi)$ of a graph $G = (V, E)$ consists of a tree T and a mapping χ from each vertex $t \in T$ to a *bag* $\chi(t) \subseteq V$ such that

1. each vertex v is in at least one bag, i.e. $\forall v \in V : \exists t \in T : v \in \chi(t)$
2. for each edge uv there exists a bag that contains u and v, i.e. $\forall uv \in E : \exists t \in T : \{u, v\} \subseteq \chi(t)$
3. for each $r, s, t \in T$ where there exists a path from r to t containing s: $\chi(r) \cap \chi(t) \subseteq \chi(s)$.

The size of the largest bag of a tree decomposition minus one is its *width*. Then, the treewidth $tw(G)$ of a graph G is defined as the smallest possible width of any tree decomposition of G.

Treewidth can also be defined by *elimination orders*. To avoid confusion with elimination orderings for twin-width as defined in Section 3.1.1, elimination orders are going to be referred to as *treewidth elimination orders*. An elimination in this context entails forming a clique of all neighbors of a vertex v of a graph G and then removing v as well as all edges incident to v. More formally, a new graph $G' = (V(G)\backslash\{v\}, (E(G) \cup E_1)\backslash E_2)$ with $E_1 = \{uw \mid u, w \in N_G(v)\}$ and $E_2 = \{e \in E(G) \mid v \in e\}$ is created from G. Then, a treewidth elimination order is a bijective function $\pi : V \to \{1, ..., n\}$ where $n = |V(G)|$. From this, a sequence of graphs can be obtained by starting with G and eliminating vertices in the order given by π. The maximum degree of all graphs in this sequence is the width of π. A graph G has treewidth at most k if and only if here exists an elimination order π of width at most k.

3.3.2 TurbochargedMinDegree

Gaspers et al. [18] propose the TurbochargedMinDegree algorithm shown in Algorithm 3.1. It is based on the GreedyDegree heuristic. GreedyDegree works by always adding the vertex with minimum degree to the treewidth elimination order.

To turbocharge a greedy treewidth heuristic, the *incremental conservative (IC) treewidth problem* or IC-Treewidth has to be solved. As an input it receives a graph G, two integers k and c, and a partial treewidth elimination order π with length l and width $\leq k$.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

The problem is to determine whether there exists a partial treewidth elimination order π' with length $l+1$ and width $\leq k$ such that the first $l-c$ items of π and π' are the same. Gaspers et al. [18] show that IC-Treewidth is FPT when parameterized by c and k.

Algorithm 3.1: TurbochargedMinDegree [18]

Data: Graph $G = (V, E)$, integer k, integer c.
Result: Elimination order of width $\leq k$ or *no* if none was found.

```
1  H ← G;
2  π ← ();
3  for i ← 1 to |V| do
4  |   choose vertex v with minimum degree;
5  |   if d(v) ≤ k then
6  |   |   π ← π + (v);
7  |   |   H ← eliminate(H, v);
8  |   else
9  |   |   G' ← eliminate(G, π[1], ..., π[i − c − 1]);
10 |   |   W ← {v ∈ V(G') | d(v) ≤ k};
11 |   |   (H, π') ← IC-Treewidth(G', W, k, c + 1);
12 |   |   if π' is empty then
13 |   |   |   return no;
14 |   |   else
15 |   |   |   π ← (π[1], ..., π[i − c − 1]) + π';
16 |   |   end
17 |   end
18 end
19 return π
```

Finally, the TurbochargedMinDegree algorithm has to be embedded into a search algorithm. This is due to the fact, that it only solves the decision problem of finding a treewidth elimination order given the parameter k, which is unknown. For this, Gaspers et al. [18] use a biased binary search approach. Based on their experimental evaluation, they found an improvement of treewidth of 3-5%. Therefore, they typically used the search range $[0.94 \times k', k' - 1]$ where k' is the best treewidth heuristically found without turbocharging. The authors conclude that the improvements yielded by their approach are a reasonable trade-off for the increased running time.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 4

Design

This chapter discusses the design of a new turbocharged greedy heuristic for twin-width. Section 4.1 presents the algorithmic problem involved in the turbocharging algorithm and Section 4.2 proposes two SAT encodings to solve it. Section 4.3 covers the heuristics we use in our algorithm. Finally, Section 4.4 develops a heuristic algorithm for twin-width using these pieces.

4.1 IC-Twin-Width

To improve an existing greedy heuristic using local search, a subproblem has to be solved. We call this problem the *incremental conservative twin-width* problem. It is concerned with extending an elimination sequence without exceeding a maximal red degree. Before giving a formal definition, some preliminaries are needed.

We first give a modified definition of d-elimination sequences by Schidler and Szeider [25]. For a graph G, instead of using a tree to represent the parent relationship, a rooted forest T, in which each tree is a rooted tree, is used. The vertices of T are a subset $V(T) \subseteq V(G)$ of the vertices of G. Further, the set of non-roots is denoted as $N(T)$ and $\prec$ is a linear ordering such that $u \prec v$ for any $u, v \in N(T)$ if v is the parent of u in T. Schidler and Szeider [25] then refer to T as a *contraction forest*, $\prec$ as an *elimination ordering* and define a *twin-width decomposition* of G as the pair $(T, \prec)$.

Let V' be $V(G) \backslash N(T)$ and $k = |N(T)|$. The ordering of vertices $v_1, ..., v_k$ together with T and G then define a sequences of trigraphs $G_0, ..., G_k$ such that $V(G_i) = \{v_{i+1}, ..., v_k\} \cup V'$. p_i refers to the parent of v_i in T. G_0 is defined as G and G_i is obtained from G_{i-1} by eliminating v_i with its parent p_i. An *elimination sequence* is now defined as the sequence $G_0, ..., G_k$ of graphs. It is a d-elimination sequence if the maximal red degree of each G_i is at most d.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

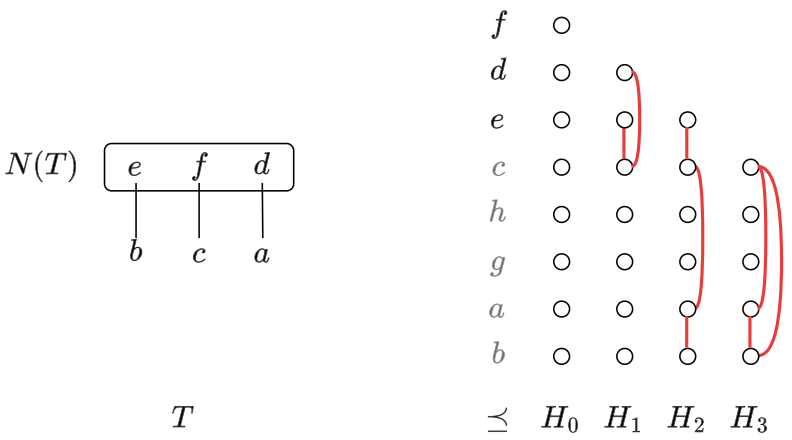

Figure 4.1: T and $H_0, ..., H_3$ derived from a twin-width decomposition of width 2 of the Wagner graph consisting of 3 eliminations.

Note that $\prec$ in general only orders a subset of all vertices of G. If $|V(G)| = k$, $(T, \prec)$ is called a *full twin-width decomposition* consisting of a *full contraction forest* T and a *full elimination ordering* $\prec$. Further, $G_0, ..., G_k$ then is a *full elimination sequence*. Note that in this case, T is a tree with a single root vertex. Figure 4.1 depicts an example of a twin-width decomposition that is not a full twin-width decomposition.

We define the sequence of graph $H_0, ..., H_k$ in the same way as in Section 3.1.1: H_i is the red graph of G_i defined as $H_i = (V(G_i), R(G_i))$. The only difference is, that length of the sequence does not have to contain $|V(G)|$ graphs, as G does not have to be a full elimination sequence.

We can then also define the set of edges of each H_i recursively in the exact same way for elimination sequences, as they are for full elimination sequences for $1 \leq i \leq k$ and $E(H_0) = \emptyset$:

$$E(H_i) = \{uv \in E(H_{i-1}) \mid u, v \in V(H_i)\} \tag{4.1}$$

$$\cup \{up_i \mid v_i u \in E(H_{i-1})\} \tag{4.2}$$

$$\cup \{up_i \mid v_i u \in E(G), p_i u \notin E(G), u \in V(H_i)\} \tag{4.3}$$

$$\cup \{up_i \mid v_i u \notin E(G), p_i u \in E(G), u \in V(H_i)\} \tag{4.4}$$

IC-Twin-Width

Instance: Graph G, an elimination sequence $G_0, ..., G_i$ of width $\leq k$ defined by $(T, \prec)$, integer k and c.

Problem: Does there exist an elimination sequence $G_0, ..., G_{i+c}$ of width $\leq k$ for G?

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

It is easy to see that IC-Twin-Width is NP-complete. This is the case, since it can be reduced to SAT (e.g. using one of the encodings described below) and twin-width can be reduced to IC-Twin-Width: An instance (G, d) of twin-width can be solved by passing it to an algorithm for IC-Twin-Width by providing G, an elimination sequence consisting of just $H_0 = (V(G), \emptyset)$, d, and $|V(G)| - 1$. Finally, $(T, \prec)$ serves as a certificate, which can be checked in polynomial time by executing the eliminations and computing the red degree at each step.

Further, IC-Twin-Width is solvable in FPT time when parameterized by c. For each elimination, $O(|V(G)|^2)$ pairs of vertices can be considered as candidates. Since only c steps have to be computed, there are $O(|V(G)|^{2^c})$ possible sequences of eliminations to be considered, which is polynomial for a fixed value of c. Just as for twin-width, checking the width of such a sequence is also possible in polynomial time. Given the size of the exponent, however, a naive algorithm would likely not be reasonably fast for any practical value of c.

4.2 Turbocharging Algorithm

This section covers turbocharging algorithms for twin-width. As discussed in Section 3.3, a turbocharging algorithm is used to exactly compute a number of steps of an iteratively constructed solution. In the case of twin-width, this is an elimination sequence which should be extended by a number of eliminations without exceeding the target twin-width k. This is the problem of IC-Twin-Width defined above. For this, we use SAT encodings based on the encodings proposed by Schidler and Szeider [24].

We therefore assume that a k-elimination sequence $G'_0, ..., G'_i$ for a graph G' is provided as an input. In this section, we want to extend this sequence by $c + 1$ eliminations. As the steps $G'_0, ..., G'_{i-1}$ are not required to be known for our implementation, we are using the following definitions for simplicity: $H'_j = (V(G'_j), R(G'_j))$ for all j, $H_0 = H'_i$, and $n = |V(G')|$. Instead of writing that the part of the sequence $G'_{i+1}, ..., G'_{i+c+2}$ is being computed, we write that $G_1, ..., G_{c+1}$ is being computed. Since G_i can be obtained from H_i and G', we are mostly concerned with computing $H_1, ..., H_{c+1}$. The input for generating the encodings then is the trigraph $G = G'_i$. We denote the vertices of G as $\{v_1, ..., v_n\}$.

4.2.1 General Approach

To encode IC-Twin-Width such that it can be solved by a SAT solver, we closely follow the definition of elimination sequences given above. They, therefore, need to encode the following elements for the values k and c:

- H_0: As $E(H_0)$ does not have to be empty, we need to encode the edges it contains. This represents the state after reversing c steps after the moment of regret in the turbocharged algorithm.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Name	Range	Meaning
$a_{i,j}$	$0 \leq i < j \leq n$	$v_i v_j \in E_k$ for some k
$o_{i,j}$	$0 \leq i < j \leq n$	$v_i \prec v_j$
$p_{i,j}$	$0 \leq i < j \leq n$	$p_i = v_j$
$r_{i,j,k}$	$0 \leq i, j \leq n$ and $j < k \leq n$	$v_j v_k \in E(H_{\varphi \prec v_i})$ after eliminating v_i
s_i	$0 \leq i \leq n$	v_i is eliminated in the elimination sequence

Table 4.1: Variables of the relative encoding

- **Elimination ordering**: When encoding the elimination ordering, we need to encode $c+1$ steps $v_1 \prec .., \prec v_{c+1}$. These can be appended to $v'_1 \prec ... \prec v'_i \prec v_1 \prec .., \prec v_{c+1}$, yielding an elimination ordering.
- **Contraction forest**: Next, we need to encode the contraction forest. For this, we encode the parents of the eliminated vertices $v_1, ..., v_{c+1}$. Since each eliminated vertex has exactly one parent that has not been eliminated at that step, the parent relationship is a rooted forest. Combining it with T leads to a contraction forest.
- **Elimination sequence**: Using the recursive definition of H_i and the definition of H_0, we encode $H_1, ..., H_{c+1}$. Per definition, $G_1, ..., G_{c+1}$ can the be constructed from all H_i and G. The sequence $G'_0, ..., G'_i, G_1, ..., G_{c+1}$ then is an elimination sequence.
- **Bounds**: Finally, we have to ensure that the degree of each H_i is at most k. Combined with the assertion that $G'_0, ..., G'_i$ has width at most k, this ensures that $G'_0, ..., G'_i, G_1, ..., G_{c+1}$ has width at most k as well.

The encodings presented below both follow this approach, but encode it in different ways.

4.2.2 Relative Encoding

This encoding extends and modifies the relative encoding presented in Section 3.1.1. The variables of this encoding are listed in Table 4.1. To suit the IC-Twin-Width problem, we made two central modifications:

- New variables s_i are introduced. They are used to track vertices that are eliminated in the elimination sequence.
- A new imaginary vertex v_0 is introduced that has no adjacent edges and is always eliminated first. This enables $r_{0,j,k}$ to carry information about red edges in the input trigraph. The range of all variables is also extended down to 0 to accommodate this vertex.

To produce an elimination ordering of length $c+1$, exactly $c+2$ vertices have to be selected (since v_0 is always selected). For this, at-most and at-least constraints are used

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

to enforce that s_i is true for exactly $c+1$ vertices v_i with $i \geq 1$ and the clause s_0 ensures that v_0 is selected. The following clauses for $i \geq 1$ ensure that v_0 is always eliminated first and that it its parent is a selected vertex. This makes sure that the red edges in the input trigraph are transferred to the first vertex in the elimination ordering (via clauses transferring red edges discussed later):

$$o_{0,i}$$
$$\neg p_{0,i} \vee s_i.$$

If a vertex v_i is not eliminated in the elimination sequence, it does not need to be part of the order and parent relationships, nor do the red edges after eliminating it have to be tracked (since it is never eliminated). Therefore, $p_{i,j}$, $o_{i,j}$ and $r_{i,j,k}$ can be set to false for all j and k. This is encoded using the following clauses for all $i < j$:

$$s_i \vee \neg p_{i,j}$$
$$s_i \vee \neg o_{i,j}$$

and for all i and $j < k$

$$s_i \vee \neg r_{i,j,k}.$$

For each $i < j$, the existence of a red edge $ij \in R(G)$ or lack thereof is encoded by the clauses $r_{0,i,j}$ and $\neg r_{0,i,j}$, respectively.

Since vertices that are not selected do not need to have a parent, the at-least and at-most one constraints on $p_{i,j}$ need to be modified. This is done such that for every i exactly one $p_{i,j}$ or $\neg s_i$ has to be true.

Next, the transfer and creation of red edges has to be adapted, since $r_{i,j,k}$ has been set to false for vertices v_i which are not selected. This is done by adding $\neg s_i$ to all clauses containing $r_{i,j,k}$. The resulting clauses are

$$\neg s_i \vee \neg o^*_{i,k} \vee \neg o^*_{j,k} \vee \neg o^*_{j,m} \vee \neg r^*_{i,k,m} \vee r_{j,k,m}$$

for mutually distinct i, j, k, m with $k < m$, encoding Subset (4.1) of $E(H_i)$,

$$\neg s_i \vee \neg p_{i,j} \vee \neg o^*_{i,k} \vee \neg a^*_{i,k} \vee r_{i,j,k}$$

for mutually distinct i, j, k with $i < j$ encoding Subset (4.2) of $E(H_i)$, and finally, subsets Subset (4.3) and Subset (4.4) of $E(H_i)$ for each i if $v_k \in (N_G(v_i) \triangle N_G(v_j)) \backslash \{v_i, v_j\}$ are encoded by the clauses

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

$$\neg s_i \vee \neg p_{i,j} \vee \neg o^*_{i,k} \vee r_{i,j,k}.$$

The remaining encoding follows the relative encoding covered in Section 3.1.1 closely. The only differences are that indices down to 0 are used to ensure $\{r_{i,j,k} \mid 0 \leq n\}$ as well as when encoding $\neg p_{i,j} \vee o^*_{i,j}$ and the semantics of $a^*_{i,j}$.

A major disadvantage of this encoding is its size. There are $O(n^3)$ variables $r_{i,j,k}$ and encoding their semantics requires $O(n^4)$ clauses. For any assignment of all s_i, however, the search space is greatly reduced by in particular the clauses $s_i \vee \neg r_{i,j,k}$, leaving just $O(n^2 c)$ variables without an immediately forced assignment.

4.2.3 Absolute Encoding

Like the absolute encoding presented in Section 3.1.1, this encoding assigns an absolute position to each vertex in the sequence. This is encoded in the variables $o_{i,j}$ where i is the position and can not exceed $c+1$, as IC-Twin-Width only requires an elimination sequence of length $c+1$ to be computed. The same goes for the variables $p_{i,j}$, which denotes that v_j is the parent of the vertex eliminated a the ith position, and for $r_{i,j,k}$ where, again, i represents the position and j and k represent v_j and v_k. An overview of all variables of this encoding is given in Table 4.2.

Name	Range	Meaning
$o_{i,j}$	$1 \leq i \leq c+1, 1 \leq j \leq n$	$\varphi_\prec(v_j) = i$
$el_{i,j}$	$1 \leq i \leq c+1, 1 \leq j \leq n$	$\varphi_\prec(v_j) \leq i$
$p_{i,j}$	$1 \leq i \leq c+1, 1 \leq j \leq n$	$p_m = v_n : i = \varphi_\prec(m) \wedge j = \varphi_\prec(n)$
$r_{i,j,k}$	$0 \leq i \leq c+1, 1 \leq j < k \leq n$	$\exists v_j v_k \in E(H_i)$
$a1_{i,j}$	$1 \leq i \leq c+1, 1 \leq j \leq n$	$\exists v_m : \varphi_\prec(v_m) = i \wedge v_j v_m \in E(H_i)$
$a2_{i,j}$	$1 \leq i \leq c+1, 1 \leq j \leq n$	$\exists v_m : \varphi_\prec(v_m) = i \wedge v_j v_m \in E(G)$

Table 4.2: Variables of the absolute encoding

To make sure $o_{i,j}$ is assigned exactly once for each i, at-most one and at-least one constraints are used. Further, for each j, at most one $o_{i,j}$ must be true, which is ensured using at-most one constraints.

Each vertex eliminated in the elimination sequence needs to have exactly one parent. This is again achieved by using at-most one and at-least one constraints for $p_{i,j}$ for each i. Additionally, the following clauses ensure that no vertex that has been eliminated already can be the parent of a vertex eliminated later for $i \leq m$:

$$o_{i,j} \vee \neg p_{m,j}.$$

Just like in the relative encoding discussed above, the red edges of the input trigraph have to be encoded. For each $i < j$, the existence of a red edge $ij \in R(G)$ or lack thereof is encoded by the clauses $r_{0,i,j}$ and $\neg r_{0,i,j}$, respectively.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Next, the creation and transfer of red edges has to be encoded. For this, the auxiliary variable $el_{i,j}$ represents that $v_j \notin V(H_i)$, i.e. that v_j has already been eliminated at step i. Let $r^*_{i,j,k}$ be $r_{i,j,k}$ if $j < k$ and $r_{i,k,j}$ otherwise. The naive way to encode red edges the is as follows:

Subset (4.1) of $E(H_i)$ can be encoded for mutually distinct j, k and $i \geq 1$ by the clauses

$$el_{i,j} \vee el_{i,k} \vee \neg r^*_{i-1,j,k} \vee r^*_{i,j,k}.$$

Note that there is no constraint of $i > 1$ so that red edges from H_0 are transferred to H_1. Subset (4.2) can be encoded by the following clauses for mutually distinct j, k, m and $i \geq 1$:

$$\neg o_{i,m} \vee \neg p_{i,j} \vee \neg r^*_{i-1,m,k} \vee r^*_{i,j,k}.$$

Subset (4.3) and Subset (4.4) can be encoded by adding the following clauses if $v_k \in (N_G(v_j) \triangle N_G(v_m)) \backslash \{v_j, v_m\}$

$$el_{i,k} \vee \neg o_{i,m} \vee \neg p_{i,j} \vee r^*_{i,j,k}.$$

However, this approach requires $O(n^3 c)$ clauses. This can be reduced by introducing additional auxiliary variables: $a1_{i,k}$ is true if there exists an i, m, k with $m \neq k$ such that $\varphi(v_m) = i$, there exists a red edge $mk \in H_{i-1}$, and k is still part of H_i. It is encoded using the following clauses;

$$el_{i,k} \vee \neg o_{i,m} \vee \neg r^*_{i-1,m,k} \vee a1_{i,k}.$$

Subset (4.2) can then be encoded more succinctly for all i, j, k with $j < k$ by the clauses

$$\neg a1_{i,k} \vee \neg p_{i,j} \vee r^*_{i,j,k}.$$

Next, $a2_{i,k}$ is true if and only if there exists an edge $mk \in G$ and $\varphi(v_m) = i$. This is encoded in the following clauses if $mk \in G$ and $mk \notin G$, respectively:

$$\neg o_{i,m} \vee a2_{i,k},$$
$$\neg o_{i,m} \vee \neg a2_{i,k}.$$

Now Subset (4.3) and Subset (4.4) can be encoded for all $v_k \in (N_G(v_j) \triangle N_G(v_m)) \backslash \{v_j, v_m\}$:

$$el_{i,k} \vee \neg a2_{i,k} \vee \neg p_{i,j} \vee r^*_{i,j,k}.$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Finally, it has to be ensured that the degree of all H_i is at most k. This is encoded using at-most constraints on $r_{i,j,k}$ for all i and j.

This encoding requires $O(n^2c)$ variables and $o_{i,j}$, $el_{i,j}$ and $p_{i,j}$ require only $O(nc)$ variables each. Black edges do not have to be encoded explicitly at all. By using the auxiliary variables $a1_{i,j}$ and $a2_{i,j}$ to encode red edges, the number of clauses is reduced from $O(n^3c)$ to $O(n^2c)$.

4.3 Heuristics

For the greedy heuristic of our turbocharged algorithm, we select two heuristics: MinIntersectionSize and MinRedDegree.

4.3.1 MinIntersectionSize

MinIntersectionSize greedily selects the pair of vertices u, v such that $\triangle(u, v)$ is minimized. This is equivalent to Heuristic1 of Touiouidth by Berthe et al. [8]. Since we are using eliminations instead of contractions, we eliminate the vertex of u, v with the lower index and select the other one as its parent.

4.3.2 MinRedDegree

MinRedDegree is inspired by Zygosity by Arrighi et al. [5]. It uses two measures to select the next vertex v_i to eliminate and its parent p_i:

- **Red Degree**: The red degree here refers to the red degree of the entire graph after eliminating v_i. This is the primary measure.
- **Intersection size**: If multiple candidate eliminations lead to the same red degree, the intersection size is used as a tie breaker. It refers to the size of the intersection of the neighborhoods of v_i and p_i. It is increased by one if v_i and p_i are connected by an edge.

Like in MinIntersectionSize, we define $v_i < p_i$ for simplicity. Unlike Zygosity, we do not sample candidates. Instead, all possible eliminations are evaluated. This is due to the fact that Zygosity is geared more towards speed than MinRedDegree. The loss of speed is expected to be overshadowed by the run time of the turbocharging algorithm in most cases. Therefore, checking all candidates with the expectation of finding a better sequence outweighs the cost in speed in our case.

4.4 Turbocharged Algorithm

We have presented SAT encodings which can be used to solve IC-TwinWidth as well as greedy heuristics for twin-width. From these, we now construct a turbocharged heuristic.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Algorithm 4.1 shows the resulting algorithm using the MinRedDegree heuristic as an example.

The elimination sequence is constructed iteratively by a greedy heuristic. If the desired width of k is exceeded, a moment of regret is reached. The last c steps of the solution are reversed and an exact algorithm for IC-TwinWidth using one of the SAT encodings discussed before is used to compute the next $c+1$ steps without exceeding width k. If this fails, the algorithm returns *no*, as it cannot find an elimination sequence of width $\leq k$. Otherwise, the heuristic continues construction a solution.

Contrary to the turbocharging approach for treewidth by Gaspers et al. [18], we limit the number of steps that are reversed at the moment of regret to the number of steps the greedy algorithm constructed since the last moment of regret (or the start of the algorithm). This prevents the possibility of the heuristic repeatedly not being able to find a single contraction within the constraints of k causing IC-TwinWidth being called repeatedly and extending the sequence by just one contraction each time. This is done because the run time of IC-TwinWidth tends to be much longer compared to constructing a single step greedily.

Since the optimal value for k is not known beforehand, we need to wrap TurbochargedMinRedDegree into a search algorithm. We start by obtaining an upper bound for the twin-width by running the heuristic without turbocharging. Then k is decremented by one and TurbochargedMinRedDegree constructs a new elimination sequence s of width $\leq k$. k is set to the width of s minus one and the process is repeated until TurbochargedMinRedDegree fails to find an elimination sequence for the given k. As a preprocessing step we merge all twins. Algorithm 4.2 outlines this approach using MinRedDegree as the heuristic. TurbochargedMinIntersectionSize works similarly, but uses MinIntersectionSize instead of MinRedDegree.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Algorithm 4.1: TurbochargedMinRedDegree

Data: Graph $G = (V, E)$, integer k, integer c.
Result: An elimination sequence of width $\leq k$ encoded as a list of pairs (v_i, p_i) of eliminated vertices and their parents or *no* if none was found.

```
1  G' ← G;
2  s ← ();
3  greedySteps ← 0;
4  for i ← 1 to |V| do
5  |  choose vertex v and parent p using MinRedDegree;
6  |  s ← s + (v, p);
7  |  G' ← eliminate(H, (v, p));
8  |  greedySteps ← greedySteps + 1;
9  |  if redDegree(H) > k then
10 |  |  stepsBack ←min(c, greedySteps);
11 |  |  G' ← eliminate(G, s[1], ..., s[i − stepsBack]);
12 |  |  s' ← IC-TwinWidth(G', k, c + 1);
13 |  |  if s' is empty then
14 |  |  |  return no;
15 |  |  else
16 |  |  |  s ← (s[1], ..., s[i − stepsBack]) + s';
17 |  |  |  G' ← eliminate(G, s[1], ..., s[i]);
18 |  |  end
19 |  |  greedySteps ← 0;
20 |  end
21 end
22 return s;
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Algorithm 4.2: Turbocharged Twin-Width Heuristic using MinRedDegree

Data: Graph $G = (V, E)$, integer c.
Result: An elimination sequence encoded as a list of pairs (v_i, p_i) of eliminated vertices and their parents.

```
s ← MinRedDegree(G);
k ← width(s) − 1;
while k > 0 do
    s′ ← TurbochargedMinRedDegree(G, k, c);
    if s′ == "no" then
        return s;
    end
    s ← s′;
    k ← width(s) − 1;
end
return s;
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 5

Implementation

This chapter discusses the implementation of our turbocharged algorithm described in Chapter 4. Section 5.1 lists technologies used. Then, Section 5.2 discusses how models returned from a SAT solver are decoded. Section 5.3 covers implementation details of the greedy heuristics. Finally, Section 5.4 discusses premature termination of our turbocharged algorithm.

The source code is available on GitHub.[1]

5.1 Technologies

We implement our algorithm in Python 3.6.9 using PySat 1.8[2]. To manage graphs we use NetworkX[3]. For the SAT solver, we use Gluecard 4, as it performed well on our encoding in preliminary experiments.

5.2 Decoding SAT Models

If the SAT solver can find a model for an encoding, we need to extract the solution from it, i.e., we need to determine the contractions that the model represents. This process differs slightly between the encodings.

5.2.1 Relative Encoding

To extract the order of vertices to eliminate from a model in the relative encoding, we start with the sequence $1, ..., n$. We then order them according to the assignments of

[1] https://github.com/DamianJaeger/turbocharged-twin-width
[2] https://pysathq.github.io/
[3] https://networkx.org/

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

the variables $o_{i,j}$: if $o_{i,j}$ is true, i has to be before j, otherwise j has to be before i. So far, this is identical to the way models are decoded in the approach by Schidler and Szeider [24]. The only difference is that for any selected vertex v_i, and vertex v_j that is not selected, $o_{i,j}$ is true, leading vertices that have not been selected to be at the end of the order. Therefore, the first $c+1$ vertices in this order - excluding the imaginary vertex v_0 which is always first - are the ones that are eliminated. To find the parents of the eliminated vertices, the assignment of $p_{i,j}$ is used: for each $1 \leq i \leq c+1$ exactly one $p_{i,j}$ has to be true, meaning that v_j is the parent of the ith vertex in the sequence.

5.2.2 Absolute Encoding

In the absolute encoding we can extract the sequence of eliminated vertices directly from the variable assignment of $o_{i,j}$: The vertex v_j, for which $o_{1,j}$ is assigned to true is the first one, the vertex v_k for which $o_{2,k}$ is assigned to true is eliminated second, and so on. The parents can be determined in the same way using the assignments of $p_{i,j}$: The vertex v_i, for which $p_{1,i}$ is assigned to true is the parent of the vertex eliminated first and so forth. As for each i exactly one $o_{i,j}$ and $p_{i,j}$ has to be assigned to true, we get a sequence of vertices with their parent vertices.

5.3 Heuristics

To simplify maintaining additional data on the graph a heuristic is working on between steps, the heuristics are not called for each step. Instead, they receive k as a parameter and compute as many steps as they can until reaching a moment of regret. They then return all the computed eliminations at once. Our implementation of MinIntersectionSize is fairly straightforward and does not warrant closer discussion.

5.3.1 MinRedDegree

As discussed in Section 4.3.2, MinRedDegree selects the next vertex to be eliminated and its parent based on the red degree and uses the intersection size as a tie-breaker. A naive implementation would consider all pairs of vertices, calculate the intersection size, carry out the elimination, and calculate the red degree of the resulting graph.

However, to lower the computational effort, we do not calculate the red degree after each elimination from scratch. Instead we use the fact that edges in the definition of H_i (which are red edges in the trigraph G_i) can only be removed relative to H_{i-1} if one of its incident vertices is eliminated. Further, one of the vertices incident to any new red edge in H_i is always either adjacent to v_i or its parent p_i in H_{i-1} or G. Therefore, the degrees of all vertices in H_i that are not adjacent to v_i or p_i in H_{i-1} or G are equal to their degrees in H_{i-1}. For vertices v_j (excluding v_i), we therefore make the following case distinction:

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

- $v_j \notin N_{H_{i-1}}(v_i) \wedge v_j \in N_G(v_i) \wedge v_j \notin N_G(p_i)$: v_j is adjacent to v_i in G but not to p_i and is not connected to v_i with a red edge. The red degree of v_j is increased by one relative to H_{i-1}, as a new red edge is created.
- $v_j \notin N_{H_{i-1}}(p_i) \wedge v_j \in N_G(p_i) \wedge v_j \notin N_G(v_i)$: As above, but with v_i and p_i switching places. The red degree of v_j is again increased by one relative to H_{i-1}.
- $v_j \in N_{H_{i-1}}(v_i) \wedge v_j \in N_{H_{i-1}}(p_i)$: v_j is adjacent to p_i and v_i in H_{i-1}. Since v_i is eliminated, the red degree of v_j is decreased by one.
- Otherwise, the red degree of v_j remains unchanged.

Now, the red degrees of all vertices can be computed once and only the changes in red degrees of v_i and all neighbors of v_i and p_i in G and H_{i-1} have to be computed and added to their previous values.

5.4 Anytime Algorithm

Since our turbocharged algorithm continuously tries to improve the solution and since it is not possible to accurately estimate its runtime for an instance in general, it makes sense to implement it as an anytime algorithm. We therefore implement the possibility to cancel the execution at any time. Before the first invocation of the base heuristic without turbocharging has completed, the algorithm uses the upper bound $|V(G)| - 1$. Once a heuristic solution is found, it continues attempting to find better solutions by lowering the value of k and terminates once it fails to find a solution satisfying k. However, if it is interrupted after computing the initial upper bound, it returns the best solution found up until the moment of the interruption. This way, the algorithm can, e.g., be used to compute upper bounds in the context of an exact algorithm for twin-width within a fixed time budget.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 6

Experiments

This chapter covers the experimental evaluation of our turbocharged algorithm. Section 6.1 discusses the algorithms we use and evaluate against. Section 6.2 and Section 6.3 describe the parameters and setup we choose as well as how our experiment is structured. The dataset we use is described in Section 6.4. Finally, we cover the results of the experiment in Section 6.5 and discuss them in Section 6.6.

6.1 Algorithms

We tested our algorithm with both MinRedDegree and MinIntersectionSize as the base heuristic. In our preliminary testing, our absolute encoding outperformed our relative encoding significantly in almost every scenario. Therefore, we excluded the relative encoding from our experiments and used the absolute encoding exclusively.

To provide a better comparison to existing approaches, we implemented the randomized anytime heuristic Heuristic2 from Touiouidth in Python as well. This prevents the original implementation from having an advantage due to not being implemented in Python, which tends to be slower at runtime.

6.2 Parameters

In our preliminary testing, the value of the parameter c had a high impact on run time, while the impact on the quality of the solution was not proportional. We use the values $c = 2$, $c = 5$ and $c = 10$ in our experiments. $c = 2$ is the minimum value to be able improve on a step made by MinRedDegree. For $c = 10$ we saw a high impact on runtime, and $c = 5$ intends to strike a balance between being able to resolve moments of regret and needing an excessive amount of computational time.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

6.3 Setup

For each turbocharged run we assign a time budged t_{budget}. For each instance and greedy heuristic, we start by merging twins and running the greedy heuristic without turbocharging. This takes some time t_{greedy}. Then, the turbocharged algorithm is executed for each value of c which takes some time $t_{turbocharged}$ for each c. Each of these invocations has a time budget of $t_{budget} - t_{greedy}$. Finally, Heuristic2 is run with a time budget equal to the maximal $t_{turbocharged}$ for all values of c for this instance.

For our experiments we used a server with two AMD EPYC 7402 CPUs with 24 cores clocked at 2.80GHz. The operating system was Ubuntu 18.04. We used a memory limit of 32GB. For t_{budget} we chose a value of 8 hours.

6.4 Dataset

To evaluate our algorithm, we use the dataset from the exact track of PACE 2023[1]. It contains a total of 200 instances of various sizes with different properties. The smallest graph has 19 vertices with graphs ranging up to 20000 vertices. Similarly, the number of edges ranges from 46 all the way to 35052. The average graph has 1550 vertices and 5596 edges. This dataset also contains different graph classes: about 12% of the instances are bipartite, 9% planar and 1% chordal. Of these instances, 50 could be solved by the approach by Schidler and Szeider [25] within 30 minutes, 50 within 8 hours and the remaining 100 could not be solved within this time frame [2, 6].

6.5 Results

For the analysis of our results we at times divide the instances into categories: small instances (instances 1 to 50 of the dataset), medium instances (instances 51 to 70 of the dataset), large instances (instances 71 to 100 of the dataset), and very large instances (instances 101 to 200 of the dataset).

6.5.1 MinRedDegree vs MinIntersectionSize

When comparing the quality of results of all instances for which both base heuristics completed with in the time budget, MinRedDegree was able to find significantly lower bounds for twin-width than MinIntersectionSize on average. The upper bounds computed by both heuristics are shown in Figure 6.1. The difference is particularly pronounced for large instances. On very large instances, however, MinIntersectionSize was able to outperform MinRedDegree on solution quality for many instances. In terms of speed, MinIntersectionSize was able to complete on average almost 6 times faster, as can be seen in Table 6.3. In total, MinIntersectionSize completed within the time budget on 156 instances, while MinRedDegree only completed on 114 instances.

[1] https://pacechallenge.org/2023/

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

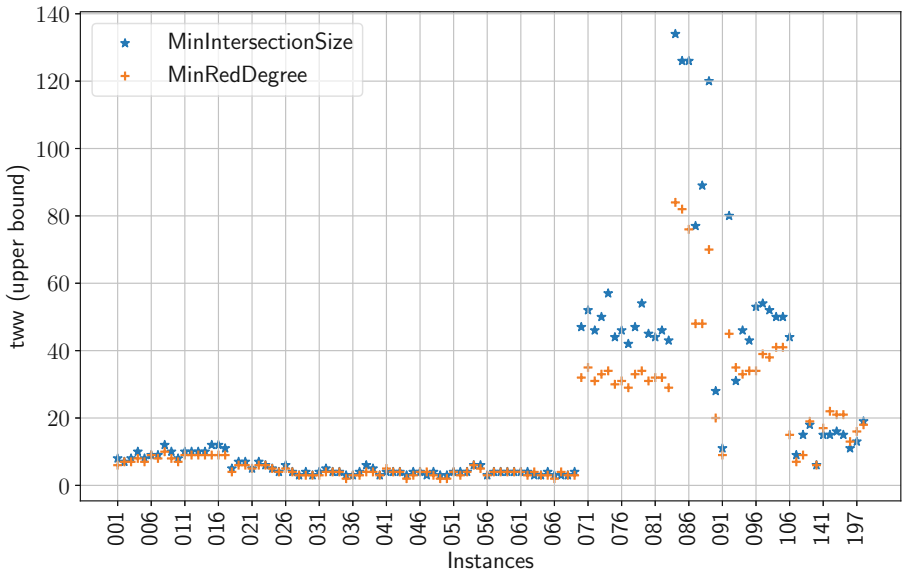

Figure 6.1: Upper bounds for tww obtained by MinRedDegree and MinIntersectionSize on the 112 instances for which both completed within the time limit

6.5.2 MinRedDegree

Relative to the results obtained by MinRedDegree, none of the improvement algorithms we tested was able to improve the solution by more than 1 for any instance. As shown in Table 6.1, for small and medium instances Heuristic2 significantly outperformed the turbocharged heuristics, but it failed to find any improved solution for large and very large instances. For the turbocharged heuristic, larger values for k lead to better solutions (with the exception of $k = 10$ for large instances).

Algorithm	small	medium	large	very large	total
$k = 2$	4	0	2	1	7
$k = 5$	7	1	**5**	**3**	16
$k = 10$	15	2	1	**3**	21
Heuristic2	**21**	**6**	0	0	**27**

Table 6.1: Comparison of the number of improved instances over MinRedDegree obtained by all tested algorithms for the 114 instances with results for MinRedDegree grouped by instance category

6.5.3 MinIntersectionSize

Given the worse initial solutions obtained by MinIntersectionSize, the improvement algorithms were able to improve the solution considerably. Table 6.2 gives an overview

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

of the number of instances improved and the sum of by how much the initial solution was improved for all algorithms. Here, Heuristic2 was only able to outperform the turbocharged heuristics regarding solution quality on small instances. Particularly on the larger instances, TurbochargedMinIntersectionSize with $k = 2$ outperformed the remaining algorithms. Unlike for TurbochargedMinRedDegree, higher values for k had no or next to no benefit on any category of instances.

Algorithm	small	medium	large	very large	total
$k = 2$	35 (47)	14 **(30)**	**29 (589)**	**38 (158)**	**116 (824)**
$k = 5$	35 (46)	**16** (29)	28 (444)	30 (117)	109 (636)
$k = 10$	33 (46)	15 (19)	27 (208)	22 (60)	97 (333)
Heuristic2	**41 (64)**	11 (24)	23 (257)	0 (0)	75 (345)

Table 6.2: Comparison of the improvement over MinIntersectionSize obtained by all tested algorithms for the 156 instances with results for MinIntersectionSize grouped by instance category. Values are the numbers of instances improved and in parenthesis are the sums of improvement of the upper bound through turbocharging.

6.5.4 Improvement Speed

When using MinIntersectionSize, on average, a lower value for k lead to an increased rate of improvement over time. Figure 6.2 shows an example of the rate of improvement for all algorithms on a large instance. Note that MinIntersectionSize completes significantly before MinRedDegree. Therefore, TurbochargedMinIntersectionSize is able to make multiple improvement steps before MinRedDegree even terminates. In this example, TurbochargedMinIntersectionSize managed to find a better solution than TurbochargedMinRedDegree just after MinRedDegree terminated and none of the improvement algorithms were able to improve upon the solution by MinRedDegree. The rate of improvement of TurbochargedMinIntersectionSize tends to slow down continuously as the bound of twin-width decreases. Heuristic2, on the other hand, has a close to linear rate of improvement that only flattens out towards the end. This trend is observable on many instances.

Next, we look at the amount of time it took the improvement algorithms to find the best solution they obtained within the time budget. For this, we only consider the 112 instances that both base heuristics completed on within the time budget for a fair comparison. Note that a high amount of time taken does not necessarily reflect negatively on an algorithm, as it means that it is able to utilize the available time. Given that two algorithms have achieved the same solution quality, however, less time taken is preferable. Therefore, we also show the quality of the result with each time.

Table 6.3 gives an overview over the average upper bound for twin-width and the average amount of time taken to obtain it for all algorithms grouped into small to medium and large to very large instances. For small to medium instances, Heuristic2 stands out by utilizing the available time and finding the best solutions. For this, the base heuristic

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

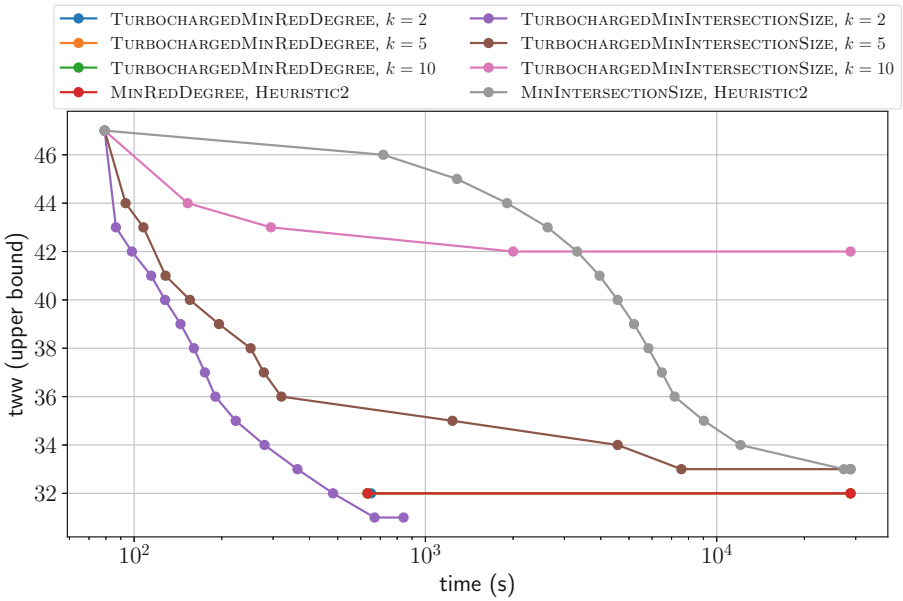

Figure 6.2: Upper bounds for twin-width obtained by TurbochargedMinRedDegree, TurbochargedMinIntersectionSize, and MinRedDegree with Heuristic2 over time on a large instance. Repeated values for tww indicate that the heuristic failed to find a better solution or a timeout has been reached.

does not have a significant impact, as they achieve similar outcomes, as discussed in Section 6.5.1. For TurbochargedMinRedDegree, better solutions naturally lead to more time spent. On the other hand, TurbochargedMinIntersectionSize with $k = 2$ is noteworthy as it achieves the lowest bounds for twin-width among all turbocharged algorithms while also requiring the least amount of time on average.

	Base heuristic							
	MinRedDegree				MinIntersectionSize			
	small - medium		large - very large		small - medium		large - very large	
Algorithm	tww	time (s)	tww	time (s)	tww	time (s)	tww	time (s)
Base heuristic	5.27	11.06	32.74	8642	6.11	1.62	47.19	1481
$k = 2$	5.21	11.07	32.68	8645	5.01	10.64	**32.45**	7114
$k = 5$	5.16	11.21	32.55	8890	5.04	109.77	36	13438
$k = 10$	5.03	127.98	32.64	8808	5.19	73.61	41.83	10296
Heuristic2	4.89	392.92	32.74	8642	**4.86**	415.65	41.07	11930

Table 6.3: Average upper bound for twin-width and average amount of time taken to obtain it over all 112 instances on which both base heuristics completed successfully for all tested algorithms grouped by instance categories.

For the larger instances, TurbochargedMinIntersectionSize with $k = 2$ also stands

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

out. It finds the best solutions among all tested algorithms while taking the least amount of time (excluding MinIntersectionSize of course) on average. For higher values of k, the time required increases quickly, especially when taking the relative loss of solution quality into account. The average time values of improvement algorithms using MinRedDegree as the base heuristics are not very meaningful, as they are dragged down by the fact that hardly any improvements have been made.

6.5.5 Comparison of Best Performing Heuristics

Finally, we compare the two overall best performing improvement algorithms evaluated: TurbochargedMinIntersectionSize with $k = 2$ and MinRedDegree with Heuristic2. Figure 6.3 gives an overview of the twin-width bounds obtained by both algorithms for instances where both base heuristics completed. On small instances, MinRedDegree with Heuristic2 manages to find better solutions on average, reducing the bound by one on multiple instances compared to TurbochargedMinIntersectionSize with $k = 2$. On medium instances, TurbochargedMinIntersectionSize with $k = 2$ manages to find slightly better solutions than MinRedDegree with Heuristic2, the difference does, however, not exceed 2 for any instance. On large instances, both algorithms perform very similarly on average with a difference of about 0.5%. The differences for single instances, however are much greater, with the highest difference in bounds being 9. On very large instances, TurbochargedMinIntersectionSize with $k = 2$ performs significantly better. The average bound for twin-width was about 10% lower than those obtained by MinRedDegree with Heuristic2. Additionally, the base heuristic MinIntersectionSize completed on 56 very large instances, while MinRedDegree only completed on 14 instances.

6.6 Discussion

Overall, Heuristic2 turned out to be very effective on small and medium instances in our experiments. On larger instances, however, it fell behind our turbocharged approach. This is likely due to the larger search space of possible eliminations sequences. Our turbocharging algorithms still managed to make improvements on these instances and therefore largely managed to outperform Heuristic2. Using MinIntersectionSize as the base heuristic has the advantage of finding an initial solution much sooner while still delivering good results after more time.

6.6.1 Selection of k

For TurbochargedMinRedDegree, increasing the value for k tended to yield better solutions in our experiments. This makes intuitive sense, as it allows resolving problems that occurred further back. However, this was not the case for TurbochargedMinIntersectionSize. Here, increasing k had next to no effect or a negative effect. The negative effect can be largely attributed to larger values of k leading a lower rate of improvement, resulting in more timeouts and termination at a higher upper bound once

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

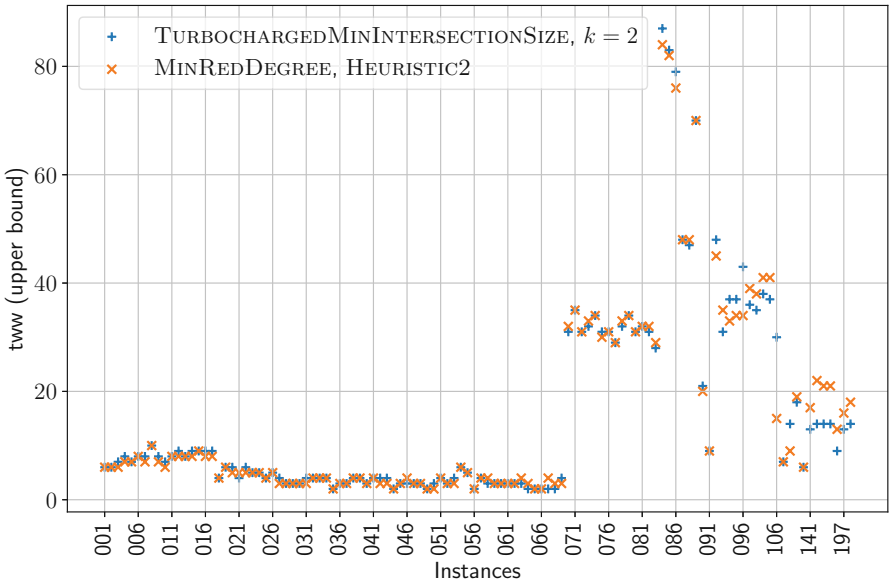

Figure 6.3: Upper bounds for twin-width obtained by MinRedDegree with Heuristic2 and TurbochargedMinIntersectionSize with $k = 2$ on the 112 instances for which both base heuristics completed within the time limit.

the time budget is exhausted. For some instances, however, TurbochargedMinIntersectionSize with higher values of k simply failed to find better solutions than with lower values of k. It is not clear why higher values for k did not have a positive impact on solution quality on small and medium instances. Based on this, it appears to be beneficial to select higher values for k when using TurbochargedMinRedDegree and to stick to low values for k when using TurbochargedMinIntersectionSize.

6.6.2 Efficacy of Turbocharging

When using MinIntersectionSize as the base heuristic, turbocharging yielded improvements on a majority of instances. These were particularly significant on the larger instances. The same was not the case for MinRedDegree. Especially for large instances, this could be explained by the fact that MinRedDegree already found lower bounds than MinIntersectionSize. However, this is not the case for very large instances, where turbocharging lead to significantly more improvement over MinIntersectionSize compared to MinRedDegree. A possible reason for this could be that the turbocharging algorithm is concerned with the red degree, much like MinRedDegree, and just looks at multiple steps at once instead of only one. MinIntersectionSize, on the other hand, optimizes toward the intersection size, which is fundamentally different. It is possible that the resulting alternation of priorities at the moments of regret improves results.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 7

Conclusions

In this thesis, we developed the two turbocharged heuristics TurbochargedMinRedDegree and TurbochargedMinIntersectionSize which compute upper bounds for twin-width on graphs. They use the base heuristics MinRedDegree and MinIntersectionSize, respectively. As the turbocharging algorithm we developed two SAT encodings, one of which significantly outperformed the other. To compare our algorithms to the state of the art, we benchmarked them with different parameters against the upper bound heuristic Heuristic2 from Touiouidth, one of the best performing exact submissions to PACE 2023.

In our experiments, the turbocharged heuristics performed well, particularly TurbochargedMinIntersectionSize with the parameter k set to 2. While it was narrowly outperformed by MinRedDegree with Heuristic2 on small instances, they performed similarly on medium and large instances. On very large instances, our approach significantly outperformed Heuristic2.

We believe that turbocharging twin-width heuristics is a viable strategy for obtaining upper bounds for twin widths, especially on larger graphs. On smaller graphs, however, randomized approaches such as Heuristic2 are still preferable. TurbochargedMinIntersectionSize in particular could be used in exact algorithms for twin-width in large graphs to obtain tighter upper bounds. It is also suited for obtaining better upper bounds for larger graphs in general.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Figures

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Tables

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Algorithms

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Acronyms

BCP Boolean Constraint Propagation. 9

CDCL Conflict-Driven Clause Learning. 11

CNF conjunctive normal form. 9

DPLL Davis-Putnam-Logemann-Loveland. 9

IC incremental conservative. 26

PACE 2023 8th Parameterized Algorithms and Computational Experiments Challenge. 1, 13, 25, 46, 53

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Bibliography

[1] https://github.com/ASchidler/twin_width/blob/main/encoding2.py. Accessed: 2024-07-06.

[2] https://pacechallenge.org/2023/properties/. Accessed: 2024-09-01.

[3] F. N. Abu-Khzam, S. Cai, J. Egan, P. Shaw, and K. Wang. Turbo-charging dominating set with an fpt subroutine: Further improvements and experimental analysis. In *Theory and Applications of Models of Computation: 14th Annual Conference, TAMC 2017, Bern, Switzerland, April 20-22, 2017, Proceedings 14*, pages 59–70. Springer, 2017.

[4] S. Alouneh, S. Abed, M. H. Al Shayeji, and R. Mesleh. A comprehensive study and analysis on sat-solvers: advances, usages and achievements. *Artificial Intelligence Review*, 52:2575–2601, 2019.

[5] E. Arrighi, P. G. Drange, K. Langedal, F. Vadiee, M. Vatshelle, and P. Wolf. Pace solver description: Zygosity. In *18th International Symposium on Parameterized and Exact Computation (IPEC 2023)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2023.

[6] M. Bannach and S. Berndt. Pace solver description: The pace 2023 parameterized algorithms and computational experiments challenge: Twinwidth. In *18th International Symposium on Parameterized and Exact Computation (IPEC 2023)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2023.

[7] P. Bergé, E. Bonnet, and H. Déprés. Deciding Twin-Width at Most 4 Is NP-Complete. In M. Bojańczyk, E. Merelli, and D. P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 229 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:20, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[8] G. Berthe, Y. Coudert-Osmont, A. Dobler, L. Morelle, A. Reinald, and M. Rocton. Pace solver description: Touiouidth. In *IPEC 2023-18th International Symposium on Parameterized and Exact Computation*, volume 285, page 4, 2023.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[9] É. Bonnet and J. Duron. Pace solver description: Redalert-heuristic track. In *18th International Symposium on Parameterized and Exact Computation (IPEC 2023)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2023.

[10] É. Bonnet, C. Geniet, E. J. Kim, S. Thomassé, and R. Watrigant. Twin-width ii: small classes. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1977–1996. SIAM, 2021.

[11] E. Bonnet, C. Geniet, E. J. Kim, S. Thomassé, and R. Watrigant. Twin-width III: Max Independent Set, Min Dominating Set, and Coloring. In N. Bansal, E. Merelli, and J. Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 35:1–35:20, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[12] É. Bonnet, C. Geniet, R. Tessera, and S. Thomassé. Twin-width vii: groups. *arXiv preprint arXiv:2204.12330*, 2022.

[13] É. Bonnet, U. Giocanti, P. O. de Mendez, P. Simon, S. Thomassé, and S. Toruńczyk. Twin-width iv: ordered graphs and matrices. *Journal of the ACM*, 71(3):1–45, 2024.

[14] É. Bonnet, E. J. Kim, A. Reinald, and S. Thomassé. Twin-width vi: the lens of contraction sequences. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1036–1056. SIAM, 2022.

[15] É. Bonnet, E. J. Kim, S. Thomassé, and R. Watrigant. Twin-width i: tractable fo model checking. *ACM Journal of the ACM (JACM)*, 69(1):1–46, 2021.

[16] A. Dobler, M. Sorge, and A. Villedieu. Turbocharging heuristics for weak coloring numbers. *arXiv preprint arXiv:2203.03358*, 2022.

[17] R. G. Downey, J. Egan, M. R. Fellows, F. A. Rosamond, and P. Shaw. Dynamic dominating set and turbo-charging greedy heuristics. *Tsinghua Science and Technology*, 19(4):329–337, 2014.

[18] S. Gaspers, J. Gudmundsson, M. Jones, J. Mestre, and S. Rümmele. Turbocharging treewidth heuristics. *Algorithmica*, 81:439–475, 2019.

[19] P. Hliněnỳ and J. Jedelskỳ. Twin-width of planar graphs is at most 8, and at most 6 when bipartite planar. *arXiv preprint arXiv:2210.08620*, 2022.

[20] H. Jacob and M. Pilipczuk. Bounding twin-width for bounded-treewidth graphs, planar graphs, and bipartite graphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 287–299. Springer, 2022.

[21] A. Leonhardt, H. Dell, A. Haak, F. Kammer, J. Meintrup, U. Meyer, and M. Penschuck. Pace solver description: Exact (guthmi) and heuristic (guthm). In *18th International Symposium on Parameterized and Exact Computation (IPEC 2023)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2023.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[22] Y. Mizutani, D. Dursteler, and B. D. Sullivan. Pace solver description: Hydra prime. In *18th International Symposium on Parameterized and Exact Computation (IPEC 2023)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2023.

[23] M. R. Prasad, A. Biere, and A. Gupta. A survey of recent advances in sat-based formal verification. *International Journal on Software Tools for Technology Transfer*, 7:156–173, 2005.

[24] A. Schidler and S. Szeider. A sat approach to twin-width. In *2022 Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX)*, pages 67–77. SIAM, 2022.

[25] A. Schidler and S. Szeider. Computing twin-width with sat and branch & bound. In *IJCAI*, pages 2013–2021, 2023.

[26] L. Zhang, C. F. Madigan, M. H. Moskewicz, and S. Malik. Efficient conflict driven learning in a boolean satisfiability solver. In *IEEE/ACM International Conference on Computer Aided Design. ICCAD 2001. IEEE/ACM Digest of Technical Papers (Cat. No. 01CH37281)*, pages 279–285. IEEE, 2001.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.