SIAM J. COMPUT.
Vol. 53, No. 4, pp. 1085–1131

# FAST FPT-APPROXIMATION OF BRANCHWIDTH[*]

FEDOR V. FOMIN[†] AND TUUKKA KORHONEN[†]

**Abstract.** Branchwidth determines how graphs and, more generally, arbitrary connectivity (symmetric and submodular) functions can be decomposed into a tree-like structure by specific cuts. We develop a general framework for designing fixed-parameter tractable 2-approximation algorithms for branchwidth of connectivity functions. The first ingredient of our framework is combinatorial. We prove a structural theorem establishing that either a sequence of particular refinement operations can decrease the width of a branch decomposition or the width of the decomposition is already within a factor of 2 from the optimum. The second ingredient is an efficient implementation of the refinement operations for branch decompositions that support efficient dynamic programming. We present two concrete applications of our general framework. The first is an algorithm that, for a given $n$-vertex graph $G$ and integer $k$, in time $2^{2^{\mathcal{O}(k)}} n^2$ either constructs a rank decomposition of $G$ of width at most $2k$ or concludes that the rankwidth of $G$ is more than $k$. It also yields a $(2^{2k+1}-1)$-approximation algorithm for cliquewidth within the same time complexity, which in turn improves to $f(k) \cdot n^2$ the running times of various algorithms on graphs of cliquewidth $k$. Breaking the "cubic barrier" for rankwidth and cliquewidth was an open problem in the area. The second application is an algorithm that, for a given $n$-vertex graph $G$ and integer $k$, in time $2^{\mathcal{O}(k)} n$ either constructs a branch decomposition of $G$ of width at most $2k$ or concludes that the branchwidth of $G$ is more than $k$. This improves over the 3-approximation that follows from the recent treewidth 2-approximation of Korhonen [FOCS 2021].

**Key words.** branchwidth, rankwidth, cliquewidth, graph algorithms, parameterized algorithms, approximation algorithms

**MSC codes.** 68Q25, 68R10, 68Q27

**DOI.** 10.1137/22M153937X

## 1. Introduction.

The branchwidth of a connectivity function $f$ (that is, $f$ is symmetric and submodular and $f(\emptyset) = 0$) was introduced by Robertson and Seymour [41]. Let $V$ be a finite set and $f : 2^V \to \mathbb{Z}_{\geq 0}$ be a connectivity function on $V$. A *branch decomposition* of $f$ is a pair $(T, L)$, where $T$ is a cubic tree (the degree of each nonleaf node of $T$ is 3) and $L$ is a bijection mapping $V$ to the leaves of $T$. (If $|V| \leq 1$, then $f$ admits no branch decomposition.) For every edge $e$ of $T$, the connected components of $T \setminus \{e\}$, the graph obtained from $T$ by deleting $e$, induce a partition $(X, Y)$ of the set of leaves of $T$. The *width* of $e$ is $f(e) = f(L^{-1}(X)) = f(L^{-1}(Y))$. The *width of* $(T, L)$ is the maximum width of all edges of $T$. The *branchwidth* $\mathtt{bw}(f)$ of $f$ is the minimum width of a branch decomposition of $f$. In this paper, we develop a framework for designing fixed-parameter tractable (FPT) 2-approximation algorithms for computing branch decompositions of connectivity functions. We provide a detailed overview of the framework in section 2. Here we discuss some of its concrete algorithmic consequences.

*Rankwidth.* Rankwidth was introduced by Oum and Seymour [37]. A *rank decomposition* of a graph $G$ is a branch decomposition of the following connectivity

[*]Received by the editors December 12, 2022; accepted for publication (in revised form) April 30, 2024; published electronically August 8, 2024. A preliminary version of this paper appeared in the Proceedings of STOC 2022.
https://doi.org/10.1137/22M153937X
**Funding:** This work was supported by the Research Council of Norway via the project BWCA grant 314528.

[†]Department of Informatics, University of Bergen, Bergen, Norway (fomin@ii.uib.no, tuukka.korhonen@uib.no).

function $f$ defined on the vertex set $V(G)$. For a graph $G$ and a pair $A, B$ of disjoint subsets of $V(G)$, let $G[A,B]$ be the bipartite graph induced by edges between $A$ and $B$. Let $M_G[A,B]$ be the $|A| \times |B|$ 0-1 matrix representing $G[A,B]$. Then the value $f(A) = f(V(G) \setminus A) = \mathrm{rk}(M_G[A, V(G) \setminus A])$ is the GF(2)-rank of $M_G[A, V(G) \setminus A]$. By making use of our algorithmic framework for connectivity functions, we prove the following theorem about approximation of rankwidth.

THEOREM 1.1. *There is an algorithm that, given an $n$-vertex graph $G$ and an integer $k$, in time $2^{2^{\mathcal{O}(k)}} n^2$ either computes a rank decomposition of $G$ of width at most $2k$ or correctly concludes that the rankwidth of $G$ is more than $k$.*

Several algorithms computing rankwidth exactly or approximately are known in the literature. After a number of improvements, the best running times of algorithms computing rankwidth are of the form $f(k) \cdot n^3$ [28, 30, 35, 38, 37]. Theorem 1.1 affirmatively answers the open question of Oum [36, Question 3], who asked whether there exists an algorithm with functions $f(k)$, $g(k)$, and a constant $c < 3$ that finds a rank decomposition of width at most $f(k)$ or confirms that the rankwidth of a graph is larger than $k$, in time $g(k) \cdot n^c$. Pipelined with the previous work on rankwidth and cliquewidth, Theorem 1.1 has several important consequences.

*Cliquewidth.* A common approach in graph algorithms is to decompose a graph by making use of small separations. Perhaps the most popular measure of graph decomposition is the treewidth of a graph [17, Chapter 7]. Many NP-hard optimization problems can be solved efficiently on graphs of bounded treewidth. The seminal result of Courcelle [10, 11] (see also [1, 6, 12]), combined with the algorithm of Bodlaender [2], states that every decision problem on graphs expressible in monadic second order logic ($\mathbf{MSO_2}$) is solvable in linear time on graphs of bounded treewidth. However, the average vertex degree of a graph of treewidth $k$ does not exceed $k$, limiting the algorithmic applicability of treewidth to "sparse" graph classes. Arguably, the most successful project of extending the meta-algorithmic results from graphs of bounded treewidth to "nonsparse" graphs is by making use of the cliquewidth of a graph defined by Courcelle, Engelfriet, and Rozenberg [13]. We give the formal definition due to its technicality in the appendix. Informally, a graph is of cliquewidth at most $k$ if it can be built from single vertices following a $k$-expression, which identifies how to systematically join the already constructed parts of the graph. Moreover, in each constructed part, the vertices can be partitioned into at most $k$ types such that vertices of the same type will be indistinguishable in later steps of the construction. Cliquewidth generalizes treewidth in the following sense. Every graph $G$ of treewidth at most $k$ has cliquewidth at most $\mathcal{O}(2^k)$. On the other hand, for example, the $n$-vertex complete graph $K_n$ has treewidth $n-1$ but constant cliquewidth [29].

Oum and Seymour proved that for any graph of rankwidth $k$, its cliquewidth is between $k$ and $2^{k+1} - 1$ [37]. Moreover, their proof gives an algorithm that in time $2^{\mathcal{O}(k)} n^2$ converts a rank decomposition of width $k$ into a $(2^{k+1} - 1)$-expression for the cliquewidth. By combining the construction of Oum and Seymour with Theorem 1.1, we derive the following corollary.

COROLLARY 1.2. *There is an algorithm that, given an $n$-vertex graph $G$ and an integer $k$, in time $2^{2^{\mathcal{O}(k)}} n^2$ either computes a $(2^{2k+1} - 1)$-expression of $G$ or correctly concludes that the cliquewidth of $G$ is more than $k$.*

The importance of cliquewidth is due to its algorithmic properties. Courcelle, Makowsky, and Rotics in [14] identified a variation of $\mathbf{MSO_2}$, called $\mathbf{MSO_1}$, and showed how to extend Courcelle's theorem for $\mathbf{MSO_1}$ to graphs of bounded cliquewidth. Informally, $\mathbf{MSO_1}$ is the class of $\mathbf{MSO_2}$ formulas that allow quantification over subsets of vertices, but not of edges. While being less expressive than $\mathbf{MSO_2}$, $\mathbf{MSO_1}$ still captures a broad class of optimization problems on graphs, including vertex cover, dominating set, domatic number for fixed $k$, $k$-colorability for fixed $k$, partition into cliques for fixed $k$, clique, independent set, and induced path. The meta-theorem of Courcelle, Makowsky, and Rotics [14] states that every $\mathbf{MSO_1}$-definable problem on graphs is solvable in time $f(k) \cdot (n + m)$ when a $k$-expression is provided with the input. The theorem of Courcelle, Makowsky, and Rotics applies also to a more general class of problems, like optimization problems searching for sets of vertices that are optimal concerning some linear evaluation function (for example, a clique of the maximum weight) or counting [14, 15].

The applicability of the meta-theorem of Courcelle, Makowsky, and Rotics crucially depends on the efficiency of computing the cliquewidth of a graph and constructing the corresponding $k$-expression. The only known way of constructing (an approximately optimal) $k$-expression for cliquewidth, as well as for related graph parameters like NLC-width [44] or Boolean width [8], is by making use of rankwidth. Combining Corollary 1.2 with the meta-theorem of Courcelle, Makowsky, and Rotics [14] implies that every $\mathbf{MSO_1}$-definable problem is solvable in quadratic $f(k) \cdot n^2$ running time on graphs of cliquewidth at most $k$. This is the first improvement on the time complexity of this meta-theorem since the $f(k) \cdot n^3$ algorithm given by Oum in 2005 [35, 33].

*Exact rankwidth.* Oum [34] proved that the set of graphs having rankwidth at most $k$ is characterized by excluded vertex-minors with at most $(6^{k+1} - 1)/5$ vertices. Thus for every $k$, there are only finitely many graphs, such that a graph $G$ does not contain any of them as a vertex-minor if and only if the rankwidth of $G$ is at most $k$, and this set of graphs is computable by exhaustive enumeration and brute-force testing of the rankwidth. Courcelle and Oum [16] proved that for every fixed graph $H$, the property that $H$ is isomorphic to a vertex-minor of an input graph $G$ is expressible in a variant of $\mathbf{MSO_1}$ that can be checked in time $f(k) \cdot (n + m)$ if a $k$-expression is provided. Combined with Corollary 1.2, this implies the following.

Corollary 1.3. *Deciding whether the rankwidth of a given $n$-vertex graph is at most $k$ can be done in time $f(k) \cdot n^2$.*

The function $f(k)$ in Corollary 1.3 is huge as it depends on the set of forbidden vertex-minors as well as checking a generalization of $\mathbf{MSO_1}$. Also this approach does not provide the rank decomposition.

Jeong, Kim, and Oum [30] developed an alternative framework for computing branch decompositions of finite-dimensional vector spaces over a fixed finite field. As one of the applications of their method, they gave an FPT algorithm that for an input graph $G$ and integer $k$ in time $2^{2^{O(k^2)}} n^3$ either constructs a rank decomposition of width $\leq k$ or concludes that the rankwidth of $G$ is more than $k$. The algorithm of Jeong, Kim, and Oum does not rely on vertex-minors and logic and can be seen as a (very nontrivial) adaptation to vector spaces of the dynamic programming algorithm of Bodlaender and Kloks for treewidth [4]. The cubic running time of their algorithm is due to the application of iterative compression. However, if instead of iterative compression we combine Theorem 1.1 with the dynamic programming algorithm of Jeong, Kim, and Oum, we immediately obtain the following corollary.

Corollary 1.4. *There is an algorithm that, given an $n$-vertex graph $G$ and an integer $k$, in time $2^{2^{O(k^2)}} n^2$ either constructs a rank decomposition of width at most $k$ or correctly concludes that the rankwidth of $G$ is more than $k$.*

*Branchwidth of graphs.* As another application of the new algorithmic framework for connectivity functions, we obtain a 2-approximation algorithm for branchwidth of graphs. In this case, the connectivity function $f$ is defined on the edge set $E(G)$ of a graph $G$. For $A \subseteq E(G)$, $f(A)$ is the number of vertices that are incident to both $A$ and $E(G) \setminus A$. The branchwidth of graphs is a "close relative" of the treewidth: For every graph of branchwidth $k$, its treewidth is between $k-1$ and $3k/2-1$ [41]. Thus any $c$-approximation of treewidth is also a $3c/2$-approximation for branchwidth, and vice versa. It appears that in certain situations, branchwidth could be more convenient to work with than treewidth, for example, when it comes to some dynamic programming algorithms [9, 19, 21]. The previously best-known approximation of branchwidth of running time $2^{\mathcal{O}(k)}n$ is a 3-approximation that follows from the treewidth 2-approximation algorithm of Korhonen [32]. We improve this to 2-approximation.

THEOREM 1.5. *There is an algorithm that, given an $n$-vertex graph $G$ and an integer $k$, in time $2^{\mathcal{O}(k)}n$ either computes a branch decomposition of $G$ of width at most $2k$ or correctly concludes that the branchwidth of $G$ is more than $k$.*

## 1.1. Previous work.

An algorithm of running time $\mathcal{O}(n^{8k+12}\log n)$ deciding whether the branchwidth of a connectivity function is at most $k$ was given by Oum [35]. Oum and Seymour gave a 3-approximation algorithm of running time $O(n^6\delta\log n)$, where $\delta$ is the time for each evaluation of an "interpolation" of $f$ [37] (see [37] for the definition of "interpolation").

Rankwidth was introduced by Oum and Seymour as a tool for FPT-approximation of cliquewidth [37]. In particular, they defined rankwidth, showed that rankwidth and cliquewidth are functionally equivalent parameters, and gave an $\mathcal{O}(8^k n^9 \log n)$ time algorithm for outputting a rank decomposition of width at most $3k+1$ or concluding that the rankwidth is more than $k$. For unbounded $k$, Fellows, Rosamond, Rotics, and Szeider showed that cliquewidth is NP-complete [20], and Oum observed that rankwidth is NP-complete [35]. Since the algorithm of Oum and Seymour, a number of FPT exact and approximation algorithms were developed for rankwidth [16, 28, 30, 35, 38, 37]; see Table 1 for an overview. The running times of all these algorithms are at least cubic in $n$. The existence of an FPT-approximation in subcubic time was widely open; see [36, Question 3]. Rankwidth 1 graphs are known as distance-hereditary graphs [34]. There is a (nontrivial) linear time $\mathcal{O}(n+m)$ recognition algorithm for distance-hereditary graphs [18]. To the best of our knowledge,

TABLE 1
*Overview of rankwidth algorithms. Here $k$ is the rankwidth and $n$ is the number of vertices of an input graph $G$. Unless otherwise specified, each of the algorithms outputs in $\mathcal{O}(TIME)$ a decomposition of width given in the Approx column. The function $h(k)$ is a huge function whose bound depends on list of forbidden minors in matroids or vertex-minors in graphs and the length of the logic formula describing such minors.*

| Reference | Approx | TIME | Remarks |
|---|---|---|---|
| Oum and Seymour [37] | $3k+1$ | $8^k n^9 \log n$ | Works for connectivity functions |
| Oum [35] | $3k+1$ | $8^k n^4$ | |
| Oum [35] | $3k-1$ | $h(k)n^3$ | |
| Courcelle and Oum [16] | exact | $h(k)n^3$ | Does not provide decomposition |
| Hliněný and Oum [28] | exact | $h(k)n^3$ | |
| Jeong, Kim, and Oum [30] | exact | $2^{2^{O(k^2)}}n^3$ | Works for spaces over finite fields |
| This paper | $2k$ | $2^{2^{\mathcal{O}(k)}}n^2$ | |
| This paper | exact | $2^{2^{O(k^2)}}n^2$ | Our approximation + [30] |

already for rankwidth $k = 2$ no algorithm faster than $\mathcal{O}(n^3)$ was known. Algorithms for computing the branchwidth of certain matroids were studied by Hliněný [26, 27].

Following the work of Courcelle, Makowsky, and Rotics [14], there is a lot of literature on developing algorithms (FPT and XP) on graphs of bounded cliquewidth [24, 25, 31, 39, 43]. The design of algorithms on rank decompositions [7, 22, 23] and related graph parameters functionally equivalent to rankwidth like NLC-width [44] and Boolean width [8] is also an active research area in graph algorithms. For each of these width parameters, the only known way to compute the corresponding expression or decomposition is through rank decompositions. By Theorem 1.1, the running times of all FPT algorithms on graphs of $k$-cliquewidth, $k$-rankwidth, $k$-Boolean width, or $k$-NLC width improve from $f(k) \cdot n^3$ to $f(k) \cdot n^2$.

The branchwidth of a planar graph is computable in polynomial time, but in general the problem is NP-hard [42]. Bodlaender and Thilikos in developed an algorithm computing the branchwidth of a graph in time $2^{\mathcal{O}(k^3)}n$ [5].

As we already mentioned, the branchwidth of graphs is an invariant similar to treewidth and within a constant factor of treewidth. By the seminal algorithm of Bodlaender [2], deciding whether the treewidth of a graph is at most $k$ can be done in time $2^{\mathcal{O}(k^3)}n$. There are several linear time FPT algorithms with single-exponential dependence in $k$ that reach a constant approximation ratio [3, 32]. Korhonen obtained an algorithm that in time $2^{\mathcal{O}(k)}n$ computes a tree decomposition of width at most $2k + 1$ or concludes that the treewidth of the graph is more than $k$ [32]. When it comes to branchwidth, the result of Korhonen is incomparable with Theorem 1.5. Theorem 1.5 implies a 3-approximation for treewidth, which is worse than Korhonen's algorithm, while Korhonen's algorithm yields a 3-approximation for branchwidth, which is worse than Theorem 1.5.

*Organization of the paper.* We overview our framework for designing FPT 2-approximation algorithms for computing branch decomposition in section 2, outlining the main novel techniques. Section 3 contains definitions and simple facts about branch decompositions and connectivity functions. In section 4 we develop the combinatorial tools of our framework which will be used in the next sections. Section 5 is devoted to the algorithmic part of our framework. Section 6 implements our framework for rankwidth, and section 7 for branchwidth of graphs.

## 2. Overview of our framework.

Our framework for computing branch decompositions consists of two parts: the combinatorial and the algorithmic. We give an overview first of the combinatorial part, then of the algorithmic part, and then we provide the specific applications for approximating rankwidth and branchwidth of graphs.

### 2.1. Combinatorial framework.

Combinatorial results about how and when a branch decomposition can be improved form the core of our framework. For a branch decomposition $(T, L)$ of a connectivity function $f$, we want to decide whether $(T, L)$ could be refined into a "better" branch decomposition $(T', L')$. By better, we mean the following. Let $k$ be the width of $(T, L)$ and $h$ be the number of *heavy edges* of $T$, that is, the edges $e$ where $f(e)$ is the width of $(T, L)$. Then $(T', L')$ is *better* than $(T, L)$ if the width of $(T', L')$ is at most $k$ and the number of edges of width $k$ in $T'$ is less than $h$.

Our central combinatorial insight is that if the width of $(T, L)$ is more than $2\mathtt{bw}(f)$, then for any heavy edge $e$, there is a partition of $V$ into three sets $(C_1, C_2, C_3)$ with some particular properties, such that the quadruple $(e, C_1, C_2, C_3)$ can be used to refine $(T, L)$ into a better branch decomposition. We start first with explaining how edge $e$ and tripartition $(C_1, C_2, C_3)$ is used to refine $(T, L)$.

Let $(T, L)$ be a branch decomposition of a connectivity function $f : 2^V \to \mathbb{Z}_{\geq 0}$. Let $uv$ be an edge of $T$ and $(C_1, C_2, C_3)$ be a tripartition of $V$ (one of the sets $C_i$ could be empty). We denote by $(T, L\restriction_{C_i})$ the partial branch decomposition that has the same tree $T$ as $(T, L)$, but the mapping $L\restriction_{C_i}$ is a restriction of mapping $L$ to $C_i$. (We say that a *partial branch decomposition* is a branch decomposition where the labeling function $L$ is only required to be an injection.)

The *refinement* $(T, L')$ *of* $(T, L)$ *with* $(uv, C_1, C_2, C_3)$ is obtained by first taking the partial branch decompositions $(T_i, L_i) = (T, L\restriction_{C_i})$ for each $i \in \{1, 2, 3\}$. Let $u_iv_i$ be the copy of the edge $uv$ in $T_i$. The partial branch decompositions $(T_1, L_1)$, $(T_2, L_2)$, and $(T_3, L_3)$ are combined into a new partial branch decomposition by inserting a new node $w_i$ on each edge $u_iv_i$, and then connecting the nodes $w_1$, $w_2$, and $w_3$ to a new center node $t$. Finally, the obtained partial branch decomposition is transformed into a branch decomposition by iteratively pruning leaves that are not labeled and suppressing degree-2 nodes. See Figure 1.

How do the widths of edges change after the refinement? First, note that if $C_i$ is nonempty, then there will be an edge $tw_i \in E(T')$ corresponding to a bipartition $(C_i, \overline{C_i})$ in the refinement; see Figure 2. (For $X \subseteq V$, we use $\overline{X}$ to denote $V \setminus X$.) Thus the width of the refinement will be at least $f(C_i)$. Let $(W, \overline{W})$ be the bipartition corresponding to the edge $uv$ in $T$. Then in the refinement there will be edges $u_iw_i$ and $v_iw_i$ corresponding to bipartitions $(C_i \cap W, \overline{C_i \cap W})$ and $(C_i \cap \overline{W}, \overline{C_i \cap \overline{W}})$ for each $i$, provided that they are nonempty. Therefore, the width of the refinement will

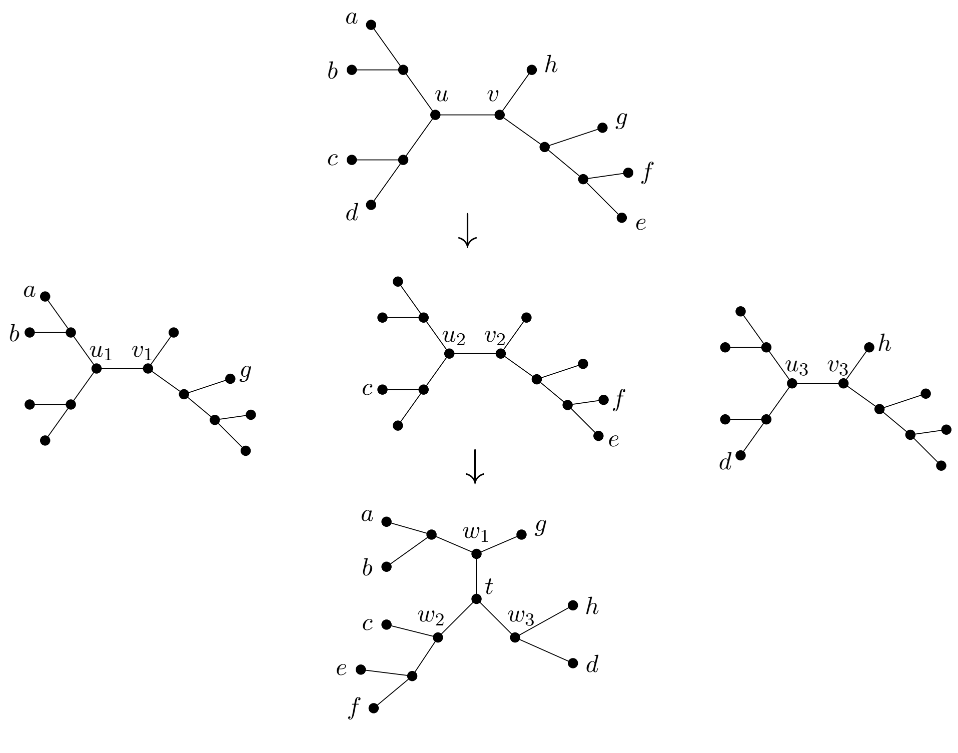

FIG. 1. *An example of the refinement operation. A branch decomposition* $(T, L)$ *on a set* $V = \{a, b, c, d, e, f, g, h\}$ *(top). For a tripartition* $(C_1 = \{a, b, g\}, C_2 = \{c, e, f\}, C_3 = \{d, h\})$*, we have the partial branch decompositions* $(T_1, L_1) = (T, L\restriction_{\{a,b,g\}})$*,* $(T_2, L_2) = (T, L\restriction_{\{c,e,f\}})$*, and* $(T_3, L_3) = (T, L\restriction_{\{d,h\}})$ *(middle), and the refinement of* $(T, L)$ *with* $(uv, C_1, C_2, C_3)$ *(bottom).*

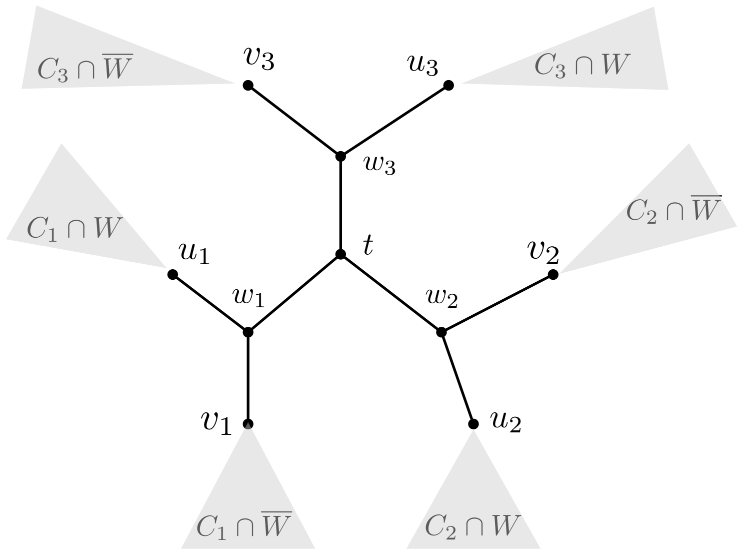

FIG. 2. *Changes of the width in a "neighborhood" of $uv$.*

be at least $\max(f(C_i \cap W), f(C_i \cap \overline{W}))$. Our first result is that if $f(uv) > 2\texttt{bw}(f)$, then there exists a refinement that "locally improves" the branch decomposition around the edge $uv$ in the sense that the widths of all of the aforementioned edges of the refinement are smaller than the width of the edge $uv$ in $T$.

THEOREM 2.1. *Let $f : 2^V \to \mathbb{Z}_{\geq 0}$ be a connectivity function. If $W \subseteq V$ is a set with $f(W) > 2\texttt{bw}(f)$, then there exists a tripartition $(C_1, C_2, C_3)$ of $V$ so that for each $i \in \{1, 2, 3\}$ it holds that $f(C_i) < f(W)/2$, $f(C_i \cap W) < f(W)$, and $f(C_i \cap \overline{W}) < f(W)$.*

Theorem 2.1 will be proved in section 4 by considering a branch decomposition $T^*$ of width $\texttt{bw}(f)$ and showing that we can either take a bipartition $(C_1, \overline{C_1}, \emptyset)$ of $V$ corresponding to an edge of $T^*$ or take a tripartition $(C_1, C_2, C_3)$ of $V$ corresponding to an internal node of $T^*$.

We call any tripartition $(C_1, C_2, C_3)$ satisfying the conclusion of Theorem 2.1 for a set $W$ a *$W$-improvement,* signifying that the refinement operation with this tripartition and the edge $uv$ corresponding to $(W, \overline{W})$ improves the branch decomposition around the edge $uv$. However, we cannot guarantee that the new decomposition $T'$ would be better than $T$. The reason is that $W$-improvement can increase the widths of edges deeper in $(T_i, L_i)$. It is a nontrivial statement that the existence of a $W$-improvement on a heavy edge implies the existence of a refinement that is better than $(T, L)$. The following theorem states this more formally.

THEOREM 2.2. *Let $f : 2^V \to \mathbb{Z}_{\geq 0}$ be a connectivity function and $(T, L)$ be a branch decomposition of $f$ of width $k$, having $h \geq 1$ edges of width $k$. Let $uv$ be an edge of $(T, L)$ corresponding to a partition $(W, \overline{W})$ and having width $f(uv) = k$. If there exists a $W$-improvement, then there exists a $W$-improvement $(C_1, C_2, C_3)$ such that the refinement of $(T, L)$ with $(uv, C_1, C_2, C_3)$ has width at most $k$ and fewer than $h$ edges of width $k$.*

The combination of Theorems 2.1 and 2.2 shows that when we consider a heavy edge $uv$, we either prove that $k = f(uv) \leq 2\texttt{bw}(f)$ (if there is no $W$-improvement, then $f(uv) \leq 2\texttt{bw}(f)$), or that there exists a $W$-improvement that could be used to globally improve $(T, L)$ by reducing the number of edges of width $k$. We refer to a $W$-improvement $(C_1, C_2, C_3)$ whose existence is guaranteed by Theorem 2.2 as a *global* $T$-*improvement* . We stated Theorem 2.2 in a nonconstructive manner, but its proof gives a constructive way of finding a global $T$-improvement. In particular, we show that a global $T$-improvement can be found by selecting a $W$-improvement which optimizes four explicit criteria over all $W$-improvements. More precisely, let $(C_1, C_2, C_3)$ be a $W$-improvement. The first three conditions that $(C_1, C_2, C_3)$ optimizes are

(i) $\max\{f(C_1), f(C_2), f(C_3)\}$ is minimized;
(ii) subject to (i), the number of nonempty sets $C_i$ in $(C_1, C_2, C_3)$ is minimized; and
(iii) subject to (i) and (ii), $f(C_1) + f(C_2) + f(C_3)$ is minimized.

(In section 4 we refer to a $W$-improvements satisfying (i)–(iii) as a minimum $W$-improvement.)

To state the fourth property we need the following definition. Let $r = uv \in E(T)$ be the edge of $T$ corresponding to the partition $(W, \overline{W})$. By slightly abusing the notation, let us view $r$ as the root of $T$. Then for every node $w$ of $T$, we define $T_r[w]$ as the set of leaves of the subtree of $T$ rooted in $w$. We say that a tripartition $(C_1, C_2, C_3)$ *intersects* a node $w \in V(T)$ if $T_r[w]$ contains elements from at least two different sets $C_i$. Now the fourth condition is

(iv) subject to (i), (ii), and (iii), $(C_1, C_2, C_3)$ intersects the minimum number of nodes of $T$.

The main result of section 4 will be to show that any $W$-improvement satisfying (i)–(iv) is a global $T$-improvement. The properties (i)–(iv) are very useful from an algorithmic perspective: For several functions $f$, $W$-improvements with properties (i)–(iv) can be computed by making use of dynamic programming over a branch decomposition of $f$.

The combinatorial characterization of improving refinements brings us to the following generic algorithm for 2-approximating branchwidth. The algorithm repeatedly takes a heavy edge $uv$ corresponding to a partition $(W, \overline{W})$. Then it checks if there exists a $W$-improvement and, if not, concludes that $(T, L)$ is a 2-approximation and returns $(T, L)$. If a $W$-improvement exists, the algorithm refines $(T, L)$ using a global $T$-improvement. In the next subsection we outline how to implement this generic algorithm efficiently for branch decompositions that support efficient dynamic programming for finding global $T$-improvements.

**2.2. Algorithmic framework.** In order to efficiently apply refinement operations to improve a branch decomposition, we need to find global $T$-improvements efficiently. For this, the main idea is that finding a global $T$-improvement corresponds to a partitioning problem, with the goal of finding a tripartition $(C_1, C_2, C_3)$ of $V$ satisfying the conditions of Theorem 2.1 and (i)–(iv). Such types of problems can be solved efficiently on branch decompositions that support dynamic programming, like rank decompositions or branch decompositions of graphs. The application of a sequence of at most $n$ refinement operations guarantees that the width of the resulting decomposition would decrease by at least one. Thus starting with a branch decomposition of width $k$ and repeatedly using dynamic programming to find a global $T$-improvement in $f(k) \cdot n$ time would result in a total time complexity of $f(k) \cdot k \cdot n^2$ to reduce the width down to $2\mathtt{bw}(f)$. Of course, this is not sufficient to obtain our applications. Let us recall that we target a linear time algorithm for the branchwidth of graphs. For rankwidth, where we aim for a quadratic running time, we also have to implement iterative compression, resulting in cubic running time.

The algorithmic ingredient of our general framework strongly uses the combinatorial properties of global $T$-improvements: It appears that refinements with global $T$-improvements "interplay well" with dynamic programming. This allows us to implement a sequence of refinement operations that reduces the width from $k$ to $k-1$ in a total of $t(k) \cdot 2^{\mathcal{O}(k)}n$ time, where $t(k)$ is a function representing the time complexity of dynamic programming on a single node of the branch decomposition. This part is inspired by the amortization techniques from [32].

We exploit the fact that when we refine a branch decomposition $(T, L)$ on an edge $uv$, only some connected subtree of $T$ around $uv$ changes. We call the nodes in this connected subtree the *edit set* of the refinement operation. In particular, we show that a refinement operation can be implemented by removing its edit set $R$ and inserting a connected subtree of $|R|$ new nodes in its place. Then we define a potential function $\phi$ of a branch decomposition so that $\phi(T, L)$ is initially bounded by $2^{\mathcal{O}(k)}n$, where $k$ is the width of $(T, L)$. We show that when refining with a global $T$-improvement, a refinement operation with edit set $R$ decreases $\phi(T, L)$ by at least $|R|$, and therefore the total sum of $|R|$ across a series of refinement operations is bounded by $2^{\mathcal{O}(k)}n$.

We exploit the bound on the sizes of edit sets by only recomputing the dynamic programming tables of the nodes in the edit set and reusing the rest. We formalize the notion of performing dynamic programming operations in time $t(k)$ with a definition of a "refinement data structure" in section 5.2. This leads to the following theorem.

THEOREM 2.3. *Let $f$ be a connectivity function for which there exists a refinement data structure with time complexity $t(k)$. There is an algorithm that, given a branch decomposition $(T, L)$ of $f$ of width $k$, in time $t(k) \cdot 2^{\mathcal{O}(k)}n$ either outputs a branch decomposition of $f$ of width at most $k-1$ or correctly concludes that $k \leq 2\mathtt{bw}(f)$.*

## 2.3. Applications.

*Rankwidth.* We show that for rank decompositions, the refinement data structure from Theorem 2.3 can be implemented in $t(k) = 2^{2^{\mathcal{O}(k)}}$ time, yielding a $2^{2^{\mathcal{O}(k)}}n$ time implementation of the algorithm in Theorem 2.3 for rankwidth. To obtain Theorem 1.1, we then apply this algorithm with the technique of iterative compression of Reed, Smith, and Vetta [40]; see also [17, Chapter 4]. In particular, by iteratively obtaining a rank decomposition of $G$ of width $\leq 2\mathtt{rw}(G) + 1$ from a rank decomposition of $G \setminus \{v\}$ of width $\leq 2\mathtt{rw}(G \setminus \{v\})$, and then applying Theorem 2.3 to decrease the width or to prove that it is already within a factor of 2 from the optimum.

A crucial detail of this iterative compression and the implementation of Theorem 2.3 for rankwidth is that we always maintain an *augmented rank decomposition,* i.e., a rank decomposition that for each edge corresponding to a partition $(A, B)$ of $V(G)$ stores representatives of the twin classes of the bipartite graph $G[A, B]$. In particular, the implementation of Theorem 2.3 for rank decompositions requires an augmented rank decomposition as an input and outputs an augmented rank decomposition. The augmentation is required for implementing dynamic programming on rank decompositions, and the reason for maintaining the augmentation throughout the iterative compression is that we are not aware of any $f(k) \cdot n$ time algorithms for making a given rank decomposition augmented. Instead, in every step of the iterative compression we use a $2^{\mathcal{O}(k)}n$ time algorithm for obtaining an augmented rank decomposition of $G$ from an augmented rank decomposition of $G \setminus \{v\}$.

*Branchwidth of graphs.* We show that for branch decompositions of graphs, the refinement data structure from Theorem 2.3 can be implemented in $t(k) = 2^{\mathcal{O}(k)}$ time, thus yielding a $2^{\mathcal{O}(k)}n$ time implementation of the algorithm for branchwidth of graphs. By using a 2-approximation algorithm for treewidth with time complexity

$2^{\mathcal{O}(k)}n$ [32] and converting the tree decomposition into a branch decomposition [41], we can assume that we initially have a branch decomposition of width at most $3k$, where $k$ is the branchwidth. We then obtain Theorem 1.5 by $k$ applications of the algorithm of Theorem 2.3. We note that this algorithm can be made more self-contained, to only depend on the compression technique of Bodlaender [2] instead of treewidth algorithms, by applying the compression technique of Bodlaender in the same manner as it is applied in [3, 32].

## 3. Preliminaries.

We denote the set of vertices of a graph $G$ by $V(G)$ and the set of edges by $E(G)$. For a vertex $v$ of a graph $G$, we denote by $N_G(v)$ the neighborhood of $v$ in $G$, and for a vertex set $X$, we let $N_G(X) = \bigcup_{v\in X} N_G(v) \setminus X$. We drop the subscript $G$ when it is clear from the context. For a graph $G$ and a set $X \subseteq V(G)$, we denote by $G[X]$ the subgraph induced by $X$. Let $v$ be a degree-2 vertex in $G$, with neighbors $N(v) = \{u, w\}$. The *suppression* of $v$ deletes the vertex $v$ and edges incident to it and inserts the edge $uw$ if it does not exist.

For a graph $G$ and a pair $A, B$ of disjoint subsets of $V(G)$, $G[A, B]$ is the bipartite graph induced by edges between $A$ and $B$ and labeled with the bipartition $(A, B)$. The notation $M_G[A, B]$ denotes the $|A| \times |B|$ 0-1 matrix representing $G[A, B]$. All matrices in this paper are 0-1 matrices, and the GF(2) rank of matrix $M$ is denoted by $\mathrm{rk}(M)$.

A *bipartition* of a set $V$ is an ordered pair $(X, Y)$ of disjoint sets such that $X \cup Y = V$. A *tripartition* of a set $V$ is an ordered triple $(X, Y, Z)$ of disjoint sets so that $X \cup Y \cup Z = V$. Note that in these definitions a partition can contain empty parts. We say that a tripartition $(X, Y, Z)$ of $V$ *intersects* a set $A \subseteq V$ if at least two of $X, Y, Z$ have a nonempty intersection with $A$.

*Connectivity functions.* Let $V$ be a finite set and $f : 2^V \to \mathbb{Z}_{\geq 0}$ be a function. The function $f$ is *submodular* if

$$f(U \cup W) + f(U \cap W) \leq f(U) + f(W) \tag{3.1}$$

for all $U, W \subseteq V$. For a set $W \subseteq V$, we denote $\overline{W} = V \setminus W$. The function $f$ is *symmetric* if for every $W \subseteq V$, $f(W) = f(\overline{W})$. Function $f$ is a *connectivity function* if (i) $f$ is symmetric, (ii) $f$ is submodular, and (iii) $f(\emptyset) = 0$.

We note that here the requirements that $f(\emptyset) = 0$ and $f$ takes only nonnegative values are actually not severe restrictions: For any function $f : 2^V \to \mathbb{Z}$ satisfying (i) and (ii), we can prove that $f(\emptyset) \leq f(A)$ for any $A \subseteq V$ by letting $U = A$ and $W = \overline{A}$ in (3.1), implying that the function $f - f(\emptyset)$ satisfies all (i), (ii), (iii) and is nonnegative.

We will need the following two observations about connectivity functions. In the following, let $f : 2^V \to \mathbb{Z}_{\geq 0}$ be a connectivity function.

*Observation* 3.1. For any sequence of subsets $A_1, \dots, A_n \subseteq V$, it holds that $f(A_1 \cap \cdots \cap A_n) = f(\overline{A_1} \cup \cdots \cup \overline{A_n})$.

*Proof.* Note that $\overline{A_1 \cap \cdots \cap A_n} = \overline{A_1} \cup \cdots \cup \overline{A_n}$. ∎

The following inequalities give bounds on a situation when a bipartition $(B, \overline{B})$ partitions a set $A$.

*Observation* 3.2. For any $A, B \subseteq V$, it holds that $f(A) \leq f(A \cap B) + f(A \cap \overline{B}) \leq f(A) + 2f(B)$.

*Proof.* For the first inequality, we have by submodularity

$$f(A) = f((A \cap B) \cup (A \cap \overline{B})) \leq f(A \cap B) + f(A \cap \overline{B}) - f(\emptyset). \tag{3.2}$$

For the second inequality, we have that

$$\begin{aligned} f(A\cap B)+f(A\cap \overline{B}) &\le 2f(A)+2f(B)-f(A\cup B)-f(A\cup \overline{B}) && \text{(submodularity)}\\ &= 2f(A)+2f(B)-f(\overline{A}\cap \overline{B})-f(\overline{A}\cap B) && \text{(Observation 3.1)}\\ &\le 2f(A)+2f(B)-f(A). && \text{(from (3.2))} \qquad \square \end{aligned}$$

*Branch decompositions.* A *cubic tree* is a tree where each node has degree 1 or 3. A *branch decomposition* of a connectivity function $f: 2^V \to \mathbb{Z}_{\ge 0}$ is a pair $(T,L)$, where $T$ is a cubic tree and $L$ is a bijection mapping $V$ to the leaves of $T$. If $|V| \le 1$, then $f$ admits no branch decomposition. Usually, by slightly abusing the notation, we will treat the leaves of the branch decomposition as the elements of $V$, and thus denote a branch decomposition by only the tree $T$ instead of the pair $(T,L)$.

Let $e = uv \in E(T)$ be an edge of a branch decomposition $T$. We denote by $T[uv] \subseteq V$ the set of leaves of $T$ that are closer to the node $u$ than to the node $v$, and by $T[vu] \subseteq V$ the set of leaves that are closer to $v$ than $u$. Hence $(T[uv], T[vu])$ is the bipartition of $V$ induced by the connected components of the forest $T \setminus \{e\}$ obtained from $T$ by deleting $e$. The *width* of an edge $e = uv$ of $T$ is $f(e) = f(T[uv]) = f(T[vu])$, and the *width of a branch decomposition* $(T,L)$, denoted by $\mathtt{bw}(T,L)$, is the maximum width of an edge. Finally, the *branchwidth* of $f$, denoted by $\mathtt{bw}(f)$, is the minimum width of a branch decomposition of $f$.

Let $r \in E(T)$ be an edge of a branch decomposition $T$. We introduce notation with the intuition that $T$ is rooted at the edge $r$. The *$r$-subtree* of a node $w \in V(T)$ is the subtree of $T$ induced by nodes $x \in V(T)$ such that $w$ is on the unique $x-r$ path (including $w$). For any node $w \in V(T)$, we denote by $T_r[w] \subseteq V$ the set of leaves in the $r$-subtree of $w$. Note that when $r = uv$, it holds that $T_r[u] = T[uv]$, $T_r[v] = T[vu]$, and for any $w \in V(T)$ it holds that either $T_r[w] \subseteq T_r[u]$ or $T_r[w] \subseteq T_r[v]$. The *$r$-parent* of a node $w \in V(T) \setminus \{u,v\}$ is the node next to $w$ on the unique $w-r$ path. If $p$ is the $r$-parent of $w$, then $w$ is an *$r$-child* of $p$. Note that every nonleaf node has exactly two $r$-children, and every leaf node has no $r$-children. A set $X \subseteq V$ *$r$-intersects* a node $w \in V(T)$ if $X \cap T_r[w]$ is nonempty. Similarly, a tripartition $(C_1, C_2, C_3)$ of $V$ *$r$-intersects* a node $w \in V(T)$ if it intersects the set $T_r[w]$.

## 4. Combinatorial results.

In this section we prove our combinatorial results. Throughout the section we use the convention that $f: 2^V \to \mathbb{Z}_{\ge 0}$ is a connectivity function.

### 4.1. Refinement operation.

The central concept of our results is the refinement operation. Before defining it, we need the definition of a *partial branch decomposition.*

DEFINITION 4.1 (partial branch decomposition). *A* partial branch decomposition *on a set $C$ is a pair $(T,L)$, where $T$ is a cubic tree and $L$ is an injection from $C$ to the leaves of $T$.*

Let $(T,L)$ be a branch decomposition of $f$, and let $C_i \subseteq V$. We denote by $(T, L\restriction_{C_i})$ the partial branch decomposition on the set $C_i$ obtained by restricting the mapping $L$ to only $C_i$. Now we can define the refinement operation (see also Figure 1).

DEFINITION 4.2 (refinement). *Let $(T,L)$ be a branch decomposition, let $uv \in E(T)$ an edge of $T$, and let $(C_1, C_2, C_3)$ be a tripartition of $V$. We define the* refinement *of $T$ with $(uv, C_1, C_2, C_3)$ as the following branch decomposition.*

*For each $i \in \{1,2,3\}$, let $(T_i, L_i) = (T, L\restriction_{C_i})$, and let $u_i v_i$ be the copy of the edge $uv$ in $T_i$. Now, let $(T', L')$ be a partial branch decomposition on $V$ obtained by* *first inserting a node $w_i$ on the edge $u_iv_i$ of each $T_i$ (i.e., $V(T_i) \leftarrow V(T_i) \cup \{w_i\}$ and $E(T_i) \leftarrow E(T_i) \cup \{u_iw_i, w_iv_i\} \setminus \{u_iv_i\}$), then taking the disjoint union of $(T_1, L_1)$, $(T_2, L_2)$, $(T_3, L_3)$, and then inserting a node $t$ adjacent to $w_1$, $w_2$, $w_3$, connecting the disjoint union into a tree. Finally, the refinement is obtained by simplifying $(T', L')$ by iteratively removing degree-1 nodes that are not labeled by $L'$, and suppressing degree-2 nodes.*

We observe that if $T'$ is a refinement of $T$ with $(r, C_1, C_2, C_3)$, then every edge of $T'$ corresponds either to a bipartition $(C_i, \overline{C_i})$ or to a bipartition $(T_r[w] \cap C_i, \overline{T_r[w] \cap C_i})$, where $w \in V(T)$. The following states this more formally.

*Observation* 4.3. Let $T$ be a branch decomposition, let $r \in E(T)$, and let $(C_1, C_2, C_3)$ be a tripartition of $V$. Let $T'$ be the refinement of $T$ with $(r, C_1, C_2, C_3)$. It holds that

$$\begin{aligned}&\{\{T'[u'v'], T'[v'u']\} \mid uv' \in E(T')\}\\ &\quad = \bigcup_{i \in \{1,2,3\}} \Big(\{\{C_i, \overline{C_i}\}\} \cup \{\{T_r[w] \cap C_i, \overline{T_r[w] \cap C_i}\} \mid w \in V(T)\}\Big) \setminus \{\{\emptyset, V\}\}.\end{aligned}$$

Note that as a branch decomposition is a cubic tree, and no two edges correspond to the same bipartition, i.e., the set $\{\{T[uv], T[vu]\} \mid uv \in E(T)\}$ has cardinality $|E(T)|$.

**4.2. Improving refinement.** Now we provide the formal definitions and results of our combinatorial framework, postponing the proofs to subsections 4.3 and 4.4. First we define a $W$-improvement of a set $W \subseteq V$.

DEFINITION 4.4 ($W$-improvement). *Let $W \subseteq V$. A tripartition $(C_1, C_2, C_3)$ of $V$ is a* $W$-improvement *if for every $i \in \{1, 2, 3\}$,*

- *$f(C_i) < f(W)/2$;*
- *$f(C_i \cap W) < f(W)$; and*
- *$f(C_i \cap \overline{W}) < f(W)$.*

*The* width *of a $W$-improvement is $\max\{f(C_1), f(C_2), f(C_3)\}$, the* sum-width *of a $W$-improvement is $f(C_1) + f(C_2) + f(C_3)$, and the* arity *of a $W$-improvement is the number of nonempty parts in the partition.*

Let us note that because of the condition $f(C_i \cap W) < f(W)$, we have that for every $i \in \{1, 2, 3\}$, $C_i \neq V$. Thus the arity of every $W$-improvement is always 2 or 3. Also, $W$-improvements are symmetric in the sense that the ordering of the sets $C_1, C_2, C_3$ as well as replacing $W$ by $\overline{W}$ does not change any properties of a $W$-improvement.

The following lemma is a restatement of Theorem 2.1.

LEMMA 4.5. *If $W \subseteq V$ such that $f(W) > 2\mathtt{bw}(f)$, then there exists a $W$-improvement.*

We postpone the proof of Lemma 4.5 to subsection 4.3.

If $uv$ is an edge of a branch decomposition $T$ with $(T[uv], T[vu]) = (W, \overline{W})$, then refining with $(uv, C_1, C_2, C_3)$ where $(C_1, C_2, C_3)$ is a $W$-improvement "locally improves" the branch decomposition around the edge $uv$ in the sense that the widths of the new edges corresponding to the edge $uv$ will be of the forms $f(C_i \cap W) < f(W)$ and $f(C_i \cap \overline{W}) < f(W)$. Towards proving that the existence of a $W$-improvement implies the existence of a $W$-improvement that globally improves the branch decomposition, we define minimum $W$-improvements.

DEFINITION 4.6 (minimum $W$-improvement). *For $W \subseteq V$ a $W$-improvement $(C_1, C_2, C_3)$ is a* minimum $W$-improvement *if*

(1) $(C_1, C_2, C_3)$ *is of minimum width among all $W$-improvements;*
(2) *subject to* (1), $(C_1, C_2, C_3)$ *is of minimum arity; and*
(3) *subject to* (1) *and* (2), $(C_1, C_2, C_3)$ *is of minimum sum-width.*

Let $uv = r$ be an edge of a branch decomposition $T$. Recall that for any node $w$ of $T$ it holds that either $T_r[w] \subseteq T[uv]$ or $T_r[w] \subseteq T[vu]$, and recall that by Observation 4.3, when refining with $(uv, C_1, C_2, C_3)$, the edge corresponding to $T_r[w]$ will turn into edges corresponding to $T_r[w] \cap C_1$, $T_r[w] \cap C_2$, and $T_r[w] \cap C_3$. The following lemma shows that when refining with a minimum $W$-improvement, the width of a branch decomposition does not increase.

LEMMA 4.7. *Let $W \subseteq V$, and let $(C_1, C_2, C_3)$ be a minimum $W$-improvement. Then for any set $W'$ such that $W' \subseteq W$ or $W' \subseteq \overline{W}$ and every $i \in \{1, 2, 3\}$, it holds that $f(W' \cap C_i) \leq f(W')$.*

We postpone the proof of Lemma 4.7 to subsection 4.4.

Next we define the notion of global $T$-improvement, which will finally be the type of improvement that we actually use to perform the refinement operation.

DEFINITION 4.8 (global $T$-improvement). *Let $T$ be a branch decomposition, and let $r = uv \in E(T)$ be an edge of $T$. A $T$-*improvement *on $r$ is a tuple $(r, C_1, C_2, C_3)$, where $W = T[uv]$, and $(C_1, C_2, C_3)$ is a minimum $W$-improvement. We say that a $T$-improvement on $r$* intersects *a node $w \in V(T)$ if $(C_1, C_2, C_3)$ $r$-intersects it, i.e., the set $T_r[w]$ intersects at least two sets from $\{C_1, C_2, C_3\}$. A $T$-improvement $(r, C_1, C_2, C_3)$ is a* global $T$-improvement *if it intersects the minimum number of nodes of $T$ among all $T$-improvements on $r$.*

The following theorem is our main combinatorial result. In particular, Theorem 2.2 will be a straightforward application of it.

THEOREM 4.9. *Let $T$ be a branch decomposition, $r \in E(T)$ an edge of $T$, and $(r, C_1, C_2, C_3)$ a global $T$-improvement. Then for every $i \in \{1, 2, 3\}$ and every node $w \in V(T)$, it holds that $f(T_r[w] \cap C_i) \leq f(T_r[w])$. Moreover, if $T_r[w] \cap C_i \neq \emptyset$, then $f(T_r[w] \cap C_i) = f(T_r[w])$ if and only if $T_r[w] \subseteq C_i$.*

We postpone the proof of Theorem 4.9 to subsection 4.4. Let us prove Theorem 2.2 using Theorem 4.9 and Observation 4.3.

THEOREM 2.2. *Let $f : 2^V \to \mathbb{Z}_{\geq 0}$ be a connectivity function and $(T, L)$ be a branch decomposition of $f$ of width $k$, having $h \geq 1$ edges of width $k$. Let $uv$ be an edge of $(T, L)$ corresponding to a partition $(W, \overline{W})$ and having width $f(uv) = k$. If there exists a $W$-improvement, then there exists a $W$-improvement $(C_1, C_2, C_3)$ such that the refinement of $(T, L)$ with $(uv, C_1, C_2, C_3)$ has width at most $k$ and fewer than $h$ edges of width $k$.*

*Proof.* Denote $r = uv$. Take a global $T$-improvement $(r, C_1, C_2, C_3)$, which exists by the existence of a $W$-improvement. Let $T'$ be the refinement of $T$ with $(r, C_1, C_2, C_3)$. By Observation 4.3, each edge of $T'$ corresponds to a bipartition of the form $(T_r[w] \cap C_i, \overline{T_r[w] \cap C_i})$, where $w \in V(T)$, or of the form $(C_i, \overline{C_i})$. Because $(C_1, C_2, C_3)$ is a $W$-improvement, all edges of the form $(C_i, \overline{C_i})$ have width less than $k/2$. By the first part of Theorem 4.9, all edges of the form $(T_r[w] \cap C_i, \overline{T_r[w] \cap C_i})$ have width at most $f(T_r[w]) \leq k$, and therefore the width of $T'$ is at most $k$.

By the second part of Theorem 4.9, if $T'$ contains an edge corresponding to a bipartition $(T_r[w] \cap C_i, \overline{T_r[w] \cap C_i})$ with width $k$, then $T_r[w] \cap C_i = T_r[w]$, and thus an edge corresponding to the exact same bipartition also exists in $T$. This gives an injective mapping from the edges of $T'$ of width $k$ to edges of $T$ of width $k$. It remains to observe that by the definition of $W$-improvement, this mapping maps no edge of $T'$ to the edge $uv$ of $T$, and thus $T'$ contains strictly fewer edges of width $k$ than $T$. ◻

**4.3. Proof of Lemma 4.5.** The idea of the proof of Lemma 4.5 is that we take an optimal branch decomposition $T^*$ of $f$, i.e., a branch decomposition $T^*$ of width $\mathtt{bw}(f) < f(W)/2$, and argue that either the bipartition corresponding to some edge of $T^*$ or the tripartition corresponding to some node of $T^*$ results in an $W$-improvement.

We define an orientation of a set $C \subseteq V$ with respect to a set $W \subseteq V$. This will be used for orienting the edges of the optimal branch decomposition based on $W$.

Definition 4.10 (orientation). *Let $C, W \subseteq V$. We say that the set $W$* directly orients *$C$ if $f(C \cap W) < f(\overline{C} \cap W)$ and $f(C \cap \overline{W}) < f(\overline{C} \cap \overline{W})$. The set $W$* inversely orients *$C$ if it directly orients $\overline{C}$. The set $W$* disorients *$C$ if it neither directly nor inversely orients $C$.*

Note that the definition of orienting is symmetric with respect to complementing $W$, i.e., $W$ (directly, inversely, dis)-orients $C$ if and only if $\overline{W}$ (directly, inversely, dis)-orients $C$. Next we show that if there is an edge of an optimal branch decomposition that cannot be oriented according to Definition 4.10, then it corresponds to a $W$-improvement.

Lemma 4.11. *Let $C, W \subseteq V$. If $W$ disorients $C$ and $f(C) < f(W)/2$, then $(C, \overline{C}, \emptyset)$ is a $W$-improvement.*

*Proof.* By possibly interchanging $C$ with $\overline{C}$, without loss of generality we can assume that $f(C \cap W) \leq f(\overline{C} \cap W)$ and $f(C \cap \overline{W}) \geq f(\overline{C} \cap \overline{W})$. Since we have $f(C) = f(\overline{C}) < f(W)/2$, to show that $(C, \overline{C}, \emptyset)$ is a $W$-improvement, it suffices to prove that $f(\overline{C} \cap W) < f(W)$ and $f(C \cap \overline{W}) < f(W)$. Assume, by way of contradiction, that $f(\overline{C} \cap W) \geq f(W)$. Then by the submodularity of $f$, $f(\overline{C} \cup W) \leq f(C)$, hence $f(C \cap \overline{W}) \leq f(C)$. Therefore, $f(C \cap \overline{W}) + f(\overline{C} \cap \overline{W}) \leq 2f(C) < f(W)$. But this contradicts Observation 3.2. The proof of $f(C \cap \overline{W}) < f(W)$ is symmetric. ◻

Now, to prove Lemma 4.5, we take an optimal branch decomposition $T^*$ of $f$ and orient each edge $uv$ with $(T^*[uv], T^*[vu]) = (C, \overline{C})$ towards $v$ if $W$ directly orients $C$ and towards $u$ if $W$ inversely orients $C$. If no orientation can be found, then Lemma 4.11 shows that $(C, \overline{C}, \emptyset)$ is a $W$-improvement and we are done.

Lemma 4.12. *No edge of $T^*$ is oriented towards a leaf.*

*Proof.* Suppose there is an edge of $T^*$ oriented towards a leaf $v \in V$. This means that $W$ directly orients $V \setminus \{v\}$, implying that $f(W \setminus \{v\}) < f(W \cap \{v\})$ and that $f(\overline{W} \setminus \{v\}) < f(\overline{W} \cap \{v\})$. However, one of the sets $W \cap \{v\}$ or $\overline{W} \cap \{v\}$ must be empty and therefore either $f(W \cap \{v\}) = 0$ or $f(\overline{W} \cap \{v\}) = 0$, but $f$ cannot take values less than 0, so we get a contradiction. ◻

Now, by walking in $T^*$ according to the orientation, we end up finding an internal node towards which all incident edges are oriented. In particular, the internal node corresponds to a tripartition $(C_1, C_2, C_3)$ of $V$, so that $f(C_i) < f(W)/2$ for all $i \in \{1,2,3\}$ (as the width of $T^*$ is $< f(W)/2$), and $W$ directly orients each $C_i$. The following lemma shows that this node indeed gives a $W$-improvement, and therefore completes the proof of Lemma 4.5.

Lemma 4.13. *Let $W \subseteq V$, and let $(C_1, C_2, C_3)$ be a tripartition of $V$ such that for each $i \in \{1,2,3\}$, $f(C_i) < f(W)/2$ and $W$ directly orients $C_i$. Then $(C_1, C_2, C_3)$ is a $W$-improvement.*

*Proof.* Since $W$ directly orients $C_i$, we have that $f(C_i \cap W) < f(\overline{C}_i \cap W)$. By Observation 3.2, $f(C_i \cap W) + f(\overline{C}_i \cap W) \leq f(W) + 2f(C_i)$. Therefore, we have $f(C_i \cap W) \leq f(W)/2 + f(C_i) < f(W)$. The proof that $f(C_i \cap \overline{W}) < f(W)$ is similar. ☐

**4.4. Proofs of Lemma 4.7 and Theorem 4.9.** Let $(C_1, C_2, C_3)$ be a minimum $W$-improvement for a set $W \subseteq V$. The main idea behind the proofs of Lemma 4.7 and Theorem 4.9 is that if $f(W' \cap C_1) \geq f(W')$ for some $W' \subseteq W$ or $W' \subseteq \overline{W}$, then we prove that $(C_1', C_2', C_3') = (C_1 \cup W', C_2 \setminus W', C_3 \setminus W')$ is also a minimum $W$-improvement (and symmetrically for $C_2$ and $C_3$). In particular, this will be used to contradict the minimality of $(C_1, C_2, C_3)$ or the fact that $(r, C_1, C_2, C_3)$ is a global $T$-improvement.

The proof is different for minimum $W$-improvements of arity 2 and of 3. The following lemma will be used in both cases, in particular, it will be useful for arguing that a set $C_1' = C_1 \cup W'$ satisfies the condition $f(C_1' \cap W) < f(W)$ of $W$-improvements.

LEMMA 4.14. *Let $W \subseteq V$, and let $(C_1, C_2, C_3)$ be a $W$-improvement. Suppose that for some $W' \subseteq W$, $f(C_1 \cap W') \geq f(W')$. Then $f((C_1 \cup W') \cap W) < f(W)$.*

*Proof.* Because $W' \subseteq W$, we have that

$$f((C_1 \cup W') \cap W) = f(W' \cup (C_1 \cap W)).$$

Then

$$\begin{aligned} f(W' \cup (C_1 \cap W)) &\leq f(W') + f(C_1 \cap W) - f(W' \cap (C_1 \cap W)) && \text{(submodularity)} \\ &= f(W') + f(C_1 \cap W) - f(W' \cap C_1) && \text{(by } W' \subseteq W) \\ &\leq f(C_1 \cap W) < f(W), \end{aligned}$$

where the last line follows from the assumption $f(C_1 \cap W') \geq f(W')$ and the definition of $W$-improvement. ☐

Note that by the symmetry of $W$-improvements, Lemma 4.14 holds also when replacing $C_1$ by $C_2$ or $C_3$, and also for $W' \subseteq \overline{W}$.

**4.4.1. $W$-improvements of arity 2.** The following lemma completes the proof of Lemma 4.7 for $W$-improvements of arity 2 (note symmetry). It also sets up the proof of Theorem 4.9 for $W$-improvements of arity 2, which will be completed in subsection 4.4.3.

LEMMA 4.15. *Let $W' \subseteq W \subseteq V$, and let $(C, \overline{C}, \emptyset)$ be a minimum $W$-improvement. Then $f(C \cap W') \leq f(W')$, and, moreover, if the equality $f(C \cap W') = f(W')$ holds, then $(C \cup W', \overline{C} \setminus W', \emptyset)$ is also a minimum $W$-improvement.*

*Proof.* Let $(C, \overline{C}, \emptyset)$ be a minimum $W$-improvement and suppose that for some $W' \subseteq W$ we have $f(C \cap W') \geq f(W')$. To prove the lemma, we first show that in this case $(C', \overline{C'}, \emptyset) = (C \cup W', \overline{C} \setminus W', \emptyset)$ is also a $W$-improvement.

Let us check that $(C', \overline{C'}, \emptyset)$ satisfies all the conditions of a $W$-improvement.

First, by $f(C \cap W') \geq f(W')$ and submodularity of $f$, we have that

$$f(\overline{C'}) = f(C') = f(C \cup W') \leq f(C) < f(W)/2. \tag{4.1}$$

Then by $W' \subseteq W$ we have that $f(C' \cap \overline{W}) = f(C \cap \overline{W}) < f(W)$ and that $f(\overline{C'} \cap \overline{W}) = f(\overline{C} \cap \overline{W}) < f(W)$. By Lemma 4.14, we have $f(C' \cap W) < f(W)$. Finally,

$$\begin{aligned} f(\overline{C'} \cap W) = f(C' \cup \overline{W}) &= f((C \cup W') \cup (C \cup \overline{W})) && \text{(Obs. 3.1)} \\ &\le f(C \cup W') + f(C \cup \overline{W}) - f((C \cup W') \cap (C \cup \overline{W})) && \text{(submodularity)} \\ &= f(C \cup W') + f(C \cup \overline{W}) - f(C) && (W' \subseteq W) \\ &\le f(C) + f(\overline{C} \cap W) - f(C) < f(W). && \text{(Obs. 3.1 + (4.1))} \end{aligned}$$

This completes the proof that $(C', \overline{C'}, \emptyset)$ is a $W$-improvement.

Now, if $f(C \cap W') > f(W')$, then by the submodularity of $f$, we would get $f(C') = f(C \cup W') < f(C)$. But this contradicts the minimality of $(C, \overline{C}, \emptyset)$. If $f(C \cap W') = f(W')$, then $f(C') \le f(C)$. In this case, since $(C', \overline{C'}, \emptyset)$ is a $W$-improvement and $(C, \overline{C}, \emptyset)$ is a minimum $W$-improvement, we conclude that $(C', \overline{C'}, \emptyset)$ is also a minimum $W$-improvement . ∎

Note again that due to symmetry of $W$-improvements, Lemma 4.15 holds also when swapping the roles of $C$ and $\overline{C}$ and for $W' \subseteq \overline{W}$.

**4.4.2. $W$-improvements of arity 3.** For $W$-improvements of arity 3, we will heavily exploit the minimality of arity, i.e., the condition on minimum $W$-improvements that there should be no $W$-improvement with smaller or equal width and with smaller arity. We proceed with a sequence of auxiliary lemmas establishing properties of minimum $W$-improvements of arity 3.

LEMMA 4.16. *Let $(C_1, C_2, C_3)$ be a minimum $W$-improvement of arity 3. Then for every $i \in \{1,2,3\}$, $W$ directly orients $C_i$.*

*Proof.* If $W$ disorients $C_i$, then by Lemma 4.11, $(C_i, \overline{C}_i, \emptyset)$ is a $W$-improvement contradicting the minimality of $(C_1, C_2, C_3)$. On the other hand, $W$ cannot inversely orient $C_i$. Indeed, the assumption $f(W \cap C_i) > f(W \cap \overline{C}_i)$ and $f(\overline{W} \cap C_i) > f(\overline{W} \cap \overline{C}_i)$ yields that $f(W \cap \overline{C}_i) < f(W)$ and $f(\overline{W} \cap \overline{C}_i) < f(W)$ by the fact that $(C_1, C_2, C_3)$ is a $W$-improvement. In this case, again, $(C_i, \overline{C}_i, \emptyset)$ is a $W$-improvement contradicting the minimality of $(C_1, C_2, C_3)$. ∎

LEMMA 4.17. *Let $(C_1, C_2, C_3)$ be a minimum $W$-improvement of arity 3. Then for every $i \in \{1,2,3\}$ it holds that $f(W \cup C_i) > f(W)/2$ and $f(\overline{W} \cup C_i) > f(W)/2$.*

*Proof.* By Lemma 4.16, $W$ directly orients $C_i$. Hence $f(\overline{W} \cap C_i) < f(\overline{W} \cap \overline{C}_i)$. By Observation 3.2, $f(W) = f(\overline{W}) \le f(\overline{W} \cap C_i) + f(\overline{W} \cap \overline{C}_i)$. Therefore, we get $f(W \cup C_i) = f(\overline{W} \cap \overline{C}_i) > f(W)/2$. The proof of $f(\overline{W} \cup C_i) > f(W)/2$ is symmetric. ∎

The following lemma shows that any set $C_i$ in a minimum $W$-improvement cannot be easily "separated," in some sense, from any set $C_i \cup (W \cap C_j)$.

LEMMA 4.18. *Let $(C_1, C_2, C_3)$ be a minimum $W$-improvement of arity 3. Then for every set $P$ and $i, j \in \{1,2,3\}$ such that $C_i \subseteq P \subseteq C_i \cup (W \cap C_j)$, it holds that $f(P) \ge f(C_i)$.*

*Proof.* For $i = j$ the lemma is trivial. For $i \ne j$, without loss of generality, we can assume that $i = 1$ and $j = 2$. Aiming towards a contradiction, let us assume that for some $P$ with $C_1 \subseteq P \subseteq C_1 \cup (W \cap C_2)$ it holds that $f(P) < f(C_1)$. We claim that then $(C_1', C_2', C_3') = (P, C_2 \setminus P, C_3)$ is a $W$-improvement contradicting the minimality of $(C_1, C_2, C_3)$.

First, by our assumption, $f(C_1') = f(P) < f(C_1) < f(W)/2$. To verify that $f(C_2') = f(C_2 \setminus P) = f(C_2 \cap \overline{P}) \le f(C_2)$, first note $f(C_2 \cup \overline{P}) = f(C_2 \cup C_3) = f(C_1)$. Then by the submodularity of $f$,

$$f(C_2 \cap \overline{P}) + f(C_1) = f(C_2 \cap \overline{P}) + f(C_2 \cup \overline{P}) \leq f(C_2) + f(P).$$

By our assumption, $f(P) < f(C_1)$. Thus $f(C_2') = f(C_2 \cap \overline{P}) < f(C_2)$. As $C_3' = C_3$, we also have $f(C_3') = f(C_3)$.

It remains to show that $f(C_i' \cap W) < f(W)$ and $f(C_i' \cap \overline{W}) < f(W)$ for all $i \in \{1,2,3\}$. Again, for $i = 3$ this is trivial as $C_3' = C_3$.

Because $P \subseteq C_1 \cup (W \cap C_2)$, we have that $C_1' \cap \overline{W} = C_1 \cap \overline{W}$ and $C_2' \cap \overline{W} = C_2 \cap \overline{W}$. Thus $f(C_i' \cap \overline{W}) < f(W)$ for all $i$.

To prove that $f(P \cap W) < f(W)$, first observe that $P \cup W = C_1 \cup W$. Then by Lemma 4.17,

$$f(P \cup W) = f(C_1 \cup W) > f(W)/2.$$

By the submodularity of $f$,

$$f(P \cap W) + f(P \cup W) \leq f(P) + f(W).$$

Since $f(P) < f(C_1) < f(W)/2$, we have that $f(C_1' \cap W) = f(P \cap W) < f(W)$.

Similarly, to prove that $f(C_2' \cap W) < f(W)$, we note that $C_2' \cup W = C_2 \cup W$. Then by Lemma 4.17, $f(C_2' \cup W) > f(W)/2$. We have already proved that $f(C_2') < f(C_2)$. Then by making use of the submodularity of $f$, we have that

$$\frac{f(W)}{2} + f(C_2' \cap W) < f(C_2' \cap W) + f(C_2' \cup W) \leq f(C_2') + f(W) < \frac{f(W)}{2} + f(W).$$

This concludes the proof that $(C_1', C_2', C_3')$ is a $W$-improvement. Finally, the width of $(C_1', C_2', C_3')$ is at most the width of $(C_1, C_2, C_3)$, but its sum-width is strictly less. This contradicts the minimality of $(C_1, C_2, C_3)$. ∎

The following lemma with Lemma 4.15 complete the proof of Lemma 4.7 (note symmetry). It also sets up the proof of Theorem 4.9 for $W$-improvements of arity 3, which will be completed in subsection 4.4.3.

LEMMA 4.19. *Let $(C_1, C_2, C_3)$ be a minimum $W$-improvement of arity* 3. *Then for every $W' \subseteq W$, $f(C_1 \cap W) \leq f(W')$. Moreover, if the equality holds, then $(C_1 \cup W', C_2 \setminus W', C_3 \setminus W')$ is also a minimum $W$-improvement.*

*Proof.* Let $W' \subseteq W$, and let $(C_1, C_2, C_3)$ be a minimum $W$-improvement of arity 3 and assume $f(C_1 \cap W') \geq f(W')$. We define $(C_1', C_2', C_3') = (C_1 \cup W', C_2 \setminus W, C_3 \setminus W')$. First we will prove that $(C_1', C_2', C_3')$ is a $W$-improvement.

We start by showing that $f(C_i') \leq f(C_i)$ for all $i$. The inequality

$$f(C_1') = f(C_1 \cup W') \leq f(C_1) \tag{4.2}$$

follows directly from our assumption and the submodularity of $f$. Moreover, (4.2) turns into an equality only if $f(C_1 \cap W') = f(W')$.

To prove that

$$f(C_2') \leq f(C_2), \tag{4.3}$$

we do the following:

$$\begin{aligned} f(C_2') = f(C_2 \setminus W') &= f(\overline{C}_2 \cup W') = f(\overline{C}_2 \cup (C_1 \cup W')) \\ &\leq f(C_2) + f(C_1 \cup W') - f(\overline{C}_2 \cap (C_1 \cup W')) && \text{(submodularity)} \\ &\leq f(C_2) + f(C_1) - f(\overline{C}_2 \cap (C_1 \cup W')) && \text{(by (4.2))} \\ &= f(C_2) + f(C_1) - f(C_1 \cup (C_3 \cap W')) \leq f(C_2). && \text{(Lemma 4.18)} \end{aligned}$$

The proof of $f(C_3') \leq f(C_3)$ is symmetric.

Next we prove that $f(C_i' \cap W) < f(W)$ and that $f(C_i' \cap \overline{W}) < f(W)$. First, note that for each $i \in \{1,2,3\}$, $C_i' \cap \overline{W} = C_i \cap \overline{W}$. So the cases with $\overline{W}$ are trivial. The inequality $f(C_1' \cap W) < f(W)$ is proved in Lemma 4.14.

The bound on $f(C_2 \cap W)$ follows from the following chain of inequalities:

$$\begin{aligned}
f(C_2' \cap W) &= f(C_2 \cap \overline{W'} \cap W) = f(\overline{C}_2 \cup W' \cup \overline{W}) && \text{(Observation 3.1)}\\
&\le f(\overline{C}_2 \cup W') + f(W) - f((\overline{C}_2 \cup W') \cap \overline{W}) && \text{(submodularity)}\\
&= f(C_2 \cap \overline{W'}) + f(W) - f(\overline{C}_2 \cap \overline{W}) && \text{(by } W' \subseteq W)\\
&= f(C_2 \cap \overline{W'}) + f(W) - f(C_2 \cup W) && \text{(Observation 3.1)}\\
&< f(C_2 \setminus W') + f(W) - f(W)/2 = f(C_2') + f(W)/2 && \text{(Lemma 4.17)}\\
&\le f(C_2) + f(W)/2 < f(W). && \text{(by (4.3))}
\end{aligned}$$

The proof of $f(C_3' \cap W) < f(W)$ is symmetric. This completes the proof that $(C_1', C_2', C_3')$ is a $W$-improvement with $f(C_i') \le f(C_i)$ for all $i$.

Now, if $f(C_1 \cap W') > f(W')$, then $f(C_1') < f(C_1)$ and $(C_1', C_2', C_3')$ contradicts the minimality of $(C_1, C_2, C_3)$. If $f(C_1 \cap W') = f(W')$, then $(C_1', C_2', C_3')$ has the same width and the sum-width as $(C_1, C_2, C_3)$, which is a minimum $W$-improvement. Thus in the case of $f(C_1 \cap W') = f(W')$, $(C_1', C_2', C_3')$ is also a minimum $W$-improvement. ∎

**4.4.3. Completing the proof of Theorem 4.9.** The following lemma is just Lemmas 4.15 and 4.19 put together.

LEMMA 4.20. *Let $(C_1, C_2, C_3)$ be a minimum $W$-improvement. Then for every $W' \subseteq W$ it holds that $f(C_1 \cap W') \le f(W')$. Moreover, if the equality holds, then $(C_1 \cup W', C_2 \setminus W', C_3 \setminus W')$ is also a minimum $W$-improvement.*

As discussed above, the first part of Lemma 4.20 directly gives Lemma 4.7 by the symmetry of $W$-improvements. Next we prove Theorem 4.9 by using also the second part of Lemma 4.20.

THEOREM 4.9. *Let $T$ be a branch decomposition, $r \in E(T)$ an edge of $T$, and $(r, C_1, C_2, C_3)$ a global $T$-improvement. Then for every $i \in \{1,2,3\}$ and every node $w \in V(T)$, it holds that $f(T_r[w] \cap C_i) \le f(T_r[w])$. Moreover, if $T_r[w] \cap C_i \ne \emptyset$, then $f(T_r[w] \cap C_i) = f(T_r[w])$ if and only if $T_r[w] \subseteq C_i$.*

*Proof.* The first part follows from combining the fact that either $T_r[w] \subseteq T[uv]$ or $T_r[w] \subseteq T[vu]$ with Lemma 4.7.

To prove the second part, suppose that for some node $w \in V(T)$ it holds that $T_r[w] \cap C_1 \ne \emptyset$, $T_r[w] \nsubseteq C_1$, and $f(T_r[w] \cap C_1) = f(T_r[w])$. Take a tripartition $(C_1', C_2', C_3') = (C_1 \cup T_r[w], C_2 \setminus T_r[w], C_3 \setminus T_r[w])$. By Lemma 4.20, this tripartition is a minimum $W$-improvement. We argue that this minimum $W$-improvement contradicts the fact that $(r, C_1, C_2, C_3)$ is a global $T$-improvement by showing that it $r$-intersects fewer nodes of $T$ than $(r, C_1, C_2, C_3)$.

The $W$-improvement $(C_1, C_2, C_3)$ $r$-intersects the node $w$ but $(C_1', C_2', C_3')$ does not, so it suffices to prove the implication that if $(C_1, C_2, C_3)$ does not $r$-intersect a node $w' \in V(T)$, then $(C_1', C_2', C_3')$ does not $r$-intersect $w'$.

As both $T_r[w]$ and $T_r[w']$ represent leaves of $r$-subtrees of $T$, it follows that either $T_r[w'] \subseteq T_r[w]$, $T_r[w] \subseteq T_r[w']$, or $T_r[w]$ and $T_r[w']$ are disjoint. If $T_r[w'] \subseteq T_r[w]$, then $(C_1', C_2', C_3')$ does not $r$-intersect $w'$. If $T_r[w] \subseteq T_r[w']$, then because $(C_1, C_2, C_3)$ $r$-intersects $w$, it also $r$-intersects $w'$. If $T_r[w']$ and $T_r[w]$ are disjoint, then the intersections of $(C'_1, C'_2, C'_3)$ with $T_r[w']$ are the same as the intersections of $(C_1, C_2, C_3)$ with $T_r[w']$, so $w'$ $r$-intersects $(C'_1, C'_2, C'_3)$ if and only if it $r$-intersects $(C_1, C_2, C_3)$. ◻

## 5. Algorithmic properties of refinement.

In this section we present our algorithmic framework for designing fast FPT 2-approximation algorithms for computing branch decompositions. In particular, we show that a sequence of refinement operations decreasing the width of a branch decomposition from $k$ to $k-1$ or concluding $k \leq 2\mathtt{bw}(f)$ can be implemented in time $t(k) \cdot 2^{\mathcal{O}(k)} \cdot n$ for connectivity functions $f$ whose branch decompositions support dynamic programming with time complexity $t(k)$ per node. That is, we prove Theorem 2.3. The concrete implementation of this framework for rankwidth, with $t(k) = 2^{2^{\mathcal{O}(k)}}$, is provided in section 6 and for graph branchwidth, with $t(k) = 2^{\mathcal{O}(k)}$, in section 7.

### 5.1. Amortized analysis of refinement.

A naive implementation of the refinement operation (see Definition 4.2) would use $\Omega(n)$ time on each refinement, which would result in the time complexity of $\Omega(n^2)$ over the course of $n$ refinements. In this section we show that the refinement operations can be implemented so that over any sequence of refinement operations using global $T$-improvements on a branch decomposition of width at most $k$, the total work done in refining the branch decomposition amortizes to $2^{\mathcal{O}(k)}n$.

The efficient implementation of refinements is based on the notion of the edit set of a global $T$-improvement. Informally, the edit set corresponding to a global $T$-improvement $(r, C_1, C_2, C_3)$ are the nodes of the subtree of $T$ obtained after pruning all subtrees whose leaves are entirely from one of the sets $C_i$. We state this formally as follows.

DEFINITION 5.1 (edit set). *Let $T$ be a branch decomposition, let $r \in E(T)$, and let $(r, C_1, C_2, C_3)$ be a global $T$-improvement. We say that the* edit set *of $(r, C_1, C_2, C_3)$ is the set $R \subseteq V(T)$ of nodes of $T$ that $r$-intersect $(C_1, C_2, C_3)$, i.e.,*

$$R = \{w \in V(T) \mid T_r[w] \textit{ intersects at least two sets from } \{C_1, C_2, C_3\}\}.$$

Note that for a global $T$-improvement $(uv, C_1, C_2, C_3)$, both $u$ and $v$ are necessarily in the edit set. We formalize the intuition about edit sets in the following lemma. It will be implicitly used in many of our arguments.

LEMMA 5.2. *Let $T$ be a branch decomposition, $r = uv \in E(T)$, $(r, C_1, C_2, C_3)$ a global $T$-improvement, $R$ the edit set of $(r, C_1, C_2, C_3)$, and $T'$ the refinement of $T$ with $(r, C_1, C_2, C_3)$. It holds that*

(1) *every node in $R$ is nonleaf;*
(2) *the nodes of $R$ induce a connected subtree $T[R]$ of $T$; and*
(3) *there exists an edge $r' \in E(T')$ so that for every $w \in V(T) \setminus R$ there is a node $w' \in V(T')$ with $T_r[w] = T'_{r'}[w']$.*

*Proof.* For (1), the set $T_r[w]$ of a leaf $w$ consists of one element. Thus $w$ does not $r$-intersect $(C_1, C_2, C_3)$. For (2), first note that $\{u, v\} \subseteq R$. Then consider a node $w \in R \setminus \{u, v\}$, and let $p$ be the $r$-parent of $w$. It holds that $T_r[w] \subseteq T_r[p]$, so $p$ must also be in $R$.

For (3), observe that by the definition of edit set for every node $w \in V(T) \setminus R$ it holds that $T_r[w] \subseteq C_i$ for some $i$. This implies that the $r$-subtree of $w$ appears identically in $T'$, and therefore $T'$ consists of the $r$-subtrees of all $w \in N_T(R)$ and a connected subtree inserted in the place of $R$ and connected to $N_T(R)$. As $\{u, v\} \subseteq |R|$, this inserted subtree contains at least one edge, which we can designate as $r'$. ◻

Next we define the *neighbor partition* of an edit set $R$.

DEFINITION 5.3 (neighbor partition). *Let $(r, C_1, C_2, C_3)$ be a global $T$-improvement and $R$ its edit set. The* neighbor partition *of $R$ is the partition $(N_1, N_2, N_3)$ of the neighbors $N_T(R)$ of $R$, where $N_i = \{w \in N_T(R) \mid T_r[w] \subseteq C_i\}$.*

Note that the neighbor partition is indeed a partition of $N_T(R)$ by the definition of edit set.

Next we give an algorithm for performing the refinement operation in $\mathcal{O}(|R|)$ time, given the edit set $R$ and its neighbor partition.

LEMMA 5.4. *Let $T$ be a branch decomposition, let $r = uv \in E(T)$, and let $(r, C_1, C_2, C_3)$ be a global $T$-improvement. Given the edit set $R$ of $(r, C_1, C_2, C_3)$ and the neighbor partition $(N_1, N_2, N_3)$ of $R$, $T$ can be turned into the refinement of $T$ with $(r, C_1, C_2, C_3)$ in $\mathcal{O}(|R|)$ time.*

*Proof.* We create three copies $T_1$, $T_2$, $T_3$ of the induced subtree $T[R]$. We denote the copy of a vertex $x \in R$ in $T_i$ by $x_i$, and denote $R_i = \{x_i \mid x \in R\}$. To each $T_i$ we also insert a new node $w_i$ on the edge $u_i v_i$, i.e., let $V(T_i) \leftarrow V(T_i) \cup \{w_i\}$ and $E(T_i) \leftarrow E(T_i) \setminus \{u_i v_i\} \cup \{u_i w_i, w_i v_i\}$. We then insert a new center node $t$ and connect each $w_i$ to it.

For each node $y \in N_i$, let $p \in R$ be the $r$-parent of $y$. We remove the edge $yp$ and insert the edge $yp_i$. It remains to remove all nodes of $R$, and then iteratively remove degree-1 nodes and suppress degree-2 nodes in $R_1 \cup R_2 \cup R_3 \cup \{t, w_1, w_2, w_3\}$.

For the time complexity, these operations can be done in time linear in $|R| + |R_1| + |R_2| + |R_3| + |N_1| + |N_2| + |N_3| = \mathcal{O}(|R|)$ because $T$ is represented as an adjacency list and the maximum degree of $T$ is 3. ∎

The outline of the refinement operation in our framework is that the dynamic programming outputs the edit set $R$ and its neighbor partition in $\mathcal{O}(t(k)|R|)$ time, then the algorithm of Lemma 5.4 computes the refinement in $\mathcal{O}(|R|)$ time, and then the dynamic programming tables of the $|R|$ new nodes are computed in $\mathcal{O}(t(k)|R|)$ time, making use of the previous dynamic programming tables of the nodes not in the edit set, which are preserved throughout the refinement operation. To bound the sum of the sizes of the edit sets $R$ over the course of the algorithm, we introduce the following potential function.

DEFINITION 5.5 ($k$-potential). *Let $T$ be a branch decomposition of function $f$. The* $k$-potential *of $T$ is*

$$\phi_k(T) = \sum_{\substack{e \in E(T) \\ f(e) < k}} f(e) \cdot 3^{f(e)} + \sum_{\substack{e \in E(T) \\ f(e) \geq k}} 3f(e) \cdot 3^{f(e)}.$$

When working with $k$-potentials, we will use the following notation. For $x \geq 0$, let

$$\phi_k(x) = \begin{cases} x \cdot 3^x & \text{if } x < k, \\ 3x \cdot 3^x & \text{otherwise.} \end{cases}$$

For $W \subseteq V$, we will use $\phi_k(W)$ to denote $\phi_k(f(W))$. With this notation, the $k$-potential of $T$ is

$$\phi_k(T) = \sum_{uv \in E(T)} \phi_k(T[uv]).$$

Note that for any $k$, the $k$-potential of a branch decomposition $T$ is at most $\mathcal{O}(3^{\mathtt{bw}(T)} \mathtt{bw}(T)|E(T)|)$, which is $2^{\mathcal{O}(k)} n$ when $\mathtt{bw}(T) = \mathcal{O}(k)$.

Next we show that performing a refinement operation with an edit set $R$ decreases the $k$-potential by at least $|R|$.

LEMMA 5.6. *Let $T$ be a branch decomposition with an edge $r = uv \in E(T)$ so that $f(uv) = \mathtt{bw}(T) = k$. Let $(r, C_1, C_2, C_3)$ be a global $T$-improvement, $R$ be the edit set of $(r, C_1, C_2, C_3)$, and $T'$ be the refinement of $T$ with $(r, C_1, C_2, C_3)$. Then it holds that $\phi_k(T') \le \phi_k(T) - |R|$.*

*Proof.* We use the notation that $W = T[uv]$. Note that

$$\phi_k(T) = \phi_k(k) + \sum_{w \in (V(T) \setminus \{u,v\})} \phi_k(T_r[w]) \tag{5.1}$$

and that by Observation 4.3

$$\phi_k(T') = \sum_{i \in \{1,2,3\}} \left( \phi_k(C_i) + \sum_{w \in V(T)} \phi_k(C_i \cap T_r[w]) \right). \tag{5.2}$$

Then

$$\begin{aligned}
\phi_k(T) - \phi_k(T') &= \phi_k(T) - \sum_{i \in \{1,2,3\}} \left( \phi_k(C_i) + \sum_{w \in V(T)} \phi_k(C_i \cap T_r[w]) \right) \\
&\ge \phi_k(T) - \sum_{i \in \{1,2,3\}} \Bigg( \phi_k(C_i) + \phi_k(C_i \cap W) + \phi_k(C_i \cap \overline{W}) \\
&\qquad\qquad + \sum_{w \in (V(T) \setminus \{u,v\})} \phi_k(C_i \cap T_r[w]) \Bigg),
\end{aligned}$$

where the last line is obtained by taking $C_i \cap T_r[u] = C_i \cap W$ and $C_i \cap T_r[v] = C_i \cap \overline{W}$ out of the sum.

By the definition of $W$-improvement, we have that $\phi_k(C_i) \le \phi_k((k-1)/2)$ and $\phi_k(C_i \cap W) \le \phi_k(k-1)$. Then by interleaving the sums (5.1) and (5.2), we have that $\phi_k(T) - \phi_k(T')$ is at least

$$\begin{aligned}
&\phi_k(k) - 3\phi_k((k-1)/2) - 6\phi_k(k-1) \\
&\quad + \sum_{w \in (V(T) \setminus \{u,v\})} \left( \phi_k(T_r[w]) - \sum_{i \in \{1,2,3\}} \phi_k(C_i \cap T_r[w]) \right).
\end{aligned}$$

By simplifying $\phi_k(k) - 3\phi_k((k-1)/2) - 6\phi_k(k-1) \ge 3 \cdot 3^k$, we lower bound the latter by

$$3 \cdot 3^k + \sum_{w \in (V(T) \setminus \{u,v\})} \left( \phi_k(T_r[w]) - \sum_{i \in \{1,2,3\}} \phi_k(C_i \cap T_r[w]) \right).$$

Let us note that for $w \notin R$, $\phi_k(T_r[w]) = \sum_{i \in \{1,2,3\}} \phi_k(C_i \cap T_r[w])$ because $T_r[w]$ is a subset of some $C_i$ and $\phi_k(\emptyset) = 0$. Also by Theorem 4.9, for any node $w$ in the edit set and every $i$ it holds that $f(T_r[w] \cap C_i) < f(T_r[w])$. By making use of these observations, we conclude that

$$\phi_k(T) - \phi_k(T') \ge 2 + \sum_{w \in (R \setminus \{u,v\})} \left( \phi_k(T_r[w]) - \sum_{i \in \{1,2,3\}} \phi_k(C_i \cap T_r[w]) \right) \ge |R|. \qquad \square$$

In particular, when performing a sequence of refinement operations with global $T$-improvements on edges $r$ of width $f(r) = \texttt{bw}(T) = k$, the sum of the sizes of edit sets across all of the operations is at most $\mathcal{O}(3^k \cdot k \cdot |E(T)|)$.

## 5.2. Refinement data structure.

We define a *refinement data structure* to formally capture what is required from the underlying dynamic programming in our framework.

DEFINITION 5.7 (refinement data structure). *Let $f$ be a connectivity function. A* refinement data structure *of $f$ with time complexity $t(k)$ maintains a branch decomposition $T$ of $f$ with $\texttt{bw}(T) \le k$ rooted on an edge $r = uv \in E(T)$ and supports the following operations:*

1. *Init($T$, $uv$): Given a branch decomposition $T$ of $f$ with $\texttt{bw}(T) \le k$ and an edge $uv \in E(T)$, initialize the data structure in $\mathcal{O}(t(k)|V(T)|)$ time.*
2. *Move($vw$): Move the root edge $r = uv$ to an incident edge $vw$, i.e., set $r \leftarrow vw$. Runs in $\mathcal{O}(t(k))$ time.*
3. *Width(): Return $f(uv)$ in $\mathcal{O}(t(k))$ time.*
4. *CanRefine(): Returns true if there exists a $W$-improvement where $W = T[uv]$ and false otherwise. Runs in time $\mathcal{O}(t(k))$. Once CanRefine() has returned true, the following can be invoked:*
   (a) *EditSet(): Let $(r, C_1, C_2, C_3)$ be a global $T$-improvement, $R$ the edit set of $(r, C_1, C_2, C_3)$, and $(N_1, N_2, N_3)$ the neighbor partition of $R$. Returns $R$ and $(N_1, N_2, N_3)$. Runs in $\mathcal{O}(t(k)|R|)$ time.*
   (b) *Refine($R$, $(N_1, N_2, N_3)$): Implements the refinement operation described in Lemma 5.4. That is, computes the refinement of $T$ by removing the edit set $R$ and inserting a connected subtree of $|R|$ nodes in its place. Sets $r$ to an arbitrary edge between two newly inserted nodes (such an edge exists because $|R| \ge 2$). Runs in $\mathcal{O}(t(k)|R|)$ time.*
5. *Output(): Outputs $T$ in $\mathcal{O}(t(k)|V(T)|)$ time.*

We explain how our algorithm uses the refinement data structure in subsection 5.3. Let us here informally explain how the refinement data structure is typically implemented using dynamic programming. The formal descriptions for rankwidth and graph branchwidth constitute sections 6 and 7.

For each node $w$ of $T$, the refinement data structure stores a dynamic programming table of size $\mathcal{O}(t(k))$ that represents information of the $r$-subtree of $w$ in such a way that the dynamic programming tables of the nodes $u$ and $v$ combined together can be used to detect the existence of a $W$-improvement on $W = T[uv]$. Now, the Init($T$, $uv$) operation is to compute these dynamic programming tables in a bottom-up fashion for all nodes from the leaves towards the root $uv$, using $\mathcal{O}(t(k))$ time per node. The Move($vw$) operation changes the root edge $uv$ to an incident edge $vw$. For implementing Move($vw$), we observe the following useful property.

*Observation* 5.8. Let $T$ be a branch decomposition, $r = uv \in E(T)$ an edge of $T$, and $r' = vw \in E(T)$ another edge of $T$. For all nodes $x \in V(T) \setminus \{v\}$, the $r$-subtree of $x$ is the same as the $r'$-subtree of $x$.

In particular, as the dynamic programming table of a node depends only on its $r$-subtree, it suffices to recompute only the dynamic programming table of the node $v$ in $\mathcal{O}(t(k))$ time when using Move($vw$). The Width() operation is typically implemented without dynamic programming, using some other auxiliary data structure. The CanRefine() operation is implemented by combining the information of the dynamic programming tables of $u$ and $v$ in an appropriate way. Then the EditSet()

operation is implemented by tracing the dynamic programming backwards, working with a representation of the global $T$-improvement $(r, C_1, C_2, C_3)$ that allows for efficiently determining whether $C_i$ and $T_r[w]$ intersect. The Refine($R$, $(N_1, N_2, N_3)$) operation is a direct application of Lemma 5.4 followed by computing the dynamic programming tables of the $|R|$ new nodes inserted by Lemma 5.4 in $\mathcal{O}(t(k)|R|)$ time, and possibly also updating other auxiliary data structures. The implementation of Output() is typically straightforward, as it just amounts to outputting the branch decomposition that the data structure has been maintaining.

**5.3. General algorithm.** We present a general algorithm that uses the refinement data structure to either improve the width of a given branch decomposition from $k$ to $k-1$ or conclude that it is already a 2-approximation, and runs in time $t(k)2^{\mathcal{O}(k)}n$, where $t(k)$ is the time complexity of the refinement data structure.

Our algorithm is described as a pseudocode in Algorithm 5.1. The algorithm is a depth-first search on the given branch decomposition $T$, where whenever we return from a subtree via an edge $uv$ of width $f(uv) = k$, we check if there exists a global $T$-improvement $(uv, C_1, C_2, C_3)$. If there is no such global $T$-improvement, then we conclude that $T$ is already a 2-approximation. If there is such a global $T$-improvement, then we refine $T$ using it. We need to be careful to proceed so that the refinement does not break invariants of depth-first search, and the extra work caused by refining with an edit set $R$ can be bounded by $\mathcal{O}(t(k)|R|)$.

Let us explain how Algorithm 5.1 is implemented with the refinement data structure. We always maintain that the root edge $uv$ in the refinement data structure corresponds to the edge $uv$ in Algorithm 5.1. We start by calling Init($T$, $uv$) after line 7. In the cases of line 9 and line 15 the edge $uv$ is changed to an adjacent edge $vw$, which is done by the Move($vw$) operation. The edge $uv$ is changed also after the refinement operation. There we can move to the appropriate edge with $|R|$ Move($vw$) operations. Now that the edge $uv$ of Algorithm 5.1 corresponds to the edge $uv$ of the refinement data structure, all nonelementary operations of Algorithm 5.1 can be performed with the refinement data structure. In particular, checking $f(uv)$ on line 13 is done by Width(), line 22 corresponds to CanRefine(), and line 26 corresponds to EditSet() and Refine(). The returned edit set is also used to determine the node $v$ on line 24.

The rest of this section is devoted to proving the correctness and the time complexity of Algorithm 5.1. The next lemma shows that adding the refinement operation does not significantly change the properties of depth-first search and provides the key argument for proving the correctness.

LEMMA 5.9. *Algorithm* 5.1 *maintains the invariant that the nodes with state* `open` *form a path $w_1, \ldots, w_l$ in $T$, where $l \geq 2$, $w_1 = s$, $w_{l-1} = v$, and $w_l = u$.*

*Proof.* This invariant is satisfied at the beginning of the algorithm. There are three cases in the if-else structure that do not terminate the algorithm and alter $u$, $v$, or the states, i.e., the cases of line 9, line 15, and line 22. The case of line 9 maintains the invariant by extending the path by one node. The case of line 15 maintains the invariant by removing the last node of the path. In the case of line 22, recall that both $u$ and $v$ are in the edit set $R$ and the edit set is a connected subtree of $T$, so the refinement removes some suffix $w_j, \ldots, w_l$ of the path. Together with the fact that $w_1$ is a leaf and thus $w_1 \notin R$ (see Lemma 5.2), this implies that the node $v'$ determined in line 24 must be the node $w_{j-1}$ of the path. Finally, the path is extended by one node in lines 28–31. ∎

**Algorithm 5.1** Iterative refinement.

```
Input: Branch decomposition $T$ of a connectivity function $f$.
Output: A branch decomposition of $f$ of width at most $\mathtt{bw}(T) - 1$ or the conclusion
that $\mathtt{bw}(T) \le 2\mathtt{bw}(f)$.
 1: Let $k \leftarrow \mathtt{bw}(T)$
 2: Let state be an array initialized with the value unseen for all nodes of $T$,
    including new nodes that will be created by the refinement operation.
 3: Let $s$ be an arbitrary leaf node of $T$
 4: $v \leftarrow s$
 5: $u \leftarrow$ the neighbor of $v$
 6: state[$v$] $\leftarrow$ open
 7: state[$u$] $\leftarrow$ open
 8: while state[$u$] = open do
 9:     if exists $w \in N_T(u)$ with state[$w$] = unseen then
10:         $v \leftarrow u$
11:         $u \leftarrow w$
12:         state[$u$] $\leftarrow$ open
13:     else if $f(uv) < k$ then
14:         if $v = s$ then return $T$
15:         else
16:             state[$u$] $\leftarrow$ closed
17:             $u \leftarrow v$
18:             $v \leftarrow$ the node $v \in N_T(u)$ with state[$v$] = open
19:             ▷ Such a node $v$ is unique.
20:         end if
21:     else
22:         if exists a global $T$-improvement $(uv, C_1, C_2, C_3)$ then
23:             $R \leftarrow$ the edit set
24:             $v' \leftarrow$ the node $v' \in N_T(R)$ with state[$v'$] = open
25:             ▷ Such node $v'$ is unique.
26:             $T \leftarrow$ Refine($T$, $(uv, C_1, C_2, C_3)$)
27:             ▷ Where the refinement operation works as in Lemma 5.4, i.e., by
    removing the edit set $R$ and inserting a connected subtree of $|R|$ nodes in its
    place.
28:             $v \leftarrow v'$
29:             $u \leftarrow$ the node $u \in N_T(v)$ that was inserted by the refinement
30:             ▷ Such node $u$ is unique.
31:             state[$u$] $\leftarrow$ open
32:         else
33:             conclude $\mathtt{bw}(T) \le 2\mathtt{bw}(f)$
34:         end if
35:     end if
36: end while
```

The next lemma will be used to prove the correctness of Algorithm 5.1 in the case when it returns an improved branch decomposition.

LEMMA 5.10. *If Algorithm* 5.1 *reaches line* 14*, i.e., terminates by returning a branch decomposition, then all nodes except* $v$ *and* $u$ *have state* `closed`.

*Proof.* We show that Algorithm 5.1 maintains the invariant that if a node $w$ is closed, then all nodes $w'$ in the $s$-subtree of $w$ are also closed. This is trivially maintained by the case of line 9. In other cases all of the neighbors of $u$ except $v$ are closed due to Lemma 5.9. This implies that the case of line 15 also maintains the invariant. The case of line 22, i.e., refinement, maintains this because the union of the edit set $R$ and the path $w_1, \ldots, w_l$ defined in Lemma 5.9 is a connected subtree that contains $u$, $v$, and $s$.

This invariant implies the conclusion of the lemma because at line 14 all neighbors of $u$ except $v$ are closed. ∎

We are ready to prove the correctness and the time complexity of Algorithm 5.1. In particular, next we complete the proof of the main theorem of this section.

THEOREM 2.3. *Let $f$ be a connectivity function for which there exists a refinement data structure with time complexity $t(k)$. There is an algorithm that, given a branch decomposition $(T, L)$ of $f$ of width $k$, in time $t(k) \cdot 2^{\mathcal{O}(k)}n$ either outputs a branch decomposition of $f$ of width at most $k-1$ or correctly concludes that $k \leq 2\mathtt{bw}(f)$.*

*Proof.* It suffices to prove that Algorithm 5.1 is correct and runs in time $t(k)2^{\mathcal{O}(k)}n$ provided a refinement data structure with time complexity $t(k)$.

We first show the correctness. The algorithm terminates with the conclusion $\mathtt{bw}(T) \leq 2\mathtt{bw}(f)$ if and only if there is no $W$-improvement of $W = T[uv]$, where $f(W) = \mathtt{bw}(T)$. Therefore, by Lemma 4.5, $\mathtt{bw}(T) = f(W) \leq 2\mathtt{bw}(f)$. For the other case, let $w \neq s$ be a node of $T$ and $p$ be the $s$-parent of $w$. Note that the state of $w$ can become closed only if $f(wp) < k$, and after that, $f(wp)$ will not change unless $w$ and $p$ are in the edit set, in which case they are replaced by new nodes that are unseen. Therefore, by Lemma 5.10, when Algorithm 5.1 reaches line 14, we have that $f(wp) < k$ for all $w \in V(T) \setminus \{u, v\}$, and by line 13 we also have $f(uv) < k$. Therefore we have that $f(e) < k$ for all edges $e$ of $T$, implying that Algorithm 5.1 is correct when it returns a branch decomposition.

Then we prove the running time. By the definition of $W$-improvement and Theorem 4.9, the width of $T$ never increases. By Lemma 5.6, with every refinement, the potential function drops by at least $|R|$, the size of the edit set. While we cannot control the size of the edit set for each new refinement, the total sum of the sizes of the edit sets over all the sequence of refinements does not exceed $\phi_k(T) = 2^{\mathcal{O}(k)}n$. Thus the total time complexity of the refinement operations is $t(k)2^{\mathcal{O}(k)}n$ and the total number of new nodes created over the course of the algorithm in refinement operations is $2^{\mathcal{O}(k)}n$. All cases of the algorithm advance the state of some node either from unseen to open or from open to closed, and therefore the total number of operations is $2^{\mathcal{O}(k)}n$ and their total time $t(k)2^{\mathcal{O}(k)}n$. ∎

## 6. Approximating rankwidth.

In this section we prove Theorem 1.1, that is, there is an algorithm that for an $n$-vertex graph $G$ and an integer $k$ in time $2^{2^{\mathcal{O}(k)}}n^2$ either computes a rank decomposition of width at most $2k$ or correctly concludes that the rankwidth of $G$ is more than $k$. To prove that theorem, we define "augmented rank decompositions," show how to implement the refinement data structure of Theorem 2.3 for augmented rank decompositions with time complexity $t(k) = 2^{2^{\mathcal{O}(k)}}$, and then apply the algorithm of Theorem 2.3 with iterative compression.

### 6.1. Preliminaries on rank decompositions.

We start with preliminaries on rank decompositions. Most of the material in this subsection is well known and is commonly used in dynamic programming over rank decompositions [7, 22, 23]. For the reader's convenience, we provide short proofs here.

Let us recall that a *rank decomposition* of a graph $G$ is a branch decomposition of the function $\mathrm{cutrk}_G : 2^{V(G)} \to \mathbb{Z}_{\geq 0}$ defined on vertex sets of a graph $G$. For a graph $G$ and a subset $A$ of the vertex set $V(G)$, we define $\mathrm{cutrk}_G(A)$ as the rank of the $|A| \times |\overline{A}|$ $0-1$ matrix $M_G[A, \overline{A}]$ over the binary field where the entry $m_{i,j}$ of $M_G[A, \overline{A}]$ on the $i$th row and the $j$th column is 1 if and only if the $i$th vertex in $A$ is adjacent to the $j$th vertex in $\overline{A}$. That is, $\mathrm{cutrk}_G(A) = \mathrm{rk}(M_G[A, \overline{A}])$. Then the *rankwidth* of a graph $G$, $\mathtt{rw}(G)$, is the minimum width of a rank decomposition of $G$.

Oum and Seymour showed that $\mathrm{cutrk}_G$ is a connectivity function [37].

Proposition 6.1 ([37]). *The function* $\mathrm{cutrk}_G$ *is a connectivity function.*

In the rest of this section we will always assume that we are computing the rankwidth of a graph $G$, and therefore we drop the subscript in $\mathrm{cutrk}_G$. We also assume that a representation of $G$ that allows for checking the existence of an edge in $\mathcal{O}(1)$ time (e.g., an adjacency matrix) is available.

The following definition of a representative will be used for dynamic programming on rank decompositions.

Definition 6.2 (representative). *Let $A \subseteq V(G)$. A set $R \subseteq A$ is a* representative *of $A$ if for every vertex $v \in A$ there is a vertex $u \in R$ with $N(v) \setminus A = N(u) \setminus A$. The representative $R$ is* minimal *if for each $v \in A$ there exists exactly one such $u \in R$.*

Note that there always exists a minimal representative. The property of cutrk that we exploit in dynamic programming is that it bounds the size of any minimal representative.

Proposition 6.3 ([37]). *Let $A \subseteq V(G)$. If $\mathrm{cutrk}(A) \leq k$, then any minimal representative of $A$ has size at most $2^k$.*

To do computations on minimal representatives, we usually need to work with representatives of bipartitions $(A, \overline{A})$.

Definition 6.4 (representative of $(A, \overline{A})$). *Let $A \subseteq V(G)$. A pair $(R, Q)$ with $R \subseteq A$ and $Q \subseteq \overline{A}$ is a* (minimal) representative *of $(A, \overline{A})$ if $R$ is a (minimal) representative of $A$ and $Q$ is a (minimal) representative of $\overline{A}$.*

Given some representative of $(A, \overline{A})$, a minimal representative can be found in polynomial time by the following lemma.

Lemma 6.5. *Let $A \subseteq V(G)$. Given a representative $(R, Q)$ of $(A, \overline{A})$, a minimal representative of $(A, \overline{A})$ can be computed in $(|R| + |Q|)^{\mathcal{O}(1)}$ time.*

*Proof.* One can use the partition refinement algorithm on the graph $G[R, Q]$. ☐

We will also need the following lemma to work with representatives.

Lemma 6.6. *Let $R_A$ be a representative of $A$, and let $R_B$ be a representative of $B$. Then $R_A \cup R_B$ is a representative of $A \cup B$.*

*Proof.* If $N(v) \setminus A = N(u) \setminus A$, then $N(v) \setminus (A \cup B) = N(u) \setminus (A \cup B)$. ☐

Next define the $A_R$-representative of a vertex.

Definition 6.7 ($A_R$-representative of a vertex). *Let $A \subseteq V(G)$, let $R$ be a minimal representative of $A$, and let $v$ be a vertex $v \in A$. The* $A_R$-representative *of $v$, denoted by $\mathrm{rep}_{A_R}(v)$, is the vertex $u \in R$ with $N(u) \setminus A = N(v) \setminus A$.*

By the definition of a minimal representative, there exists exactly one $A_R$-representative of a vertex, so the function $\mathrm{rep}_{A_R}(v)$ is well-defined. Using a minimal representative of $(A, \overline{A})$ we can compute $\mathrm{rep}_{A_R}(v)$ efficiently.

LEMMA 6.8. *Let $A \subseteq V(G)$. Given a vertex $v \in A$ and a minimal representative $(R,Q)$ of $(A,\overline{A})$, $\text{rep}_{A_R}(v)$ can be computed in $(|R|+|Q|)^{\mathcal{O}(1)}$ time.*

*Proof.* Test for each $u \in R$ if $N(u) \cap Q = N(v) \cap Q$. ∎

We also define the $A_R$-representative of a set.

DEFINITION 6.9 ($A_R$-representative of a set). *Let $A \subseteq V(G)$, let $R$ be a minimal $A$-representative, and let $X \subseteq A$. The* $A_R$-representative *of $X$, denoted by* $\text{rep}_{A_R}(X)$, *is the set* $\text{rep}_{A_R}(X) = \bigcup_{v \in X}\{\text{rep}_{A_R}(v)\}$.

Because $\text{rep}_{A_R}(v)$ is well-defined, $\text{rep}_{A_R}(X)$ is also well-defined. By applying Lemma 6.8, the $A_R$-representative of a set $X$ can be computed in $|X|(|R|+|Q|)^{\mathcal{O}(1)}$ time. As any $A_R$-representative of a set is a subset of $R$, there are at most $2^{|R|}$ different $A_R$-representatives of sets. Therefore if $\text{cutrk}(A) \leq k$, there are at most $2^{2^k}$ different $A_R$-representatives of sets.

Many computations on $A_R$-representatives of sets rely on the following observation.

*Observation* 6.10. Let $A \subseteq V(G)$ be a set of vertices, $X \subseteq A$, $Y \subseteq A$, and let $R_A$ be a minimal representative of $A$, let $R_X$ be a minimal representative of $X$, and let $R_Y$ be a minimal representative of $Y$. Also let $X' \subseteq X$ and $Y' \subseteq Y$. Then it holds that $\text{rep}_{A_{R_A}}(X' \cup Y') = \text{rep}_{A_{R_A}}(\text{rep}_{X_{R_X}}(X') \cup \text{rep}_{Y_{R_Y}}(Y'))$.

## 6.2. Augmented rank decompositions.

In order to do dynamic programming efficiently on a rank decomposition, we define the notion of an *augmented rank decomposition.*

DEFINITION 6.11 (augmented rank decomposition). *An* augmented rank decomposition *is a pair $(T,\mathcal{R})$, where $T$ is a rank decomposition and $\mathcal{R}$ is an auxiliary array that stores for each edge $uv \in E(T)$ a minimal representative $(\mathcal{R}[uv],\mathcal{R}[vu])$ of the bipartition $(T[uv],T[vu])$.*

For an augmented rank decomposition $(T,\mathcal{R})$, root edge $r \in E(T)$, and a node $w \in V(T)$ we will also use the notation $\mathcal{R}_r[w]$ to denote the minimal representative of $T_r[w]$ stored in $\mathcal{R}$. Note that by Lemma 6.3 an augmented rank decomposition can be represented in $\mathcal{O}(2^k n)$ space, where $k$ is the width.

Next we show that we can maintain an augmented rank decomposition in our iterative compression.

LEMMA 6.12. *Let $v \in V(G)$. Given an augmented rank decomposition $(T,\mathcal{R})$ of $G \setminus \{v\}$ of width $k$, an augmented rank decomposition of $G$ of width at most $k+1$ can be computed in $2^{\mathcal{O}(k)}n$ time.*

*Proof.* We obtain a rank decomposition $T'$ of $G$ by subdividing an arbitrary edge of $T$ and inserting $v$ as a leaf connected to the node created by subdividing. The width of $T'$ is at most $k+1$ because adding one vertex to $A$ increases $\text{cutrk}(A)$ by at most one.

For the new edge incident with $v$ a minimal representative is easy to compute in $\mathcal{O}(n)$ time: $\{v\}$ is a minimal representative of $\{v\}$ and a minimal representative of $V(G)\setminus\{v\}$ has one vertex from $N(v)$ if it is nonempty and one from $V(G)\setminus(N(v)\cup\{v\})$ if it is nonempty.

Other edges of $T'$ correspond to edges of $T$ in the sense that for each $x'y' \in E(T')$ nonincident to $v$, we have $(T'[x'y'],T'[y'x']) = (A',\overline{A'}) = (T[xy] \cup \{v\}, T[yx])$ for the corresponding $xy \in E(T)$. We start by setting for each such $A'$ the representative as

$\mathcal{R}[xy] \cup \{v\}$. This is not necessarily a minimal representative of $A'$, but we turn it into minimal later. Now we have representatives of size at most $2^k + 1$ for the sides of bipartitions containing $v$.

For the sides of bipartitions not containing $v$, we compute minimal representatives by dynamic programming. We root the decomposition at $v$ and proceed from leaves to the root. For leaves, the minimal representatives have exactly one vertex, the leaf. For nonleaves, we want to compute a minimal representative of a set $B$ corresponding to a subtree with $v \notin B$ such that $B = B_1 \cup B_2$, where we have already computed minimal representatives $R_1$ and $R_2$ of $B_1$ and $B_2$, and a representative $R_A$ of $\overline{B}$ of size $|R_A| \leq 2^k + 1$. By Lemma 6.6, $R_1 \cup R_2$ is a representative of $B$, so $(R_1 \cup R_2, R_A)$ is a representative of $(B, \overline{B})$ of size $2^{\mathcal{O}(k)}$, so we use Lemma 6.5 to compute a minimal representative of $(B, \overline{B})$ in time $2^{\mathcal{O}(k)}$. ∎

**6.3. Refinement data structure for rankwidth.** This subsection is devoted to proving the following lemma.

LEMMA 6.13. *There is a refinement data structure for rank decompositions with time complexity $t(k) = 2^{2^{\mathcal{O}(k)}}$, where the Init operation requires an augmented rank decomposition and the Output operation outputs an augmented rank decomposition.*

Combined with Theorem 2.3 we get the following corollary.

COROLLARY 6.14. *There is an algorithm that, given an augmented rank decomposition of $G$ of width $k$, outputs an augmented rank decomposition of $G$ of width at most $k-1$ or correctly concludes that $k \leq 2\mathtt{rw}(G)$ in time $2^{2^{\mathcal{O}(k)}} n$.*

And by applying iterative compression with Corollary 6.14 and Lemma 6.12 we obtain the main result of this section.

THEOREM 1.1. *There is an algorithm that, given an $n$-vertex graph $G$ and an integer $k$, in time $2^{2^{\mathcal{O}(k)}} n^2$, either computes a rank decomposition of $G$ of width at most $2k$ or correctly concludes that the rankwidth of $G$ is more than $k$.*

Our refinement data structure is based on characterizing global $T$-improvements by dynamic programming on the augmented rank decomposition $(T, \mathcal{R})$ directed towards the edge $r \in E(T)$. In subsections 6.3.1, 6.3.2, 6.3.3, and 6.3.4 we introduce the objects manipulated in this dynamic programming and prove properties of them, and in subsection 6.3.5 we apply this dynamic programming to provide the refinement data structure.

**6.3.1. Embeddings.** Bui-Xuan, Telle, and Vatshelle in [7] characterized the cut-rank of a bipartition $(A, \overline{A})$ by the existence of an embedding of $G[A, \overline{A}]$ into a certain graph $R_k$. Next we define this notion of embedding. In our definition the function describing the embedding is in some sense inversed. This inversion will make manipulating embeddings in dynamic programming easier.

DEFINITION 6.15 (embedding). *Let $G$ and $H$ be bipartite graphs and let $(A_G, B_G)$ and $(A_H, B_H)$ be bipartitions of their vertex sets. A function $f : V(H) \to 2^{V(G)}$ is an embedding of $G$ into $H$ if*
- *$f(u) \cap f(v) = \emptyset$ for $u \neq v$;*
- *$A_G = \bigcup_{v \in A_H} f(v)$, $B_G = \bigcup_{v \in B_H} f(v)$; and*
- *for every pair $(a_H, b_H) \in A_H \times B_H$ and every $(a, b) \in f(a_H) \times f(b_H)$, it holds that $ab \in E(G)$ if and only if $a_H b_H \in E(H)$.*

When using the notation $G[X, Y]$ to construct a bipartite graph, we always assume that the bipartition of $G[X, Y]$ is $(X, Y)$. Note that the embedding completely defines the edges of $G[X,Y]$ in terms of the edges of $H$, and in particular gives a representative of $(X,Y)$ of size at most $|V(H)|$, as we formalize as follows.

*Observation* 6.16. Let $G$ be a graph and $(A,\overline{A})$ be a bipartition of $V(G)$. Let $H$ be a bipartite graph with bipartition $(A_H,B_H)$. Let $f : V(H) \to 2^{V(G)}$ be an embedding of $G[A,\overline{A}]$ into $H$. Let $g$ be a function mapping each $v \in V(H)$ to a subset of $f(v)$ as follows:

$$g(v) = \begin{cases} \{u\} \text{ where } u \in f(v) & \text{if } f(v) \text{ is nonempty,} \\ \emptyset & \text{otherwise.} \end{cases}$$

Then $(\bigcup_{v\in A_H} g(v), \bigcup_{v\in B_H} g(v))$ is a representative of $(A,\overline{A})$ of size at most $|V(H)|$.

Next we define the graph $R_k$ that will be used to characterize cut-rank.

Definition 6.17 (graph $R_k$ [7]). *For each $k \geq 0$, we denote by $R_k$ the bipartite graph with a bipartition $(A,B)$, having for each subset $X \subseteq \{1,\dots,k\}$ a vertex $a_X \in A$ and a vertex $b_X \in B$ (in particular, having $|A| = 2^k$ and $|B| = 2^k$), and having an edge between $a_X$ and $b_Y$ if and only if $|X \cap Y|$ is odd.*

Proposition 6.18 ([7]). *Let $A \subseteq V(G)$. It holds that* $\operatorname{cutrk}(A) \leq k$ *if and only if there is an embedding of $G[A,\overline{A}]$ into $R_k$.*

We will find global $T$-improvements by computing embeddings into $R_k$ by dynamic programming. In order to manipulate embeddings and objects related to embeddings we introduce some notation that naturally extends the definitions of intersections and unions of sets.

Definition 6.19 (intersection $f \cap X$). *Let $f : V(H) \to 2^A$ be a function and $X \subseteq A$ be a set. We denote by $f \cap X$ the function $f \cap X : V(H) \to 2^X$ with $(f \cap X)(v) = f(v) \cap X$.*

We note that an intersection of an embedding and a set is again an embedding.

*Observation* 6.20. Let $A$ be a set, let $C \subseteq A$, and let $X \subseteq A$. If $f : V(H) \to 2^A$ is an embedding of $G[A\cap C, A\setminus C]$ into $H$, then $f \cap X$ is an embedding of $G[X \cap C, X \setminus C]$ into $H$.

Finally, we define the union of functions.

Definition 6.21 (union $f \cup g$). *Let $f : V(H) \to 2^X$ and $g : V(H) \to 2^Y$ be functions. We define function $f \cup g : V(H) \to 2^{X\cup Y}$ with $(f \cup g)(v) = f(v) \cup g(v)$.*

#### 6.3.2. Representatives of embeddings.

Next we define the $A_R$-representative of an embedding, extending the definition of the $A_R$-representative of a set.

Definition 6.22 ($A_R$-representative of an embedding). *Let $A \subseteq V(G)$, and let $R$ be a minimal representative of $A$. Also let $f : V(H) \to 2^A$ be an embedding. The* $A_R$-representative *of $f$ is the function $g : V(H) \to 2^R$, where $g(v) = \operatorname{rep}_{A_R}(f(v))$.*

The $A_R$-representative of an embedding is well-defined because the $A_R$-representative of a set is well-defined. If $\operatorname{cutrk}(A) \leq k$, then the number of $A_R$-representatives of embeddings into $H$ is at most $(2^{2^k})^{|V(H)|}$. In particular, the number of $A_R$-representatives of embeddings into $R_k$ is at most $(2^{2^k})^{2\cdot 2^k} = 2^{2^{\mathcal{O}(k)}}$.

We will define the compatibility and the composition of two representatives of embeddings. The intuition is that embeddings $f_X$ and $f_Y$ of disjoint subgraphs can be merged into an embedding $f_X \cup f_Y$ if and only if their representatives are compatible. Moreover, the representative of $f_X \cup f_Y$ will be the composition of the representatives of $f_X$ and $f_Y$.

Let $X$ and $Y$ be disjoint subsets of $V(G)$ and let $A = X \cup Y$. Also let $R$ be a minimal representative of $A$, $R_X$ a minimal representative of $X$, and $R_Y$ a minimal representative of $Y$. Also let $H$ be a bipartite graph, $g_X : V(H) \to 2^{R_X}$ an $X_{R_X}$-representative an embedding, and $g_Y : V(H) \to 2^{R_Y}$ a $Y_{R_Y}$-representative of an embedding.

Definition 6.23 (compatibility). *Let $(A_H, B_H)$ be the bipartition of $H$. The representatives $g_X$ and $g_Y$ are* compatible *if for every pair $(a_H, b_H) \in A_H \times B_H$ it holds that*

1. *for every $(a,b) \in g_X(a_H) \times g_Y(b_H)$ it holds that $ab \in E(G) \Leftrightarrow a_H b_H \in E(H)$; and*
2. *for every $(a,b) \in g_Y(a_H) \times g_X(b_H)$ it holds that $ab \in E(G) \Leftrightarrow a_H b_H \in E(H)$.*

Note that compatibility can be tested in $(|V(H)| + |R_X| + |R_Y|)^{\mathcal{O}(1)}$ time. Next we show that two embeddings can be merged into an embedding only if their representatives are compatible.

Lemma 6.24. *Let $C \subseteq A$ be a set and let $f_A$ be an embedding of $G[A \cap C, A \setminus C]$ into $H$. If $g_X$ is the $X_{R_X}$-representative of $f_A \cap X$ and $g_Y$ is the $Y_{R_Y}$-representative of $f_A \cap Y$, then $g_X$ and $g_Y$ are compatible.*

*Proof.* Let $f_X = f_A \cap X$, $f_Y = f_A \cap Y$, and let $(A_H, B_H)$ be the bipartition of $H$. For the case (1) of compatibility it suffices to prove for every pair $(a_H, b_H) \in A_H \times B_H$ and every $(a,b) \in g_X(a_H) \times g_Y(b_H)$ that $ab \in E(G)$ if and only if $a_H b_H \in E(H)$.

As $g_X(a_H)$ is an $X_{R_X}$-representative of $f_X(a_H)$, there exists $a' \in f_X(a_H)$ with $N(a') \setminus X = N(a) \setminus X$. Similarly there exists $b' \in f_Y(b_H)$ with $N(b') \setminus Y = N(b) \setminus Y$. Therefore $ab \in E(G)$ if and only if $a'b' \in E(G)$. Since $f_X(a_H) = f_A(a_H) \cap X$ and $f_Y(b_H) = f_A(b_H) \cap Y$, we have that $a' \in f_A(a_H)$ and that $b' \in f_A(b_H)$. Because $f_A$ is an embedding it holds that $a'b' \in E(G)$ if and only if $a_H b_H \in E(H)$. This concludes the proof that the case (1) of compatibility holds.

For (2) it suffices to prove the same but for every pair $(a,b) \in g_Y(a_H) \times g_X(b_H)$. This proof is symmetric. ∎

The composition is defined as the representative of the union.

Definition 6.25 (composition). *The* composition *of $g_X$ and $g_Y$ is the function $g_A : V(H) \to 2^A$ defined by $g_A(v) = \mathrm{rep}_{A_R}(g_X(v) \cup g_Y(v))$ for all $v \in V(H)$.*

By using a minimal $(A, \overline{A})$-representative $(R, Q)$ the composition can be computed in time $(|V(H)| + |R| + |Q| + |R_X| + |R_Y|)^{\mathcal{O}(1)}$.

Next we prove that two embeddings can be merged into an embedding if their representatives are compatible, and in this case the composition gives the resulting representative.

Lemma 6.26. *Let $C \subseteq A$ be a set. Let $g_X$ be the $X_{R_X}$-representative of an embedding $f_X$ of $G[X \cap C, X \setminus C]$ into $H$ and $g_Y$ the $Y_{R_Y}$-representative of an embedding $f_Y$ of $G[Y \cap C, Y \setminus C]$ into $H$. If $g_X$ and $g_Y$ are compatible, then $f_X \cup f_Y$ is an embedding of $G[A \cap C, A \setminus C]$ into $H$ so that the composition of $g_X$ and $g_Y$ is the $A_R$-representative of $f_X \cup f_Y$.*

*Proof.* Let $(A_H, B_H)$ be the bipartition of $H$. We let $f_A = f_X \cup f_Y$ and observe that $f_A$ is a function $f_A : V(H) \to 2^A$ that satisfies $f_A(u) \cap f_A(v) = \emptyset$ for $u \neq v$, $(X \cap C) \cup (Y \cap C) = (A \cap C) = \bigcup_{v \in A_H} f_A(v)$, and $(X \setminus C) \cup (Y \setminus C) = (A \setminus C) = \bigcup_{v \in B_H} f_A(v)$. Therefore $f_A$ is an embedding of $G[A \cap C, A \setminus C]$ into $H$ if for every pair $(a_H, b_H) \in A_H \times B_H$ and every $(a,b) \in f_A(a_H) \times f_A(b_H)$ it holds that $ab \in E(G)$

if and only if $a_H b_H \in E(H)$. We say that $f_A$ is *good* for a pair $(a,b) \in (A \cap C) \times (A \setminus C)$ if this holds for this pair. Now $f_A$ is an embedding of $G[A \cap C, A \setminus C]$ into $H$ if it is good for every pair $(a,b) \in (A \cap C) \times (A \setminus C)$.

Because $f_X$ is an embedding of $G[X \cap C, X \setminus C]$ into $H$ we have that $f_X$ is good for all pairs in $(X \cap C) \times (X \setminus C)$ and therefore $f_A$ is good for all pairs in $(X \cap C) \times (X \setminus C)$. Analogously because $f_Y$ is an embedding of $G[Y \cap C, Y \setminus C]$ into $H$ we have that $f_A$ is good for all pairs in $(Y \cap C) \times (Y \setminus C)$. It remains to prove that $f_A$ is good for all pairs in $(X \cap C) \times (Y \setminus C)$ and in $(Y \cap C) \times (X \setminus C)$.

Let $(a,b) \in (X \cap C) \times (Y \setminus C)$. The set $g_X(a_H)$ is an $X_{R_X}$-representative of $f_X(a_H) \supseteq \{a\}$, so there exists $a' \in g_X(a_H)$ so that $N(a') \setminus X = N(a) \setminus X$. Similarly there exists $b' \in g_Y(b_H)$ so that $N(b') \setminus Y = N(b) \setminus Y$. Therefore $ab \in E(G)$ if and only if $a'b' \in E(G)$. Now, by compatibility (1) it holds that $a'b' \in E(G)$ if and only if $a_H b_H \in E(H)$. Therefore $f_A$ is good for $(a,b)$.

The proof for $(a,b) \in (Y \cap C) \times (X \setminus C)$ is symmetric, using compatibility (2) instead. Therefore $f_A$ is an embedding of $G[A \cap C, A \setminus C]$ into $H$. Finally, by Observation 6.10 the composition of $g_X$ and $g_Y$ is the $A_R$-representative of $f_A$. ◻

**6.3.3. Improvement embeddings.** In order to construct a minimum $W$-improvement $(C_1, C_2, C_3)$, we build six embeddings simultaneously in the dynamic programming, three to bound $\operatorname{cutrk}(C_i)$ and three to bound $\operatorname{cutrk}(C_i \cap W)$. We of course also need to bound $\operatorname{cutrk}(C_i \cap \overline{W})$, but note that $G[(C_i \cap \overline{W}) \cap W, \overline{(C_i \cap \overline{W})} \cap W] = G[\emptyset, W]$, so dynamic programming is not required for building the side of $W$ of the embedding $G[C_i \cap \overline{W}, \overline{C_i \cap \overline{W}}]$.

DEFINITION 6.27 ($A$-restricted improvement embedding). *Let $A \subseteq V(G)$. A 10-tuple*

$$\bar{E} = (f_1^C, f_2^C, f_3^C, f_1^W, f_2^W, f_3^W, k_1, k_2, k_3, l)$$

*is an $A$-*restricted improvement embedding *if there exists a tripartition $(C_1, C_2, C_3)$ of $A$ so that*

1. *for each $i \in \{1,2,3\}$, $f_i^C$ is an embedding of $G[A \cap C_i, A \cap \overline{C_i}]$ into $R_{k_i}$; and*
2. *for each $i \in \{1,2,3\}$, $f_i^W$ is an embedding of $G[A \cap C_i, A \cap \overline{C_i}]$ into $R_l$.*

Note that an $A$-restricted improvement embedding $\bar{E}$ uniquely defines such a tripartition $(C_1, C_2, C_3)$ of $A$. We call such a tripartition the *tripartition of* $\bar{E}$. We say that $\bar{E}$ *intersects a set* if its tripartition $(C_1, C_2, C_3)$ intersects it. We call the quadruple $(k_1, k_2, k_3, l)$ the *shape of* $\bar{E}$. An $A$-restricted improvement embedding is $k$-bounded if $k_1, k_2, k_3, l \le k$.

In the following lemma we introduce the notation $\bar{E} \cap X$, where $\bar{E}$ is an $A$-restricted improvement embedding and $X \subseteq A$.

LEMMA 6.28. *Let $A \subseteq V(G)$, $X \subseteq A$, and*

$$\bar{E} = (f_1^C, f_2^C, f_3^C, f_1^W, f_2^W, f_3^W, k_1, k_2, k_3, l)$$

*be an $A$-restricted improvement embedding with tripartition $(C_1, C_2, C_3)$. The tuple*

$$\bar{E} \cap X = (f_1^C \cap X, f_2^C \cap X, f_3^C \cap X, f_1^W \cap X, f_2^W \cap X, f_3^W \cap X, k_1, k_2, k_3, l)$$

*is an $X$-restricted improvement embedding with tripartition $(C_1 \cap X, C_2 \cap X, C_3 \cap X)$.*

*Proof.* By Observation 6.20, for each $i$, $f_i^C \cap X$ is an embedding of $G[X \cap C_i, X \cap \overline{C_i}]$ to $R_{k_i}$ and $f_i^W$ is an embedding of $G[X \cap C_i, X \cap \overline{C_i}]$ to $R_l$. ◻

We also define the union of improvement embeddings, extending the definition of the union of embeddings.

DEFINITION 6.29. *Let $X$ and $Y$ be disjoint subsets,*

$$\bar{E}_1 = (f_1^C, f_2^C, f_3^C, f_1^W, f_2^W, f_3^W, k_1, k_2, k_3, l)$$

*an $X$-restricted improvement embedding, and*

$$\bar{E}_2 = (g_1^C, g_2^C, g_3^C, g_1^W, g_2^W, g_3^W, k_1, k_2, k_3, l)$$

*a $Y$-restricted improvement embedding with the same shape. We denote by $\bar{E}_1 \cup \bar{E}_2$ the* 10*-tuple*

$$\bar{E}_1 \cup \bar{E}_2 = (f_1^C \cup g_1^C, f_2^C \cup g_2^C, f_3^C \cup g_3^C, f_1^W \cup g_1^W, f_2^W \cup g_2^W, f_3^W \cup g_3^W, k_1, k_2, k_3, l).$$

Note that if the tripartition of $X$ is $(C_1 \cap X, C_2 \cap X, C_3 \cap X)$, the tripartition of $Y$ is $(C_1 \cap Y, C_2 \cap Y, C_3 \cap Y)$, and $\bar{E}_1 \cup \bar{E}_2$ is indeed an $X \cup Y$-restricted improvement embedding, then the tripartition of $\bar{E}_1 \cup \bar{E}_2$ is $(C_1, C_2, C_3)$.

We will use dynamic programming on the rank decomposition $T$ to compute improvement embeddings, minimizing the number of nodes of $T$ intersected.

### 6.3.4. Representatives of improvement embeddings.

The definition of the $A_R$-representative of an improvement embedding extends the definition of the $A_R$-representative of an embedding.

DEFINITION 6.30 ($A_R$-representative of improvement embedding). *Let $A \subseteq V(G)$, and let $R$ be a minimal representative of $A$. Also let*

$$\bar{E} = (f_1^C, f_2^C, f_3^C, f_1^W, f_2^W, f_3^W, k_1, k_2, k_3, l)$$

*be an $A$-restricted improvement embedding. The $A_R$*-representative *of $\bar{E}$ is the tuple*

$$(g_1^C, g_2^C, g_3^C, g_1^W, g_2^W, g_3^W, k_1, k_2, k_3, l),$$

*where each such $g$ is the $A_R$-representative of the corresponding embedding $f$.*

The shape of the $A_R$-representative of $\bar{E}$ is the same as the shape of $\bar{E}$. When $\text{cutrk}(A) \leq k$, the number of $A_R$-representatives of $k$-bounded improvement embeddings is $(2^{2^{\mathcal{O}(k)}})^6 k^4 = 2^{2^{\mathcal{O}(k)}}$. We naturally extend the definitions of composition and compatibility to representatives of improvement embeddings.

Let $G$ be a graph, $X$ and $Y$ disjoint subsets of $V(G)$, and $A = X \cup Y$. Also let $R$ be a minimal representative of $A$, $R_X$ a minimal representative of $X$, and $R_Y$ a minimal representative of $Y$. Let $\tilde{E}_X = (f_1^C, f_2^C, f_3^C, f_1^W, f_2^W, f_3^W, k_1, k_2, k_3, l)$ be the $X_{R_X}$-representative of an $X$-restricted improvement embedding and $\tilde{E}_Y = (g_1^C, g_2^C, g_3^C, g_1^W, g_2^W, g_3^W, k_1', k_2', k_3', l')$ be the $Y_{R_Y}$-representative of a $Y$-restricted improvement embedding.

DEFINITION 6.31 ($C$-compatibility). *$\tilde{E}_X$ and $\tilde{E}_Y$ are* $C$-compatible *if they have the same shape and for each $i \in \{1,2,3\}$, $f_i^C$ and $g_i^C$ are compatible.*

DEFINITION 6.32 (compatibility). *$\tilde{E}_X$ and $\tilde{E}_Y$ are* compatible *if they are $C$-compatible and for each $i \in \{1,2,3\}$, $f_i^W$ and $g_i^W$ are compatible.*

We derive the following lemma directly from Lemma 6.24.

LEMMA 6.33. *Let $\bar{E}$ be an $A$-restricted improvement embedding. If $\tilde{E}_X$ is the $X_{R_X}$-representative of $\bar{E} \cap X$ and $\tilde{E}_Y$ is the $Y_{R_Y}$-representative of $\bar{E} \cap Y$, then $\tilde{E}_X$ and $\tilde{E}_Y$ are compatible.*

*Proof.* Apply Lemma 6.24 to each embedding in $\bar{E}$. ∎

Next we define the composition of two representatives of improvement embeddings, extending the definition of the composition of representatives of embeddings.

DEFINITION 6.34 (composition). *If $\tilde{E}_X$ and $\tilde{E}_Y$ are compatible, then the* composition *of $\tilde{E}_X$ and $\tilde{E}_Y$ is the pointwise composition of $\tilde{E}_X$ and $\tilde{E}_Y$, i.e., the composition is the* 10*-tuple*

$$\tilde{E}_A = (h_1^C, h_2^C, h_3^C, h_1^W, h_2^W, h_3^W, k_1, k_2, k_3, l),$$

*where for each $i \in \{1,2,3\}$ $h_i^C$ is the composition of $f_i^C$ and $g_i^C$, and $h_i^W$ is the composition of $f_i^W$ and $g_i^W$.*

Next we prove the main lemma for computing improvement embeddings by dynamic programming, which is an extension of Lemma 6.26.

LEMMA 6.35. *Let $X$ and $Y$ be disjoint sets, $A = X \cup Y$, $R$ a minimal representative of $A$, $R_X$ a minimal representative of $X$, and $R_Y$ a minimal representative of $Y$. Let $\bar{E}_1$ be an $X$-restricted improvement embedding and $\bar{E}_2$ a $Y$-restricted improvement embedding. Let $\tilde{E}_X$ be the $X_{R_X}$-representative of $\bar{E}_1$, and let $\tilde{E}_Y$ be the $Y_{R_Y}$-representative of $\bar{E}_2$. If $\tilde{E}_X$ and $\tilde{E}_Y$ are compatible, then $\bar{E}_1 \cup \bar{E}_2$ is an $A$-restricted improvement embedding and the composition of $\tilde{E}_X$ and $\tilde{E}_Y$ is the $A_R$-representative of $\bar{E}_1 \cup \bar{E}_2$.*

*Proof.* Denote

$$\bar{E}_1 = (f_1^C, f_2^C, f_3^C, f_1^W, f_2^W, f_3^W, k_1, k_2, k_3, l)$$

and

$$\bar{E}_2 = (g_1^C, g_2^C, g_3^C, g_1^W, g_2^W, g_3^W, k_1, k_2, k_3, l).$$

Let $(C_1^X, C_2^X, C_3^X)$ be the tripartition of $\bar{E}_1$ and $(C_1^Y, C_2^Y, C_3^Y)$ the tripartition of $\bar{E}_2$. By Lemma 6.26, $f_i^C \cup g_i^C$ is an embedding of $G[C_i^X \cup C_i^Y, A \setminus (C_i^X \cup C_i^Y)]$ to $R_{k_i}$ and $f_i^W \cup g_i^W$ is an embedding of $G[C_i^X \cup C_i^Y, A \setminus (C_i^X \cup C_i^Y)]$ to $R_l$ for every $i$. Therefore

$$\bar{E} = (f_1^C \cup g_1^C, f_2^C \cup g_2^C, f_3^C \cup g_3^C, f_1^W \cup g_1^W, f_2^W \cup g_2^W, f_3^W \cup g_3^W, k_1, k_2, k_3, l)$$

is an improvement embedding with tripartition $(C_1^X \cup C_1^Y, C_2^X \cup C_2^Y, C_3^X \cup C_3^Y)$. By Lemma 6.26 the composition of $\tilde{E}_X$ and $\tilde{E}_Y$ is the $A_R$-representative of $\bar{E}$. ∎

*Finding the $W$-improvement.* In the dynamic programming we will determine if there exists a $W$-improvement based on the representatives of $W$-restricted improvement embeddings and the representatives of $\overline{W}$-restricted improvement embeddings. For this purpose we will define the root-compatibility of two representatives of improvement embeddings, characterizing whether they can be merged to yield a $W$-improvement.

Let $W \subseteq V(G)$, let $R_W$ be a minimal representative of $W$, and let $R_{\overline{W}}$ be a minimal representative of $\overline{W}$. Also let

$$\tilde{E}_W = (f_1^C, f_2^C, f_3^C, f_1^W, f_2^W, f_3^W, k_1, k_2, k_3, l)$$

be a $W_{R_W}$-representative of a $W$-restricted improvement embedding and

$$\tilde{E}_{\overline{W}} = (g_1^C, g_2^C, g_3^C, g_1^W, g_2^W, g_3^W, k_1, k_2, k_3, l)$$

be a $\overline{W}_{R_{\overline{W}}}$-representative of an $\overline{W}$-restricted improvement embedding so that $\tilde{E}_W$ and $\tilde{E}_{\overline{W}}$ have the same shape.

DEFINITION 6.36 (root-compatibility). *$\tilde{E}_W$ and $\tilde{E}_{\overline{W}}$ are* root-compatible *if they are $C$-compatible and*

1. *for each $i$, there exists an embedding $f_i^{\overline{W}}$ of $G[\emptyset, \overline{W}]$ into $R_l$ so that $f_i^W$ is compatible with the $\overline{W}_{R_{\overline{W}}}$-representative of $f_i^{\overline{W}}$; and*
2. *for each $i$, there exists an embedding $g_i^{\overline{W}}$ of $G[\emptyset, W]$ into $R_l$ so that $g_i^W$ is compatible with the $W_{R_W}$-representative of $g_i^{\overline{W}}$.*

The definition does not directly provide an efficient algorithm for checking root-compatibility, but a couple of observations and brute force yields a $2^{2^{\mathcal{O}(k)}}$ time algorithm as follows.

LEMMA 6.37. *Let* $\operatorname{cutrk}(W) \leq k$ *and suppose that the improvement embeddings are $k$-bounded. Given $\tilde{E}_W$, $\tilde{E}_{\overline{W}}$, $R_W$, and $R_{\overline{W}}$, the root-compatibility of $\tilde{E}_W$ and $\tilde{E}_{\overline{W}}$ can be checked in time $2^{2^{\mathcal{O}(k)}}$.*

*Proof.* We prove the case (1); the other case is symmetric.

Suppose there exists such an embedding $f_i^{\overline{W}}$. Let $(A_H, B_H)$ be the bipartition of $R_l$ and let $u_H, v_H \in B_H$ be a pair of distinct vertices in $B_H$. Suppose that there exists $u, v \in \overline{W}$ such that $N(u) \cap W = N(v) \cap W$, $u \in f_i^{\overline{W}}(u_H)$, and $v \in f_i^{\overline{W}}(v_H)$. Note that now, if we remove $u$ from $f_i^{\overline{W}}(u_H)$ and insert it into $f_i^{\overline{W}}(v_H)$, we obtain an embedding whose $\overline{W}_{R_{\overline{W}}}$-representative is compatible with the same $W_{R_W}$-representatives as $f_i^{\overline{W}}$. Therefore, we can assume for any $u, v \in \overline{W}$ that if $N(u) \cap W = N(v) \cap W$, then there exists $v_H \in B_H$ so that $\{u, v\} \subseteq f_i^{\overline{W}}(v_H)$.

As for any $v \in \overline{W}$ there is a vertex $v_R \in R_{\overline{W}}$ with $N(v) \cap W = N(v_R) \cap W$, it is sufficient to enumerate all intersections $f_i^{\overline{W}} \cap R_{\overline{W}}$ and assign each $v$ to the same vertex of $B_H$ as $v_R$. Note that in this case, the $\overline{W}_{R_{\overline{W}}}$-representative of $f_i^{\overline{W}}$ depends only on the intersection $f_i^{\overline{W}} \cap R_{\overline{W}}$. The number of the intersections is at most $\left(2^{|R_{\overline{W}}|}\right)^{|B_H|} \leq (2^{2^k})^{2^k} = 2^{2^{\mathcal{O}(k)}}$ and each can be checked in time $2^{\mathcal{O}(k)}$. ∎

We also introduce the definition of $C_i$-emptiness to denote whether none of the vertices has been assigned to $C_i$ in the tripartition.

DEFINITION 6.38 ($C_i$-empty). *Let $\tilde{E}_R = (f_1^C, f_2^C, f_3^C, \ldots)$ be the $A_R$-representative of an $A$-restricted improvement embedding. Let $i \in \{1, 2, 3\}$, and let $(A_H, B_H)$ be the bipartition of $R_{k_i}$. $\tilde{E}_R$ is $C_i$*-empty *if for every $v \in A_H$ it holds that $f_i^C(v) = \emptyset$.*

The definition of $C_i$-emptiness is for determining which of the parts of a corresponding tripartition intersect $A$.

*Observation* 6.39. Let $\tilde{E}_R$ be the $A_R$-representative of an $A$-restricted improvement embedding $\bar{E}$ and $(C_1, C_2, C_3)$ the tripartition of $\bar{E}$. It holds that $C_i = \emptyset$ if and only if $\tilde{E}_R$ is $C_i$-empty.

Finally, we describe how to find $W$-improvements based on representatives of improvement embeddings.

LEMMA 6.40. *Let $T$ be a rank decomposition, $uv = r \in E(T)$, and $W = T[uv]$. Denote $X = W = T_r[u]$, $Y = \overline{W} = T_r[v]$, and let $R_X$ be a minimal representative of $X$*

*and $R_Y$ a minimal representative of $Y$. There exists a $W$-improvement $(C_1, C_2, C_3)$ with arity $\alpha$ and* $\operatorname{cutrk}(C_i) \le k_i$ *for each $i \in \{1,2,3\}$ if and only if there exist $\tilde{E}_u$ and $\tilde{E}_v$ such that*

1. *$\tilde{E}_u$ is the $X_{R_X}$-representative of an $X$-restricted improvement embedding with shape $(k_1, k_2, k_3, l)$ whose tripartition is $(C_1 \cap X, C_2 \cap X, C_3 \cap X)$;*
2. *$\tilde{E}_v$ is the $Y_{R_Y}$-representative of a $Y$-restricted improvement embedding with shape $(k_1, k_2, k_3, l)$ whose tripartition is $(C_1 \cap Y, C_2 \cap Y, C_3 \cap Y)$;*
3. *$\tilde{E}_u$ and $\tilde{E}_v$ are root-compatible;*
4. *$k_i <$ $\operatorname{cutrk}(W)/2$ for all $i$, $l <$ $\operatorname{cutrk}(W)$; and*
5. *$\alpha = 2$ if there exists $i$ such that both $\tilde{E}_u$ and $\tilde{E}_v$ are $C_i$-empty and otherwise $\alpha = 3$.*

*Proof. If direction:*
Let

$$\bar{E}_X = (f_1^C, f_2^C, f_3^C, f_1^W, f_2^W, f_3^W, k_1, k_2, k_3, l)$$

be an $X$-restricted improvement embedding whose $X_{R_X}$-representative is $\tilde{E}_u$ and whose tripartition is $(C_1 \cap X, C_2 \cap X, C_3 \cap X)$. Also let

$$\bar{E}_Y = (g_1^C, g_2^C, g_3^C, g_1^{\overline{W}}, g_2^{\overline{W}}, g_3^{\overline{W}}, k_1, k_2, k_3, l)$$

be a $Y$-restricted improvement embedding whose $Y_{R_Y}$-representative is $\tilde{E}_u$ and whose tripartition is $(C_1 \cap Y, C_2 \cap Y, C_3 \cap Y)$.

As $\tilde{E}_u$ and $\tilde{E}_v$ are $C$-compatible, Lemma 6.26 implies that for each $i \in \{1,2,3\}$, $f_i^C \cup g_i^C$ is an embedding of $G[C_i, \overline{C_i}]$ into $R_{k_i}$. Therefore by Proposition 6.18 it holds that $\operatorname{cutrk}(C_i) \le k_i$.

As $\tilde{E}_u$ and $\tilde{E}_v$ are root-compatible, by the definition of root-compatibility and Lemma 6.26 for each $i$ there exists an embedding $f_i^{\overline{W}}$ of $G[\emptyset, \overline{W}]$ to $R_l$ so that $f_i^W \cup f_i^{\overline{W}}$ is an embedding of $G[C_i \cap W, (\overline{C_i} \cap W) \cup \overline{W}] = G[C_i \cap W, \overline{C_i \cap W}]$ to $R_l$. Therefore by Proposition 6.18 it holds that $\operatorname{cutrk}(C_i \cap W) \le l$.

Symmetrically, by root-compatibility there exists an embedding $g_i^W$ of $G[\emptyset, W]$ to $R_l$ so that $g_i^{\overline{W}} \cup g_i^W$ is an embedding of $G[C_i \cap \overline{W}, (\overline{C_i} \cap \overline{W}) \cup W] = G[C_i \cap \overline{W}, \overline{C_i \cap \overline{W}}]$ to $R_l$. Therefore by Proposition 6.18 it holds that $\operatorname{cutrk}(C_i \cap \overline{W}) \le l$.

Finally note that $C_i = \emptyset$ if and only if both $\tilde{E}_u$ and $\tilde{E}_v$ are $C_i$-empty.

*Only if direction:* Let $(C_1, C_2, C_3)$ be a $W$-improvement of arity $\alpha$, where $\operatorname{cutrk}(C_i) = k_i$. Also let $l = \operatorname{cutrk}(W) - 1$. By Proposition 6.18, for each $i$ there exists an embedding $f_i^C$ of $G[C_i, \overline{C_i}]$ to $R_{k_i}$, an embedding $f_i^W$ of $G[C_i \cap W, \overline{C_i \cap W}]$ to $R_l$, and an embedding $f_i^{\overline{W}}$ of $G[C_i \cap \overline{W}, \overline{C_i \cap \overline{W}}]$ to $R_l$.

Let

$$\bar{E}_X = (f_1^C \cap X, f_2^C \cap X, f_3^C \cap X, f_1^W \cap X, f_2^W \cap X, f_3^W \cap X, k_1, k_2, k_3, l)$$

and note that the tripartition of $\bar{E}_X$ is $(C_1 \cap X, C_2 \cap X, C_3 \cap X)$. Also let

$$\bar{E}_Y = (f_1^C \cap Y, f_2^C \cap Y, f_3^C \cap Y, f_1^{\overline{W}} \cap Y, f_2^{\overline{W}} \cap Y, f_3^{\overline{W}} \cap Y, k_1, k_2, k_3, l)$$

and note that the tripartition of $\bar{E}_Y$ is $(C_1 \cap Y, C_2 \cap Y, C_3 \cap Y)$. Let $\tilde{E}_u$ be the $X_{R_X}$-representative of $\bar{E}_X$ and $\tilde{E}_v$ the $Y_{R_Y}$-representative of $\bar{E}_Y$. Note that now $\tilde{E}_u$ and $\tilde{E}_v$ satisfy (1) and (2).

By Lemma 6.24, $\tilde{E}_u$ and $\tilde{E}_v$ are $C$-compatible. By Lemma 6.24, for each $i$, $f_i^W \cap Y$ is an embedding of $G[\emptyset, \overline{W}]$ to $R_l$ whose $Y_{R_Y}$-representative is compatible with the

$X_{R_X}$-representative of $f_i^W \cap X$, and $f_i^{\overline{W}} \cap X$ is an embedding of $G[\emptyset, W]$ whose $X_{R_X}$-representative is compatible with the $Y_{R_Y}$-representative of $f_i^{\overline{W}} \cap Y$. Therefore $\tilde{E}_u$ and $\tilde{E}_v$ are root-compatible by the definition of root-compatibility.

If $(C_1, C_2, C_3)$ has arity 2, then there is $i$ such that $C_i = \emptyset$, which implies that both $\tilde{E}_u$ and $\tilde{E}_v$ are $C_i$-empty. Otherwise each $C_i$ intersects with at least one of $X$ or $Y$, and therefore for each $i$ at least one of $\tilde{E}_u$ or $\tilde{E}_v$ is not $C_i$-empty. ◻

**6.3.5. Refinement data structure for augmented rank decompositions.** In this subsection we provide the refinement data structure for augmented rank decompositions using dynamic programming building on the previous subsections. Throughout this subsection, we assume that we are maintaining an augmented rank decomposition $(T, \mathcal{R})$ of width $\mathtt{rw}(T) \leq k$, and therefore we are only interested in $k$-bounded improvement embeddings.

*Concrete representatives.* In order to maintain an augmented rank decomposition in the refinement operation, we need to construct a minimal representative of each set $C_i$ of the $W$-improvement $(C_1, C_2, C_3)$. To do this, we maintain "concrete representatives" in the dynamic programming.

DEFINITION 6.41 (concrete representative). *Let $H$ be bipartite graph, let $A \subseteq V(G)$, and let $f : V(H) \to 2^A$ be a function. A* concrete representative *of $f$ is a function $g$ defined in Observation* 6.16*, i.e., a function $g$ mapping each $v \in V(H)$ to a subset of $f(v)$ as follows:*

$$g(v) = \begin{cases} \{u\} \text{ where } u \in f(v) & \text{if } f(v) \text{ is nonempty,} \\ \emptyset & \text{otherwise.} \end{cases}$$

*Let $\bar{E} = (f_1^C, f_2^C, f_3^C, \ldots)$ be an $A$-restricted improvement embedding. A concrete representative of $\bar{E}$ is a triple $(g_1^C, g_2^C, g_3^C)$, where each $g_i^C$ is a concrete representative of $f_i^C$.*

Note that a concrete representative can be represented in $\mathcal{O}(|V(H)|)$ space. By Observation 6.16, a concrete representative of an embedding of $G[C_i, \overline{C_i}]$ into $H$ can be turned into a representative of $(C_i, \overline{C_i})$ of size at most $|V(H)|$. In particular, using a concrete representative of a $V(G)$-restricted improvement embedding with tripartition $(C_1, C_2, C_3)$ we can compute a minimal representative of each $C_i$ in time $2^{\mathcal{O}(k)}$.

We define the union of two concrete representatives naturally.

DEFINITION 6.42 (union of concrete representatives). *Let $f_R$ be a concrete representative of a function $f : V(H) \to 2^X$ and $g_R$ a concrete representative of a function $g : V(H) \to 2^Y$. We denote by $f_R \cup g_R$ the function mapping each $v \in V(H)$ to a subset of $f(v) \cup g(v)$ as follows:*

$$(f_R \cup g_R)(v) = \begin{cases} f_R(v) & \text{if } f_R(v) \neq \emptyset, \\ g_R(v) & \text{otherwise.} \end{cases}$$

*The* union *of concrete representatives of improvement embeddings is the pointwise union of such triples of concrete representatives.*

Observe that such $f_R \cup g_R$ is a concrete representative of $f \cup g$.

*Dynamic programming tables.* In the refinement data structure we maintain a dynamic programming table for each node $w \in V(T)$. We call this table the *$r$-table* of the node $w$ to signify that this table contains information about the $r$-subtree of $w$. Next we formally define an $r$-table.

DEFINITION 6.43 ($r$-table). *For an augmented rank decomposition $(T,\mathcal{R})$, root edge $r \in E(T)$, node $w \in V(T)$, and $A = T_r[w]$, $R = \mathcal{R}_r[w]$, an* $r$-table *of $w$ is a triple $(\mathcal{E},\mathcal{I},\mathcal{C})$, where*

1. *$\mathcal{E}$ is the set of all $A_R$-representatives of $k$-bounded $A$-restricted improvement embeddings;*
2. *$\mathcal{I}$ is a function mapping each $\tilde{E}_R \in \mathcal{E}$ to the least integer $i$ such that there exists an $A$-restricted improvement embedding $\bar{E}$ whose $A_R$-representative is $\tilde{E}_R$ and which $r$-intersects $i$ nodes of the $r$-subtree of $w$; and*
3. *$\mathcal{C}$ is a function mapping each $\tilde{E}_R \in \mathcal{E}$ to a concrete representative of an $A$-restricted improvement embedding $\bar{E}$ such that $\tilde{E}_R$ is the $A_R$-representative of $\bar{E}$ and $\bar{E}$ $r$-intersects $\mathcal{I}(\tilde{E}_R)$ nodes of the $r$-subtree of $w$.*

Note that an $r$-table can be represented by making use of $2^{2^{\mathcal{O}(k)}}$ space.

To correctly construct the refinement that matches the concrete representation obtained, we need to spell out some additional properties of $r$-tables, which will be naturally satisfied by the way the $r$-tables will be constructed.

DEFINITION 6.44 (local and global representation). *Let $(T,\mathcal{R})$ be an augmented rank decomposition, $r \in E(T)$, $w \in V(T)$, $A = T_r[w]$, and $R = \mathcal{R}_r[w]$. An $r$-table $(\mathcal{E},\mathcal{I},\mathcal{C})$ of $w$* locally represents *an $A$-restricted improvement embedding $\bar{E}$ if there exists $\tilde{E}_R \in \mathcal{E}$ so that $\tilde{E}_R$ is the $A_R$-representative of $\bar{E}$, $\mathcal{I}(\tilde{E}_R)$ is the number of nodes in the $r$-subtree of $w$ $r$-intersected by $\bar{E}$, and $\mathcal{C}(\tilde{E}_R)$ is a concrete representative of $\bar{E}$. The $r$-table of $w$* globally represents *$\bar{E}$ if the $r$-tables of all nodes $w'$ in the $r$-subtree of $w$ (including $w$ itself) locally represent $\bar{E} \cap T_r[w']$.*

DEFINITION 6.45 (linked $r$-table). *A* linked *$r$-table of $w$ is a 4-tuple $(\mathcal{E},\mathcal{I},\mathcal{C},\mathcal{L})$ so that $(\mathcal{E},\mathcal{I},\mathcal{C})$ is an $r$-table of $w$ and for each $\tilde{E}_R \in \mathcal{E}$ there exists an $A$-restricted improvement embedding $\bar{E}$ so that*

1. *$\tilde{E}_R$ is the $A_R$-representative of $\bar{E}$;*
2. *$(\mathcal{E},\mathcal{I},\mathcal{C})$ globally represents $\bar{E}$; and*
3. *if $w$ is nonleaf and has $r$-children $w_1$ and $w_2$ with $r$-tables $(\mathcal{E}_1,\mathcal{I}_1,\mathcal{C}_1)$ and $(\mathcal{E}_2,\mathcal{I}_2,\mathcal{C}_2)$, then $\mathcal{L}$ is a function mapping $\tilde{E}_R$ to a pair $\mathcal{L}(\tilde{E}_R) = (\tilde{E}_1, \tilde{E}_2)$ such that $\tilde{E}_1 \in \mathcal{E}_1$, $\tilde{E}_2 \in \mathcal{E}_2$, $\tilde{E}_1$ is the $T_r[w_1]_{\mathcal{R}_r[w_1]}$-representative of $\bar{E} \cap T_r[w_1]$, and $\tilde{E}_2$ is the $T_r[w_2]_{\mathcal{R}_r[w_2]}$-representative of $\bar{E} \cap T_r[w_2]$.*

The existence of a linked $r$-table will be formally proved in Lemma 6.47 when also its construction is given. Note that a linked $r$-table of a node $w$ can be represented in $2^{2^{\mathcal{O}(k)}}$ space. We start implementing the dynamic programming from the leaves.

LEMMA 6.46. *Let $(T,\mathcal{R})$ be an augmented rank decomposition, let $r \in E(T)$, and let $w$ be a leaf node of $T$. A linked $r$-table of $w$ can be constructed in time $2^{\mathcal{O}(k)}$.*

*Proof.* Let $A = T_r[w]$. As $|A| = 1$, the number of $A$-restricted improvement embeddings is $2^{\mathcal{O}(k)}$, and we can iterate over all of them and construct the $r$-table directly by definition. Moreover, the $r$-table is by definition linked because $A$ is the minimal representative of itself, and so there is bijection between $A$-restricted improvement embeddings and their $A_A$-representatives. ∎

The next lemma is the main dynamic programming lemma. It specifies the computation of linked $r$-tables for nonleaf nodes.

LEMMA 6.47. *Let $(T,\mathcal{R})$ be an augmented rank decomposition, $r \in E(T)$, $w$ a nonleaf node of $T$, and $w_1$ and $w_2$ the $r$-children of $w$. Given linked $r$-tables of the nodes $w_1$ and $w_2$, a linked $r$-table $(\mathcal{E},\mathcal{I},\mathcal{C},\mathcal{L})$ of $w$ can be constructed in time $2^{2^{\mathcal{O}(k)}}$.*

*Proof.* Let $A = T_r[w]$, $X = T_r[w_1]$, $Y = T_r[w_2]$, $R = \mathcal{R}_r[w]$, $R_X = \mathcal{R}_r[w_1]$, and $R_Y = \mathcal{R}_r[w_2]$. Let $(\mathcal{E}_1, \mathcal{I}_1, \mathcal{C}_1)$ be the given $r$-table of $w_1$ and $(\mathcal{E}_2, \mathcal{I}_2, \mathcal{C}_2)$ the given $r$-table of $w_2$.

Let $\bar{E}$ be a $k$-bounded $A$-restricted improvement embedding. Note that the number of nodes in the $r$-subtree of $w$ that $\bar{E}$ $r$-intersects is $i_1 + i_2 + i_w$, where $i_1$ is the number of nodes in the $r$-subtree of $w_1$ that $\bar{E} \cap X$ $r$-intersects, $i_2$ is the number of nodes in the $r$-subtree of $w_2$ that $\bar{E} \cap Y$ $r$-intersects, and $i_w = 1$ if $\bar{E}$ $r$-intersects $w$ and 0 otherwise. Moreover, the fact whether $\bar{E}$ $r$-intersects $w$ can be determined only by considering the $A_R$-representative of $\bar{E}$, in particular by whether it is $C_i$-empty for at most one $i$.

We enumerate all pairs $(\tilde{E}_1, \tilde{E}_2) \in \mathcal{E}_1 \times \mathcal{E}_2$. If $\tilde{E}_1$ is compatible with $\tilde{E}_2$, then by Lemma 6.35, for any $\bar{E}_X$ and $\bar{E}_Y$ such that $\tilde{E}_1$ is a $X_{R_X}$-representative of $\bar{E}_X$ and $\tilde{E}_2$ is a $Y_{R_Y}$-representative of $\bar{E}_Y$ it holds that $\bar{E} = \bar{E}_X \cup \bar{E}_Y$ is an $A$-restricted improvement embedding such that $\bar{E} \cap X = \bar{E}_X$, $\bar{E} \cap Y = \bar{E}_Y$, and the composition $\tilde{E}_R$ of $\tilde{E}_1$ and $\tilde{E}_2$ is the $A_R$-representative of $\bar{E}$. Let us fix such $\bar{E}_X$ and $\bar{E}_Y$ so that they are globally represented by $(\mathcal{E}_1, \mathcal{I}_1, \mathcal{C}_1)$ and $(\mathcal{E}_2, \mathcal{I}_2, \mathcal{C}_2)$, respectively. If $\tilde{E}_R \notin \mathcal{E}$, or $\tilde{E}_R \in \mathcal{E}$ but $\mathcal{I}(\tilde{E}_R) > \mathcal{I}_1(\tilde{E}_1) + \mathcal{I}_2(\tilde{E}_2) + i_w$, we insert $\tilde{E}_R$ to $\mathcal{E}$ and set $\mathcal{I}(\tilde{E}_R) \leftarrow \mathcal{I}_1(\tilde{E}_1) + \mathcal{I}_2(\tilde{E}_2) + i_w$, $\mathcal{C}(\tilde{E}_R) \leftarrow \mathcal{C}_1(\tilde{E}_1) \cup \mathcal{C}_2(\tilde{E}_2)$ and $\mathcal{L}(\tilde{E}_R) \leftarrow (\tilde{E}_1, \tilde{E}_2)$. Now $(\mathcal{E}, \mathcal{I}, \mathcal{C})$ globally represents $\bar{E}$.

As the number of such pairs is $2^{2^{\mathcal{O}(k)}}$, this algorithm clearly works in time $2^{2^{\mathcal{O}(k)}}$.

The fact that for all $A$-restricted improvement embeddings $\bar{E}$ we actually considered a pair $(\tilde{E}_1, \tilde{E}_2) \in \mathcal{E}_1 \times \mathcal{E}_2$ so that $\tilde{E}_1$ is a $X_{R_X}$-representative of $\bar{E} \cap X$ and $\tilde{E}_2$ is a $Y_{R_Y}$-representative of $\bar{E} \cap Y$ follows from the definition of $r$-state and Lemma 6.28, i.e., the fact that $\bar{E} \cap X$ is an $X$-restricted improvement embedding and $\bar{E} \cap Y$ is a $Y$-restricted improvement embedding. ◻

Now the Init($T$, $r$) operation can be implemented in $|V(T)| 2^{2^{\mathcal{O}(k)}}$ time by constructing a linked $r$-table of each node in the order from leaves to root with $|V(T)|$ applications of Lemma 6.47. For the Move($vw$) operation, observe that the linked $r$-table of a node $x \in V(T)$ depends only on the $r$-subtree of $x$, and therefore by Observation 5.8 it suffices to recompute only the linked $r$-table of $v$ when using Move($vw$). Therefore Move($vw$) can be implemented in $2^{2^{\mathcal{O}(k)}}$ time by a single application of Lemma 6.47. The operation Width() is implemented without $r$-tables. It amounts to finding the smallest $k'$ for which $G[\mathcal{R}[uv], \mathcal{R}[vu]]$ has an embedding to $R_{k'}$, which can be done by brute force in time $2^{2^{\mathcal{O}(k)}}$. The Output() operation also is straightforward as we just output the augmented rank decomposition we are maintaining.

It remains to implement CanRefine(), EditSet(), and Refine($R$, $(N_1, N_2, N_3)$). The following is the main lemma for them, providing the implementations of CanRefine() and EditSet() and setting the stage for Refine($R$, $(N_1, N_2, N_3)$).

LEMMA 6.48. *Let $(T, \mathcal{R})$ be an augmented rank decomposition, $uv = r \in E(T)$, and $W = T[uv]$. For each node $w \in V(T)$, let $(\mathcal{E}_w, \mathcal{I}_w, \mathcal{R}_w, \mathcal{L}_w)$ be the linked $r$-table of $w$. There is an algorithm that returns $\perp$ if there is no $W$-improvement, and otherwise returns a tuple $(R, N_1, N_2, N_3, C_1^R, C_2^R, C_3^R)$, where $(r, C_1, C_2, C_3)$ is a global $T$-improvement, $R$ is the edit set of $(r, C_1, C_2, C_3)$, $(N_1, N_2, N_3)$ is the neighbor partition of $R$, and for each $i \in \{1, 2, 3\}$, $C_i^R$ is a minimal representative of $C_i$. The algorithm runs in time $2^{2^{\mathcal{O}(k)}}(|R| + 1)$.*

*Proof.* Denote $X = W = T_r[u]$, $Y = \overline{W} = T_r[v]$, and let $R_X = \mathcal{R}_r[u]$ and $R_Y = \mathcal{R}_r[v]$.

We iterate over all pairs $(\tilde{E}_u, \tilde{E}_v) \in \mathcal{E}_u \times \mathcal{E}_v$ satisfying the conditions in Lemma 6.40 and determine the width, the arity, and the sum-width of a corresponding $W$-improvement as in Lemma 6.40. Also, the minimum number of nodes of $T$ $r$-intersected by a $W$-improvement corresponding to $(\tilde{E}_u, \tilde{E}_v)$ can be determined as $\mathcal{I}_u(\tilde{E}_u)+\mathcal{I}_v(\tilde{E}_v)$. If no such pair is found, we return $\perp$. Otherwise we find a pair $(\tilde{E}_u, \tilde{E}_v)$ so that $\tilde{E}_u$ is the $X_{R_X}$-representative of an $X$-restricted improvement embedding $\bar{E}_X$ that is globally represented by $(\mathcal{E}_u, \mathcal{I}_u, \mathcal{C}_u)$ and whose tripartition is $(C_1 \cap X, C_2 \cap X, C_3 \cap X)$, $\tilde{E}_v$ is the $Y_{R_Y}$-representative of a $Y$-restricted improvement embedding $\bar{E}_Y$ that is globally represented by $(\mathcal{E}_v, \mathcal{I}_v, \mathcal{C}_v)$ and whose tripartition is $(C_1 \cap Y, C_2 \cap Y, C_3 \cap Y)$, and $(r, C_1, C_2, C_3)$ is a global $T$-improvement of $T$. As $|\mathcal{E}_u||\mathcal{E}_v| = 2^{2^{\mathcal{O}(k)}}$ and each pair can be checked in time $2^{2^{\mathcal{O}(k)}}$ (root-compatibility by Lemma 6.37), this phase has time complexity $2^{2^{\mathcal{O}(k)}}$.

Let $(f_1^C, f_2^C, f_3^C) = \mathcal{C}_u(\tilde{E}_u) \cup \mathcal{C}_v(\tilde{E}_v)$. Now $f_i^C$ is a concrete representative of an embedding of $G[(X \cap C_i) \cup (Y \cap C_i), (X \setminus C_i) \cup (Y \setminus C_i)] = G[C_i, \overline{C_i}]$ to $R_{k_i}$, and therefore by Observation 6.16 we obtain a representative of $(C_i, \overline{C_i})$ of size at most $2 \cdot 2^{k_i}$ from $f_i^C$. We compute a minimal representative $C_i^R$ of $C_i$ in time $2^{\mathcal{O}(k)}$ by Lemma 6.5.

We compute the edit set and the neighbor partition with a BFS-type algorithm that maintains a queue $Q$ containing pairs $(w, \tilde{E}_w)$, where $w \in V(T)$, with the invariant that if $w$ is in the $r$-subtree of $u$, then $\tilde{E}_w$ is the $T_r[w]_{\mathcal{R}_r[w]}$-representative of $\bar{E}_X \cap T_r[w]$ and if $w$ is in the $r$-subtree of $v$, then $\tilde{E}_w$ is the $T_r[w]_{\mathcal{R}_r[w]}$-representative of $\bar{E}_Y \cap T_r[w]$. We start by inserting the pairs $(u, \tilde{E}_u)$ and $(v, \tilde{E}_v)$ to $Q$. Then we iteratively pop a pair $(w, \tilde{E}_w)$ from the queue. If there is $i$ such that $\tilde{E}_w$ is $C_j$-empty for all $j \neq i$, then it holds that $T_r[w] \subseteq C_i$, and therefore we insert $w$ to $N_i$. Otherwise, we insert $w$ to $R$, let $w_1$ and $w_2$ be the $r$-children of $w$, and let $(\tilde{E}_1, \tilde{E}_2) = \mathcal{L}_w(\tilde{E}_w)$. We insert $(\tilde{E}_1, w_1)$ and $(\tilde{E}_2, w_2)$ into $Q$. This maintains the invariant by the definition of a linked $r$-table.

For each node $w \in R \cup N_1 \cup N_2 \cup N_3$ we access tables indexed by $\tilde{E}_w$ and determine $C_i$-emptiness, so the work is bounded by $|R \cup N_1 \cup N_2 \cup N_3| \cdot 2^{2^{\mathcal{O}(k)}} = |R| \cdot 2^{2^{\mathcal{O}(k)}}$. □

What is left is the Refine($R$, $(N_1, N_2, N_3)$) operation. Computing the refinement $T'$ of $T$ itself is a direct application of Lemma 5.4 and computing the linked $r$-tables of the new nodes is done by $|R|$ applications of Lemma 6.47, but in order to compute the linked $r$-tables we must first make $T'$ augmented, that is, for each new edge $uv$ compute a minimal representative of $(T'[uv], T'[vu])$. We use the minimal representatives of $C_1$, $C_2$, and $C_3$ returned by the algorithm of Lemma 6.48 for this.

LEMMA 6.49. *Let $T$ be a rank decomposition, let $r \in E(T)$, and let $(r, C_1, C_2, C_3)$ be a global $T$-improvement. Let $w$ be a nonleaf node of $T$, and let $w_1$ and $w_2$ be the $r$-children of $w$. Let $i$,$j$, and $l$ be such that $\{i, j, l\} = \{1, 2, 3\}$. Given minimal representatives of $T_r[w_1] \cap C_i$, $T_r[w_2] \cap C_i$, $\overline{T_r[w]}$, $C_j$, and $C_l$, a minimal representative of $(T_r[w] \cap C_i, \overline{T_r[w] \cap C_i})$ can be computed in $2^{\mathcal{O}(k)}$ time.*

*Proof.* As $(r, C_1, C_2, C_3)$ is a global $T$-improvement of $T$, by Theorem 4.9 and Proposition 6.3 each of the given minimal representatives has size at most $2^k$.

It holds that

$$T_r[w] \cap C_i = (T_r[w_1] \cap C_i) \cup (T_r[w_2] \cap C_i),$$

so by Lemma 6.6 we obtain a representative of $T_r[w] \cap C_i$ as the union of the given minimal representatives of $T_r[w_1] \cap C_i$ and $T_r[w_2] \cap C_i$. It also holds that

$$\overline{T_r[w] \cap C_i} = \overline{T_r[w]} \cup \overline{C_i} = \overline{T_r[w]} \cup C_j \cup C_l,$$

so again by Lemma 6.6 we obtain a representative of $\overline{T_r[w] \cap C_i}$ as the union of the given minimal representatives of $\overline{T_r[w]}$, $C_j$, and $C_l$. Now we have a representative of $(T_r[w] \cap C_i, \overline{T_r[w] \cap C_i})$ of size $2^{\mathcal{O}(k)}$, so we can use Lemma 6.5 to compute a minimal representative in time $2^{\mathcal{O}(k)}$. ∎

In particular, a minimal representative of $\overline{T_r[w]}$ is available in $(T, \mathcal{R})$ as $\mathcal{R}[pw]$, where $p$ is the $r$-parent of $w$, and minimal representatives of $T_r[w_1] \cap C_i$ and $T_r[w_2] \cap C_i$ are available by doing the construction in an order towards the root $r$.

This completes the description of the refinement data structure for augmented rank decompositions and thus also the proof of Theorem 6.13.

## 7. Approximating graph branchwidth.

In this section we prove the following theorem.

THEOREM 1.5. *There is an algorithm that, given an $n$-vertex graph $G$ and an integer $k$, in time $2^{\mathcal{O}(k)}n$ either computes a branch decomposition of $G$ of width at most $2k$ or correctly concludes that the branchwidth of $G$ is more than $k$.*

We start with the definition of branch decomposition of a graph.

DEFINITION 7.1 (border). *Let $G$ be a graph and $X \subseteq E(G)$. The border $\delta_G(X)$ of $X$ is the set of vertices of $G$ that are incident to an edge in $X$ and to an edge in $E(G) \setminus X$.*

The following result is well known.

PROPOSITION 7.2. *The function $|\delta_G| : 2^{E(G)} \to \mathbb{Z}_{\geq 0}$ assigning for each $X \subseteq E(G)$ the cardinality $|\delta_G(X)|$ of the border of $X$ is a connectivity function.*

A *branch decomposition* of a graph $G$ is a branch decomposition of function $|\delta_G|$, and the *branchwidth* of $G$, denoted by $\texttt{bw}(G)$, is the branchwidth of $|\delta_G|$. In the rest of this section we assume that we are computing the branchwidth of a graph $G$ and drop the subscript.

For branchwidth we do not need iterative compression as we can use the algorithm of Korhonen [32] and the connection between branchwidth and treewidth [41] to obtain a branch decomposition of $G$ of width at most $3k$ in $2^{\mathcal{O}(k)}n$ time.

The following is the main lemma, and the whole section is devoted to its proof.

LEMMA 7.3. *There is a refinement data structure for branch decompositions of graphs with time complexity $t(k) = 2^{\mathcal{O}(k)}$.*

With Theorem 2.3 and the aforementioned connections to treewidth, this will prove Theorem 1.5.

The remaining part of this section is organized as follows. In subsection 7.1 we define augmented branch decompositions, in subsection 7.2 we introduce the objects manipulated in our dynamic programming and prove some properties of them. Then in subsection 7.3 we give the refinement data structure for branch decompositions using this dynamic programming.

### 7.1. Augmented branch decompositions.

In our refinement data structure we maintain an augmented branch decomposition. An augmented branch decompositions stores the *border description* of each bipartition corresponding to an edge of it.

DEFINITION 7.4 (border description). *Let $X \subseteq E(G)$. The* border description *of $X$ is the pair $(\delta(X), f)$, where $f : \delta(X) \to \mathbb{Z}_{\geq 0}$ is the function so that $f(v)$ is the number of edges in $X$ incident to $v$.*

An *augmented branch decomposition* is a branch decomposition $T$ where for each edge $uv \in E(T)$ the border descriptions of $T[uv]$ and $T[vu]$ are stored. Note that an augmented branch decomposition can be represented in $\mathcal{O}(|V(T)|\mathtt{bw}(T))$ space.

The following lemma leads to an algorithm for computing the border descriptions.

LEMMA 7.5. *Let $X, Y$ be disjoint subsets of $E(G)$. Given the border descriptions of $X$ and $Y$, the border description of $X \cup Y$ can be computed in $\mathcal{O}(|\delta(X)| + |\delta(Y)|)$ time.*

*Proof.* Let $(\delta(X), f)$ be the border description of $X$ and $(\delta(Y), g)$ the border description of $Y$. It holds that $\delta(X \cup Y) \subseteq \delta(X) \cup \delta(Y)$, where $(\delta(X) \cup \delta(Y)) \setminus \delta(X \cup Y)$ can be identified as the vertices $v$ for which $f(v) + g(v)$ is the degree of $v$. Now $(\delta(X \cup Y), f + g)$ is the border description of $X \cup Y$. ◻

Because $\delta(\overline{X}) = \delta(X)$ and the number of edges in $\overline{X}$ incident to $v$ is the degree of $v$ minus the number of edges in $X$ incident to $v$, the border description of $\overline{X}$ can be computed from the border description of $X$ in $\mathcal{O}(|\delta(X)|)$ time. It follows that, given a branch decomposition $T$, a corresponding augmented branch decomposition can be computed in $\mathcal{O}(|V(T)|\mathtt{bw}(T))$ time by using Lemma 7.5 $\mathcal{O}(|V(T)|)$ times.

**7.2. Borders of tripartitions.** We define the partial solutions stored in dynamic programming tables.

DEFINITION 7.6 (border of a tripartition). *Let $A \subseteq E(G)$, and let $(C_1, C_2, C_3)$ be a tripartition of $A$. The* border *of $(C_1, C_2, C_3)$ is the 9-tuple*

$$(R_1, R_2, R_3, r_1, r_2, r_3, k_1, k_2, k_3),$$

*where for each $i \in \{1,2,3\}$ it holds that $R_i = \delta(C_i) \cap \delta(A)$, $r_i = 0$ if $C_i = \emptyset$ and otherwise $r_i = 1$, and $k_i$ is the number of vertices $v \in V(G) \setminus \delta(A)$ such that there exists an edge $e_1 \in C_i$ incident to $v$ and an edge $e_2 \in A \setminus C_i$ incident to $v$, i.e., $k_i = |\delta(C_i) \cap \delta(A \setminus C_i) \setminus \delta(A)|$.*

We call a border of tripartition $k$-bounded if for each $i$ it holds that $k_i \leq k$. Note that if $|\delta(A)| \leq k$, then the number of $k$-bounded borders of tripartitions of $A$ is $\leq (2^k)^3 2^3 k^3 = 2^{\mathcal{O}(k)}$, and each of them can be represented in $\mathcal{O}(k)$ space.

We define the *composition* of borders of tripartitions to combine partial solutions.

DEFINITION 7.7 (composition). *Let $X$ and $Y$ be disjoint subsets of $E(G)$, and let $A = X \cup Y$. Let $R_X = (R_1^X, R_2^X, R_3^X, r_1^X, r_2^X, r_3^X, k_1^X, k_2^X, k_3^X)$ be the border of a tripartition of $X$ and $R_Y = (R_1^Y, R_2^Y, R_3^Y, r_1^Y, r_2^Y, r_3^Y, k_1^Y, k_2^Y, k_3^Y)$ the border of a tripartition of $Y$. Denote $F = (\delta(X) \cup \delta(Y)) \setminus \delta(A)$. The* composition *of $R_X$ and $R_Y$ is the 9-tuple $(R_1, R_2, R_3, r_1, r_2, r_3, k_1, k_2, k_3)$, where for each $i \in \{1,2,3\}$,*

1. *$R_i = \delta(A) \cap (R_i^X \cup R_i^Y)$;*
2. *$r_i = \max(r_i^X, r_i^Y)$; and*
3. *$k_i = k_i^X + k_i^Y + |F \cap (R_i^X \cup R_i^Y) \cap (R_j^X \cup R_l^X \cup R_j^Y \cup R_l^Y)|$, where $\{i, j, l\} = \{1, 2, 3\}$.*

Note that when the sets $\delta(X)$, $\delta(Y)$, and $\delta(A)$ are given and have size $\leq k$, the composition can be computed in $\mathcal{O}(k)$ time.

Next we prove that the composition operation really combines partial solutions, as expected.

LEMMA 7.8. *Let $X$ and $Y$ be disjoint subsets of $E(G)$, and let $A = X \cup Y$. If $R_X$ is the border of a tripartition $(C_1^X, C_2^X, C_3^X)$ and $R_Y$ is the border of a tripartition*

*$(C_1^Y, C_2^Y, C_3^Y)$, then the composition of $R_X$ and $R_Y$ is the border of the tripartition $(C_1^X \cup C_1^Y, C_2^X \cup C_2^Y, C_3^X \cup C_3^Y)$.*

*Proof.* Let $V^X$ be the set of vertices incident to $X$, and $V^Y$ the set of vertices incident to $Y$. Also let $F = (\delta(X) \cup \delta(Y)) \setminus \delta(A)$. Let $i \in \{1,2,3\}$, and let $j$ and $l$ be such that $\{i,j,l\} = \{1,2,3\}$. It holds that

$$\delta(C_i^X \cup C_i^Y) = (\delta(C_i^X) \cup \delta(C_i^Y)) \cap (\delta(C_j^X) \cup \delta(C_l^X) \cup \delta(C_j^Y) \cup \delta(C_l^Y) \cup \delta(\overline{A})),$$

which by $\delta(\overline{A}) = \delta(A)$ implies

$$\delta(C_i^X \cup C_i^Y) \cap \delta(A) = (\delta(C_i^X) \cup \delta(C_i^Y)) \cap \delta(A).$$

Then, because $\delta(A) \cap V^X \subseteq \delta(X)$ and $\delta(A) \cap V^Y \subseteq \delta(Y)$, we have

$$\begin{aligned}\delta(C_i^X \cup C_i^Y) \cap \delta(A) &= (\delta(C_i^X) \cup \delta(C_i^Y)) \cap \delta(A)\\ &= ((\delta(C_i^X) \cap \delta(X)) \cup (\delta(C_i^Y) \cap \delta(Y))) \cap \delta(A)\\ &= (R_i^X \cup R_i^Y) \cap \delta(A),\end{aligned}$$

so the sets $R_1$, $R_2$, and $R_3$ in the composition are correct.

Let us then show that the numbers $k_1$, $k_2$, and $k_3$ in the composition are correct. First, note that

$$\begin{aligned}&|\delta(C_i^X \cup C_i^Y) \cap \delta(A \setminus (C_i^X \cup C_i^Y)) \setminus \delta(A)|\\ &\quad = |\delta(C_i^X \cup C_i^Y) \setminus \delta(A)|\\ &\quad = |\delta(C_i^X \cup C_i^Y) \cap F| + |\delta(C_i^X \cup C_i^Y) \setminus (F \cup \delta(A))|.\end{aligned} \tag{7.1}$$

By observing that $V^X \cap F \subseteq \delta(X)$ and $V^Y \cap F \subseteq \delta(Y)$, we have that

$$\begin{aligned}\delta(C_i^X \cup C_i^Y) \cap F &= F \cap (\delta(C_i^X) \cup \delta(C_i^Y)) \cap (\delta(C_j^X) \cup \delta(C_l^X) \cup \delta(C_j^Y) \cup \delta(C_l^Y))\\ &= F \cap (R_i^X \cup R_i^Y) \cap (R_j^X \cup R_l^X \cup R_j^Y \cup R_l^Y).\end{aligned}$$

Since $V^X \setminus (F \cup \delta(A)) = V^X \setminus \delta(X)$ and $V^Y \setminus (F \cup \delta(A)) = V^Y \setminus \delta(Y)$ are disjoint, we get that

$$\begin{aligned}&|\delta(C_i^X \cup C_i^Y) \setminus (F \cup \delta(A))|\\ &\quad = |(\delta(C_i^X) \cap (\delta(C_j^X) \cup \delta(C_l^X)) \setminus \delta(X)| + |(\delta(C_i^Y) \cap (\delta(C_j^Y) \cup \delta(C_l^Y)) \setminus \delta(Y)|\\ &\quad = |\delta(C_i^X) \cap \delta(X \setminus C_i^X) \setminus \delta(X)| + |\delta(C_i^Y) \cap \delta(Y \setminus C_i^Y) \setminus \delta(Y)|\\ &\quad = k_i^X + k_i^Y.\end{aligned}$$

Hence, by (7.1) the numbers $k_1$, $k_2$, and $k_3$ in the composition are correct. The numbers $r_1$, $r_2$, and $r_3$ are correct by observing that $C_i^X \cup C_i^Y$ is empty if and only if both $C_i^X$ and $C_i^Y$ are empty. ☐

The next lemma gives the method for determining if there exists a $W$-improvement based on borders of tripartitions.

LEMMA 7.9. *Let $T$ be a branch decomposition, and let $uv = r \in E(T)$ and $W = T[uv]$. Denote $X = W$ and $Y = \overline{W}$. There exists a $W$-improvement $(C_1, C_2, C_3)$ with arity $\alpha$ and $|\delta(C_i)| = k_i$ for each $i \in \{1,2,3\}$ if and only if there exists $R_X$ and $R_Y$ such that*

1. *$R_X = (R_1^X, R_2^X, R_3^X, r_1^X, r_2^X, r_3^X, k_1^X, k_2^X, k_3^X)$ is the border of the tripartition $(C_1 \cap X, C_2 \cap X, C_3 \cap X)$;*
2. *$R_Y = (R_1^Y, R_2^Y, R_3^Y, r_1^Y, r_2^Y, r_3^Y, k_1^Y, k_2^Y, k_3^Y)$ is the border of the tripartition $(C_1 \cap Y, C_2 \cap Y, C_3 \cap Y)$;*
3. *the composition of $R_X$ and $R_Y$ is $(\emptyset, \emptyset, \emptyset, r_1, r_2, r_3, k_1, k_2, k_3)$, where it holds that $r_1 + r_2 + r_3 = \alpha$ and $k_i < |\delta(W)|/2$ for each $i$; and*
4. *for each $i$ it holds that $k_i^X + |R_i^X| < |\delta(W)|$ and $k_i^Y + |R_i^Y| < |\delta(W)|$.*

*Proof.* Suppose that such $R_X$ and $R_Y$ exist. By Lemma 7.8 and the fact that $\delta(E(G)) = \emptyset$, $(\emptyset, \emptyset, \emptyset, r_1, r_2, r_3, k_1, k_2, k_3)$ is the border of $(C_1, C_2, C_3)$. By the definition of border we have that $k_i = |\delta(C_i)|$ and $r_1 + r_2 + r_3$ is the number of nonempty sets in $(C_1, C_2, C_3)$. It remains to prove that $|\delta(C_i \cap W)| = k_i^X + |R_i^X|$ and that $|\delta(C_i \cap \overline{W})| = k_i^Y + |R_i^Y|$ for each $i$. We have that

$$\begin{aligned}\delta(W \cap C_i) &= (\delta(W \cap C_i) \cap \delta(\overline{W})) \cup (\delta(W \cap C_i) \cap \delta(W \setminus C_i)) \\ &= (\delta(X \cap C_i) \cap \delta(X)) \cup (\delta(X \cap C_i) \cap \delta(X \setminus C_i)) \\ &= (\delta(X \cap C_i) \cap \delta(X)) \cup (\delta(X \cap C_i) \cap \delta(X \setminus C_i) \cap \delta(X)) \\ &\quad \cup (\delta(X \cap C_i) \cap \delta(X \setminus C_i) \setminus \delta(X)) \\ &= (\delta(X \cap C_i) \cap \delta(X)) \cup (\delta(X \cap C_i) \cap \delta(X \setminus C_i) \setminus \delta(X)).\end{aligned}$$

Therefore, by the definition of border,

$$\begin{aligned}&|(\delta(X \cap C_i) \cap \delta(X)) \cup (\delta(X \cap C_i) \cap \delta(X \setminus C_i) \setminus \delta(X))| \\ &\quad = |\delta(X \cap C_i) \cap \delta(X)| + |\delta(X \cap C_i) \cap \delta(X \setminus C_i) \setminus \delta(X)| = |R_i^X| + k_i^X.\end{aligned}$$

The other case is symmetric.

The above is the proof of the if direction. The proof for the only if direction is the same but starting from the supposition that such $W$-improvement $(C_1, C_2, C_3)$ exists and letting $R_X$ be the border of $(C_1 \cap X, C_2 \cap X, C_3 \cap X)$ and $R_Y$ the border of $(C_1 \cap Y, C_2 \cap Y, C_3 \cap Y)$. ◻

**7.3. Refinement data structure for graph branch decompositions.** In the refinement data structure for branch decompositions we maintain an augmented branch decomposition $T$ rooted at edge $r \in E(T)$ and a dynamic programming table that stores for each node $w \in V(T)$ all $k$-bounded borders of tripartitions of $T_r[w]$ and information about how many nodes in the $r$-subtree of $w$ they intersect. We call this dynamic programming table an $r$-table to signify that it is directed towards $r$.

In this subsection we always assume that $k$ is an integer such that $\mathtt{bw}(T) \leq k$, and therefore we only care about $k$-bounded borders of tripartitions. Next we formally define the contents of an $r$-table.

DEFINITION 7.10 ($r$-table). *Let $T$ be a branch decomposition, $r \in E(T)$ an edge of $T$, $w \in V(T)$, and $A = T_r[w]$. The* $r$-table *of $w$ is the pair $(\mathcal{B}, \mathcal{I})$, where $\mathcal{B}$ is the set of all $k$-bounded borders of tripartitions of $A$, and $\mathcal{I}$ is a function mapping each $R \in \mathcal{B}$ to the least integer $i$ such that there exists a tripartition of $A$ whose border is $R$ and that $r$-intersects $i$ nodes of the $r$-subtree of $w$.*

As there are $2^{\mathcal{O}(k)}$ $k$-bounded tripartitions of $T_r[w]$, the $r$-table of $w$ can be represented in $2^{\mathcal{O}(k)}$ space.

LEMMA 7.11. *Let $T$ be an augmented branch decomposition, let $r \in E(T)$, and let $w$ be a nonleaf node of $T$ with $r$-children $w_1$ and $w_2$. Given the $r$-tables of $w_1$ and $w_2$, the $r$-table of $w$ can be constructed in $2^{\mathcal{O}(k)}$ time.*

*Proof.* Denote $A = T_r[w]$, $X = T_r[w_1]$, and $Y = T_r[w_2]$. Let $(\mathcal{B}_{w_1}, \mathcal{I}_{w_1})$ and $(\mathcal{B}_{w_2}, \mathcal{I}_{w_2})$ be the $r$-tables of $w_1$ and $w_2$.

We construct the $r$-table $(\mathcal{B}, \mathcal{I})$ of $w$ as follows. We iterate over all pairs

$$(R_X, R_Y) \in \mathcal{B}_{w_1} \times \mathcal{B}_{w_2}$$

and let $\mathcal{B}$ be the set of compositions of those pairs. This correctly constructs $\mathcal{B}$ by Lemma 7.8 and the observation that if $(C_1, C_2, C_3)$ is a tripartition of $A$ whose border is $k$-bounded, then $(C_1 \cap X, C_2 \cap X, C_3 \cap X)$ is a tripartition of $X$ whose border is $k$-bounded and $(C_1 \cap Y, C_2 \cap Y, C_3 \cap Y)$ is a tripartition of $Y$ whose border is $k$-bounded. For each $R \in \mathcal{B}$ we set $\mathcal{I}(R)$ as the minimum value of $\mathcal{I}_{w_1}(R_X) + \mathcal{I}_{w_2}(R_Y) + i_w$ over such pairs $R_X$, $R_Y$ whose composition is $R$, where $i_w = 1$ if $R$ is of the form $(\ldots, r_1, r_2, r_3, \ldots)$ where $r_1 + r_2 + r_3 \geq 2$, and $i_w = 0$ otherwise. This correctly constructs $\mathcal{I}$ by the observations that $(C_1, C_2, C_3)$ $r$-intersects a node $w'$ in the $r$-subtree of $w_1$ if and only if $(C_1 \cap X, C_2 \cap X, C_3 \cap X)$ intersects $w'$, $(C_1, C_2, C_3)$ $r$-intersects a node $w'$ in the $r$-subtree of $w_2$ if and only if $(C_1 \cap Y, C_2 \cap Y, C_3 \cap Y)$ intersects $w'$, and that $(C_1, C_2, C_3)$ intersects $w$ if and only if $r_1 + r_2 + r_3 \geq 2$.

As $|\mathcal{B}_{w_1}||\mathcal{B}_{w_2}| = 2^{\mathcal{O}(k)}$, the time complexity is $2^{\mathcal{O}(k)}$. ∎

Now the Init($T$, $r$) operation can be implemented in $|V(T)|2^{\mathcal{O}(k)}$ time by first making $T$ augmented by $2|V(T)|$ applications of Lemma 7.5 and then constructing the $k$-bounded $r$-tables of all nodes in the order from leaves to root by $|V(T)|$ applications of Lemma 7.11. For the Move($vw$) operation, we note the $r$-table of a node $x \in V(T)$ depends only on the $r$-subtree of $x$, and therefore by Observation 5.8 it suffices to recompute only the $r$-table of the node $v$ when using Move($vw$). Therefore Move($vw$) can be implemented in $2^{\mathcal{O}(k)}$ time by a single application of Lemma 7.11. The Width() operation returns $|\delta(T[uv])|$, which is available because $T$ is augmented. The Output() operation is also straightforward as it just returns the branch decomposition we are maintaining.

The following lemma implements the operations CanRefine() and EditSet() based on Lemma 7.9.

LEMMA 7.12. *Let $T$ be a branch decomposition, and let $uv = r \in E(T)$ and $W = T[uv]$. For each node $w \in V(T)$ let $(\mathcal{B}_w, \mathcal{I}_w)$ be the $r$-table of $w$. There is an algorithm that returns $\perp$ if there is no $W$-improvement, and otherwise returns a tuple $(R, N_1, N_2, N_3)$, where $(r, C_1, C_2, C_3)$ is a global $T$-improvement, $R$ is the edit set of $(r, C_1, C_2, C_3)$, and $(N_1, N_2, N_3)$ is the neighbor partition of $R$. The algorithm runs in time $2^{\mathcal{O}(k)}(|R| + 1)$.*

*Proof.* Denote $X = W = T_r[u]$, $Y = \overline{W} = T_r[v]$. We iterate over all pairs $(R_X, R_Y) \in \mathcal{B}_u \times \mathcal{B}_v$, using Lemma 7.9 to either conclude that there exists no $W$-improvement or to find a pair $(R_X, R_Y)$ such that there is a global $T$-improvement $(r, C_1, C_2, C_3)$ so that $R_X$ is the border of $(C_1 \cap X, C_2 \cap X, C_3 \cap X)$, $R_Y$ is the border of $(C_1 \cap Y, C_2 \cap Y, C_3 \cap Y)$, and the number of nodes of $T$ $r$-intersected by $(C_1, C_2, C_3)$ is $\mathcal{I}_u(R_X) + \mathcal{I}_v(R_Y)$. In time $2^{\mathcal{O}(k)}$ we either find such a pair or conclude that there is no $W$-improvement.

We compute the edit set and the neighbor partition with a BFS-type algorithm that maintains a queue $Q$ containing pairs $(w, R_w)$, where $w \in V(T)$, with the invariant that there exists a global $T$-improvement $(r, C_1, C_2, C_3)$ so that for all $(w, R_w)$ that appear in the queue, $R_w$ is the border of $(C_1 \cap T_r[w], C_2 \cap T_r[w], C_3 \cap T_r[w])$. We start by inserting the pairs $(u, R_X)$ and $(v, R_Y)$ to $Q$. We iteratively pop a pair $(w, R_w)$ from the queue. Denote $R_w = (\ldots, r_1, r_2, r_3, \ldots)$. If there is $i$ such that $r_j = 0$ for both $j \neq i$, then it holds that $T_r[w] \subseteq C_i$, and therefore we insert $w$

into $N_i$. Otherwise, we insert $w$ into $R$, let $w_1$ and $w_2$ be the $r$-children of $w$, and find a pair $(R_{w_1}, R_{w_2}) \in \mathcal{B}_{w_1} \times \mathcal{B}_{w_2}$ so that the composition of $R_{w_1}$ and $R_{w_2}$ is $R_w$ and $\mathcal{I}_{w_1}(R_{w_1}) + \mathcal{I}_{w_2}(R_{w_2}) + 1 = \mathcal{I}_w(R_w)$. This kind of pair exists and maintains the invariant by the definition of $r$-table and Lemma 7.8.

For each node $w \in R \cup N_1 \cup N_2 \cup N_3$ we iterate over $\mathcal{B}_{w_1} \times \mathcal{B}_{w_2}$ and access some tables, so the total amount of work is bounded by $|R \cup N_1 \cup N_2 \cup N_3| 2^{\mathcal{O}(k)} = |R| 2^{\mathcal{O}(k)}$. ∎

What is left is the Refine($R$, $(N_1, N_2, N_3)$) operation. Computing the refinement of $T$ is done in $\mathcal{O}(|R|)$ time by applying Lemma 5.4. Computing the border descriptions of the newly inserted edges can be done in a bottom-up fashion starting from $N_1 \cup N_2 \cup N_3$ by $2|R|$ applications of Lemma 7.5. Then computing the $r$-tables of the newly inserted nodes can be done also in a similar fashion in $|R|2^{\mathcal{O}(k)}$ time by $|R|$ applications of Lemma 7.11.

This completes the description of the refinement data structure for branch decompositions and thus also the proof of Theorem 7.3.

## Appendix. Definitions of treewidth and cliquewidth.

*Treewidth.* A *tree decomposition* of a graph $G$ is a pair $(X, T)$, where $T$ is a tree whose vertices we will call *nodes* and $X = (\{X_i \mid i \in V(T)\})$ is a collection of subsets of $V(G)$ such that

1. $\bigcup_{i \in V(T)} X_i = V(G)$;
2. for each edge $(v, w) \in E(G)$, there is an $i \in V(T)$ such that $v, w \in X_i$; and
3. for each $v \in V(G)$ the set of nodes $\{i \mid v \in X_i\}$ forms a subtree of $T$.

The *width* of a tree decomposition $(\{X_i \mid i \in V(T)\}, T)$ equals $\max_{i \in V(T)}\{|X_i| - 1\}$. The *treewidth* of a graph $G$ is the minimum width over all tree decompositions of $G$.

*Cliquewidth.* Let $G$ be a graph, and let $k$ be a positive integer. A *$k$-graph* is a graph whose vertices are labeled by integers from $\{1, 2, \ldots, k\}$. We call the $k$-graph consisting of exactly one vertex labeled by some integer from $\{1, 2, \ldots, k\}$ an initial $k$-graph. The *cliquewidth* is the smallest integer $k$ such that $G$ can be constructed by means of repeated application of the following four operations on $k$-graphs:

- *introduce*: construction of an initial $k$-graph labeled by $i$ and denoted by $i(v)$ (that is, $i(v)$ is a $k$-graph with a single vertex);
- *disjoint union* (denoted by $\oplus$);
- *relabel*: changing all labels $i$ to $j$ (denoted by $\rho_{i \to j}$); and
- *join*: connecting all vertices labeled by $i$ with all vertices labeled by $j$ by edges (denoted by $\eta_{i,j}$).

Using the symbols of these operation, we can construct well-formed expressions. An expression is called *$k$-expression* for $G$ if the graph produced by performing these operations, in the order defined by the expression, is isomorphic to $G$ when labels are removed, and the *cliquewidth of* $G$ is the minimum $k$ such that there is a $k$-expression for $G$.

**Acknowledgments.** We thank Eunjung Kim and Sang-il Oum for answering our questions about their paper. We thank Michał Pilipczuk for suggesting the problem of approximating rankwidth. We also thank the anonymous referees for their comments.

## REFERENCES

[1] S. Arnborg, J. Lagergren, and D. Seese, *Easy problems for tree-decomposable graphs*, J. Algorithms, 12 (1991), pp. 308–340, https://doi.org/10.1016/0196-6774(91)90006-K.
[2] H. L. Bodlaender, *A linear-time algorithm for finding tree-decompositions of small treewidth*, SIAM J. Comput., 25 (1996), pp. 1305–1317, https://doi.org/10.1137/S0097539793251219.

[3] H. L. Bodlaender, P. G. Drange, M. S. Dregi, F. V. Fomin, D. Lokshtanov, and M. Pilipczuk, *A $c^k n$ 5-approximation algorithm for treewidth*, SIAM J. Comput., 45 (2016), pp. 317–378, https://doi.org/10.1137/130947374.

[4] H. L. Bodlaender and T. Kloks, *Efficient and constructive algorithms for the pathwidth and treewidth of graphs*, J. Algorithms, 21 (1996), pp. 358–402, https://doi.org/10.1006/jagm.1996.0049.

[5] H. L. Bodlaender and D. M. Thilikos, *Constructive linear time algorithms for branchwidth*, in Proceedings of the 24th International Colloquium on Automata, Languages and Programming (ICALP 1997), Lecture Notes in Comput. Sci. 1256, Springer, 1997, pp. 627–637, https://doi.org/10.1007/3-540-63165-8_217.

[6] R. B. Borie, R. G. Parker, and C. A. Tovey, *Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families*, Algorithmica, 7 (1992), pp. 555–581, https://doi.org/10.1007/BF01758777.

[7] B. Bui-Xuan, J. A. Telle, and M. Vatshelle, *H-join decomposable graphs and algorithms with runtime single exponential in rankwidth*, Discrete Appl. Math., 158 (2010), pp. 809–819, https://doi.org/10.1016/j.dam.2009.09.009.

[8] B. Bui-Xuan, J. A. Telle, and M. Vatshelle, *Boolean-width of graphs*, Theoret. Comput. Sci., 412 (2011), pp. 5187–5204, https://doi.org/10.1016/j.tcs.2011.05.022.

[9] W. Cook and P. D. Seymour, *Tour merging via branch-decomposition*, INFORMS J. Comput., 15 (2003), pp. 233–248, https://doi.org/10.1287/ijoc.15.3.233.16078.

[10] B. Courcelle, *The monadic second-order logic of graphs.* I. *Recognizable sets of finite graphs*, Inform. and Comput., 85 (1990), pp. 12–75, https://doi.org/10.1016/0890-5401(90)90043-H.

[11] B. Courcelle, *The expression of graph properties and graph transformations in monadic second-order logic*, in Handbook of Graph Grammars and Computing by Graph Transformation, World Scientific, 1997, pp. 313–400, https://doi.org/10.1142/9789812384720_0005.

[12] B. Courcelle and J. Engelfriet, *Graph Structure and Monadic Second-Order Logic: A Language-Theoretic Approach*, Cambridge University Press, 2012, https://doi.org/10.1017/CBO9780511977619.

[13] B. Courcelle, J. Engelfriet, and G. Rozenberg, *Handle-rewriting hypergraph grammars*, J. Comput. System Sci., 46 (1993), pp. 218–270, https://doi.org/10.1016/0022-0000(93)90004-G.

[14] B. Courcelle, J. A. Makowsky, and U. Rotics, *Linear time solvable optimization problems on graphs of bounded clique-width*, Theory Comput. Syst., 33 (2000), pp. 125–150, https://doi.org/10.1007/s002249910009.

[15] B. Courcelle, J. A. Makowsky, and U. Rotics, *On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic*, Discrete Appl. Math., 108 (2001), pp. 23–52, https://doi.org/10.1016/S0166-218X(00)00221-3.

[16] B. Courcelle and S. Oum, *Vertex-minors, monadic second-order logic, and a conjecture by Seese*, J. Combin. Theory Ser. B, 97 (2007), pp. 91–126, https://doi.org/10.1016/j.jctb.2006.04.003.

[17] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh, *Parameterized Algorithms*, Springer, 2015, https://doi.org/10.1007/978-3-319-21275-3.

[18] G. Damiand, M. Habib, and C. Paul, *A simple paradigm for graph recognition: Application to cographs and distance hereditary graphs*, Theoret. Comput. Sci., 263 (2001), pp. 99–111, https://doi.org/10.1016/S0304-3975(00)00234-6.

[19] C. C. Fast and I. V. Hicks, *A branch decomposition algorithm for the p-median problem*, INFORMS J. Comput., 29 (2017), pp. 474–488, https://doi.org/10.1287/ijoc.2016.0743.

[20] M. R. Fellows, F. A. Rosamond, U. Rotics, and S. Szeider, *Clique-width is NP-complete*, SIAM J. Discrete Math., 23 (2009), pp. 909–939, https://doi.org/10.1137/070687256.

[21] F. V. Fomin and D. M. Thilikos, *Dominating sets in planar graphs: Branch-width and exponential speed-up*, SIAM J. Comput., 36 (2006), pp. 281–309, https://doi.org/10.1137/S0097539702419649.

[22] R. Ganian and P. Hliněný, *On parse trees and Myhill-Nerode-type tools for handling graphs of bounded rank-width*, Discrete Appl. Math., 158 (2010), pp. 851–867, https://doi.org/10.1016/j.dam.2009.10.018.

[23] R. Ganian, P. Hliněný, and J. Obdržálek, *Better algorithms for satisfiability problems for formulas of bounded rank-width*, Fund. Inform., 123 (2013), pp. 59–76, https://doi.org/10.3233/FI-2013-800.

[24] M. U. Gerber and D. Kobler, *Algorithms for vertex-partitioning problems on graphs with fixed clique-width*, Theoret. Comput. Sci., 299 (2003), pp. 719–734, https://doi.org/10.1016/S0304-3975(02)00725-9.

[25] B. Godlin, T. Kotek, and J. A. Makowsky, *Evaluations of graph polynomials*, in Proceedings of the 34th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2008), Lecture Notes in Comput. Sci. 5344, Springer, 2008, pp. 183–194, https://doi.org/10.1007/978-3-540-92248-3_17.

[26] P. Hliněný, *A parametrized algorithm for matroid branch-width*, SIAM J. Comput., 35 (2005), pp. 259–277, https://doi.org/10.1137/S0097539702418589.

[27] P. Hliněný, *Branch-width, parse trees, and monadic second-order logic for matroids*, J. Combin. Theory Ser. B, 96 (2006), pp. 325–351, https://doi.org/10.1016/j.jctb.2005.08.005.

[28] P. Hliněný and S. Oum, *Finding branch-decompositions and rank-decompositions*, SIAM J. Comput., 38 (2008), pp. 1012–1032, https://doi.org/10.1137/070685920.

[29] P. Hliněný, S. Oum, D. Seese, and G. Gottlob, *Width parameters beyond tree-width and their applications*, Comput. J., 51 (2008), pp. 326–362, https://doi.org/10.1093/comjnl/bxm052.

[30] J. Jeong, E. J. Kim, and S. Oum, *Finding branch-decompositions of matroids, hypergraphs, and more*, SIAM J. Discrete Math., 35 (2021), pp. 2544–2617, https://doi.org/10.1137/19M1285895.

[31] D. Kobler and U. Rotics, *Edge dominating set and colorings on graphs with fixed clique-width*, Discrete Appl. Math., 126 (2003), pp. 197–221, https://doi.org/10.1016/S0166-218X(02)00198-1.

[32] T. Korhonen, *A single-exponential time* 2*-approximation algorithm for treewidth*, in Proceedings of the 62nd IEEE Annual Symposium on Foundations of Computer Science (FOCS 2021), IEEE, 2021, pp. 184–192, https://doi.org/10.1109/FOCS52979.2021.00026; to appear in SIAM J. Comput.

[33] S. Oum, *Graphs of Bounded Rank-width*, Ph.D. thesis, Princeton University, 2005.

[34] S. Oum, *Rank-width and vertex-minors*, J. Combin. Theory Ser. B, 95 (2005), pp. 79–100, https://doi.org/10.1016/j.jctb.2005.03.003.

[35] S. Oum, *Approximating rank-width and clique-width quickly*, ACM Trans. Algorithms, 5 (2008), 10, https://doi.org/10.1145/1435375.1435385.

[36] S. Oum, *Rank-width: Algorithmic and structural results*, Discrete Appl. Math., 231 (2017), pp. 15–24, https://doi.org/10.1016/j.dam.2016.08.006.

[37] S. Oum and P. Seymour, *Approximating clique-width and branch-width*, J. Combin. Theory Ser. B, 96 (2006), pp. 514–528, https://doi.org/10.1016/j.jctb.2005.10.006.

[38] S. Oum and P. D. Seymour, *Testing branch-width*, J. Combin. Theory Ser. B, 97 (2007), pp. 385–393, https://doi.org/10.1016/j.jctb.2006.06.006.

[39] M. Rao, *MSOL partitioning problems on graphs of bounded treewidth and clique-width*, Theoret. Comput. Sci., 377 (2007), pp. 260–267, https://doi.org/10.1016/j.tcs.2007.03.043.

[40] B. A. Reed, K. Smith, and A. Vetta, *Finding odd cycle transversals*, Oper. Res. Lett., 32 (2004), pp. 299–301, https://doi.org/10.1016/j.orl.2003.10.009.

[41] N. Robertson and P. D. Seymour, *Graph minors.* X. *Obstructions to tree-decomposition*, J. Combin. Theory Ser. B, 52 (1991), pp. 153–190, https://doi.org/10.1016/0095-8956(91)90061-N.

[42] P. D. Seymour and R. Thomas, *Call routing and the ratcatcher*, Combinatorica, 14 (1994), pp. 217–241, https://doi.org/10.1007/BF01215352.

[43] K. Suchan and I. Todinca, *On powers of graphs of bounded NLC-width (clique-width)*, Discrete Appl. Math., 155 (2007), pp. 1885–1893, https://doi.org/10.1016/j.dam.2007.03.014.

[44] E. Wanke, *k-NLC graphs and polynomial algorithms*, Discrete Appl. Math., 54 (1994), pp. 251–266, https://doi.org/10.1016/0166-218X(94)90026-4.