An Improved Parameterized Algorithm for Treewidth*

Tuukka Korhonen†
tuukka.korhonen@uib.no
Department of Informatics, University of Bergen
Norway

Daniel Lokshtanov‡
daniello@ucsb.edu
Department of Computer Science, University of California
Santa Barbara
USA

ABSTRACT

We give an algorithm that takes as input an n-vertex graph G and an integer k, runs in time $2^{O(k^2)}n^{O(1)}$, and outputs a tree decomposition of G of width at most k, if such a decomposition exists. This resolves the long-standing open problem of whether there is a $2^{o(k^3)}n^{O(1)}$ time algorithm for treewidth. In particular, our algorithm is the first improvement on the dependency on k in algorithms for treewidth since the $2^{O(k^3)}n^{O(1)}$ time algorithm given by Bodlaender and Kloks [ICALP 1991] and Lagergren and Arnborg [ICALP 1991].

We also give an algorithm that given an n-vertex graph G, an integer k, and a rational $\varepsilon \in (0,1)$, in time $k^{O(k/\varepsilon)}n^{O(1)}$ either outputs a tree decomposition of G of width at most $(1+\varepsilon)k$ or determines that the treewidth of G is larger than k. Prior to our work, no approximation algorithms for treewidth with approximation ratio less than 2, other than the exact algorithms, were known. Both of our algorithms work in polynomial space.

CCS CONCEPTS

• **Theory of computation → Graph algorithms analysis**; **Parameterized complexity and exact algorithms**.

KEYWORDS

treewidth, parameterized complexity

ACM Reference Format:
Tuukka Korhonen and Daniel Lokshtanov. 2023. An Improved Parameterized Algorithm for Treewidth. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC '23), June 20–23, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3564246.3585245

1 INTRODUCTION

A *tree decomposition* of a graph G is a pair (T, bag) where T is a tree and bag is a function assigning to each node t of T a set $\mathsf{bag}(t)$ (called a *bag*) of vertices of G. The function bag must satisfy the tree decomposition axioms: *(i)* for every edge uv of G at least one bag $\mathsf{bag}(t)$ contains both u and v, and *(ii)* for every vertex v of G, the set $\{t \in V(T) \mid v \in \mathsf{bag}(t)\}$ induces a *non-empty* and *connected* subtree of T. The *width* of a tree decomposition (T, bag) is the size of a largest bag minus one, and the *treewidth* of a graph G is the minimum width of a tree decomposition of G. The treewidth of a graph G measures, in some sense, how far away G is from being a tree. The treewidth of G is at most 1 if and only if every connected component of G is a tree, while the treewidth of a complete graph on n vertices is equal to $n-1$ [23].

Treewidth and tree decompositions play a central role in graph theory and graph algorithms, and the concept has been independently rediscovered several times [7, 33, 41] under different names in different contexts. It is a key tool in the celebrated Graph Minors project of Robertson and Seymour [41–43]. Many problems that are intractable on general graphs are solvable in linear time if a tree decomposition of the input graph G of constant width is provided as an input (see e.g. [9] and references within). Indeed, the classic Courcelle's Theorem [21] states that such an algorithm exists for every problem expressible in Monadic Second Order Logic (see also [19]).

Therefore it should not come as a surprise that a significant amount of attention has been devoted to designing algorithms to determine, given as input a graph G and an integer k, whether the treewidth of G is at most k (and to produce a tree decomposition of width at most k in the "yes" case). This problem is known to be NP-complete [3], however, in many settings tree decompositions are only relevant if the treewidth of the input graph is sufficiently small, directing research towards algorithms with running times of the form $f(k)\cdot n^{g(k)}$ or $f(k)\cdot n^{O(1)}$. Algorithms with running time of the first type are called *slicewise polynomial*, since they run in polynomial time when k is considered a constant. Algorithms of the second type are called *fixed-parameter tractable (FPT)* as they run in polynomial time if k is considered a constant, and furthermore the exponent of the polynomial remains the same for different values of k. We refer to the textbooks [22, 25, 29, 39] for an introduction to parameterized algorithms.

The first slicewise polynomial algorithm for treewidth was given by Arnborg, Corneil and Proskurowski [3], with running time $O(n^{k+2})$. Subsequently, Robertson and Seymour [42], gave a non-constructive (see Bodlaender [10] for a discussion of the non-constructive nature of [42]) $f(k)n^2$ time algorithm for treewidth, and Bodlaender [10], building on work of Fellows and Langston [28] made this algorithm constructive. The function f in the running time both of the algorithm of Robertson and Seymour [42] and of Bodlaender [10] is unspecified and was not even known to be computable at the time of publication.

The algorithm of Robertson and Seymour [42] follows a "two-step" approach. In the first step they compute a tree decomposition of G of width at most $4k+3$ in time $O(3^{3k}n^2)$, or conclude that the

*Due to space limits, most of technicals details are omitted or just sketched. The full version of the paper is available on arXiv [35].
†Supported by Research Council of Norway via the project BWCA (grant no. 314528).
‡Supported by NSF award CCF-2008838.

STOC '23, June 20–23, 2023, Orlando, FL, USA

ACM ISBN 978-1-4503-9913-5/23/06.
https://doi.org/10.1145/3564246.3585245

treewidth of G is more than k. In the second step they do dynamic programming over the tree decomposition found in the first step. The second step is the only non-constructive part of their algorithm, and runs in time $f(k)n$ where the function f is unspecified.

Matoušek and Thomas [38], Lagergren [36], and Reed [40] gave improved algorithms for the first step. The algorithms of Matoušek and Thomas and Lagergren run in time $k^{O(k)}n\log^2 n$, and the algorithm of Reed runs in time $k^{O(k)}n\log n$. All three algorithms either conclude that the treewidth of G is more than k, or produce a tree decomposition of width at most $O(k)$. The algorithm of Lagergren [36] is given as a parallel algorithm with $k^{O(k)}\log^3 n$ running time on $O(k^2n)$ processors.

For the second step, constructive $2^{O(k^3)}n$ time dynamic programming algorithms were discovered in 1991 independently by Lagergren and Arnborg [37], and Bodlaender and Kloks [16, 17]. None of [16, 17, 37] explicitly mention the dependence on k, but the $2^{O(k^3)}$ bound directly follows from the analysis in [17] and is mentioned in [11]. Combined with the algorithm for the first step by Lagergren [36], this led to a $2^{O(k^3)}n\log^2 n$ time algorithm for treewidth. In 1993, Bodlaender showed that the first phase of the algorithms can be replaced by an ingenious recursion scheme, and designed a linear $2^{O(k^3)}n$ time algorithm for treewidth [8, 11]. Much more recently, Elberfeld, Jakoby, and Tantau [26] gave an algorithm for treewidth that uses *space* $f(k)\log n$ and time $n^{f(k)}$.

Downey and Fellows asked in their monograph from 1999 whether the dependence on k in Bodlaender's algorithm could be improved from $2^{O(k^3)}$ to $2^{O(k)}$ [24, Chapter 6.3]. Later, in 2006, Telle [12, Problem 2.7.1] asked the less ambitious question of whether there is any fixed-parameter algorithm for treewidth whose running time as a function of k is better than $2^{O(k^3)}$. The problem of obtaining a $2^{o(k^3)}n^{O(1)}$ time algorithm was also asked by Bodlaender, Drange, Dregi, Fomin, Lokshtanov, and Pilipczuk [13] and called a "long-standing open problem" by Bodlaender, Jaffke, and Telle [15]. In this paper, we resolve this problem.

Theorem 1. *There is an algorithm that takes as input an n-vertex graph G and an integer k, and in time $2^{O(k^2)}n^4$ either outputs a tree decomposition of G of width at most k or concludes that the treewidth of G is larger than k. Moreover, the algorithm works in space polynomial in n.*

An interesting feature of our algorithm is that it runs in polynomial space, and in particular that it is not based on dynamic programming. All previously known parameterized algorithms for computing treewidth exactly [3, 11, 17, 37, 42] are based on dynamic programming and use space exponential in k. The running time dependence on n of the algorithm of Theorem 1 is significantly worse than that of Bodlaender [11]. The dependence on n of our algorithm can probably be improved, nevertheless we believe that an algorithm with running time $2^{O(k^2)}n^2$ or better should require new and interesting ideas.

Our second contribution is a new parameterized approximation algorithm for treewidth.

Theorem 2. *There is an algorithm that takes as input an n-vertex graph G, an integer k, and a rational $\varepsilon \in (0,1)$, and in time $k^{O(k/\varepsilon)}n^4$ either outputs a tree decomposition of G of width at most $(1+\varepsilon)k$ or concludes that the treewidth of G is larger than k. Moreover, the algorithm works in space polynomial in n.*

There is a rich history of approximation algorithms for treewidth. In terms of *polynomial time* approximation algorithms, the best known approximation algorithm [27] by Feige, Hajiaghayi and Lee has approximation factor $O(\sqrt{\log k})$, improving upon a $O(\log n)$-approximation algorithm [14] and a $O(\log k)$-approximation algorithm [1]. On the other hand, Wu, Austrin, Pitassi and Liu [44] showed that assuming the Small Set Expansion Conjecture (and $\mathsf{P} \neq \mathsf{NP}$), there is no constant factor approximation algorithm for treewidth.

Treewidth is one of the unusual cases where the first FPT-approximation algorithm (an approximation algorithm with running time $f(k)n^{O(1)}$) pre-dates the first polynomial time approximation algorithm. The first such algorithm, a 4-approximation algorithm running in time $O(3^{3k}n^2)$, is the "first step" of the $f(k)n^2$ time non-constructive algorithm by Robertson and Seymour for exactly computing treewidth [42]. Subsequent research, summarized in Table 1 attained different trade-offs between the running time dependence on n, the running time dependence $f(k)$ on k, and the approximation factor. The algorithm of Theorem 2 is the first FPT-approximation algorithm for treewidth with approximation ratio below 2 and running time $2^{o(k^2)}n^{O(1)}$ (or even $2^{o(k^3)}n^{O(1)}$, discounting Theorem 1). Note that by setting $\varepsilon = \frac{1}{k+1}$, the algorithm of Theorem 2 gives an exact algorithm with only a slightly slower $(k^{O(k^2)}n^4)$ running time than the algorithm of Theorem 1, in particular, being sufficient for resolving the open problem of obtaining a $2^{o(k^3)}n^{O(1)}$ time algorithm for treewidth. This is worth noting since the algorithm of Theorem 2 is in fact considerably simpler than the algorithm of Theorem 1.

Methods. Both the exact algorithm of Theorem 1 and the approximation algorithm of Theorem 2 are based on a generalization of the local improvement method introduced by Korhonen [34], which in turn was inspired by a proof of Bellenbaum and Diestel [6]. In each local improvement step we are given a tree decomposition (T, bag) of G of width more than k, and the goal is to either conclude that the treewidth of G is more than k, or to find a "better" tree decomposition of G. Here *better* means that either the width of the output tree decomposition is strictly smaller than that of (T, bag), or that the width of the output decomposition is the same as the width of (T, bag), but there are fewer bags of maximum size.

We show that the local improvement step is in fact *equivalent* to solving the following problem, which we call Subset Treewidth: given as input a graph G and a set W of vertices, conclude that the treewidth of G is at least $|W|-1$, or find a tree decomposition (T', bag') such that W is contained in the union of the non-leaf bags of (T', bag') and all non-leaf bags have size at most $|W|-1$ (the formal definition of this problem in Section 2 is worded differently, but can easily be seen to be equivalent). Observe here that if the treewidth of G is strictly less than $|W|-1$ then every tree decomposition (T', bag') of G of width at most $|W|-2$ is a valid output for Subset Treewidth (after possibly adding empty dummy leaf bags).

The first key insight behind our algorithms is that if W is a maximum size bag of the tree decomposition (T, bag), and an algorithm

Table 1: Overview of treewidth algorithms with running time $f(k) \cdot g(n)$, each either outputting a tree decomposition of width at most $\alpha(k)$ or determining that the treewidth of the input graph is larger than k. Most of the rows are based on a similar tables in [13] and [34].

Reference	Appx. $\alpha(k)$	$f(k)$	$g(n)$
Arnborg, Corneil, and Proskurowski [3]	exact	$O(1)$	n^{k+2}
Robertson and Seymour [42]	$4k+3$	$O(3^{3k})$	n^2
Matoušek and Thomas [38]	$6k+5$	$k^{O(k)}$	$n \log^2 n$
Lagergren [36]	$8k+7$	$k^{O(k)}$	$n \log^2 n$
Reed [40]	$8k+O(1)$	$k^{O(k)}$	$n \log n$
Bodlaender [11]	exact	$2^{O(k^3)}$	n
Amir [2]	$4.5k$	$O(2^{3k}k^{3/2})$	n^2
Amir [2]	$(3+2/3)k$	$O(2^{3.7k}k^3)$	n^2
Amir [2]	$O(k \log k)$	$O(k \log k)$	n^4
Feige, Hajiaghayi, and Lee [27]	$O(k\sqrt{\log k})$	$O(1)$	$n^{O(1)}$
Fomin, Todinca, and Villanger [32]	exact	$O(1)$	1.7347^n
Fomin et al. [30]	$O(k^2)$	$O(k^7)$	$n \log n$
Bodlaender et al. [13]	$3k+4$	$2^{O(k)}$	$n \log n$
Bodlaender et al. [13]	$5k+4$	$2^{O(k)}$	n
Korhonen [34]	$2k+1$	$2^{O(k)}$	n
Belbasi and Fürer [5]	$5k+4$	$2^{7.61k}$	$n \log n$
Belbasi and Fürer [4]	$5k+4$	$2^{6.755k}$	$n \log n$
This paper	exact	$2^{O(k^2)}$	n^4
This paper	$(1+\varepsilon)k$	$k^{O(k/\varepsilon)}$	n^4

for Subset Treewidth on input (G, W) outputs the decomposition (T', bag'), then a tree decomposition better than (T, bag) (in the sense above) can be computed from (T, bag) and (T', bag') in polynomial time. The proof of this statement is given in Section 5 and is a non-trivial generalization of corresponding improvement arguments by Bellenbaum and Diestel [6] and Korhonen [34]. Indeed, in retrospect, the 2-approximation algorithm of Korhonen [34] can be thought of as using this approach with the additional assumption that $|W| \geq 2k+3$, where k is the treewidth of G, and in this case a solution to Subset Treewidth whose non-leaf bags form the star $K_{1,3}$ exists. The exact algorithm of Theorem 1 is based on solving Subset Treewidth without any additional assumptions, while the approximation algorithm of Theorem 2 is based on solving Subset Treewidth with the additional assumption that $|W| \geq k(1+\varepsilon)+2$.

The second key insight is that the Subset Treewidth problem is more approachable than the treewidth problem, because the problem formulation allows us to focus on one small set W and "discard" all parts of the graph (by placing them into leaves of (T', bag')) that are not relevant for providing connectivity between vertices of W. Both the algorithm of Theorem 1 and of Theorem 2 are based on branching on important separators (see e.g. [22, Chapter 8]), a carefully chosen measure to quantify the progress made by the algorithms, and a "safe separation" reduction rule for the Subset Treewidth problem. This rule states that if the algorithm has identified two vertex sets B_1 and B_2 that can be chosen as bags of (T', bag'), and S is a minimum size (B_1, B_2)-separator, then it is safe to also make S a bag of (T', bag') and recurse on the connected components of $G \setminus S$. A generalization of this reduction rule was formulated for the treewidth problem by Bodlaender and Koster [18, Lemma 11]. However it is not clear how to utilize this reduction rule to directly obtain efficient algorithms for treewidth. On the other hand, for Subset Treewidth, this reduction rule is the main engine of our algorithms.

Organization. Due to space limits, most of technicals details are omitted or just sketched. The full version of the paper is available on arXiv [35]. The rest of the paper is organized as follows. In Section 2 we formally define the Subset Treewidth problem, give statements of intermediate theorems on how Theorems 1 and 2 follow from algorithms for Subset Treewidth, and then present an overview of the proofs. In Section 3 we present notation and preliminary results. In Section 4 we give a "pulling lemma" for tree decompositions, which will be used for our algorithms. We note that the Section 5 of the full version also contains results about important separators that we omit here for space constraints. In Section 5 we show that algorithms for Subset Treewidth imply algorithms for treewidth. In Section 6 of the full version we give the algorithm for Subset Treewidth that implies Theorem 2, and in Section 7 of the full version we give the algorithm that implies Theorem 1.

2 OVERVIEW

In this section we state the main intermediate theorems leading into Theorems 1 and 2 and overview the proofs of them. The proofs of Theorem 1 and of Theorem 2 neatly split in two parts. The first part is common to the proofs of Theorem 1 and Theorem 2, while the second part requires separate proofs. The first and common part is the overall scheme of the algorithms, namely that we proceed by "local improvement". Each local improvement step is reduced to another problem, which we call Subset Treewidth. In the second

part we give two different algorithms for the Subset Treewidth problem, one exact, leading to a proof of Theorem 1, and one approximate, leading to a proof of Theorem 2. We start by discussing the first part.

2.1 Reduction to Subset Treewidth

Suppose that we are given as input the graph G and integer τ, and the task is to either return that the treewidth of G is more than τ, or find a "good enough" tree decomposition of G. For an exact algorithm this simply means a tree decomposition of width at most τ, for a $(1+\varepsilon)$-approximation algorithm this means a tree decomposition of width at most $\tau(1+\varepsilon)$. Assume now that we are also given as input a tree decomposition (T, bag) of G of width at most $O(\tau)$. Initially such a tree decomposition can be obtained by an approximation algorithm, such as the 4-approximation algorithm of Robertson and Seymour [42] with running time $O(3^{3\tau}n^2)$. If the tree decomposition (T, bag) is already good enough, then we can output it and halt. Otherwise, a largest bag W of (T, bag) is too large. We would like to make (T, bag) better by getting rid of this bag W that is too large. More formally we want to find a tree decomposition (T'', bag'') of G of width at most $|W|-1$ and with strictly fewer bags of size $|W|$ than (T, bag) has. On the surface this does not really look any easier than trying to find a tree decomposition of width at most $|W|-2$. Somewhat miraculously it turns out that it is in fact easier, because this problem is equivalent to the Subset Treewidth problem, which we will define shortly. To define the Subset Treewidth problem we first need to introduce some notation.

Let G be a graph and $X \subseteq V(G)$. The graph $\mathtt{torso}_G(X)$ has vertices $V(\mathtt{torso}_G(X)) = X$ and has $uv \in E(\mathtt{torso}_G(X))$ if $u, v \in X$ and there is a path from u to v whose all internal vertices (if any) are in $V(G) \setminus X$. In particular, note that $E(\mathtt{torso}_G(X)) \supseteq E(G[X])$. An equivalent definition of $\mathtt{torso}_G(X)$ is that it is the graph obtained from $G[X]$ by making $N_G(C)$ a clique for every connected component C of $G \setminus X$. A *torso tree decomposition* in a graph G is a pair $(X, (T, \mathsf{bag}))$, where $X \subseteq V(G)$ and (T, bag) is a tree decomposition of $\mathtt{torso}_G(X)$. The *width* of the torso tree decomposition $(X, (T, \mathsf{bag}))$ is simply the width of (T, bag). For a set $W \subseteq V(G)$, we say that $(X, (T, \mathsf{bag}))$ *covers* W if $W \subseteq X$. We are now ready to define the Subset Treewidth problem.

Subset Treewidth	**Parameter:** k
Input: Graph G, integer k, and a set of vertices W of size $\lvert W\rvert = k+2$.	
Question: Return a torso tree decomposition of width at most k in G that covers W or conclude that the treewidth of G is at least $k+1$.	

Note that at least one of the two cases in the definition of Subset Treewidth must apply. In particular, if G has a tree decomposition (T', bag') of width at most k then $(V(G), (T', \mathsf{bag}'))$ is a torso tree decomposition of width at most k in G that covers W. The two cases need not be mutually exclusive: there exists graphs G with treewidth at least $k+1$ and sets W of size $k+2$ that nevertheless can be covered by a torso tree decomposition of width k. In such a case an algorithm for Subset treewidth may output either one of the two options.

The Subset Treewidth problem directly reduces to treewidth: using a hypothetical treewidth algorithm we can determine whether the treewidth of G is at most k. If no, then report that the treewidth of G is at least $k+1$. Otherwise output $(V(G), (T', \mathsf{bag}'))$ where (T', bag') is the width k tree decomposition returned by the treewidth algorithm. Our algorithms for treewidth are based on the result that we can reduce in the other direction as well. We encapsulate this insight in the following lemma.

Lemma 2.1. *Let (T, bag) be a tree decomposition of G and W be a largest bag of (T, bag). If there exists a torso tree decomposition $(X, (T', \mathsf{bag}'))$ in G that covers W and has width at most $|W|-2$, then there exists a tree decomposition (T'', bag'') of G of width at most $|W|-1$ with strictly fewer bags of size $|W|$. Moreover, given G, (T, bag) and $(X, (T', \mathsf{bag}'))$ we can compute (T'', bag'') in polynomial time.*

Lemma 2.1 is more carefully stated and proved as Lemma 5.6 in Section 5. Before giving a proof sketch of Lemma 2.1 in Section 2.2, we show how Lemma 2.1, together with an algorithm (or approximation algorithm) for Subset Treewidth yields an algorithm (or approximation algorithm) for treewidth.

Indeed, starting with a tree decomposition (T, bag) of width $O(\tau)$ but more than τ, we can call an algorithm for Subset Treewidth on a largest bag W of (T, bag), and either conclude that the treewidth of G is more than τ or obtain a torso tree decomposition $(X, (T', \mathsf{bag}'))$ that covers W and has width at most $|W|-2$. Lemma 2.1 now yields a tree decomposition (T'', bag'') with no larger width and strictly fewer bags of size $|W|$. We now repeat the process with (T'', bag'') as the new (T, bag). After at most $O(\tau n)$ iterations we will either have obtained a tree decomposition of G of width at most τ or concluded that the treewidth of G is more than τ. We now state this as a theorem. For the running times, we use $m = |V(G)| + |E(G)|$ to denote the size of the graph and we assume that the function $T(k)$ is increasing.

Theorem 3. *Given an algorithm for Subset Treewidth with running time $T(k) \cdot m^c$, an algorithm for treewidth with running time $T(O(k)) \cdot O((nk)^{c+1}) + k^{O(1)}n^4 + 2^{O(k)}n^2$ can be constructed. Moreover, if the algorithm for Subset Treewidth works in polynomial space, then the algorithm for treewidth works in polynomial space.*

Theorem 3 is proved in Section 5. We remark that the additive $2^{O(k)}n^2$ term comes from starting by applying the factor 4-approximation algorithm of Robertson and Seymour [42]. This additive term could be avoided by replacing this approximation algorithm by Bodlaender's recursive compression technique [11], at the expense of a $k^{O(1)}$ multiplicative factor in the running time. In light of Theorem 3 it is natural to focus on parameterized algorithms for Subset Treewidth, which is precisely our line of attack. In Section 7 of the full version we give a $2^{O(k^2)}nm$ time polynomial space algorithm for Subset Treewidth.

Theorem 4. *There is a $2^{O(k^2)}nm$ time polynomial space algorithm for Subset Treewidth.*

A proof sketch for Theorem 4 is given in Section 2.3. Putting Theorems 3 and 4 together implies Theorem 1. The argument proving Theorem 3 (assuming Lemma 2.1) also works for approximation algorithms. In particular the same argument shows that in order to

obtain a $(1+\varepsilon)$-approximation algorithm for treewidth, it is sufficient to design an algorithm for Subset Treewidth that is only required to work correctly on instances where $|W| \geq (1+\varepsilon)\tau + 2$ (and τ is the treewidth of G). Towards designing such an algorithm we define an intermediate problem, called Partitioned Subset Treewidth.

Partitioned Subset Treewidth	**Parameters:** k, t
Input: Graph G, integer k, set of vertices W of size $\|W\| = k+2$, and t cliques $W_1, \ldots, W_t$ of G such that $\bigcup_{i=1}^{t} W_t = W$.	
Question: Return a torso tree decomposition of width at most k in G that covers W or conclude that the treewidth of G is at least $k+1$.	

The Partitioned Subset Treewidth problem arises naturally when designing a recursive branching algorithm for Subset Treewidth. Every instance of Subset Treewidth is an instance of Partitioned Subset Treewidth with $t = |W| = k+2$ and $W_i = \{w_i\}$ (where $W = \{w_1, w_2, \ldots, w_{|W|}\}$). However, Partitioned Subset Treewidth appears substantially easier when t is much smaller than k. The following theorem, proved in Section 6 of the full version, formalizes this intuition.

Theorem 5. *There is a $k^{O(kt)}nm$ time polynomial space algorithm for Partitioned Subset Treewidth.*

We give a proof sketch of Theorem 5 in Section 2.3. In light of Theorem 5 it is natural to ask whether it is possible to reduce Subset Treewidth to Partitioned Subset Treewidth with t much smaller than k. While we do not know of a way to do this for exact algorithms, we obtain such a reduction for the variant of Subset Treewidth which is sufficient for $(1+\varepsilon)$-approximating treewidth. In particular, it is possible to show, using standard methods, that for every graph G, vertex set W and positive integer t there exists a partition of W into t sets $W_1, \ldots, W_t$ such that making all of $W_1, \ldots, W_t$ into cliques increases the treewidth of G by at most $3\lceil |W|/t \rceil$ (we give a proof of essentially this fact with parameter values relevant to our applications in Lemma 5.7). Thus, for instances of Subset Treewidth where $\tau(1+\varepsilon)+2 \leq |W| = O(\tau)$ (and τ is the treewidth of G), setting $t = O(1/\varepsilon)$ shows that there exists a partition of W into at most $t = O(1/\varepsilon)$ sets such that making all of $W_1, \ldots, W_t$ into cliques increases the treewidth of G to at most $\tau(1+\varepsilon)$. The proof of Theorem 3 (assuming Lemma 2.1) coupled with this partitioning argument yields a reduction from $(1+\varepsilon)$-approximating treewidth to solving Partitioned Subset Treewidth exactly with $t = O(1/\varepsilon)$.

Theorem 6. *Given an algorithm for Partitioned Subset Treewidth with running time $T(k,t) \cdot m^c$, we can construct an $(1+\varepsilon)$-approximation algorithm for treewidth with running time*

$$T(O(k), O(1/\varepsilon)) \cdot O((nk)^{c+1}) \cdot (1+1/\varepsilon)^{O(k)} + k^{O(1)}n^4 + 2^{O(k)}n^2.$$

Moreover, if the algorithm for Partitioned Subset Treewidth works in polynomial space, then the algorithm for treewidth works in polynomial space.

Putting Theorems 6 and 5 together implies Theorem 2. We now give a proof sketch of the main engine behind the proofs of Theorems 3 and 6, namely Lemma 2.1.

2.2 Proof Sketch of Lemma 2.1

The proof of Lemma 2.1 proceeds as follows. Given G, (T, bag), W and $(X, (T', \mathsf{bag}'))$ as in the premise of Lemma 2.1 we will construct a tree decomposition (T'', bag'') of G in polynomial time. We will always succeed in making (T'', bag''), but (T'', bag'') might not be "better" than (T, bag), in the sense that its width might be more than $|W|-1$, or it may have at least as many bags of size $|W|$ as (T, bag). We will show that in this case we can "improve" $(X, (T', \mathsf{bag}'))$ instead! In particular we will find a torso tree decomposition $(X^\star, (T^\star, \mathsf{bag}^\star))$ in G that covers W, with $|X^\star| < |X|$ and no larger width. Since G, (T, bag), W together with $(X^\star, (T^\star, \mathsf{bag}^\star))$ again satisfy the premise of Lemma 2.1 we can repeat the process with $(X^\star, (T^\star, \mathsf{bag}^\star))$ as the new $(X, (T', \mathsf{bag}'))$. Since $|X|$ cannot keep decreasing forever, eventually the tree decomposition (T'', bag'') will satisfy the conclusion of the Lemma. Thus it remains to sketch *(i)* how we construct (T'', bag'') and *(ii)* how to improve $(X, (T', \mathsf{bag}'))$ when (T'', bag'') is not better than (T, bag). We start by describing the construction of (T'', bag'').

Constructing (T'', bag''). We root (T, bag) at the node r of T corresponding to W. For each connected component C of $G \setminus X$ we make a tree decomposition (T_C, bag_C) of $G[N[C]]$ as follows. The decomposition tree T_C is simply a copy of T. For each node t of T let t_C be the copy of t in T_C. Thus r_C is the copy of the root r of T in T_C. For every node t_C of T_C we set $\mathsf{bag}_C(t_C) = \mathsf{bag}(t) \cap N[C]$ *plus all vertices of $N(C)$ that appear in at least one bag below t in T*. In other words (T_C, bag_C) is simply the restriction of the tree decomposition (T, bag) to the vertex set $N[C]$, but additionally for every vertex v of $N(C)$ we add v to all bags on the path from r_C to the subtree of T_C where this vertex already occurs. Observe that C is disjoint from X, which contains W, and therefore $\mathsf{bag}_C(r_C) = N(C)$.

Now the tree decomposition (T'', bag'') consists of a copy of (T', bag') together with the tree decomposition (T_C, bag_C) of the graph $G[N[C]]$ for every connected component C of $G \setminus X$. For each component C of $G \setminus X$ we have that $N(C)$ is a clique in $\mathtt{torso}_G(X)$, and that therefore (see e.g. [22, Chapter 6]) at least one bag of (T', bag') contains $N(C)$. We add an edge from r_C to this bag. See Figure 1 for a visualization of this construction.

It is quite easy to verify that (T'', bag'') is indeed a tree decomposition of G, and that it can be constructed in polynomial time. However it is not at all obvious that it should be better than (T, bag) - it could even be worse, because we added vertices to the bags $\mathsf{bag}_C(t_C)$ that were not there in the corresponding bag $\mathsf{bag}(t)$ of (T, bag). Thus, all that remains is to show how to improve $(X, (T', \mathsf{bag}'))$ when (T'', bag'') is not better than (T, bag). Working towards this goal we first state the main tool that we will use to improve $(X, (T', \mathsf{bag}'))$.

A pulling lemma. The following lemma is very useful to improve $(X, (T', \mathsf{bag}'))$, and also in many other arguments in this paper. We call it the "pulling lemma" because in the proof the separator S will be "pulled" along disjoint paths into a bag of the (torso) tree decomposition. To state the pulling lemma we need to define the notions of separations and linkedness. A separation in a graph G is a partition of $V(G)$ into three parts (A, S, B) such that no edge of G has one endpoint in A and the other in B. We call S the *separator* of the separation (A, S, B) and $|S|$ is the *order* of the separation. A

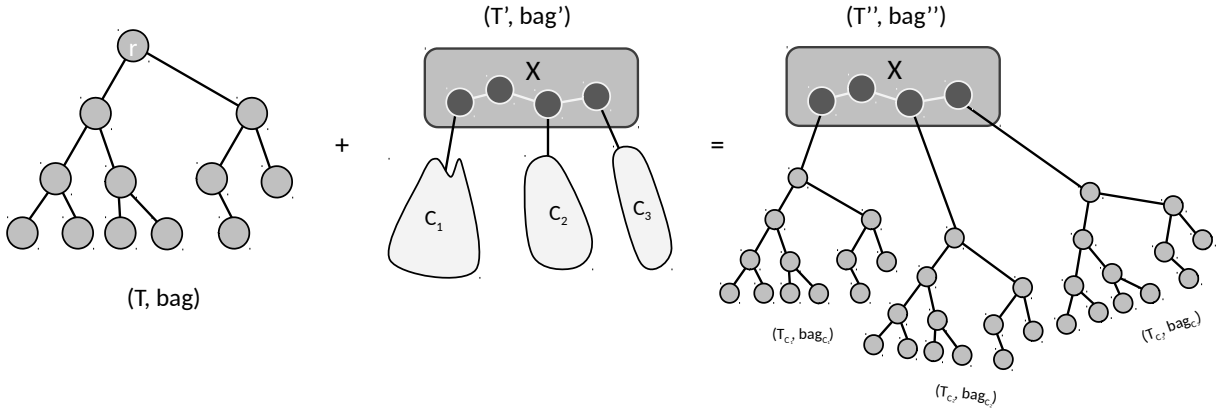

Figure 1: Construction of (T'', bag'').

vertex set A is *linked* into a vertex set B in G if there exist $|A|$ vertex disjoint paths with one endpoint in A and one in B (paths on a single vertex that start and end in $A \cap B$ count).

Lemma 2.2 (Pulling lemma, simplified variant)**.** *Let G be a graph, k be an integer, $(X, (T, \mathsf{bag}))$ be a torso tree decomposition in G of width k, $Z \subseteq X$ be a vertex set in G that is a subset of at least one bag of (T, bag), and (A, S, B) be a separation of G so that $Z \subseteq S \cup B$ and S is linked into Z. There exists a torso tree decomposition $((X \cap A) \cup S, (T', \mathsf{bag}'))$ of width at most k. Moreover, S is a bag of (T', bag'). Furthermore, when G, $(X, (T, \mathsf{bag}))$, (A, S, B), and Z are given as inputs, the torso tree decomposition $((X \cap A) \cup S, (T', \mathsf{bag}'))$ can be constructed in polynomial time.*

The formal version of Lemma 2.2 is proved as Lemma 4.1 in Section 4. Analogous lemmas with similar proofs have been used in the context of tree decompositions, for example by [6, 18], and so we do not give a sketch of Lemma 2.2 in this overview.

Improving X when (T'', bag'') is not better than (T, bag). We now return our focus to the setting where we started with (T, bag), a maximum size bag W, and $(X, (T', \mathsf{bag}'))$, and we used them to make the new tree decomposition (T'', bag'') of G. If (T'', bag'') is better than (T, bag) (in the sense of having strictly fewer bags of size $|W|$ and no larger bags) then (T, bag) already satisfies the conclusion of Lemma 2.1. Hence, assume that (T'', bag'') is not better than (T, bag). Our goal is to find a component C of $G \setminus X$ and a separation (P', S', Q') such that $N(C) \subseteq S' \cup Q'$, $W \subseteq S' \cup P'$, S' is linked to $N(C)$, and $|S'| < |N(C)|$. Then $(X, (T', \mathsf{bag}'))$, $Z = N(C)$ and the separation (P', S', Q') will satisfy the premise of Lemma 2.2. Setting $X^\star = (X \cap P') \cup S'$, Lemma 2.2 implies that there exists a torso tree decomposition $(X^\star, (T^\star, \mathsf{bag}^\star))$ of width at most that of $(X, (T', \mathsf{bag}'))$. Moreover, because $N(C) \subseteq X$ and $N(C)$ is disjoint from P', it holds that

$$|X^\star| = |X| - |X \setminus P'| + |S'| \leq |X| - |N(C)| + |S'| < |X|.$$

Since $W \subseteq S' \cup P'$ and $W \subseteq X$, we have that $X^\star$ covers W, and we have found our improved torso tree decomposition $(X^\star, (T^\star, \mathsf{bag}^\star))$ that covers W. We now show how such a component C and separation (P', S', Q') can be identified.

Note that (T', bag') has no bags of size at least $|W|$. Therefore every bag of size at least $|W|$ in (T'', bag'') appears in (T_C, bag_C) for some component C of $G \setminus X$. Observe also that for the root r of T and every component C of $G \setminus X$ we have $|\mathsf{bag}_C(r_C)| < |W| = |\mathsf{bag}(r)|$. Indeed, for every copy r_C of the root we have that C is disjoint from X, and that therefore $\mathsf{bag}_C(r_C) = N(C)$. But $N(C)$ is a subset of some bag of (T', bag'), all of which have size at most $|W| - 1$. Therefore, since (T'', bag'') is not better than (T, bag) at least one of the two following statements must hold. *(i)* There exists a node t in $V(T)$ and component C of $G \setminus X$ such that $|\mathsf{bag}_C(t_C)| > |\mathsf{bag}(t)|$, or *(ii)* There exists a node t in $V(T)$ and two distinct components C_1, C_2 of $G \setminus X$ such that $|\mathsf{bag}_{C_1}(t_{C_1})| = |\mathsf{bag}_{C_2}(t_{C_2})| = |\mathsf{bag}(t)| = |W|$.

We show how to improve X in the first case. Let t be a node in $V(T)$ and C be a component C of $G \setminus X$ such that $|\mathsf{bag}_C(t_C)| > |\mathsf{bag}(t)|$. We consider two separations of G: $(C, N(C), R)$ (where $R = V(G) \setminus N[C]$) is the "rest" and (U, B, L) where $B = \mathsf{bag}(t)$, L (the "lower" set) is the set of all non-B vertices appearing in bags of T below t, and U (the "upper") set is defined as $U = V(G) \setminus (B \cup L)$ (consult Figure 2 for a visualization of these separations and how they are used in the remainder of the argument).

We have that W is a subset of X and therefore disjoint from C. Similarly, all vertices of W appear in at least one bag above t (namely r), and therefore W is disjoint from L. It follows that S defined as

$$S = (N(C) \setminus L) \cup (B \cap R)$$

separates $N(C)$ from W. Furthermore, by choice of t and $B = \mathsf{bag}(t)$ we have that

$$\begin{aligned} |B \cap N[C]| + |B \setminus N[C]| &= |B| \\ &< |\mathsf{bag}_C(t_C)| = |B \cap N[C]| + |N(C) \cap L|. \end{aligned}$$

Here $|\mathsf{bag}_C(t_C)| = |B \cap N[C]| + |N(C) \cap L|$ follows from the construction of the function bag_C. From the above equation we have that $|B \cap R| = |B \setminus N[C]| < |N(C) \cap L|$. But then we have that

$$|N(C)| = |N(C) \setminus L| + |N(C) \cap L| > |N(C) \setminus L| + |B \cap R| \geq |S|.$$

Since S separates $N(C)$ from W, there exists a separation (P, S, Q) with $W \subseteq S \cup P$ and $N(C) \subseteq S \cup Q$, and $|S| < |N(C)|$. Let (P', S', Q') be a separation with $W \subseteq S' \cup P'$ and $N(C) \subseteq S' \cup Q'$ and $|S'|$

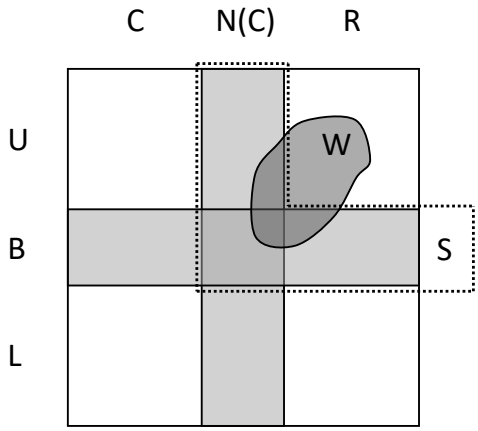

Figure 2: The set S separates $N(C)$ from W and $|S| < |N(C)|$ because $|B \cap R| < |N(C) \cap L|$.

being of minimum size. Then $|S'| \leq |S| < |N(C)|$ and (by Menger's Theorem) the set S' is linked into $N(C)$. Now the component C and separation (P', S', Q') satisfy all of the properties necessary to use Lemma 2.2 to improve $(X, (T, \mathsf{bag}))$. This concludes case *(i)* (that there exists a node t in $V(T)$ and component C of $G \setminus X$ such that $|\mathsf{bag}_C(t_C)| > |\mathsf{bag}(t)|$).

The second case (when there exists a node t in $V(T)$ and two distinct components C_1, C_2 of $G \setminus X$ with $|\mathsf{bag}_{C_1}(t_{C_1})| = |\mathsf{bag}_{C_2}(t_{C_2})| = |\mathsf{bag}(t)| = |W|$) is handled in an analogous, but even more technical way. In particular in this case we are not able to necessarily obtain an $X^\star$ with $|X^\star| < |X|$, but instead we obtain an $X^\star$ with $|X^\star| = |X|$ and a lower value of a carefully chosen potential function. This concludes the proof sketch of Lemma 2.1.

2.3 Overview of Theorems 4 and 5

We now overview our algorithms for Subset Treewidth and Partitioned Subset Treewidth. Recall that every instance of Subset Treewidth is also an instance of Partitioned Subset Treewidth, so we will only work on instances of Partitioned Subset Treewidth. This will be useful also in the algorithm for Subset Treewidth, since the recursive subproblems turn out to naturally correspond to Partitioned Subset Treewidth.

We denote an instance of Partitioned Subset Treewidth by $\mathcal{I} = (G, \{W_1, \ldots, W_t\}, k)$. We call the cliques $W_1, \ldots, W_t$ the *terminal cliques* of the instance. We say that a torso tree decomposition $(X, (T, \mathsf{bag}))$ in G is a solution of $\mathcal{I}$ if $(X, (T, \mathsf{bag}))$ covers $\bigcup_{i=1}^{t} W_i$ and has width at most k. Here we do not anymore enforce that $|\bigcup_{i=1}^{t} W_i| \leq k+2$, and it will in fact grow larger in the recursive subproblems (but k or t will not increase). Both of our algorithms will either find a solution or conclude that no solution exists. In particular, we do not use the freedom in the definitions of the problems that we could also determine that the treewidth of G is more than k without determining that no solution exists.

We will first sketch a $k^{O(k)}n^{O(1)}$ time algorithm for Partitioned Subset Treewidth in the case when there are only two terminal cliques W_1 and W_2. This algorithm showcases the most important concepts behind both the $k^{O(kt)}nm$ time algorithm of Theorem 5 and the $2^{O(k^2)}nm$ time algorithm of Theorem 4, and in fact generalizing this to the $k^{O(kt)}nm$ algorithm does not require substantial new ideas but is rather a technical step.

Reduction rule. Let W_1,W_2 be the two terminal cliques and S be a minimum size (W_1, W_2)-separator, and (A, S, B) the corresponding separation with $W_1 \subseteq A \cup S$ and $W_2 \subseteq B \cup S$. We will argue that we can make S into a new terminal clique and recursively solve the problem on the graphs $G[A \cup S]$ and $G[B \cup S]$. More formally, we denote by $G \otimes S$ the graph obtained from G by making S a clique, and then denote by $\mathcal{I} \triangleleft (A, S)$ the instance $(G[A \cup S] \otimes S, \{W_1, S\}, k)$ and by $\mathcal{I} \triangleleft (B, S)$ the instance $(G[B \cup S] \otimes S, \{W_2, S\}, k)$. We argue that there exists a solution of $\mathcal{I}$ if and only if there exists solutions of both $\mathcal{I} \triangleleft (A, S)$ and $\mathcal{I} \triangleleft (B, S)$.

Observe that because both $\mathcal{I} \triangleleft (A, S)$ and $\mathcal{I} \triangleleft (B, S)$ contain the separator S as a terminal clique but their graphs are disjoint otherwise, any solution of $\mathcal{I} \triangleleft (A, S)$ can be combined with any solution of $\mathcal{I} \triangleleft (B, S)$ into a solution of $\mathcal{I}$ by simply connecting the tree decompositions by an edge between bags containing S. To argue that if there exists a solution of $\mathcal{I}$ then there exists solutions of both $\mathcal{I} \triangleleft (A, S)$ and $\mathcal{I} \triangleleft (B, S)$, we apply the pulling lemma (Lemma 2.2). Because S is a minimum size (W_1, W_2)-separator, by Menger's theorem S is linked into W_1 and into W_2. Therefore, in order to show that a solution of $\mathcal{I} \triangleleft (A, S)$ exists, we consider a hypothetical solution $(X, (T, \mathsf{bag}))$ of $\mathcal{I}$, and apply the pulling lemma with the separation (A, S, B) and $Z = W_2$ as the subset of a bag with $Z \subseteq S \cup B$ into which S is linked. This constructs a torso tree decomposition $((X \cap A) \cup S, (T', \mathsf{bag}'))$ of width at most k where S is a bag, which can be observed to be a torso tree decomposition also in $G[A \cup S] \otimes S$ because S is a bag of (T', bag'), and to cover $W_1 \cup S$ because $W_1 \subseteq A \cup S$ and $W_1 \subseteq X$, and therefore is a solution of $\mathcal{I} \triangleleft (A, S)$. The existence of a solution of $\mathcal{I} \triangleleft (B, S)$ is proven in a symmetric way.

Observe that this reduction rule makes progress as long as $S \neq W_1$ and $S \neq W_2$, and thus we apply the rule as long as there exists any such minimum size (W_1, W_2)-separator S. Motivated by this, we say that W_1 is *strictly linked* into W_2 if W_1 is linked into W_2 and the only minimum size (W_1, W_2)-separators are W_1 and perhaps W_2 (if $|W_2| = |W_1|$).

Leaf pushing. Assume now that we cannot make any more progress by the reduction rule, and let $|W_1| \leq |W_2|$, implying that W_1 is strictly linked into W_2. Our goal is to now make progress by increasing the size of W_1. We observe that for any solution $(X, (T, \mathsf{bag}))$ that minimizes $|X|$, it holds that if l is a leaf node of T and p is the parent of l, then $\mathsf{bag}(l) \setminus \mathsf{bag}(p) \subseteq W_1 \cup W_2$. Furthermore, we can assume that $\mathsf{bag}(p) = \mathsf{bag}(l) \setminus \{w\}$, where w is a "forget-vertex" of l, and therefore $\mathsf{bag}(l) \setminus \mathsf{bag}(p) \subseteq W_1$ or $\mathsf{bag}(l) \setminus \mathsf{bag}(p) \subseteq W_2$. Then, observe that if $\mathsf{bag}(l) \setminus \mathsf{bag}(p)$ intersects W_i, it must hold that $W_i \subseteq \mathsf{bag}(l)$ because W_i is a clique. Therefore, (T, bag) either contains a bag that contains both W_1 and W_2, in which case $|W_1 \cup W_2| \leq k+1$ and there is a trivial single-bag solution, or (T, bag) has exactly two leaves and for one of them it holds that $W_1 \subseteq \mathsf{bag}(l)$ and $\mathsf{bag}(l) \setminus \mathsf{bag}(p) \subseteq W_1 \setminus W_2$.

Now, our goal will be, informally, to increase the size of W_1 by guessing a vertex in $\mathsf{bag}(l) \setminus W_1$ and adding it to W_1. We let w be

the forget-vertex of l, and observe that the parent bag $\mathsf{bag}(p) = \mathsf{bag}(l) \setminus \{w\}$ is a (W_1, W_2)-separator. This shows that $\mathsf{bag}(l) \setminus W_1$ must be non-empty, because otherwise $\mathsf{bag}(p)$ would be a (W_1, W_2)-separator of size $|W_1| - 1$, contradicting that W_1 is linked into W_2. Denote $G' = G \setminus (W_1 \setminus \{w\})$, and observe that in the graph G' the set $\mathsf{bag}(l) \setminus W_1 = \mathsf{bag}(p) \setminus W_1$ is a $(\{w\}, W_2 \setminus W_1)$-separator. We will then show that the subset $\mathsf{bag}(l) \setminus W_1$ of $\mathsf{bag}(l)$ can be replaced by an important $(\{w\}, W_2 \setminus W_1)$-separator (see Section 4.1 of the full version or [22, Chapter 8] for definitions of important separators). In particular, we will argue that there is an important $(\{w\}, W_2 \setminus W_1)$-separator $S \neq \{w\}$ in the graph G' so that there exists a solution containing a bag $W_1 \cup S$.

Let S be an important $(\{w\}, W_2 \setminus W_1)$-separator in the graph G' so that it dominates $\mathsf{bag}(l) \setminus W_1$ and minimizes $|S|$ among all such important separators. Denote the separation corresponding to S by $(A, S, B) = (R_{G'}(\{w\}, S), S, V(G) \setminus (S \cup R_{G'}(\{w\}, S)))$, where $R_{G'}(\{w\}, S)$ denotes the vertices reachable from $\{w\}$ in the graph $G' \setminus S$. It can be shown that S is linked into $(A \cup S) \cap (\mathsf{bag}(l) \setminus W_1)$. Then, by adding $W_1 \setminus \{w\}$ back to the graph and to the separation, we get that $(A, S \cup W_1 \setminus \{w\}, B)$ is a separation of G and $S \cup W_1 \setminus \{w\}$ is linked into $(A \cup S \cup W_1 \setminus \{w\}) \cap \mathsf{bag}(l)$ (the vertices in $W_1 \setminus \{w\}$ are linked by trivial one-vertex paths). We then apply the pulling lemma (Lemma 2.2) with the hypothetical solution $(X, (T, \mathsf{bag}))$, the separation $(B, S \cup W_1 \setminus \{w\}, A)$, and the subset of a bag $Z = (A \cup S \cup W_1 \setminus \{w\}) \cap \mathsf{bag}(l)$, to argue that there exists a torso tree decomposition $((X \cap B) \cup S \cup W_1 \setminus \{w\}, (T', \mathsf{bag}'))$ of width at most k, containing a bag $S \cup W_1 \setminus \{w\}$. As $|S| \leq |\mathsf{bag}(l) \setminus W_1|$, this can be turned into a solution of $\mathcal{I}$ by inserting w into the bag $S \cup W_1 \setminus \{w\}$. Therefore there exists a solution of $\mathcal{I}$ with a bag $W_1 \cup S$, and in particular it is safe to replace the terminal clique W_1 by $W_1 \cup S$, also replacing G by $G \otimes (W_1 \cup S)$.

Now, we are able to increase the size of W_1 by guessing the forget-vertex $w \in W_1$ and an important separator S and branching to $(G \otimes (W_1 \cup S), \{W_1 \cup S, W_2\}, k)$. However, by applying the reduction rule we might immediately lose most of the progress by finding a $(W_1 \cup S, W_2)$-separator S' of size $|S'| < |W_1 \cup S|$ and ending up with an instance with terminal cliques $\{S', W_2\}$. Nevertheless, we can ensure that such S' must have size $|S'| > |W_1|$ by using the facts that W_1 is strictly linked into W_2 and the way S was selected. In particular, in the end, after applying the reduction rule possibly several times, we can guarantee that if initially $|W_1| = |W_2|$, then each resulting instance has terminal cliques of sizes at least $|W_1| + 1$ and $|W_2|$, and if initially $|W_1| < |W_2|$, then each resulting instance has terminal cliques of sizes at least $|W_1| + 1$ and $|W_1| + 1$. Therefore, if we consider $\min(|W_1|, |W_2|) + \min(\min(|W_1|, |W_2|) + 1, \max(|W_1|, |W_2|))$ as our measure of progress, we are guaranteed to increase it by one by the branching.

As the sizes of terminal cliques are bounded by $k+1$, it is possible to increase this measure by at most $2k$ times. Then, as the number of important separators of size at most k is bounded by 4^k [20], this results in a branching tree of degree $k4^k$ and depth $2k$, resulting in a $(k4^k)^{2k} n^{O(1)} = 2^{O(k^2)} n^{O(1)}$ time algorithm. To improve this to $k^{O(k)} n^{O(1)}$ time, we observe that in order to make progress, it is sufficient to guess only one vertex of the important separator S and add it to W_1, instead of guessing the whole important separator S. To this end, we prove an "important separator hitting set lemma" that gives a set of size k that intersects all important separators of size at most k, and therefore allows to guess one vertex in an important separator of size at most k by a branching degree of k instead of 4^k, resulting in a $(k^2)^{2k} n^{O(1)} = k^{O(k)} n^{O(1)}$ time algorithm.

More than two terminal cliques. Generalizing the $k^{O(k)} n^{O(1)}$ time algorithm for two terminal cliques into the $k^{O(kt)} n^{O(1)}$ algorithm for t terminal cliques of Theorem 5 does not require major new ideas, but requires several technical considerations. In the algorithm for t terminal cliques, we will in addition to the leaf pushing branching do branching on merging two different terminal cliques into one, which should be done whenever we guess that there exists a solution where the two terminal cliques are in a same bag. The "real" definition of the measure of the instance will also be more involved, in particular, instead of depending on the sizes of terminal cliques, the measure depends on a notion of "flow potential" of a terminal clique. The flow potential has a technical definition, but for all terminal cliques W_i except for a uniquely largest one it will be equal to the flow from W_i into the union of the other terminal cliques. The measure of a uniquely largest terminal clique must be special to encode that we make progress, for example, in the case when there are two terminal cliques W_1 and W_2 with $|W_1| = |W_2|$ and after branching we end up with two terminal cliques of sizes $|W_1| + 1$ and $|W_2|$. The measure will also take into account the number of terminal cliques, in particular, it will "encode" that decreasing the number of terminal cliques with the expense of making the flow potential of one terminal clique worse still means making overall progress.

The $2^{O(k^2)} n^{O(1)}$ time algorithm. The $2^{O(k^2)} n^{O(1)}$ time algorithm for Subset Treewidth of Theorem 4 also uses the same reduction rule and leaf pushing arguments. In particular, even though the problem is originally Subset Treewidth, applications of the reduction rule and leaf pushing will naturally turn the problem into Partitioned Subset Treewidth.

For this algorithm, the main measure of progress will be a parameter q that states that there are no solutions that contain "internal separations" of order $< q$. Here, an internal separation of a solution $(X, (T, \mathsf{bag}))$ means a separation (A, S, B) so that S is a subset of some bag of (T, bag), and the terminal cliques intersect both A and B. The goal will be to increase q, by first pushing two terminal cliques to be of size at least $\geq q$ by using a version of leaf pushing that guesses the whole important separator instead of only one vertex, and then guessing how a hypothetical internal separation of order q would split the terminal cliques and breaking the instance by an important separator of size q pushed towards the side with two terminal cliques of size $\geq q$. We will also argue about internal separations that contain only a small number of "original" terminal vertices behind them, in particular, we will use an observation that if a solution has an internal separation (A, S, B) so that at most $k + 1 - |S|$ original terminal vertices are "behind" terminal cliques intersecting A, then the A-side of the solution can be replaced by just a single bag containing S and the original terminal vertices behind it.

3 PRELIMINARIES

We present definitions and preliminary results.

For a positive integer n we denote $[n] = \{1, 2, \ldots, n\}$ and for two integers a, b with $a \leq b$ we denote $[a, b] = \{a, a+1, \ldots, b\}$.

3.1 Graphs

We denote the set of vertices of a graph G by $V(G)$ and the set of edges by $E(G)$. When the graph G is clear from the context, we use $n = |V(G)|$ and $m = |V(G)| + |E(G)|$. For a vertex $v \in V(G)$ we denote its neighborhood in G by $N_G(v)$ and closed neighborhood by $N_G[v] = N_G(v) \cup \{v\}$. For a set of vertices $S \subseteq V(G)$ their neighborhood is $N_G(S) = \bigcup_{v \in S} N(v) \setminus S$ and the closed neighborhood $N_G[S] = N_G(S) \cup S$. We drop the subscript if the graph is clear from the context. We denote the subgraph of G induced by $S \subseteq V(G)$ by $G[S]$, and we also use the notation $G \setminus S = G[V(G) \setminus S]$. We denote by $G \otimes S$ the graph obtained from G by making S a clique.

A tripartition (A, S, B) of $V(G)$ (with possibly empty parts) is a *separation* of G if there are no edges between A and B. The *order* of the separation is $|S|$. A separation of G is a *strict separation* if both A and B are non-empty. For two sets $X, Y \subseteq V(G)$, an (X, Y)-separator is a set S so that in the graph $G \setminus S$ there are no paths from $X \setminus S$ to $Y \setminus S$. An (X, Y)-separator S is a minimal (X, Y)-separator if no proper subset of S is an (X, Y)-separator. Note that S is an (X, Y)-separator if and only if there exists a separation (A, S, B) of G with $X \subseteq A \cup S$ and $Y \subseteq B \cup S$.

For two sets of vertices $X, S \subseteq V(G)$, we denote by $R_G(X, S)$ the set of vertices in $G \setminus S$ reachable from $X \setminus S$. We define $R^N_G(X, S) = (X \cap S) \cup N(R_G(X, S)) \subseteq S$ to denote the subset of S that can be seen from X. Note that if S is an (X, Y)-separator then $R^N_G(X, S)$ is also an (X, Y)-separator and $R_G(X, R^N_G(X, S)) = R_G(X, S)$. It follows that if S is a minimal (X, Y)-separator, then $S = R^N_G(X, S)$.

For two sets of vertices $X, Y \subseteq V(G)$, we denote by $\texttt{flow}_G(X, Y)$ the maximum number of vertex-disjoint paths in G starting in X and ending in Y. We may omit the subscript if the graph is clear from the context. By Menger's theorem, $\texttt{flow}(X, Y)$ is equal to the size of a minimum size (X, Y)-separator.

We say that a set $X \subseteq V(G)$ is *linked* into a set $Y \subseteq V(G)$ if $\texttt{flow}(X, Y) = |X|$. Note that here the definition of linked is asymmetric, in particular, the fact that X is linked into Y does not imply that Y is linked into X. We say that X is *strictly linked* into Y if it is linked into Y and for all (X, Y)-separators S of size $|S| = |X|$ it holds that $S = X$ or $S = Y$.

3.2 Tree Decompositions

A tree decomposition of a graph G is a pair (T, bag), where T is a tree and bag is a function $\mathsf{bag} : V(T) \to 2^{V(G)}$ that satisfies

(1) $V(G) = \bigcup_{t \in V(T)} \mathsf{bag}(t)$,
(2) for every $uv \in E(G)$, there exists $t \in V(T)$ with $\{u, v\} \subseteq \mathsf{bag}(t)$, and
(3) for every $v \in V(G)$, the set $\{t \in V(T) \mid v \in \mathsf{bag}(t)\}$ forms a connected subtree of T.

We will call Item 1 of the definition the *vertex condition*, Item 2 the *edge condition*, and Item 3 the *connectedness condition*. The width of a tree decomposition (T, bag) is $\max_{t \in V(T)} |\mathsf{bag}(t)| - 1$ and the treewidth of a graph is the minimum width of a tree decomposition of it. We usually call the vertices of the tree T *nodes* to distinguish them from the vertices of the graph G.

We will need the following standard utility lemma that transforms a tree decomposition into a no worse tree decomposition with at most n nodes.

Lemma 3.1. *Given a tree decomposition (T, bag) of G of width k that has h bags of size $k+1$, we can in time $k^{O(1)}|V(T)|$ construct a tree decomposition of G of width k that has at most h bags of size $k+1$ and has at most n nodes.*

Proof. As long as there exists an edge $uv \in E(T)$ with $\mathsf{bag}(u) \subseteq \mathsf{bag}(v)$, we contract uv and let the bag of the resulting node be $\mathsf{bag}(v)$. This can be implemented in $k^{O(1)}|V(T)|$ time by depth-first search, and clearly does not increase the width or the number of bags of size $k+1$. This results in a tree decomposition with at most n nodes (see e.g. [31, Chapter 14.2]). □

We sometimes view a tree decomposition (T, bag) as rooted on some specific node $r \in V(T)$. In this setting we use standard rooted tree terminology, i.e., $v \in V(T)$ is an *ancestor* of $u \in V(T)$ if it is on the unique path from u to r and a *strict ancestor* if also $v \neq u$, and conversely u is a (strict) *descendant* of v. We say that a node $t \in V(T)$ is the forget-node of a vertex $v \in V(G)$ if $v \in \mathsf{bag}(t)$ and either $t = r$ or for the parent p of t it holds that $v \notin \mathsf{bag}(p)$. Note that every $v \in V(G)$ has a unique forget-node.

3.3 Torso Tree Decompositions

Let G be a graph and $X \subseteq V(G)$. The graph $\texttt{torso}_G(X)$ has set of vertices $V(\texttt{torso}_G(X)) = X$ and has $uv \in E(\texttt{torso}_G(X))$ if $u, v \in X$ and there is a path from u to v whose internal vertices are in $V(G) \setminus X$. In particular, note that $E(\texttt{torso}_G(X)) \supseteq E(G[X])$. An equivalent definition of $\texttt{torso}_G(X)$ is that it is the graph obtained from $G[X]$ by making $N_G(C)$ a clique for every connected component C of $G \setminus X$.

We will need a following lemma about the interplay of the torso operation and induced subgraphs.

Lemma 3.2. *Let X, Y be subsets of $V(G)$. Then $E(\texttt{torso}_{G[Y]}(X \cap Y)) \subseteq E(\texttt{torso}_G(X))$.*

Proof. If $uv \in E(\texttt{torso}_{G[Y]}(X \cap Y))$, then there is a path from u to v in $G[Y]$ with intermediate vertices in $Y \setminus X$. This path exists also in G, implying that $uv \in E(\texttt{torso}_G(X))$. □

A *torso tree decomposition* in a graph G is a pair $(X, (T, \mathsf{bag}))$, where $X \subseteq V(G)$ and (T, bag) is a tree decomposition of $\texttt{torso}_G(X)$. For a set $W \subseteq V(G)$, we say that $(X, (T, \mathsf{bag}))$ covers W if $W \subseteq X$.

We observe the following equivalent viewpoint of torso tree decompositions that might be useful for intuition about them.

Observation 3.3. *There exists a torso tree decomposition $(X, (T, \mathsf{bag}))$ in G if and only if there exists a tree decomposition of G whose non-leaf nodes induce the tree decomposition (T, bag).*

For tree decompositions it holds that for any connected induced subgraph $G[Y]$, the set of bags intersecting Y forms a connected subtree of the decomposition (see e.g. [31, Chapter 14.1]). We will use a corresponding property of torso tree decompositions.

Lemma 3.4. *Let $(X, (T, \mathsf{bag}))$ be a torso tree decomposition in G, and let $Y \subseteq V(G)$ so that $G[Y]$ is connected. The nodes $\{t \in V(T) \mid \mathsf{bag}(t) \cap Y \neq \emptyset\}$ induce a (possibly empty) connected subtree of T.*

Proof. By the definition of $\texttt{torso}_G(X)$, any $u-v$-path in $G[Y]$ with $u, v \in X$ can be mapped into an $u-v$-path in $\texttt{torso}_G(X)[Y \cap X]$, and therefore $\texttt{torso}_G(X)[Y \cap X]$ is connected, and therefore the lemma follows from the corresponding property of tree decompositions. □

Alternatively, Lemma 3.4 could be proven by using Lemma 3.3 and the property of tree decompositions. Lemma 3.4 implies that if there are nodes $s, x, y \in V(T)$ so that $\{s\}$ is an $(\{x\}, \{y\})$-separator in T, then $\texttt{bag}(s)$ is a $(\texttt{bag}(x), \texttt{bag}(y))$-separator in G. This implication is proven by letting Y to be any $\texttt{bag}(x) - \texttt{bag}(y)$-path and observing that by Lemma 3.4, Y must now intersect $\texttt{bag}(s)$.

4 PULLING LEMMA

We prove a lemma that will be used throughout Section 5, and the Sections 6 and 7 of the full version to argue that a separator S can be incorporated as a bag of a torso tree decomposition if it satisfies certain properties. We call it the "pulling lemma" because the separator S will be "pulled" along disjoint paths into a bag of the tree decomposition. Lemmas analogous to this have been used in the context of tree decompositions for example by [6, 18].

Lemma 4.1 (Pulling lemma)**.** *Let G be a graph and $(X, (T, \texttt{bag}))$ a torso tree decomposition in G. Let (A, S, B) be a separation of G so that there exists a node $r \in V(T)$ so that S is linked into $\texttt{bag}(r) \cap (S \cup B)$. There exists a torso tree decomposition $((X \cap A) \cup S, (T', \texttt{bag}'))$ so that*

(1) $T' = T$
(2) for all $t \in V(T)$, $|\texttt{bag}'(t)| \leq |\texttt{bag}(t)|$, and
(3) $S \subseteq \texttt{bag}'(r)$.

Moreover, when G, $(X, (T, \texttt{bag}))$, (A, S, B), and r are given as inputs, the torso tree decomposition $((X \cap A) \cup S, (T', \texttt{bag}'))$ can be constructed in $k^{O(1)}(|V(T)| + m)$ time, where k is the width of $(X, (T, \texttt{bag}))$.

Proof. Index the vertices of S by $S = \{s_1, s_2, \ldots, s_{|S|}\}$. Because S is linked into $\texttt{bag}(r) \cap (S \cup B)$, there are vertex-disjoint paths $P_1, \ldots, P_{|S|}$, so that for each $i \in [|S|]$, P_i is a path from s_i to $\texttt{bag}(r) \cap (S \cup B)$, and all vertices of P_i are contained in $S \cup B$.

To construct $(T', \texttt{bag}')$, we set $T' = T$, and for each $t \in V(T)$ we set

$$\texttt{bag}'(t) = (\texttt{bag}(t) \setminus (S \cup B)) \cup \{s_i \mid P_i \cap \texttt{bag}(t) \neq \emptyset\}.$$

We have that $|\texttt{bag}'(t)| \leq |\texttt{bag}(t)|$, because for each inserted vertex s_i we removed a vertex in P_i (note that the inserted vertex and the removed vertex could both be the same vertex s_i). By definition every P_i intersects $\texttt{bag}(r)$, and thus $S \subseteq \texttt{bag}'(r)$. Denote $X' = (X \cap A) \cup S$. It remains to show that $(T', \texttt{bag}')$ is a tree decomposition of $\texttt{torso}(X')$.

First, the tree decomposition $(T', \texttt{bag}')$ satisfies the vertex condition because no vertices in $X \cap A$ were removed, and as argued before $S \subseteq \texttt{bag}'(r)$. Second, $(T', \texttt{bag}')$ satisfies the connectedness condition because the occurrences of vertices in $X \cap A$ were not altered, and by Lemma 3.4 the sets $\{t \mid P_i \cap \texttt{bag}(t) \neq \emptyset\}$ induce connected subtrees of T.

For the edge condition, consider an edge $uv \in E(\texttt{torso}(X'))$. There is a path between u and v whose intermediate vertices are contained in $V(G) \setminus X'$. If there would be an intermediate vertex in B, then $u, v \in S$, implying $\{u, v\} \subseteq \texttt{bag}'(r)$, so it remains to consider the cases where there are no intermediate vertices or all intermediate vertices are in $A \setminus X' = A \setminus X$. It follows that if in this case $u, v \in X$, then $uv \in E(\texttt{torso}(X))$, so the edge condition of $(T', \texttt{bag}')$ in this case holds by the edge condition of $(T, \texttt{bag})$. Also if $u, v \in S$, then again $\{u, v\} \subseteq \texttt{bag}'(r)$, so the remaining case is $uv = s_i v$, where $s_i \in S \setminus X$ and $v \in X \setminus S$. Now, s_i and the intermediate vertices on the path between s_i and v are in a connected component C of $G \setminus X$. Because $v \in X$ and $\texttt{bag}(r) \subseteq X$, this implies that $N(C)$ contains both v and at least one vertex on the path P_i, and therefore as $N(C)$ is a clique in $\texttt{torso}(X)$ there is a node $t \in V(T)$ with $N(C) \subseteq \texttt{bag}(t)$ and it will hold that $\{s_i, v\} \subseteq \texttt{bag}'(t)$.

Because $(T, \texttt{bag})$ has width k and $|S| \leq k + 1$, the construction clearly can be implemented in $k^{O(1)}(|V(T)| + m)$ time. □

Note that the condition $|\texttt{bag}'(t)| \leq |\texttt{bag}(t)|$ implies that the width of $(T', \texttt{bag}')$ is at most the width of $(T, \texttt{bag})$.

5 COMPUTING TREEWIDTH BY SUBSET TREEWIDTH

In this section we show that in order to improve a tree decomposition, it is sufficient to solve Subset Treewidth. In particular, we prove Theorems 3 and 6.

5.1 Improving a Tree Decomposition

We will define a weighted version of linkedness. For a weight function $d : V(G) \to \mathbb{Z}$ and a set $S \subseteq V(G)$, we denote $d(S) = \sum_{v \in S} d(v)$.

Definition 5.1 (d-linked)**.** *Let G be a graph, $A, B \subseteq V(G)$, and $d : V(G) \to \mathbb{Z}$ a weight function. The set A is d-linked into B if for any (A, B)-separator S it holds either that $|S| > |A|$, or that $|S| = |A|$ and $d(S) \geq d(A)$.*

Note that if A is d-linked into B then A is linked into B. We say that an (A, B)-separator S with $|S| < |A|$, or with $|S| = |A|$ and $d(S) < d(A)$ *witnesses* that A is not d-linked into B. Then, we say that a torso tree decomposition $(X, (T, \texttt{bag}))$ is d-linked into a set of vertices $W \subseteq V(G)$ if for every node $t \in V(T)$ it holds that $\texttt{bag}(t)$ is d-linked into W. We say that a pair (t, S), where $t \in V(T)$ and S is a $(\texttt{bag}(t), W)$-separator witnessing that $\texttt{bag}(t)$ is not d-linked into W witnesses that $(X, (T, \texttt{bag}))$ is not d-linked into W.

Our goal is to show that any torso tree decomposition that covers W can be made to be d-linked into W. In particular, we will show that if $(X, (T, \texttt{bag}))$ is a torso tree decomposition that covers W, then given a pair (t, S) that witnesses that $(X, (T, \texttt{bag}))$ is not d-linked into W, we can, in some sense, improve $(X, (T, \texttt{bag}))$ while maintaining that it covers W and not increasing its width. We define $\phi_d(X) = |X| \cdot n(k + 1) + d(X)$ as the measure in which sense we will improve $(X, (T, \texttt{bag}))$.

Lemma 5.2. *There is an algorithm that takes as input a graph G, a set of vertices $W \subseteq V(G)$, a torso tree decomposition $(X, (T, \texttt{bag}))$ in G of width k that covers W, a weight function $d : V(G) \to [n]$, and a pair (t, S) that witnesses that $(X, (T, \texttt{bag}))$ is not d-linked into W, and in time $k^{O(1)}(|V(T)| + m)$ returns a torso tree decomposition $(X', (T', \texttt{bag}'))$ that covers W, has width at most k, has at most $|V(T)|$ nodes, and has $\phi_d(X') < \phi_d(X)$.*

Proof. After a $k^{O(1)}m$ time flow computation we may assume that S is a minimum size $(\mathsf{bag}(t), W)$-separator, because if S was not a minimum size $(\mathsf{bag}(t), W)$-separator then any minimum size $(\mathsf{bag}(t), W)$-separator also witnesses that $\mathsf{bag}(t)$ is not d-linked into W. This implies that S is linked into $\mathsf{bag}(t)$.

Let $A = R_G(W, S)$ and $B = V(G) \setminus (A \cup S)$. Note that $W \subseteq A \cup S$ and $\mathsf{bag}(t) \subseteq B \cup S$. Denote $X' = (X \cap A) \cup S$. We apply the pulling lemma (Lemma 4.1) with the torso tree decomposition $(X, (T, \mathsf{bag}))$, the separation (A, S, B), and the node t to construct a torso tree decomposition $(X', (T', \mathsf{bag}'))$ of width at most k and at most $|V(T)|$ nodes. As $W \subseteq X$ and $W \subseteq A \cup S$, we have that $W \subseteq X'$, so $(X', (T', \mathsf{bag}'))$ covers W. It remains to prove that $\phi_d(X') < \phi_d(X)$.

Because $\mathsf{bag}(t) \subseteq S \cup B$ and $\mathsf{bag}(t) \subseteq X$, we have that $|X'| \leq |X| - |\mathsf{bag}(t)| + |S|$ and $d(X') \leq d(X) - d(\mathsf{bag}(t)) + d(S)$. Therefore, if $|S| < |\mathsf{bag}(t)|$, then $|X'| < |X|$, implying $\phi_d(X') < \phi_d(X)$ because $d(S) < n(k+1)$. If $|S| = |\mathsf{bag}(t)|$ and $d(S) < d(\mathsf{bag}(t))$, then $|X'| \leq |X|$ and $d(X') < d(X)$, implying $\phi_d(X') < \phi_d(X)$. □

Then, our goal is to show that either a torso tree decomposition $(X, (T_X, \mathsf{bag}_X))$ of width $k-1$ that covers a largest bag W of a tree decomposition (T, bag) of width k can be used to improve (T, bag), or we find a pair (t, S) witnessing that $(X, (T_X, \mathsf{bag}_X))$ is not d-linked into W for a certain function d, in which case we can improve $(X, (T_X, \mathsf{bag}_X))$ by applying Lemma 5.2.

Let (T, bag) be a tree decomposition of G and $r \in V(T)$ a designated root-node of it. For a vertex $v \in V(G)$, let $f_v \in V(T)$ be the node of T with $v \in \mathsf{bag}(f_v)$ that has the smallest distance to the root r in T among all nodes whose bags contain v, that is, f_v is the forget-node of v. We define a weight function $d_{(T,\mathsf{bag},r)} : V(G) \to [|V(T)|]$ for a vertex $v \in V(G)$ as the distance from f_v to r plus one. Next we prove the main lemma of this section.

Lemma 5.3. *Let (T, bag) be a tree decomposition of G of width k, and r a node of (T, bag) with $\mathsf{bag}(r) = W$ with $|W| = k+1$. There is an algorithm that given a torso tree decomposition $(X, (T_X, \mathsf{bag}_X))$ that covers W and has width at most $k-1$, in time $k^{O(1)}(|V(T)| + |V(T_X)| + m)$ either*

(1) constructs a tree decomposition of G of width at most k, having strictly less bags of size $k+1$ than (T, bag), and having at most n nodes, or
(2) returns a pair (t, S) where $t \in V(T_X)$ and $S \subseteq V(G)$ that witnesses that $(X, (T_X, \mathsf{bag}_X))$ is not $d_{(T,\mathsf{bag},r)}$-linked into W.

Proof. We treat (T, bag) as rooted on the node r. Our goal is to construct a tree decomposition (T', bag') of G, and then show that if it does not satisfy the conditions of Lemma 1, then we find the pair (t, S) of Lemma 2.

First, for every connected component C of $G \setminus X$, we will construct a tree decomposition (T_C, bag_C) of $N[C]$, so that $N(C)$ is in the root bag of (T_C, bag_C). We again use $f_v \in V(T)$ to denote the forget-node of v in (T, bag). For a node $t \in V(T)$, denote by $t^{N(C)}$ the vertices

$$t^{N(C)} = \{v \in N(C) \mid f_v \text{ is a strict descendant of } t \text{ in } T\}.$$

To construct the tree decomposition (T_C, bag_C), we first set

$$T_C = T[\{t \in V(T) \mid C \cap \mathsf{bag}(t) \neq \emptyset\}],$$

i.e., T_C is the subtree of T induced by bags that intersect C. Observe that T_C is connected because $G[C]$ is connected. Then for each $t \in V(T_C)$ we set

$$\mathsf{bag}_C(t) = (\mathsf{bag}(t) \cap N[C]) \cup t^{N(C)}.$$

We let the root node of (T_C, bag_C) to be the node $r_C \in V(T_C)$ that is the closest to r in T. Note that because T_C is a connected subtree of T, the node r_C is uniquely defined.

Claim 5.4. *It holds that (T_C, bag_C) is a tree decomposition of $N[C]$ and $N(C) \subseteq \mathsf{bag}_C(r_C)$.*

Proof. First, for the vertices C and edges in $G[C]$ the decomposition clearly satisfies the vertex and edge conditions because (T, bag) satisfied the conditions. For edges between C and $N(C)$ and vertices in $N(C)$, note that again each such edge must be in a bag that intersects C, and because for every vertex of $N(C)$ there exists such an edge we have that every vertex of $N(C)$ must occur in some bag that intersects C. The decomposition satisfies the connectedness condition for vertices in C directly by the connectedness condition of (T, bag).

For vertices $v \in N(C)$, either (1) $v \in \mathsf{bag}(r_C)$ and v is not in $t^{N(C)}$ for any $t \in V(T_C)$, or (2) $f_v \in V(T_C) \setminus \{r_C\}$ and $v \in t^{N(C)}$ for all t on the path from the parent of f_v to the root r_C. Therefore, the connectedness condition is maintained for vertices in $N(C)$. This also shows that $N(C) \subseteq \mathsf{bag}_C(r_C)$, which finally implies the edge condition also for edges in $G[N(C)]$. □

Now, our complete construction of (T', bag') is to attach the tree decompositions (T_C, bag_C) for all components C of $G \setminus X$ from their roots to the tree decomposition (T_X, bag_X). Because $N(C)$ is a clique in $\mathtt{torso}(X)$, the decomposition (T_X, bag_X) contains a bag containing $N(C)$ to which (T_C, bag_C) can be attached.

Next we show that this construction can be implemented in $k^{O(1)}(|V(T)| + |V(T_X)| + m)$ time. In particular, first, the connected components C and their neighborhoods can be found in $k^{O(1)}m$ time. Then, we observe that the sum of $|V(T_C)|$ over all components C is at most $(k+1)|V(T)|$ because (T, bag) has width k and the components C are disjoint. By first computing pointers from vertices of G to bags containing them, and then using the fact that $|N(C)| \leq k+1$, each tree decomposition (T_C, bag_C) can be constructed in $k^{O(1)}|V(T_C)|$ time, which sums up to $k^{O(1)}|V(T)|$. Then, it remains to attach each tree decomposition (T_C, bag_C) to a node of (T_X, bag_X) whose bag contains $N(C)$. For this, observe that if we consider (T_X, bag_X) rooted, and for $v \in N(C)$ denote by f_v^X the forget-node of v in (T_X, bag_X), then $N(C)$ is contained in the bag of the node f_v^X for $v \in N(C)$ such that f_v^X maximizes the distance from the root.

Next we give the main argument for extracting the witness of Lemma 2 if (T', bag') does not satisfy Lemma 1.

Claim 5.5. *Let C be a component of $G \setminus X$ and $x \in V(T_X)$ a node of (T_X, bag_X) with $N(C) \subseteq \mathsf{bag}_X(x)$. For every node $t \in V(T_C)$ we have either that*

(1) $|\mathsf{bag}_C(t)| < |\mathsf{bag}(t)|$ or $\mathsf{bag}_C(t) = \mathsf{bag}(t)$, or that
(2) $(\mathsf{bag}_X(x) \setminus t^{N(C)}) \cup (\mathsf{bag}(t) \setminus N[C])$ witnesses that $\mathsf{bag}_X(x)$ is not $d_{(T,\mathsf{bag},r)}$-linked into W.

Proof. Lemma 1 is true if $t^{N(C)}$ is empty, so suppose $t^{N(C)}$ is non-empty and $|\mathsf{bag}_C(t)| \geq |\mathsf{bag}(t)|$. By the definition of $\mathsf{bag}_C(t)$ this implies that $|t^{N(C)}| \geq |\mathsf{bag}(t) \setminus N[C]|$. Note that $t^{N(C)} \subseteq \mathsf{bag}_X(x)$. We will show that in this case

$$S = (\mathsf{bag}_X(x) \setminus t^{N(C)}) \cup (\mathsf{bag}(t) \setminus N[C])$$

separates $\mathsf{bag}_X(x)$ from W. Therefore S witnesses that $\mathsf{bag}_X(x)$ is not $d_{(T,\mathsf{bag},r)}$-linked into W, because by $|t^{N(C)}| \geq |\mathsf{bag}(t) \setminus N[C]|$ we have that $|S| \leq |\mathsf{bag}_X(x)|$, and moreover we have $d_{(T,\mathsf{bag},r)}(S) < d_{(T,\mathsf{bag},r)}(\mathsf{bag}_X(x))$ because for every vertex $v_1 \in t^{N(C)}$ and $v_2 \in \mathsf{bag}(t)$, it holds that $d_{(T,\mathsf{bag},r)}(v_1) > d_{(T,\mathsf{bag},r)}(v_2)$ because f_{v_1} is a strict descendant of t, and f_{v_2} is an ancestor of t.

To show that S separates $\mathsf{bag}_X(x)$ from W, it is sufficient to show that it separates $t^{N(C)}$ from W because $\mathsf{bag}_X(x) \setminus S = t^{N(C)}$. Consider a shortest path in $G \setminus S$ that starts in $t^{N(C)}$ and ends in W. If this path would intersect $N[C]$ anywhere else than in its first vertex, then it would intersect $t^{N(C)}$ twice because $N(C) \setminus S = t^{N(C)}$ and $W \cap C = \emptyset$, which would contradict that it is a shortest path. Therefore, it intersects $N[C]$ only in its first vertex. Then, because for each $v \in t^{N(C)}$ the node $t \in V(T)$ separates f_v from r in T, it holds that $\mathsf{bag}(t)$ separates $t^{N(C)}$ from $\mathsf{bag}(r) = W$. Therefore, the path must intersect $\mathsf{bag}(t)$, and therefore as $\mathsf{bag}(t)$ and $t^{N(C)}$ are disjoint, it must intersect $\mathsf{bag}(t) \setminus N[C]$. However, $\mathsf{bag}(t) \setminus N[C] \subseteq S$, and therefore no such path exists in $G \setminus S$. □

Now, for all nodes of the constructed decompositions (T_C, bag_C) we check if Lemma 1 of Lemma 5.5 holds, and if it does not hold we return the pair $(x, (\mathsf{bag}_X(x) \setminus t^{N(C)}) \cup (\mathsf{bag}(t) \setminus N[C]))$. This can be done in $k^{O(1)}|V(T')| = k^{O(1)}|V(T)|$ time.

Then, it remains to prove that if Lemma 1 of Lemma 5.5 holds for all nodes of all decompositions (T_C, bag_C), then (T', bag') has width at most k and has strictly less bags of size $k+1$ than (T, bag). First, clearly (T', bag') has width at most k as none of the decompositions (T_C, bag_C) have larger width than (T, bag) and (T_X, bag_X) has smaller width than (T, bag). It remains to prove that (T', bag') has less bags of size $k+1$ than (T, bag).

Consider any node $t \in V(T)$, and suppose that there are two distinct components C_1 and C_2 of $G \setminus X$ so that both C_1 and C_2 intersect $\mathsf{bag}(t)$ and $|\mathsf{bag}_{C_1}(t)| = |\mathsf{bag}_{C_2}(t)| = |\mathsf{bag}(t)|$. Now, by Lemma 1 of Lemma 5.5 it would hold that $\mathsf{bag}_{C_1}(t) = \mathsf{bag}_{C_2}(t) = \mathsf{bag}(t)$. However, as $\mathsf{bag}_{C_1}(t) \subseteq N[C_1]$, this would contradict that $\mathsf{bag}(t)$ intersects C_2. Therefore, for any node $t \in V(T)$ there is at most one corresponding node t in the decompositions (T_C, bag_C) across all components C with a bag of size $|\mathsf{bag}_C(t)| = |\mathsf{bag}(t)|$. For the root node r, as $\mathsf{bag}(r) \subseteq X$, none of the components C intersect $\mathsf{bag}(r)$, and therefore no decomposition (T_C, bag_C) contains a node corresponding to it. All other bags of (T', bag') come from (T_X, bag_X) and have size at most k, so as $|\mathsf{bag}(r)| = k+1$, it follows that (T', bag') has strictly less bags of size $k+1$ than (T, bag).

Finally, by Lemma 3.1 we can reduce the number of nodes of (T', bag') to at most n within the same time. □

Then, we combine Lemmas 5.2 and 5.3 into a single lemma showing that to improve (T, bag) it is sufficient to find a torso tree decomposition in G that covers a largest bag of (T, bag) and has width smaller than (T, bag).

Lemma 5.6. *Let (T, bag) be a tree decomposition of G of width k and $|V(T)| \leq n$, and r a node of (T, bag) with $\mathsf{bag}(r) = W$ with $|W| = k+1$. There is an algorithm that given a torso tree decomposition $(X, (T_X, \mathsf{bag}_X))$ that covers W and has width at most $k-1$, in time $k^{O(1)}(|V(T_X)| + n^3)$ constructs a tree decomposition of G of width at most k, having strictly less bags of size $k+1$ than (T, bag), and having at most n nodes.*

Proof. First, we apply Lemma 3.1 to reduce the number of nodes of (T_X, bag_X) to at most n. Then, we repeatedly apply Lemma 5.3 together with Lemma 5.2, in particular, if Lemma 5.3 returns the tree decomposition of Lemma 1 we are done, and if it returns a pair (t, S) that witnesses that $(X, (T_X, \mathsf{bag}_X))$ is not $d_{(T,\mathsf{bag},r)}$-linked into W then we apply Lemma 5.2, which decreases $\phi_{d_{(T,\mathsf{bag},r)}}(X)$ by at least one. Because $\phi_{d_{(T,\mathsf{bag},r)}}(X)$ is initially $O(kn^2)$ and $\phi_{d_{(T,\mathsf{bag},r)}}(X)$ must be non-negative, the total number of iterations is at most $O(kn^2)$, giving a total running time of $k^{O(1)}n^3$, plus $k^{O(1)}|V(T_X)|$ from the application of Lemma 3.1. □

5.2 Reducing Treewidth to Subset Treewidth

Now we can prove Theorem 3, in particular that algorithms for Subset Treewidth imply algorithms for treewidth (for definition of Subset Treewidth see Section 2.1). Recall that the running time function $T(k)$ is assumed to be increasing on k and we denote $m = |V(G)| + |E(G)|$.

Theorem 3. *Given an algorithm for Subset Treewidth with running time $T(k) \cdot m^c$, an algorithm for treewidth with running time $T(O(k)) \cdot O((nk)^{c+1}) + k^{O(1)}n^4 + 2^{O(k)}n^2$ can be constructed. Moreover, if the algorithm for Subset Treewidth works in polynomial space, then the algorithm for treewidth works in polynomial space.*

Proof. Let G denote the input graph. First, we use the 4-approximation algorithm of [42] to obtain a tree decomposition (T, bag) of G of width at most $4k+3$ in time $2^{O(k)}n^2$ and polynomial space or to return that the treewidth of G is larger than k. By Lemma 3.1, within the same running time we assume that $|V(T)| \leq n$, and we can also assume that $m \leq O(kn)$ because otherwise the treewidth of G would be larger than k.

Then, we repeat the following process as long as the width of (T, bag) is larger than k. Let W be a largest bag of (T, bag), and note that in this case $|W| \geq k+2$. We use the algorithm for Subset Treewidth to either get a torso tree decomposition that covers W and has width at most $|W|-2$ or to conclude that the treewidth of G is larger than $|W|-2 \geq k$. If we conclude that the treewidth of G is larger than k we are ready and can immediately return. If the algorithm returns such a torso tree decomposition, we apply Lemma 5.6 to improve (T, bag), in particular to decrease the number of bags of size $|W|$ and not increase the width.

We can decrease the number of largest bags while not increasing the width at most $O(kn)$ times before the width decreases from $4k+3$ to k, and therefore the algorithm works with $O(kn)$ applications of the algorithm for Subset Treewidth and Lemma 5.6. In all of the applications, the parameter k for Subset Treewidth is at most $4k+2$, where k is the original parameter for treewidth. This results in a total running time of $T(O(k)) \cdot O((nk)^{c+1}) + k^{O(1)}n^4 + 2^{O(k)}n^2$. □

We then turn to Theorem 6, in particular, to proving that algorithms for Partitioned Subset Treewidth imply approximation algorithms for treewidth (for the definition of Partitioned Subset Treewidth see Section 2.1). The crucial lemma for this will the following.

Lemma 5.7. *Let G be a graph of treewidth at most k, $\varepsilon \in (0,1)$ a rational, and $W \subseteq V(G)$ a set of vertices of size $|W| \leq 4k+4$. There exists a partition of W into $t = O(1/\varepsilon)$ parts $W_1, \ldots, W_t$, so that after making each part into a clique the treewidth of G is at most $k + \varepsilon k$.*

Proof. If $\varepsilon < 1/k$ we can return the trivial partition of W into single vertices. Therefore we can assume that $\varepsilon k \geq 1$.

Consider a rooted tree decomposition (T, bag) of G of width k. By turning (T, bag) into a "nice tree decomposition", we can assume that the root bag of (T, bag) is empty, each node of T has at most two children, and that $|\mathsf{bag}(u) \setminus \mathsf{bag}(v)| + |\mathsf{bag}(v) \setminus \mathsf{bag}(u)| \leq 1$ holds for any two adjacent nodes $u, v \in V(T)$ (see e.g. [22, Chapter 7]). Recall that a node $t \in V(T)$ with a parent $p \in V(T)$ is a forget-node of a vertex $v \in V(T)$ if $v \in \mathsf{bag}(t) \setminus \mathsf{bag}(p)$. Respectively, such v is a forget-vertex of t. Note that each node of T has at most one forget-vertex and each vertex of G has exactly one forget-node. By further stretching (T, bag) we can also assume that each forget-node has exactly one child. We say that a node is a W-forget node if it is a forget-node of a vertex $w \in W$.

Let us process (T, bag) from the leaves towards the root, i.e., in an order of a post-order traversal, and maintain a set of "removed" nodes $R \subseteq V(T)$. Suppose we are processing a node t and let $D \subseteq V(T)$ be the nodes of T that are descendants of t and reachable from t in $T \setminus R$. Note that $t \in D$ and $D \subseteq V(T) \setminus R$. Now, if D contains at least $\varepsilon k/2$ W-forget-nodes or t is the root we add a part to the partition of W and modify the tree decomposition as follows. We let $W' \subseteq W$ be the vertices in W whose forget-nodes are in D. We add W' as a part of the partition, and add W' to the bags of all nodes in D. Then, we add all nodes in D to R.

Observe that $|W'| \leq \varepsilon k$ follows from the facts that we process the tree in post-order, each node can have at most two children, each node can be a forget-node of at most one vertex, each forget-node has one child, and $\varepsilon k \geq 1$. Therefore, the sizes of the bags of nodes in D increased by at most εk, and moreover they will not increase again because they were added to R. Therefore, the resulting tree decomposition has width at most $k + \varepsilon k$. We also observe that the resulting tree decomposition is indeed a tree decomposition after making such W' into a clique: All the new edges are contained in the bags of all nodes in D, and the subtree condition is maintained because the forget-nodes of vertices in W' are in D.

Now, each created part of the partition except the part corresponding to the root has size at least $\varepsilon k/2$, so in total the number of parts is at most $|W|/(\varepsilon k/2) + 1 \leq \frac{16}{\varepsilon} + 1 = O(1/\varepsilon)$. □

Now, by using Lemma 5.7 we can prove Theorem 6 similarly to Theorem 3.

Theorem 6. *Given an algorithm for Partitioned Subset Treewidth with running time $T(k,t) \cdot m^c$, we can construct an $(1+\varepsilon)$-approximation algorithm for treewidth with running time*

$$T(O(k), O(1/\varepsilon)) \cdot O((nk)^{c+1}) \cdot (1+1/\varepsilon)^{O(k)} + k^{O(1)} n^4 + 2^{O(k)} n^2.$$

Moreover, if the algorithm for Partitioned Subset Treewidth works in polynomial space, then the algorithm for treewidth works in polynomial space.

Proof. Let G denote the input graph. First, we use the 4-approximation algorithm of [42] to obtain a tree decomposition (T, bag) of G of width at most $4k+3$ in time $2^{O(k)} n^2$ and polynomial space or to return that the treewidth of G is larger than k. By Lemma 3.1, within the same running time we assume that $|V(T)| \leq n$, and we can also assume that $m \leq O(kn)$ because otherwise the treewidth of G would be larger than k.

Then, we repeat the following process as long as the width of (T, bag) is larger than $k + \varepsilon k$. Let W be a largest bag of (T, bag), and note that in this case $|W| \geq k + \varepsilon k + 2$ and $|W| \leq 4k+4$. We try all partitions of W into $t = O(1/\varepsilon)$ parts (where the bound for t is from Lemma 5.7). For each partition $W_1, \ldots, W_t$, we make the parts $W_1, \ldots, W_t$ into cliques in G, and then use the algorithm for Partitioned Subset Treewidth with this partition of W. By Lemma 5.7, there exists such a partition so that after making $W_1, \ldots, W_t$ into cliques the treewidth of G is at most $k + \varepsilon k$, and therefore if the algorithm for Partitioned Subset Treewidth returns for every partition that the treewidth of G is larger than $|W| - 2 \geq k + \varepsilon k$, we can return that the treewidth of G is larger than k. Otherwise, the algorithm for Partitioned Subset Treewidth returned a torso tree decomposition that covers W and has width at most $|W| - 2$, and we proceed applying Lemma 5.6 similarly as in the proof of Theorem 3.

The running time follows from the fact that there are at most $t^{O(k)} = (1+1/\varepsilon)^{O(k)}$ partitions of W into $t = O(1/\varepsilon)$ parts, and we can decrease the number of largest bags while not increasing the width at most $O(nk)$ times, and therefore there we use in total $O(nk) \cdot (1+1/\varepsilon)^{O(k)}$ applications of the algorithm for Partitioned Subset Treewidth with $t = O(1/\varepsilon)$. The parameter k for Partitioned Subset Treewidth is at most $4k+2$, where k is the original parameter for treewidth. □

REFERENCES

[1] E. Amir. 2001. Efficient approximation for triangulation of minimum treewidth. In *Uncertainty in Artificial Intelligence: Proceedings of the Seventeenth Conference (UAI-2001), San Francisco, CA*. Morgan Kaufmann Publishers, 7–15.

[2] Eyal Amir. 2010. Approximation Algorithms for Treewidth. *Algorithmica* 56, 4 (2010), 448–479. https://doi.org/10.1007/s00453-008-9180-4

[3] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. 1987. Complexity of finding embeddings in a k-tree. *SIAM J. Alg. Disc. Meth.* 8 (1987), 277–284.

[4] Mahdi Belbasi and Martin Fürer. 2021. Finding All Leftmost Separators of Size $\leq k$. In *Proceedings of 15th International Conference on Combinatorial Optimization and Applications (COCOA) (Lecture Notes in Comput. Sci., Vol. 13135)*. Springer, 273–287.

[5] Mahdi Belbasi and Martin Fürer. 2022. An Improvement of Reed's Treewidth Approximation. *J. Graph Algorithms Appl.* 26, 2 (2022), 257–282.

[6] Patrick Bellenbaum and Reinhard Diestel. 2002. Two short proofs concerning tree-decompositions. *Combinatorics, Probability and Computing* 11, 6 (2002), 541–547.

[7] Umberto Bertelè and Francesco Brioschi. 1972. *Nonserial dynamic programming*. Academic Press, New York. xii+235 pages. Mathematics in Science and Engineering, Vol. 91.

[8] Hans L. Bodlaender. 1993. A linear time algorithm for finding tree-decompositions of small treewidth. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC)*. ACM, 226–234. https://doi.org/10.1145/167088.167161

[9] H. L. Bodlaender. 1993. A tourist guide through treewidth. *Acta Cybernet.* 11, 1-2 (1993), 1–21.

[10] Hans L. Bodlaender. 1994. Improved Self-reduction Algorithms for Graphs with Bounded Treewidth. *Discret. Appl. Math.* 54, 2-3 (1994), 101–115. https://doi.org/10.1016/0166-218X(94)90018-3

[11] Hans L. Bodlaender. 1996. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Computing* 25, 6 (1996), 1305–1317.

[12] Hans L Bodlaender, Leizhen Cai, Jianer Chen, Michael R. Fellows, Jan Arne Telle, and Dániel Marx. 2006. *Open problems in parameterized and exact computation – IWPEC 2006.* Technical Report UU-CS-2006-052. Department of Information and Computing Sciences, Utrecht University.

[13] Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michał Pilipczuk. 2016. A $c^k n$ 5-approximation algorithm for treewidth. *SIAM J. Computing* 45, 2 (2016), 317–378. https://doi.org/10.1137/130947374

[14] Hans L. Bodlaender, John R. Gilbert, Hjálmtyr Hafsteinsson, and Ton Kloks. 1995. Approximating Treewidth, Pathwidth, Frontsize, and Shortest Elimination Tree. *J. Algorithms* 18, 2 (1995), 238–255.

[15] Hans L. Bodlaender, Lars Jaffke, and Jan Arne Telle. 2021. Typical Sequences Revisited - Computing Width Parameters of Graphs. *Theory Comput. Syst.* (2021). https://doi.org/10.1007/s00224-021-10030-3

[16] Hans L. Bodlaender and Ton Kloks. 1991. Better Algorithms for the Pathwidth and Treewidth of Graphs. In *Proceedings of the 18th International Colloquium of Automata, Languages and Programming (ICALP) (Lecture Notes in Comput. Sci., Vol. 510)*. Springer, 544–555. https://doi.org/10.1007/3-540-54233-7_162

[17] Hans L. Bodlaender and Ton Kloks. 1996. Efficient and Constructive Algorithms for the Pathwidth and Treewidth of Graphs. *J. Algorithms* 21, 2 (1996), 358–402.

[18] Hans L. Bodlaender and Arie M. C. A. Koster. 2006. Safe separators for treewidth. *Discret. Math.* 306, 3 (2006), 337–350.

[19] Richard B. Borie, R. Gary Parker, and Craig A. Tovey. 1992. Automatic Generation of Linear-Time Algorithms from Predicate Calculus Descriptions of Problems on Recursively Constructed Graph Families. *Algorithmica* 7, 5&6 (1992), 555–581.

[20] Jianer Chen, Yang Liu, and Songjian Lu. 2009. An Improved Parameterized Algorithm for the Minimum Node Multiway Cut Problem. *Algorithmica* 55, 1 (2009), 1–13.

[21] Bruno Courcelle. 1990. The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Information and Computation* 85 (1990), 12–75.

[22] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. 2015. *Parameterized Algorithms*. Springer. http://dx.doi.org/10.1007/978-3-319-21275-3

[23] Reinhard Diestel. 2005. *Graph theory* (3rd ed.). Graduate Texts in Mathematics, Vol. 173. Springer-Verlag, Berlin. xvi+411 pages.

[24] Rodney G. Downey and Michael R. Fellows. 1999. *Parameterized complexity*. Springer-Verlag, New York.

[25] Rodney G. Downey and Michael R. Fellows. 2013. *Fundamentals of Parameterized Complexity*. Springer.

[26] Michael Elberfeld, Andreas Jakoby, and Till Tantau. 2010. Logspace Versions of the Theorems of Bodlaender and Courcelle. In *Proceedings of the 51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010*. IEEE Computer Society, 143–152. https://doi.org/10.1109/FOCS.2010.21

[27] Uriel Feige, MohammadTaghi Hajiaghayi, and James R. Lee. 2008. Improved Approximation Algorithms for Minimum Weight Vertex Separators. *SIAM J. Computing* 38, 2 (2008), 629–657.

[28] Michael R. Fellows and Michael A. Langston. 1989. On Search, Decision and the Efficiency of Polynomial-Time Algorithms (Extended Abstract). In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing (STOC)*. ACM, 501–512.

[29] Jörg Flum and Martin Grohe. 2006. *Parameterized Complexity Theory*. Springer-Verlag, Berlin. 493 pages.

[30] Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, Michal Pilipczuk, and Marcin Wrochna. 2018. Fully Polynomial-Time Parameterized Computations for Graphs and Matrices of Low Treewidth. *ACM Transactions on Algorithms* 14, 3 (2018), 34:1–34:45. https://doi.org/10.1145/3186898

[31] Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. 2019. *Kernelization: Theory of parameterized preprocessing*. Cambridge University Press.

[32] Fedor V. Fomin, Ioan Todinca, and Yngve Villanger. 2015. Large Induced Subgraphs via Triangulations and CMSO. *SIAM J. Comput.* 44, 1 (2015), 54–87. https://doi.org/10.1137/140964801

[33] Rudolf Halin. 1976. S-functions for graphs. *J. Geometry* 8, 1-2 (1976), 171–186.

[34] Tuukka Korhonen. 2021. A Single-Exponential Time 2-Approximation Algorithm for Treewidth. In *Proceedings of the 62nd Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 184–192. https://doi.org/10.1109/FOCS52979.2021.00026

[35] Tuukka Korhonen and Daniel Lokshtanov. 2022. An Improved Parameterized Algorithm for Treewidth. *CoRR* abs/2211.07154 (2022). https://doi.org/10.48550/arXiv.2211.07154 arXiv:2211.07154

[36] Jens Lagergren. 1996. Efficient Parallel Algorithms for Graphs of Bounded Tree-Width. *Journal of Algorithms* 20, 1 (1996), 20–44. https://doi.org/10.1006/jagm.1996.0002

[37] Jens Lagergren and Stefan Arnborg. 1991. Finding Minimal Forbidden Minors Using a Finite Congruence. In *Proceedings of the 18th International Colloquium of Automata, Languages and Programming (ICALP) (Lecture Notes in Comput. Sci., Vol. 510)*. Springer, 532–543. https://doi.org/10.1007/3-540-54233-7_161

[38] Jirí Matousek and Robin Thomas. 1991. Algorithms Finding Tree-Decompositions of Graphs. *J. Algorithms* 12, 1 (1991), 1–22.

[39] Rolf Niedermeier. 2006. *Invitation to fixed-parameter algorithms*. Oxford Lecture Series in Mathematics and its Applications, Vol. 31. Oxford University Press, Oxford. xii+300 pages.

[40] Bruce A. Reed. 1992. Finding Approximate Separators and Computing Tree Width Quickly. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, STOC 1992*, S. Rao Kosaraju, Mike Fellows, Avi Wigderson, and John A. Ellis (Eds.). ACM, 221–228. https://doi.org/10.1145/129712.129734

[41] Neil Robertson and Paul D. Seymour. 1984. Graph Minors. III. Planar Tree-Width. *J. Combinatorial Theory Ser. B* 36 (1984), 49–64.

[42] Neil Robertson and Paul D. Seymour. 1995. Graph minors. XIII. The disjoint paths problem. *J. Combinatorial Theory Ser. B* 63, 1 (1995), 65–110.

[43] Neil Robertson and Paul D. Seymour. 2004. Graph Minors. XX. Wagner's conjecture. *J. Combinatorial Theory Ser. B* 92, 2 (2004), 325–357.

[44] Yu Wu, Per Austrin, Toniann Pitassi, and David Liu. 2014. Inapproximability of Treewidth and Related Problems. *J. Artif. Intell. Res.* 49 (2014), 569–600.

Received 2022-11-07; accepted 2023-02-06