Turbocharging Treewidth Heuristics*

Serge Gaspers†1, Joachim Gudmundsson[2], Mitchell Jones[3], Julián Mestre[4], and Stefan Rümmele[5]

1 UNSW, Sydney, Australia; and
Data61, CSIRO, Sydney, Australia
sergeg@cse.unsw.edu.au
2 University of Sydney, Sydney, Australia
joachim.gudmundsson@sydney.edu.au
3 University of Sydney, Sydney, Australia
mjon1572@uni.sydney.edu.au
4 University of Sydney, Sydney, Australia
julian.mestre@sydney.edu.au
5 University of Sydney, Sydney, Australia; and
UNSW, Sydney, Australia
stefan.rummele@sydney.edu.au

Abstract

A widely used class of algorithms for computing tree decompositions of graphs are heuristics that compute an elimination order, i.e., a permutation of the vertex set. In this paper, we propose to *turbocharge* these heuristics. For a *target treewidth* k, suppose the heuristic has already computed a partial elimination order of width at most k, but extending it by one more vertex exceeds the target width k. At this *moment of regret*, we solve a subproblem which is to recompute the last c positions of the partial elimination order such that it can be extended without exceeding width k. We show that this subproblem is fixed-parameter tractable when parameterized by k and c, but it is para-NP-hard and $W[1]$-hard when parameterized by only k or c, respectively. Our experimental evaluation of the FPT algorithm shows that we can trade a reasonable increase of the running time for quality of the solution.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases tree decomposition, heuristic, fixed-parameter tractability, local search

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.13

1 Introduction

The most widely used treewidth heuristics are simple algorithms that compute an elimination order, i.e., a permutation of the vertex set. For a given elimination order π of a graph G, we obtain a chordal completion G' by eliminating vertices according to this order: when we eliminate a vertex, we add edges to make its neighborhood into a clique, and then remove the vertex. Given this chordal completion G', we can obtain a tree decomposition by traversing π backwards: for each vertex v in π, let $L(v)$ denote the neighbors of v in G' that occur later than v in π; choose a bag that contains $L(v)$ and add a new neighboring bag that contains $\{v\} \cup L(v)$. Thus, for a given elimination order, the width of the tree decomposition we

* The authors acknowledge support under the ARC's Discovery Projects funding scheme (DP150101134).
† Serge Gaspers is the recipient of an Australian Research Council (ARC) Future Fellowship (FT140100048).

obtain is the largest degree of an eliminated vertex (i.e., the degree of the vertex when it is eliminated).[1]

The GreedyDegree heuristic selects a vertex of minimum degree as the next vertex of the elimination order, while the GreedyFillIn heuristic selects a vertex that has fewest non-edges in its neighborhood. These heuristics compare favorably to more involved heuristics; see [3], where a small additional improvement is achieved by turning the resulting chordal completion into a minimal chordal completion in a post-processing step.

In this paper, we propose to *turbocharge* the GreedyDegree and the GreedyFillIn heuristics. For a *target treewidth* k and a *change parameter* c, suppose the heuristic has already computed a partial elimination order π of width at most k and length l, but extending π by one more vertex exceeds the target width k. At this moment of regret, we solve a subproblem which is to compute a partial elimination order π' of width at most k and length $l+1$ which coincides with π in the first $l-c$ positions. Having solved this subproblem, we continue running the heuristic with the partial elimination order π'. In this paper, we formally study this subproblem, parameterized by combinations of l, k, and c, and experimentally evaluate what effect the turbocharging has on the width of the obtained tree decompositions.

The subproblem turns out to be para-NP-hard under Turing reductions for parameter k, $W[1]$-hard for parameters c or l, but fixed-parameter tractable for the combination of parameters k and c. Our implementation is based on this FPT algorithm, and the experiments show an improvement of the width of the obtained tree decompositions for reasonable values of the parameter c, which allows us to trade running time for quality of the solution.

Our turbocharging strategy solves a local-search subproblem when the heuristic gets stuck because all remaining vertices have degree at least $k+1$, similar to the turbocharging strategy for list coloring by Hartung and Niedermeier [7]. Another way to turbocharge the GreedyFillIn heuristic would be to select several vertices at a time and minimize the number of edges added when eliminating this set of vertices. However, we quickly run into limitations of this method, since this problem is $W[1]$-hard when parameterized by the number of vertices we would like to eliminate, and we did not pursue this strategy further.

2 Preliminaries

A graph $\mathcal{G}=(V,E)$ consists of a vertex set V and an edge set E. The neighborhood $N(v)$ of vertex v in G is defined as the set of all vertices adjacent to v, $N(v)=\{w \mid \{v,w\}\in E\}$.

A *tree decomposition* of a graph $\mathcal{G}=(V,E)$ is a pair $\mathcal{T}=(T,\chi)$, where T is a tree and χ maps each node t of T (we use $t\in T$ as a shorthand below) to a *bag* $\chi(t)\subseteq V$, such that

1. for each $v\in V$, there is a $t\in T$, s.t. $v\in\chi(t)$;
2. for each $\{v,w\}\in E$, there is a $t\in T$, s.t. $\{v,w\}\subseteq\chi(t)$;
3. for each $r,s,t\in T$, s.t. s lies on the path from r to t, $\chi(r)\cap\chi(t)\subseteq\chi(s)$.

The *width* of a tree decomposition is defined as the cardinality of its largest bag minus one. The *treewidth* of a graph $\mathcal{G}$, denoted by $tw(\mathcal{G})$, is the minimum width over all tree decompositions of $\mathcal{G}$. For a given graph $\mathcal{G}$ and integer k, deciding whether $\mathcal{G}$ has treewidth at most k is NP-complete [1]. For fixed k, one can decide in linear time whether a graph has treewidth $\leq k$ and, if so, compute a tree decomposition of width k [2].

[1] We refer the reader who is not familiar with some of these notions to the Preliminaries section.

TREEWIDTH	
Instance:	Graph G and integer k.
Problem:	Decide whether $tw(\mathcal{G}) \leq k$ holds.

One of the alternative characterizations of treewidth is based on so called *elimination orders*. Let $\mathcal{G} = (V, E)$ be a graph and $v \in V$ a vertex of $\mathcal{G}$. Eliminating v from $\mathcal{G}$ refers to the process of forming a clique out of the neighbourhood of v and removing v and its incident edges, that is, we create a new graph $\mathcal{G}' = (V \setminus \{v\}, (E \cup E_1) \setminus E_2)$, where $E_1 = \{\{u, w\} \mid u, w \in N(v)\}$ and $E_2 = \{e \in E \mid v \in e\}$. An elimination order of a graph $\mathcal{G} = (V, E)$ with $|V| = n$ is a bijective function $\pi : V \to \{1, \ldots, n\}$. Starting with $\mathcal{G}$ and iteratively eliminating the vertices in V according to the order π results in a sequence of $n + 1$ graphs with the last one being the empty graph. The width of π is the maximum degree of any vertex $v \in V$ during its elimination according to the order π. This notion leads to the following alternative characterization of treewidth.

▶ **Theorem 1** (see for example [3]). *Let $\mathcal{G} = (V, E)$ be a graph and let $k \in \mathbb{N}$. $\mathcal{G}$ has treewidth at most k if and only if there exists an elimination order π of width at most k.*

We generalize the notion of elimination orders to partial elimination orders as follows. A partial elimination order of length $l \leq n$ of a graph $\mathcal{G} = (V, E)$ with $|V| = n$ is a bijective partial function $\pi : V \to \{1, \ldots, l\}$, that is, an enumeration of l vertices of the graph. The intended meaning of a partial elimination order is that it represents the first l positions of an elimination order. Analogously to elimination orders, the width of partial elimination order π is the maximum degree of any vertex $v \in V$ during its elimination according the order π.

The two heuristics which we mentioned earlier as well as our algorithm compute the treewidth based on elimination orders. The GREEDYDEGREE heuristic as well as the GREEDYFILLIN heuristic construct an elimination order by iteratively selecting the next vertex in the elimination order and eliminating it from the graph. During the elimination step, the vertex is removed and all its neighbours are connected to a clique. The selection of the next vertex is based on a greedy criteria. GREEDYDEGREE selects the vertex with the minimal degree and GREEDYFILLIN selects the vertex whose elimination results in the fewest new edges that need to be added to the graph in order to form a clique out of its neighbours. In both cases ties are broken arbitrarily.

3 Local Search Variants of the Treewidth Problem

We are interested in how the existing greedy heuristics for TREEWIDTH can be improved via local search. We introduce the following problem, which we call the *incremental conservative (IC) treewidth problem*, since it follows the spirit of incremental conservative k-list coloring of graphs by Hartung and Niedermeier [7].

IC-TREEWIDTH	
Instance:	Graph $\mathcal{G}$, integer k and c, and partial elimination order π of length l and width $\leq k$.
Problem:	Does there exist a partial elimination order π' of length $l+1$ and width $\leq k$ such that π and π' are identical on the first $l - c$ positions?

Since we want to use an algorithm for IC-TREEWIDTH as a subroutine for improving an existing partial elimination order, we are interested in finding parameters for which the problem becomes fixed-parameter tractable. The following result shows, that the length l and hence also the change c are ineligible.

▶ **Theorem 2.** IC-TREEWIDTH *is* W[1]*-hard when parameterized by* l*.*

Proof. By reduction from INDEPENDENT SET. An instance of INDEPENDENT SET is given by a graph $\mathcal{G} = (V, E)$ and integer k. The question is whether $\mathcal{G}$ has an independent set of size k. INDEPENDENT SET is W[1]-complete when parameterized by k [4].

Let $(\mathcal{G} = (V, E), k)$ be an instance of INDEPENDENT SET with $V = \{v_1, \ldots, v_n\}$. We denote the maximum degree of $\mathcal{G}$ by $\Delta(\mathcal{G}) = d$. Now construct an instance $(\mathcal{G}', \pi, c)$ of IC-TREEWIDTH as follows. Let $U, W, X_1, \ldots, X_{k-1}, Y_1, \ldots, Y_n$ be sets of new vertices of cardinalities $|U| = n$, $|W| = n + 2d$, $|X_i| = 2d + 1$ for $1 \le i \le k - 1$, and $|Y_j| = d + 1$ for $1 \le j \le n$. Let K_X^i with $1 \le i \le k - 1$ denote the complete graph of the vertices on X_i. Analogously, we denote by K_Y^j with $1 \le j \le n$ and by K_W the complete graphs over Y_j and W, respectively. The new graph $\mathcal{G}' = (V \cup U \cup W \cup X_1 \cdots \cup X_{k-1} \cup Y_1 \cdots \cup Y_n, E')$ contains all edges of $\mathcal{G}$, and all edges of the complete graphs K_X^i, K_Y^j, and K_W. Additionally, we add the following edges. Let $U = \{u_1, \ldots, u_n\}$.

$$
\begin{aligned}
&\{u_i, v_j\} && 1 \le i, j \le n \\
&\{u_i, x\} && 1 \le i \le k-1 \text{ and } x \in X_i \\
&\{u_i, w\} && k \le i \le n \text{ and } w \in W \\
&\{v_j, y\} && 1 \le j \le n \text{ and } y \in Y_j \\
&\{x, w\} && x \in X_i, 1 \le i \le k-1 \text{ and } w \in W \\
&\{y, w\} && y \in Y_j, 1 \le j \le n \text{ and } w \in W
\end{aligned}
$$

To complete the construction of the instance for IC-TREEWIDTH, we set $\pi = (u_1, \ldots, u_{k-1})$ and $c = k - 1$. The partial elimination order π has width $n + 2d + 1$ since each vertex u_i, $1 \le i \le k - 1$, has as neighbourhood the n vertices of V and $2d + 1$ vertices of X_i at the time of its elimination. Note that this instance can be constructed in polynomial time.

We will show that $\mathcal{G}$ has an independent set of size k if and only if there exists a partial elimination order π' of width $n + 2d + 1$ and length k for graph $\mathcal{G}'$. For the first direction, assume that $\mathcal{G}$ has an independent set S of size k. Let π' be an arbitrary order of the k vertices in S. We show that π' is a partial elimination order of width $n + 2d + 1$ for graph $\mathcal{G}'$. The neighbourhood of each vertex $v_j \in S$ in $\mathcal{G}'$ consists of its original neighbourhood in $\mathcal{G}$ together with all the vertices of U and Y_j. Hence its degree is bounded by $d(v_j) \le n + 2d + 1$. Since S is an independent set, it holds for all pairs $v_i, v_j \in S$, that eliminating v_i from $\mathcal{G}'$ does not change the neighbourhood of v_j. Therefore, π' is a partial elimination order of length k and width $n + 2d + 1$.

For the second direction, assume that π' is a partial elimination order of length k and width $n + 2d + 1$ for $\mathcal{G}'$. We will show that the k vertices of π' form an independent set in $\mathcal{G}$. π' can not contain any vertex from W since they have degree $n(3 + d) + 2dk - 1$. Similar, π' can not contain any vertex from X_i, $1 \le i \le k - 1$, or from Y_j, $1 \le j \le n$, since they have degree $n + 4d + 1$ and $n + 3d + 1$, respectively. We can also exclude vertices $u_k, \ldots, u_n$ since they have degree $2n + 2d$. Starting by eliminating a vertex $u_i \in \{u_1, \ldots, u_{k-1}\}$ creates a clique of all vertices in $V \cup X_i$. This means, if π' starts with u_i, then it can not contain any vertex from V. Note that eliminating a vertex $v_j \in V$ forms a clique of all vertices in $U \cup Y_j$. Hence, if π' contains v_j, it can not be succeeded by any vertex in U. These two observations combined, say that π' either consists only of vertices of U or only of vertices of V. We can exclude the case U, since π' has length k and there are only $k - 1$ suitable vertices in U. Therefore, π' contains only vertices from V. It remains to show that these vertices form an independent set. Assume towards a contradiction, that π' contains two adjacent vertices, say v_i and v_j. W.l.o.g. we assume v_i is eliminated before v_j. Eliminating v_i introduces an edge between v_j and all $d+1$ vertices of Y_i. Hence, v_j has now degree $d(v_j) \geq n+2d+2$. But this means v_j can not be contained in π', which contradicts our assumption and the vertices in π' form indeed an independent set. ◀

This reduction from Independent Set together with a straight-forward NP-membership via a guess and check algorithm, gives us NP-completeness of IC-Treewidth.

▶ **Corollary 3.** IC-Treewidth *is NP-complete.*

A problem that is closely related to IC-Treewidth is the following Length-l-Partial-Elimination-Order problem. To solve IC-Treewidth, we can eliminate the first $l-c$ vertices of the graph and then ask for a length $c+1$ partial elimination order.

Length-l-Partial-Elimination-Order	
Instance:	Graph $\mathcal{G}$, integer l and k.
Problem:	Does there exist a partial elimination order of $\mathcal{G}$ of length l and width $\leq k$?

▶ **Theorem 4.** Length-l-Partial-Elimination-Order *is fixed-parameter tractable when parameterized by l and k.*

Proof. Let $\mathcal{G} = (V, E)$, l and k be an instance of Length-l-Partial-Elimination-Order. Let S be set of vertices of degree at most k, i.e., $S = \{v \in V \mid d_{\mathcal{G}}(v) \leq k\}$. Let $\mathcal{G}[S]$ be the subgraph $\mathcal{G}$ induced by S. Let $\mathcal{A}$ be a greedy algorithm for Independent Set that iteratively selects a minimum degree vertex and remove its closed neighborhood from the graph, until it either finds an independent set I of size l or fails to do so.

In case $\mathcal{A}$ succeeds, we show that sequencing $(v_1, v_2, \ldots, v_l)$ of I is a partial elimination order of $\mathcal{G}$ of width $\leq k$. Note that each v_i belongs to S, so it has degree at most k in $\mathcal{G}$. Since I is an independent set, eliminating v_i does not add a new edge incident on I. Hence, the partial elimination order $(v_1, \ldots, v_l)$ has width at most k.

On the other hand, if $\mathcal{A}$ fails to find an independent set of size l, we know that $|S| \leq (l-1)(k+1)$. To see this, note that adding some vertex $v \in S$ to I can block at most k other vertices (v's neighbors) from being selected by the greedy algorithm. Since the maximum independent set that $\mathcal{A}$ can find in case of failure has size $l-1$, the bound on S follows.

We can exploit this insight to design a branching algorithm for Length-l-Partial-Elimination-Order. Each node of the branching process will have associated a partial elimination order π' and a graph $\mathcal{G}'$. On the first level we only have the root node, where π' is empty and $\mathcal{G}'$ is the input graph. Consider a node $(\pi', \mathcal{G}')$ at level i of the branching process and let $S' = \{v \in V \mid d_{\mathcal{G}'}(v) \leq k\}$. If $|S'| > (l-i)(k+1)$ then we can use $\mathcal{A}$ to extend the partial elimination order by $(l-i+1)$ additional nodes, we can do just that and stop the branching process. Otherwise, if $|S'| \leq (l-i)(k+1)$, we branch on every node $v \in S'$, by adding it to π' order and eliminating it from $\mathcal{G}'$, thus generating a new node $(\pi'', \mathcal{G}'')$ on level $i+1$.

Notice that the number of branches we need to follow from a node in level i is at most $(l-i)(k+1)$. Therefore, the total number of nodes we explore is at most $\prod_{i=1}^{l}(l-i)(k+1) = (l-1)!(k+1)^l$. Hence, we can decide Length-l-Partial-Elimination-Order in $\mathcal{O}^*\left((l-1)!(k+1)^l\right)$ time. ◀

Given a partial elimination order, we can backtrack the last c choices and use this FPT result to extend it again by $c+1$ vertices. This leads to the following corollary.

▶ **Corollary 5.** IC-Treewidth *is fixed-parameter tractable when parameterized by c and k.*

Similar, the W[1]-hardness of IC-TREEWIDTH when parameterized by c carries over to LENGTH-l-PARTIAL-ELIMINATION-ORDER parameterized by l. We show next, that the combination of parameters l and k is indeed necessary, that is, we show hardness for parameter k alone.

▶ **Theorem 6.** LENGTH-l-PARTIAL-ELIMINATION-ORDER *is NP-hard even when* $k = 5$.

Proof. Our reduction is from the NP-hard problem INDEPENDENT SET ON CUBIC GRAPHS, which takes as input a 3-regular graph $\mathcal{G} = (V, E)$ and an integer k, and the question is whether $\mathcal{G}$ has an independent set of size k, i.e., a set of k vertices that are pairwise non-adjacent [5]. We construct an instance $(\mathcal{G}', l = k, 5)$ for LENGTH-l-PARTIAL-ELIMINATION-ORDER as follows. To obtain $\mathcal{G}'$, we start from $\mathcal{G}$ and add a disjoint clique W on 7 vertices. For every vertex v of $\mathcal{G}$, we add two vertices a_v and b_v to $\mathcal{G}'$ and make them adjacent to $W \cup \{v\}$. To see that a partial elimination order π of width at most 5 of $\mathcal{G}'$ corresponds to an independent set in $\mathcal{G}$, and vice-versa, first observe that π contains no vertex from W or $N(W)$; indeed, the first vertex from $W \cup N(W)$ occurring in π has more than 5 neighbors when it is eliminated. Secondly, assume the partial elimination order contains two adjacent vertices. Let v be the first vertex that is eliminated for which at least one neighbor u has already been eliminated. But then, v has degree at least 6 when it is eliminated because eliminating u added the edges $\{v, a_u\}$ and $\{v, b_u\}$. But, on the other hand, eliminating an independent set of size $l = k$ gives a partial elimination order of width 5 and length l. ◀

IC-TREEWIDTH is defined as a decision problem. We call the problem of actually computing such a partial elimination order π', the function version of IC-TREEWIDTH. As mentioned before, if it exists, computing a tree decomposition of width k can be done in linear time for fixed k [2]. This does not hold for our partial elimination orders.

▶ **Theorem 7.** *The function version* IC-TREEWIDTH *is NP-hard under Turing reductions even when* $k = 5$.

Proof. By reduction from LENGTH-l-PARTIAL-ELIMINATION-ORDER for the special case of $k = 5$. We can solve this problem by iteratively solving IC-TREEWIDTH starting with an empty partial elimination order and ending with one of length $l - 1$. ◀

Another application of Theorem 4 is a greedy algorithm where iteratively the next l vertices are selected instead of a single next vertex. A natural question is, whether we can get an FPT result if we try to select these l vertices in such a way, that number of fill-in edges is minimal. The following result shows that this is unlikely.

MIN-FILLIN-SET	
Instance:	Graph $\mathcal{G} = (V, E)$, integer l and T.
Problem:	Does there exist a set $S \subseteq V$ of size l, such that eliminating the vertices in S from $\mathcal{G}$ adds at most T new edges to $\mathcal{G}$?

▶ **Theorem 8.** MIN-FILLIN-SET *is W[1]-hard when parameterized by* l.

Proof. By reduction from CLIQUE. An instance of CLIQUE is given by a bipartite graph $\mathcal{G}$ and integer k. The question is, whether $\mathcal{G}$ contains a clique of size k. CLIQUE is W[1]-hard when parameterized by k.

Let $\mathcal{G} = (V, E)$ and k be an instance of CLIQUE. We construct an instance $(\mathcal{G}' = (V', E'), l, T)$ of MIN-FILLIN-SET as follows. The vertices V' consist of three disjoint sets $V' = X \cup Y \cup Z$, defined as follows. The set X contains a vertex for each edge in the original graph, i.e., $X = \{x_e \mid e \in E\}$. The set Y contains two copies of each vertex inthe original graph, i.e., $Y = \{y_v, y'_v \mid v \in V\}$. The set Z contains $4k$ new vertices $Z = \{z_1, \dots, z_{4k}\}$. For each edge $e = \{v, w\} \in E$ we add the following 4 edges to E': $\{x_e, y_v\}$, $\{x_e, y'_v\}$, $\{x_e, y_w\}$, and $\{x_e, y'_w\}$. Additionally, E' contains all possible edges between vertex sets Y and Z, i.e., Y and Z form a complete bipartite subgraph. Finally, we set $l = \binom{k}{2}$ and $T = 4\binom{k}{2} + k$. Clearly, $(\mathcal{G}' = (V', E'), l, T)$ can be constructed in polynomial time.

For the correctness, assume first that $C \subseteq V$ is a solution to the Clique problem, i.e., C is a clique of size k. We will show that the $\binom{k}{2}$ edges between vertices of C witness a solution for our Min-FillIn-Set instance. Let $S \subseteq X$ be the $\binom{k}{2}$ vertices corresponding to these edges. By construction, the neighbourhood of S consists of $2k$ vertices in Y that correspond to the k vertices forming the clique C and their copies, say $Y_C = \{y_{i_1}, \dots y_{i_k}\}$ and $Y'_C = \{y'_{i_1}, \dots y'_{i_k}\}$. Eliminating the vertices of S from $\mathcal{G}'$ adds a new edge between every vertex in $y \in Y_C$ and its copy $y' \in Y'_C$, resulting in k new edges. Furthermore, the elimination of a vertex $x_{\{v,w\}} \in S$ forces the following 4 edges to be added to $\mathcal{G}'$: $\{y_v, y_w\}$, $\{y_v, y'_w\}$, $\{y'_v, y_w\}$, and $\{y'_v, y'_w\}$. This results in a total of $T = 4\binom{k}{2} + k$ new edges being added to $\mathcal{G}'$. Hence, S is a solution for the Min-FillIn-Set instance.

For the other direction, assume that $S \subseteq V'$ is a solution for Min-FillIn-Set. Observe that eliminating a vertex $y \in Y$ forces us to create a clique out of the $4k$ vertices Z, resulting in more than T new edges to be added to $\mathcal{G}'$. Similar, eliminating a vertex $z \in Z$ forces us to create a clique out of the vertices in Y. Note that we can assume that Y contains at least $4k$ vertices, since otherwise we could simply blow up the Clique instance by adding at most k new isolated vertices. Hence, S contains only vertices of X. As mentioned above, the elimination of a vertex $x_{\{v,w\}} \in X$ forces the following 4 edges to be added to $\mathcal{G}'$: $\{y_v, y_w\}$, $\{y_v, y'_w\}$, $\{y'_v, y_w\}$, and $\{y'_v, y'_w\}$. By construction, these 4 edges are unique for every vertex $x \in X$. Since S is a solution of size $\binom{k}{2}$, we know that these $4\binom{k}{2}$ new edges are added to $\mathcal{G}'$. Furthermore, for every vertex $v \in V$ that is incident to an edge e corresponding to one of the eliminated vertices $x_e \in S$, the edge $\{y_v, y'_v\}$ is added to $\mathcal{G}'$. Since S is a solution, we know that $\leq T = 4\binom{k}{2} + k$ are added to $\mathcal{G}'$. Hence, S corresponds to a set of $\binom{k}{2}$ edges that is incident to at most k vertices. But this is only true, if S corresponds to the set of edges of a k-clique in $\mathcal{G}$. ◀

4 Turbocharged Treewidth Heuristics

In the previous section we showed that IC-Treewidth is fixed-parameter tractable when parameterized by c and k. We use this FPT algorithm to extend an existing partial elimination order in case a greedy heuristic gets "stuck":

We use a standard greedy algorithm, like GreedyDegree or GreedyFillIn, with one modification. In each step of the heuristic, we check if the next vertex that is to be eliminated, will cause the partial elimination order to exceed our given target width. If this is not the case, we proceed with the heuristic. On the other hand, if we would exceed the target width (we call this a point of regret), instead we backtrack the last c eliminated vertices and use our FPT algorithm to extend this shortened partial elimination order by adding $c + 1$ vertices. If the FPT algorithm is not able to produce such an extension, we abort, otherwise we switch again to the greedy heuristic and continue to the next point of regret.

Algorithm 1 explains this approach in more detail for the case of using the GreedyDegree heuristic. To change the used heuristic, only line 4 needs to be altered. The outer loop (line 3) is executed $|V|$ many times, as each iteration either extends the partial elimination order π one position or aborts the whole search. Lines 5–7 correspond to the case when the

Algorithm 1: TurbochargedMinDegree

Input : Graph $\mathcal{G} = (V, E)$, integer k, integer c.
Output : 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};            // W is bounded by c(k + 1)
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 return π
```

Algorithm 2: IC-Treewidth

Input : Graph $\mathcal{G}$, vertex set W, integer k, integer c.
Output : Pair $(\mathcal{H}, \pi)$ where π is a partial elimination order of width k and length c and $\mathcal{H}$ is the remaining graph. $(null, \emptyset)$ in case of failure.

```
1  for v ∈ W do
2      H ← eliminate(G, v);
3      if c = 1 then
4          return (H, (v));
5      W' ← {v ∈ V(H) | d(v) ≤ k};
6      (H, π') ← IC-Treewidth(H, W', k, c − 1);
7      if π' is not empty then
8          return (H, (v) + π');
9  return (null, ∅);
```

heuristic does not run into a point of regret. Here we add the selected vertex v to π and eliminate it from the graph. In case there is a point of regret, we fix the first part of the elimination order (except the last c positions) and eliminate these vertices from the graph (line 9). Vertex set W at line 10 contains all vertices of degree $\leq k$. These are the vertices which can be eliminated next without exceeding the target treewidth. The FPT algorithm from Theorem 4 is implemented as a recursive procedure outlined in Algorithm 2.

5 Experimental Evaluation

To complement our theoretical analysis of the turbocharged approach, we performed a thorough experimental evaluation of the turbocharged versions of GreedyDegree and GreedyFillIn. The experiments were run on a quad-core Intel Core i7 processor running at 2.7 GHz with 16GB of RAM. The implementation was in Java 7. We implemented and

Table 1 Comparison of average quality and average running time on different classes of randomly generated partial k-trees.

			`min-degree`		`min-fill-in`		`turbo-min-degree`		`turbo-min-fill-in`	
n	k	p	quality	time	quality	time	quality	time	quality	time
250	10	0.20	10.44	**0.12**	11.42	0.18	10.44	0.22	**10.12**	0.43
250	10	0.40	10.16	**0.10**	11.34	0.15	10.16	0.20	**10.04**	0.36
250	15	0.20	15.60	**0.17**	16.64	0.27	15.60	0.28	**15.34**	0.63
250	15	0.40	15.20	**0.14**	16.38	0.22	15.20	0.26	**15.12**	0.51
250	20	0.20	20.64	**0.22**	21.96	0.37	20.64	0.35	**20.32**	0.86
250	20	0.40	20.22	**0.18**	21.60	0.30	20.22	0.34	**20.08**	0.69
500	10	0.20	10.72	**0.36**	11.72	0.59	10.72	0.51	**10.24**	1.55
500	10	0.40	10.32	**0.28**	11.64	0.44	10.32	0.49	**10.26**	1.23
500	15	0.20	15.94	**0.63**	16.86	1.09	15.94	0.83	**15.70**	2.71
500	15	0.40	15.32	**0.46**	17.04	0.78	15.32	0.79	**15.20**	1.96
500	20	0.20	20.88	**0.94**	22.18	1.67	20.88	1.21	**20.82**	4.04
500	20	0.40	**20.32**	**0.67**	22.08	1.17	**20.32**	1.16	20.38	2.84
1000	10	0.20	10.90	**1.75**	11.94	3.11	10.90	2.08	**10.64**	7.81
1000	10	0.40	10.56	**1.29**	11.98	2.18	10.56	1.93	**10.20**	5.83
1000	15	0.20	16.04	**3.46**	17.20	6.71	16.04	3.87	**15.94**	15.46
1000	15	0.40	15.58	**2.44**	17.26	4.40	15.58	3.70	**15.46**	10.78
1000	20	0.20	**21.16**	**5.34**	22.38	10.24	**21.16**	5.58	21.54	22.38
1000	20	0.40	20.50	**3.76**	22.56	6.90	20.50	5.77	**20.34**	15.84

tested the following four algorithms:

- `min-degree`: Iteratively eliminates a vertex with minimum degree.
- `min-fill-in`: Iteratively eliminates a vertex with minimum fill-in.
- `turbo-min-degree`: The turbocharged version of `min-degre`.
- `turbo-min-fill-in`: The turbocharged version of `min-fill-in`.

When generating the elimination order for each of the above algorithms ties between vertices need to be broken. To handle this we use a fixed seed to generate a random permutation on the vertices. This permutation is then used to break ties. Using the same seed across all algorithms allows for a fair comparison between the heuristic and its turbocharged version.

The turbocharged version of the `min-degree` heuristic was implemented using the pseudo-code given in Algorithms 1 and 2, with one minor enhancement. In the first line of Algorithm 2, no specific order is given on W. To better guide the search, we first sort the vertices in W by increasing degree. The idea is that while we are now sorting the set W according to the heuristic at every call, we hope to find an extended ordering quicker. The `min-fill-in` heuristic is implemented using the corresponding versions of Algorithms 1 and 2, with the same enhancement.

We tested our algorithms on two types of instances: randomly generated partial k-trees (Section 5.1), and benchmark instances (Section 5.2).

In the rest of this section we explain how these instances were generated/sourced and analyze the experimental performance of the different algorithms.

5.1 Random instances

The partial k-trees were generated using the method by Gogate and Dechter [6, Section 7.2]. The generator takes as input a triple of parameters (n, k, p). It generates a graph of treewidth at most k having n nodes and $(1-p)\left(kn - \binom{k+1}{2}\right)$ edges. In order to ensure that the graph has a tree decomposition of width *exactly* k, we apply the Maximum-Minimum Degree (MMD) lower bound proposed by Koster et al. [8] and only keep those that are guaranteed to have treewidth k. Fifty partial k-trees were generated for each triple (n, k, p), for all combinations of the following parameters $n = \{250, 500, 1000\}$, $k = \{10, 15, 20\}$ and $p = \{0.2, 0.4\}$.

From Theorem 4 we know that the number of times the turbocharged heuristic has to backtrack might be exponential in the length of the partial elimination order (l). Therefore, to keep the computation tractable, c needs to be small. For the experiments we choose $c = 8$ as the default value.

Table 1 provides statistical summaries of the quality and running times of the different algorithms on the randomly generated instances. The running times of `turbo-min-degree` and `turbo-min-fill-in` contain the running times of `min-degree` and `min-fill-in`, respectively, since whenever the turbocharged version fails to find a decomposition of given target width, we return the result of the standard version instead. Hence, the quality of the turbocharged version will always be at least as good as the quality of the standard algorithm and the running time will always be slower. Our first observation is that `min-degree` outperforms `min-fill-in` in terms of time and quality, which is consistent with the results reported by Bodlaender and Koster [3].

For `turbo-min-degree`, we saw no improvement in the quality of the decomposition. This is probably because in most cases `min-degree` finds the optimal solution (47% of the instances) or a solution very close to the optimal. Even after setting $c = 12$ the turbocharged version failed to improve on any instances.

For `turbo-min-fill-in`, however, we observed a large improvement in quality. In this case the algorithm was able to find the optimal treewidth in 690 out of the 900 instances. In many of the smaller instances, the algorithm did not even backtrack the full $c = 8$ vertices; indeed, on average only six steps was required. This means that for `min-fill-in` we spend a few additional seconds to turbocharge the heuristic and get a considerable improvement. Note that for most of the random instances `turbo-min-fill-in` finds a treewidth that is better than the ones found by `min-degree` and `turbo-min-degree`.

5.2 Benchmark instances

Two data sets were used for the experiments: DIMACS Graph coloring networks instances,[2] and Bayesian networks repository instances.[3] In total, there are 73 instances out of which 63 are DIMACS Graph coloring networks instances and 10 are Bayesian networks repository instances. The purpose of these experiments is to test the turbocharged heuristics performance on larger instances.

Each heuristic, `min-degree` and `min-fill-in`, was executed three times on each instance. The best result (smallest treewidth) for each heuristic was selected. Finally, the heuristic producing the best result for each instance was turbocharged, using the same random seed for consistency.

For the turbocharged version the heuristic requires a target treewidth parameter (k), which is unknown. To get around this problem we chose to perform a biased binary search as follows. Let k' be the best treewidth found by either `min-degree` or `min-fill-in`. The experimental evaluation showed that the turbocharge heuristic typically improved the treewidth by 3-5%. As a result we chose to perform a binary search in the range $[0.94 \cdot k', k' - 1]$ which terminated after four iterations. In the case when this interval is non-existent (i.e. $(k' - 1)/k' \leq 0.94$), we run the turbocharged heuristic with $k' - 1$, $k' - 2$, and so on.

---

[2] `http://mat.gsia.cmu.edu/COLOR/instances.html`
[3] `http://www.cs.huji.ac.il/site/labs/compbio/Repository/`

Table 2 A subset of the experimental results on DIMACS Graph coloring networks. For instances DSJC1000.5 and DSJC500.9 we used $c = 6$, and for the other instances $c = 8$.

				`min-degree`		`min-fill-in`		`turbo`	
id	n	m	tw	quality	time	quality	time	quality	time
queen7_7	49	952	35	37	**0.056**	37	0.075	**36**	0.104
queen8_8	64	1456	46	50	**0.081**	48	0.099	**47**	0.543
queen9_9	81	2112	59	64	**0.100**	63	0.128	**62**	0.266
queen11_11	121	3960	89	97	**0.231**	95	0.283	**93**	12.49
queen13_13	169	6656	125	140	**0.610**	137	0.808	**135**	36.67
queen14_14	196	8372	143	164	**1.060**	160	1.372	**159**	95.08
myciel4	23	71	10	11	**0.011**	11	0.016	**10**	4.62
le450_5b	450	5734	309	316	**15.12**	318	19.42	**311**	500.3
le450_15c	450	16680	372	376	**21.35**	376	26.44	**372**	240.6
le450_25d	450	17425	349	367	**20.48**	363	25.18	**360**	584.4
DSJC1000.5	1000	499652	977	980	**642**	978	705	**977**	5429
DSJC125.1	125	1472	64	67	**0.144**	66	0.170	**65**	54.885
DSJC250.1	250	6436	176	180	**1.835**	177	2.300	**176**	264.46
DSJC500.1	500	24916	409	413	**31.086**	411	43.048	**410**	2089.77
DSJC500.5	500	125248	479	481	**41.024**	482	48.481	**479**	19467.95
DSJC500.9	500	224874	492	493	**45**	493	47	**492**	2662

Coloring

We ran our heuristics on 63 instances of the DIMACS Graph coloring networks. Some of the results are shown in Table 2. The fourth column shows the best known treewidth for each instance extracted from the papers by Koster et al. [8] and Gogate and Dechter [6]. Each row also lists the results obtained by the `min-degree`, the `min-fill-in` and the turbocharged version (`turbo`).

In the 31 cases where neither greedy heuristics found the best known solution, the turbocharge method was able to improve the result in 16 of the instances. These instances are listed in Table 2. Specifically, in six of the cases the turbocharged version was able to find a tree decomposition that has width equal to the best known solution.

In all but two cases the computation terminated within two hours. Note that due to the size of some of the large instances, the parameter $c = 6$ was used for four instances, $c = 4$ for one instance and, $c = 8$ for the remaining instances.

In Table 3 we list the same instances as in Table 2 to compare our results with the results reported by Koster et al. [8] and Gogate and Dechter [6]. However it should be noted that Koster et al. [8] implemented several approaches with varying quality performance and speed, but we only include the smallest treewidth result in the table. For more details see [8].

Bayesian Networks

The Bayesian network instances are directed graphs transformed into undirected graphs for the experiments. The set contains ten instances, most of these are quite small so in nine out of the ten instances either the `min-degree` or `min-fill` heuristic found the best known solution. Therefore turbocharging yielded no benefit. The only exception out of the ten instances was the Link instance, where the turbocharged algorithm was able to improve the min-fill heuristic from 15 to 13, which also improved the best known bound for this instance.

6 Conclusion and Future Work

We studied variants of the Treewidth problem that aim at modelling local search scenarios that arise in the context of tree decomposition heuristics. We have shown that IC-Treewidth,

Table 3 A comparison between turbocharged heuristics and the results reported by Gogate and Dechter [6] and Koster et al. [8]. Note that the algorithm by Gogate and Dechter [6] was terminated after 3 hours. Also note that Koster et al. [8] implemented several approaches with varying quality performance and speed, however, only the smallest treewidth result is listed in this table.

id	n	m	Gogate and Dechter [6] quality	Gogate and Dechter [6] time	Koster et al. [8] quality	Koster et al. [8] time	`turbo` quality	`turbo` time
queen7_7	49	952	**35**	543	**35**	0.51	36	**0.10**
queen8_8	64	1456	**46**	10800	**46**	1.49	47	**0.54**
queen9_9	81	2112	**59**	10800	**59**	3.91	62	**0.27**
queen11_11	121	3960	**89**	10800	**89**	23.36	93	**12.5**
queen13_13	169	6656	**125**	10800	**125**	107.6	135	**36.7**
queen14_14	196	8372	**143**	10800	145	215.4	159	**95.1**
myciel4	23	71	**10**	10800	**10**	**0.01**	**10**	4.6
le450_5b	450	5734	**309**	10800	313	7909	311	**500**
le450_15c	450	16680	**372**	10800	376	12471	**372**	**241**
le450_25d	450	17425	**349**	10800	356	11376	360	**584**
DSJC1000.5	1000	499652	**977**	10800	*	*	**977**	**5429**
DSJC125.1	125	1472	**64**	10800	67	171.5	65	**54.9**
DSJC250.1	250	6436	**176**	10800	179	5507	**176**	**264**
DSJC500.1	500	24916	**409**	10800	*	*	410	**2089**
DSJC500.5	500	125248	**479**	**10800**	*	*	**479**	19468
DSJC500.9	500	224874	**492**	10800	*	*	**492**	**2662**

the problem of extending a given partial elimination order without increasing its width by recomputing at most the last c eliminated vertices, is hard when parameterized by either the length of the partial elimination order or its width. But the problem becomes fixed-parameter tractable when parameterized by the width and c combined. We used this FPT result to turbocharge existing greedy heuristics by performing this local search whenever the heuristic would exceed some given target width. This approach was implemented and evaluated, showing that we can improve the quality of the heuristics with a modest trade off in the running time.

In future work it would be interesting to study a permissive variant of IC-Treewidth, which, for a given graph $G = (V, E)$, integer k, and partial elimination order π of length l and width at most k, asks to either compute a partial elimination order of length $l + 1$ and width at most k, or to determine that G has no elimination order (of length $|V|$) that coincides with π on the first $l - c$ vertices.

Acknowledgments. We thank Michael R. Fellows for inspiring this line of research.

References

1. Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in ak-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.
2. Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.
3. Hans L. Bodlaender and Arie M. C. A. Koster. Treewidth computations I. Upper bounds. *Inf. Comput.*, 208(3):259–275, 2010. doi:10.1016/j.ic.2009.03.008.
4. Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness II: on completeness for W[1]. *Theor. Comput. Sci.*, 141(1&2):109–131, 1995.
5. Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.
6. Vibhav Gogate and Rina Dechter. A complete anytime algorithm for treewidth. In *Proc. of UAI'04*, pages 201–208. AUAI Press, 2004.

7 Sepp Hartung and Rolf Niedermeier. Incremental list coloring of graphs, parameterized by conservation. *Theor. Comput. Sci.*, 494:86–98, 2013.

8 Arie M. C. A. Koster, Hans L. Bodlaender, and Stan P. M. van Hoesel. Treewidth: Computational experiments. *Electronic Notes in Discrete Mathematics*, 8:54–57, 2001.