APPROXIMATION ALGORITHMS FOR TREEWIDTH

EYAL AMIR
COMPUTER SCIENCE DEPARTMENT
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
URBANA, IL 61801, USA
EYAL@CS.UIUC.EDU

Abstract. This paper presents algorithms whose input is an undirected graph, and whose output is a tree decomposition of width that approximates the optimal, the *treewidth* of that graph. The algorithms differ in their computation time and their approximation guarantees. The first algorithm works in polynomial-time and finds a factor-$O(\log OPT)$, where OPT is the treewidth of the graph. This is the first polynomial-time algorithm that approximates the optimal by a factor that does not depend on n, the number of nodes in the input graph. As a result, we get an algorithm for finding *pathwidth* within a factor of $O(\log OPT \cdot \log n)$ from the optimal. We also present algorithms that approximate the treewidth of a graph by constant factors of 3.66, 4, and 4.5, respectively and take time that is exponential in the treewidth. These are more efficient than previously known algorithms by an exponential factor, and are of practical interest. Finding triangulations of minimum treewidth for graphs is central to many problems in computer science. Real-world problems in artificial intelligence, VLSI design and databases are efficiently solvable if we have an efficient approximation algorithm for them. Many of those applications rely on weighted graphs. We extend our results to weighted graphs and *weighted treewidth*, showing similar approximation results for this more general notion. We report on experimental results confirming the effectiveness of our algorithms for large graphs associated with real-world problems.

Keywords: Treewidth, Triangulation, Tree Decomposition, Network Flow.

1. Introduction. The *treewidth* of an undirected graph $G(V, E)$ is the lowest *width* achievable by a *tree decomposition* (aka *junction tree* [45, 38]) of G. A tree decomposition of G is a tree $T = \langle \mathbb{X}, \mathbb{E} \rangle$ whose nodes $X \in \mathbb{X}$ are subsets of V such that (a) every vertex of V appears in at least one $X \in \mathbb{X}$, (b) for every graph edge $(u, v) \in E$, there is $X \in \mathbb{X}$ such that $u, v \in X$; and (c) if $u \in X_1 \cap X_2$ and X_3 is on the tree-path between X_1 and X_2, then $u \in X_3$ (this is called the *running-intersection property*). The *width* of a tree decomposition $T = \langle \mathbb{X}, \mathbb{E} \rangle$ is the size of the largest $X \in \mathbb{X}$ minus 1. Figure 1.1 presents two example graphs and one possible tree decomposition per graph.

Given undirected graph $G(V, E)$ and integer k, the TREEWIDTH problem is of deciding if the *treewidth* of G is at most k [48]. An equivalent constructive problem is finding a tree decomposition of width at most k of G. Another equivalent problem is finding a *triangulation* of G (a chordal graph containing G) with a *clique number* that is at most $k + 1$ (the clique number of a graph is the size of the largest clique in this graph).

An efficient solution to TREEWIDTH is key in many applications in artificial intelligence, databases and logical-circuit design. Exact inference in Bayesian networks using the junction tree algorithm [38, 32] or the variable elimination algorithm (e.g., [20]) requires us to first find a tree decomposition (equivalently, an elimination order) and then perform inference using that tree. The time complexity of the junction tree algorithm depends exponentially on the width of the tree, so it is important to try to find a close to optimal such tree.

Reasoning with structured CSPs (constraint-satisfaction problems), propositional SAT and FOL (first-order logic) problems also benefits from close-to-optimal tree decompositions [21, 4, 44]. Also, database appalications that use the world-wide web (WWW) are practical if we use a tree decomposition of low width [30]. Finally, the solution time of many graph-related NP-hard problems that are found in the literature

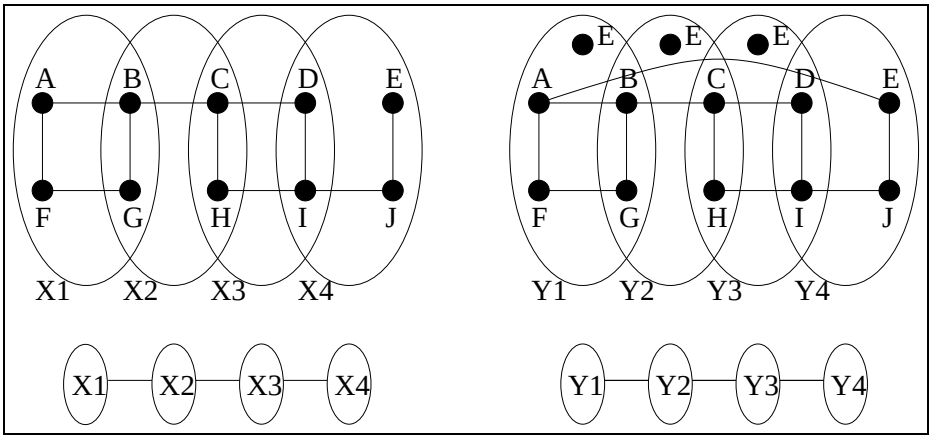

FIG. 1.1. *Two examples of tree decompositions. The given graphs are with nodes* $\{A, B, C, D, E, F, G, H, I, J\}$ *and the depicted edges. One possible tree decomposition is given for each graph. The top part depicts the sets* ($X1, .., X4$, *and* $Y1, ..., Y4$, *respectively), and the bottom part displays the tree on those sets. Their widths are* 3 *(left) and* 4 *(right). Notice that on the right-hand example* E *appears in all sets* $Y1, ..., Y4$.

is possible in polynomial time, if the graph has low treewidth and a triangulation of close to minimum treewidth is given (e.g., [6]).

This paper presents four approximation algorithms for finding triangulations of minimum treewidth. The most important of which from a theoretical perspective is the first polynomial-time algorithm that finds a tree decomposition within a factor-$O(\log OPT)$ from the optimal, where OPT is the treewidth of the given graph. This algorithm runs in time $O(n^4\, k \log k)$, for n being the number of nodes and $k = OPT$ being the treewidth of the given graph, G. The best previously known algorithm was due to [13], and produced a factor-$O(\log n)$ approximation.

Our second algorithm improves an algorithm of [49] and produces factor-4 approximations in time $O(2^{4.38k}n^2k)$. The third algorithm produces factor-$(4 + \frac{1}{2})$ triangulations in time $O(2^{3k}n^2k^{\frac{3}{2}})$. The last algorithm improves an algorithm of [7] and produces factor-$(3 + \frac{2}{3})$ approximations in time $O(2^{3.6982k}n^3k^3 \log^4 n)$. The time bounds achieved by the second and fourth algorithms are faster by factors of $O(2^{0.4k})$ and $O(2^k poly(n))$, respectively, than previously available algorithms for these approximation factors. Our third algorithm has the lowest known dependence on k amongst algorithms that produce constant-factor approximations.

We have implemented the factor-4 approximation algorithm, the factor-$(4 + \frac{1}{2})$ approximation algorithm and a reduced version of our $O(\log OPT)$-approximation algorithm. We used them to find tree decompositions of graphs used in a subset of the HPKB project [17], a subset of the CYC knowledge base [42], several CPCS Bayesian networks [46], and some SAT problems from the SATLIB benchmark set [31]. These graphs have between 100 and 60,000 nodes and between 400 and 1,000,000 edges. Our results compare favorably with the algorithms of [7, 52], and are also comparable to those produced by heuristic functions (these are known to perform well on some graphs, but are also known to produce solutions that are arbitrarily far from optimal for other graphs).

The results that we achieved here were used successfully in [43] for inference with large knowledge bases in First-Order Logic.

1.1. Overview of Our Approach. The approach that we take is similar in principle to much earlier work in the line of [48, 47, 12, 7]. The main idea there is that a recursive decomposition of the input graph can yield an approximation to the optimal tree decomposition. That approach builds on an observation of [48] that every graph of treewidth k has a balanced vertex cut of size at most $k+1$, for some notion of balance (see more formally in Lemma 2.5). Then, the recursive step uses an algorithm that finds a balanced vertex cut, also making sure that the previous separator (up the recursion) is split in some way as well.

The main observation in the current work is that the recursive step of such algorithms needs only to find a balance vertex cut of the previous separator (up the recursion), and can ignore the contribution of the rest of the graph to the balance. We then define more fully and apply a polynomial-time algorithm of [41] for finding a balanced vertex cut of a set of vertices W in a graph that is within $O(\log |W|)$ from the optimal. With a carefully defined recursive step we get an $O(\log k)$ approximation to the optimal tree decomposition. Figure 1.2 presents this approach.

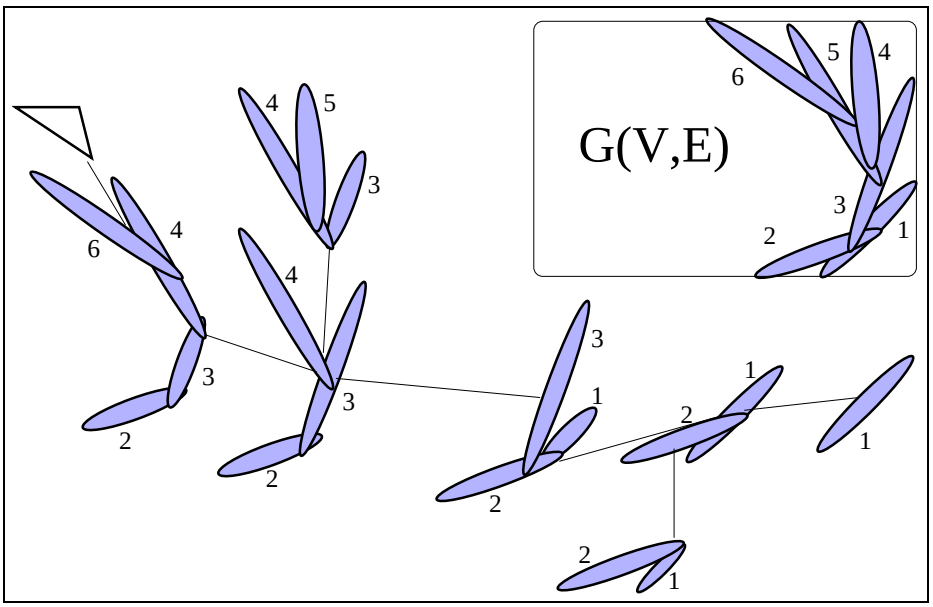

FIG. 1.2. *A recursive approach to finding close-to-optimal tree decompositions. We find a vertex cut in the graph $G(V,E)$ that is a balanced vertex cut of the previous vertex cut. Each oval in the graph on the upper right indicates a vertex cut. The numbers (1,...,6) indicate the order in which these vertex cuts were found. The tree decomposition is illustrated at the bottom-left. Only the first 6 steps of the approach are illustrated.*

We also apply the new observation together with maximum s-t-flow algorithms to give faster solutions for previously known constant-factor approximations, as described above. The main step there is setting up flow problems whose solution yields balanced vertex cuts as needed. The exponential in k remains because we connect together into cliques large fractions of the previous separator, W, and there is an exponential number in $|W|$ of possible subsets to connect into cliques. The gain that we get in speed (a smaller coefficient of k in the exponential) is due to using flow problems and our ability to ignore in the separation the rest of G.

1.2. Previous Work. An optimal solution for the TREEWIDTH problem is known to be NP-hard [5]. It is an open question whether a constant-factor approximation can be found in polynomial time. Nevertheless, several algorithms were found with either a constant approximation factor and an exponential time-dependency on k, the treewidth of the graph, or a $O(\log n)$ approximation factor and a polynomial execution time. We briefly recount some of the previous work done on finding tree decompositions of minimum width.

Prior to the current work, the best polynomial-time approximation algorithm for this problem was [35, 12]. Their algorithm yields a factor-$O(\log n)$ (specifically, $12 \cdot k \cdot \Delta \cdot \log(n+1)$ for Δ being a large unspecified constant) approximation, taking time $O(poly(n \cdot k))$, where $poly(n \cdot k)$ is the time to compute an approximate balanced vertex separator in the sense of [34] (Lemma 4.1 in [34] promises an algorithm, using the algorithm of [40] which in turn uses linear programming; the randomized algorithm of [39] can replace the linear programming part of that algorithm and runs in *expected* time $O(nm\delta \log^3 n)$, when δ is the maximum degree of any node in the graph).

For a constant k, several authors provided polynomial algorithms for the exact solution. Robertson and Seymour [48] provided the first algorithm using $O(n^{2k^2+4k+8})$ time (the complexity bound is due to [5]). Arnborg et al. [5] improved this algorithm to takes time $O(n^{k+2})$. In 1986 Robertson and Seymour [49] gave a non-constructive proof of the existence of a decision procedure that uses $O(n^2)$ time. That procedure computes a tree decomposition whose width is within a constant-factor from the treewidth. It then uses this tree decomposition to compute an optimal decomposition (of width that is the treewidth). Later, Reed [47] showed that the first part of their algorithm takes time $O(k^2 \cdot 3^{3k} \cdot n^2)$ and finds a decomposition that is of width $\leq 4k+3$.

Lagergren [36] developed an algorithm that performs the first part in $O(f(k)n \log^2 n)$ time and produces a treewidth of size $\leq 8k+7$, where $f(k)$ is

$$k \cdot 2^{k+3} \cdot (k+2))^k \cdot (k+1)!^2 \cdot k \cdot 2^k$$

which is more than $2^{k^2} \cdot (k+1)!^2$. Reed [47] proposed an algorithm that takes $O(k^2 3^{4k} n \log n)$ time to solve the first part and provides an approximation of factor 5. Both algorithms yield a valid answer that the width is larger than k, or output a tree decomposition that has treewidth bounded by a linear function in k.

Bodlaender and Kloks [13, 35] provide algorithms for the second part that work in linear time (again, with k fixed). If we have a tree decomposition of width l, and look for a tree decomposition of width k, then the time is

$$O(l^{l-1} \cdot ((2l+3)^{2l+3} \cdot (\frac{8}{3} \cdot 2^{2k+2})^{2l+3})^{2l-1} \cdot n)$$

[9, 11] (this is more than $O(2^{8k^3} n)$ time). Lagergren and Arnborg [37, 1] provided algorithms for the second part with similar super-exponential dependency on k. Bodlaender [9] provided a combined *linear-time* algorithm (for both parts) having a similar dependency on k, l, using the algorithm of [13].

Becker and Geiger [7] presented an approximation algorithm that finds a factor 3.66 approximation to the optimal decomposition in time $O(2^{4.66k} n \cdot poly(n))$, where $poly(n)$ is the running time of linear programming. Shoikhet and Geiger [52] presented an algorithm that finds an optimal triangulation in time $O(Rn^5 + R_k n^3 C_{max})$ where R is the number of all minimal separators of G, R_k is the number of minimal separators of G of size at most k, and C_{max} is the maximal number of maximal cliques in a *validity*

graph[1] of some fragment in $\mathcal{F}_G$ ($\mathcal{F}_G$ is the set of fragments of the graph obtained by a separator of G of size $\leq k$). Broersma et al. [15] presented an algorithm that finds the optimal triangulation in time $O(Rn^5 + KR^{K+1}(n+m)n \cdot \log n)$, where R is the number of all minimal separators of G and K is the *asteroidal number* of G (the algorithm does not require prior knowledge of the asteroidal number of G).

In special classes of graphs there are better results than those above. In particular, there are constant-factor approximations in AT-free graphs [14], planar graphs [51, 3], and single-crossing minor-free graphs [22].

1.3. Organization of This Paper. The paper is organized as follows. Section 2 defines the main notions involved in computing treewidth and recalls some theorems proved elsewhere. Section 3 presents our $O(\log OPT)$-approximation algorithm. Section 4 presents our algorithms that provide factor-4, factor-$(4 + \frac{1}{2})$ and factor-$(3 + \frac{2}{3})$ approximations. Section 5 draws further results for finding *path-width* and *cut-width*. Section 6 extends our results to weighted graphs. The paper concludes with experimental results in Section 7.

A good survey paper on TREEWIDTH is [10]. A good book on the subject is [35].

2. Treewidth. In this section we define treewidth formally, also recalling some of the main definitions and theorems relating to it.

DEFINITION 2.1 ([48]). *A* tree-decomposition *of a graph $G(V, E)$ is a tree $T = \langle \mathbb{X}, \mathbb{E} \rangle$ with every node $X \in \mathbb{X}$ a subset of V, such that the following three conditions are satisfied:* (1) $\bigcup_{X \in \mathbb{X}} X = V$. (2) *For all edges $(v, w) \in E$ there is $X \in \mathbb{X}$ such that both v, w are contained in X.* (3) *For each vertex $v \in V$, the set of nodes $\{X \mid x \in X, X \in \mathbb{X}\}$ forms a subtree of T (this is called the* running-intersection property*).*

The *width* of a *tree decomposition* $T = \langle \mathbb{X}, \mathbb{E} \rangle$ is $\max_{X \in \mathbb{X}}(|X| - 1)$. The *treewidth* of a graph G equals the minimum width over all tree-decompositions of G.

COROLLARY 2.2. *If $G(V, E)$ is a graph of treewidth $k + 1$, then $|E| \leq |V| \cdot k$.*

A close correspondence exists between tree decompositions and triangulations of graphs. In particular, the treewidth of G is the minimum $k \geq 0$ such that G is a subgraph of a triangulated graph with all cliques of size at most $k + 1$. A cycle in a graph is *chordless* if no proper subset of the vertices of the cycle forms a cycle.

DEFINITION 2.3. *A graph is* triangulated *(or* chordal*) if it contains no chordless cycle of length greater than three.*

A *triangulation* of a graph G is a graph H with the same set of vertices such that G is a subgraph of H and such that H is triangulated.

Any triangulation of a graph defines a tree-decomposition of a graph of the same treewidth. Similarly, every tree-decomposition of a graph defines a triangulation of it of the same treewidth. Triangulations are particularly interesting because there is a simple polynomial-time algorithm that takes a triangulated graph and finds one of its optimal tree decompositions (or equivalently, an elimination order for its vertices) iteratively: find a vertex in the triangulated graph which forms a clique with all its neighbors; create a set X for this vertex and its neighbors, and remove this vertex from the graph; then, add X to the tree that you created so far by attaching it with an edge to an already-present node such that the running intersection property still holds (one can always find such a node in the present tree (this is a simple proof that is left for the reader, or could be found in [35])).

[1] The notion of a *validity graph* is defined in their paper.

DEFINITION 2.4. *Let $G(V,E)$ be a graph, $W \subseteq V$ a subset of the vertices and $\alpha \in (0,1)$ a real number. An* α-vertex-separator *of W in G is a set of vertices $X \subseteq V$ such that every connected component of $G[V \setminus X]$ has at most $\alpha|W|$ vertices of W. A* two-way α-vertex-separator *is required in addition to have exactly two sets, S_1, S_2, separated by X such that $S_1 \cup S_2 \cup X = V$ and $|S_i| \leq \alpha|W|$, $i = 1,2$.*

LEMMA 2.5 ([48]). *Let $G(V,E)$ be a graph with n vertices and treewidth k. There exists a set X with $k+1$ vertices such that every connected component of $G[V \setminus X]$ has at most $\frac{1}{2}(n-k)$ vertices.*

The following corollary of Lemma 2.5 guarantees that there are always vertex separators with three separated subgraphs of proper sizes.

COROLLARY 2.6 ([7]). *Let $G(V,E)$ be a graph with $n \geq k+1$ vertices and treewidth k. For every $W \subseteq V$, $|W| > 1$, there is a vertex separator X and sets $A, B, C \subset V$ such that $A \cup B \cup C \cup X = V$, A, B, C are separated by X, $|X| \leq k+1$ and $|W \cap C| \leq |W \cap B| \leq |W \cap A| \leq \frac{1}{2}|W|$.*

3. $O(\log OPT)$-Approximation to Treewidth. The algorithms for finding treewidth that we present in this paper follow the recursive nature of previous works, e.g., [48, 49, 47, 12]. In this section, we present the first algorithm, which achieves an $O(\log OPT)$ factor from the optimal. The recursive step of this algorithm applies results from [41], and we describe those results first. Then, we follow with the description of the algorithm and its computational properties.

3.1. Balanced Node Cuts in $G(V,E)$ with two weight functions. First, we describe a procedure for finding balanced node cuts in $G(V,E)$ with two weight functions, π_1, π_2. The first function, π_1, defines the contribution of a node to the cost of the cut (if that node is in the vertex separator), and the second function, π_2, defines the contribution of a node to the weight of the cut (if that node is in one of the separated sets). The procedure is given in Figure 3.1.

PROCEDURE bal-node-cut(G, π_1, π_2, b, b')
$G = (V,E)$ an undirected graph, $\pi_1 \geq 0$ a cost function for nodes in V, $\pi_2 \geq 0$ a weight function on nodes in V, $b \leq 1/2$, $b' < b$ and $b' \leq 1/3$.

1. Let $G^*(V^*, E^*)$ be a directed graph with edge costs and node weights built from G as follows[a]:
   (a) Set $V^* = \{v', v'' | v \in V\}$.
   (b) Set $E^* = \{(v', v'') \mid v \in V\} \cup \{(v'', u') \mid (v,u) \in E\}$.
   (c) Set edge costs in E^*: $C(v', v'') = \pi_1(v)$, and $C(v'', u') = \infty$ for $u \neq v$.
   (d) Set node weights in V^*: $w(v') = w(v'') = \frac{1}{2}\pi_2(v)$.
2. Find a b'-balanced edge cut $(S, \bar{S})$ in G^* that is within $O(\log p)$ from the optimal b-balanced edge cut[b], for $p = |\{v \in V \mid \pi_2(v) > 0\}|$.
3. Return the node cut (A, C, B) for $C = \{v \in V \mid v' \in S,\ v'' \in \bar{S}$ or $v'' \in S,\ v' \in \bar{S}\}$, $A = \{v \in V \mid v' \in S$ or $v'' \in S\} \setminus C$, and $B = V \setminus (A \cup C)$.

[a] This step is a variant of a well-known translation of a node cut problem to an edge cut problem in a directed graph.

[b] This can be done using any algorithm for balanced edge cut in directed weighted graphs, e.g., [41].

FIG. 3.1. *An algorithm for finding a balanced node cut.*

This procedure was suggested in [41] without details on the conversion to a directed sparse edge-cut problem. That work presented (among others) a detailed al-

gorithm for directed b'-balanced edge cut of the following character.

THEOREM 3.1 ([41] §2.4 Theorem 17; §3.1; §3.4;). *Let $b \leq \frac{1}{2}$, and $b' < b$ with $b' \leq \frac{1}{3}$. There is a constructive algorithm[2] that (a) is given a directed graph G^* with edge costs and node weights, (b) returns a b'-balanced edge cut $(S, \bar{S})$ in a G^* that is within $O(\log p)$ from the optimal b-balanced edge cut, for p the number of vertices in G^* of non-zero weight, and (c) works in time $O(p^2n^2m \log p)$.*

The main bottleneck of the time taken in this theorem is the solution to a multi-commodity flow problem, for which one can use, e.g., [39] (that work presents algorithms for directed, weighted networks). The time quoted in Theorem 3.1 assumes using this algorithm as a subroutine.

COROLLARY 3.2. *Let $b \leq \frac{1}{2}$, and $b' < b$ with $b' \leq \frac{1}{3}$. Algorithm* bal-node-cut *provides a b'-balanced node cut in G that is within $O(\log p)$ from the optimal b-balanced node cut, for p the number of non-zero π_2-weights to nodes in G. It does so in time $O(p^2n^2m \log p)$.*

Some calculation and bookkeeping over the development in [41] shows that the constant factor in $O(\log p)$ in this theorem is bounded by $\beta = \frac{4\beta_1}{b-b'}$ for β_1 the constant factor given for any implementation of sparse cut in directed graphs (e.g., $\beta_1 = 115.2$ in [41]). For that we take their definition and analysis for a *3-way directed cut* ([41] §3.4), which is equivalent to the 2-way directed cut in our case (of reducing a node-cut problem to a directed edge-cut problem).

3.2. 3-way $\frac{2}{3}$-vertex-separator of $W \subset V$. We can now present the main subroutine that Section 3.3 below uses to find treewidth. It finds a $\frac{2}{3}$-vertex-separator of $W \subseteq V$ in $G(V, E)$ of size within factor-$O(\log |W|)$ from optimal $\frac{1}{2}$-vertex-separator in polynomial time. It does so using Procedure *bal-node-cut* from Figure 3.1 above.

Our procedure for finding 3-way $\frac{2}{3}$-vertex-separator of W in G calls subroutine *bal-node-cut* with G and the following weight and cost functions: Set $\pi_1(v) = 1$ for all $v \in V$, set $\pi_2(v) = 1$ for all $v \in W$, and set $\pi_2(v) = 0$ for all $v \in V \setminus W$. This ensures that the cut that is found by the algorithm ignores the weight of nodes in $V \setminus W$ (so they do not matter for the balance), but keeps count of all the nodes when computing the cost of the cut regardless of their membership in W or in $V \setminus W$. We call the overall procedure *2-3-vsep-lgk*(W,G).

COROLLARY 3.3. *Procedure* 2-3-vsep-lgk*(W,G) finds a $\frac{2}{3}$-balanced node cut of W in G of cost within $O(\log |W|)$ of the optimal.*

3.3. $O(\log OPT)$-Approximation Algorithm. Figure 3.2 presents our tree-decomposition algorithm as a triangulation one (i.e., creating cliques and a triangulated graph instead of a tree decomposition). It uses Procedure *2-3-vsep-lgk*(W,G) to find a $\frac{2}{3}$-vertex-separator X of W in V into three sets S_1, S_2, S_3 such that $|X|$ is at most $\beta \cdot \log |W|$ times the size of the optimal $\frac{1}{2}$-vertex-separator, for β the constant factor in Corollary 3.3. With the current approximation algorithm for balanced vertex cut we get $\beta = \frac{4\beta_1}{b-b'} = 2764.8$, with $\beta_1 = 115.2$ (see Section 3.1 for a summary of the analysis). It follows that $|X|$ is at most $\beta \cdot \log |W| \cdot (k+1)$ from Corollary 2.6, if k is the treewidth of G.

The following lemma implies that the algorithm *lgk-triang* outputs a triangulated graph with treewidth that approximates that of the given graph by a factor that depends inversely on the treewidth of the latter (i.e., the larger the treewidth of the graph, the better approximation we get).

[2] We bring an overview of such an algorithm in Appendix A.

PROCEDURE lgk-triang(G, W, k).
$G = (V, E)$ with $|V| = n$, $W \subseteq V$, k integer.

1. If $n \leq \beta \cdot k \cdot \log k$, then make a clique of G. Return.
2. If $|W| < 2$, then add vertices to W from G such that $|W| = 2$.
3. Find X, an approximate minimum 3-way $\frac{2}{3}$-vertex-separator of W in G, with S_1, S_2, S_3 the three parts separated by X (with $S_1, S_2 \neq \emptyset$, but possibly $S_3 = \emptyset$). If $|X| > \beta \cdot k \cdot \log |W|$, then output "*the treewidth exceeds* k" and exit.
4. For $i \leftarrow 1$ to 3 do
   (a) $W_i \leftarrow S_i \cap W$.
   (b) call lgk-triang($G[S_i \cup X]$, $W_i \cup X$, k).
5. Add edges between vertices of $W \cup X$, making a clique of $G[W \cup X]$.

FIG. 3.2. *A $O(\log OPT)$-approximation triangulation algorithm for treewidth.*

LEMMA 3.4. *Let $G(V, E)$ be a graph with n vertices, $k \geq \log n$ an integer*[3] *and $W \subseteq V$ such that $|W| \leq \gamma \cdot k \cdot \log k$, and $\gamma = c\beta$ for $c = 12 + 3\log_k \beta$ a constant*[4]*. Then,* lgk-triang*(G,W,k) either outputs correctly that the treewidth of G is more than $k - 1$ or it triangulates G such that the vertices of W form a clique and the clique number of the resulting graph is at most $\frac{4}{3}\gamma k \log k$.*

Proof. If the algorithm outputs that the treewidth is more than k, then it did not find a decomposition of W as needed. If the treewidth of G is $\leq k$, then Corollary 3.3 guarantees that subroutine *2-3-vsep-lgk* finds a $\frac{2}{3}$-vertex-separator of W in G that is within a factor of $\beta \cdot \log |W|$ of the optimal (recall, the optimal is at most k by Corollary 2.6). Thus, this separator will be found in Step 3, contradicting our assumption that the algorithm terminated this way. Thus, the treewidth must be $> k$.

Assume that the algorithm does not output that the treewidth exceeds k. Then, the algorithm eventually terminates because every recursive call to *lgk-triang* receives a graph that is strictly smaller than G ($S_1, S_2 \neq \emptyset$ (although possibly $S_3 = \emptyset$)).

Now we show that the algorithm returns a triangulated graph. We prove this by induction using the recursive structure of the algorithm. Clearly the claim is true if $n \leq \beta \cdot k \cdot \log k$ because step 1 makes a clique out of V in that case. Assume $n > \beta \cdot k \cdot \log k$. By induction, the recursive calls *lgk-triang*($G[S_i \cup X]$, $W_i \cup X$, k) return triangulations of $G[S_i \cup X]$, such that $W_i \cup X$ is a clique. Finally, the algorithm makes a clique of $W \cup X$. Thus, the graphs $G[S_i \cup W \cup X]$ are triangulated. Since the intersection of these triangulated graphs is a clique, the union is triangulated.

We show that the largest clique in this triangulation is of size at most $C \cdot k \cdot \log k$. First, we prove that always $|W| \leq \gamma k \log k$. Initially, $|W| \leq \gamma k \log k$ by our assumption in the statement of the lemma. As the algorithm is called recursively, we find a separator X such that $|X| \leq \beta k \log |W|$ and $|W_i| \leq \frac{2}{3}|W| \leq \frac{2}{3}\gamma k \log k$, by induction. Thus, $|W_i \cup X| \leq \frac{2}{3}\gamma k \log k + \beta k \log |W|$.

Now, notice that

$$|X| \leq \beta k \log |W| \leq \beta k \log(\gamma k \log k) = \beta k \log(c\beta k \log k) =$$
$$\beta k \log(ck^{(c-12)/3} k \log k) = \beta k \log(ck^{c/3-3} \log k) \leq \beta k \log k^{c/3} = \tfrac{1}{3} c\beta k \log k$$

[3] If $k < \log n$, we can use the other algorithms later in this paper with a polynomial time, constant-factor approximation guarantee.

[4] $c \leq 30$ when $n \geq 16$, and when $n < 16$ we can have a table lookup or use the other algorithms later in this paper.

It follows that $|W_i \cup X| \leq \frac{2}{3}\gamma k \log k + |X| \leq \gamma k \log k$. Thus, the in the next recursive call also $|W| \leq \gamma k \log k$.

Now, let M be a maximal clique. If M contains no vertex of $S_i \setminus W_i$, for $i = 1, 2, 3$, then M contains only vertices of $W \cup X$. Thus, in this case, $|M| \leq |W \cup X| \leq \gamma k \log k + \frac{1}{3}\gamma k \log k \leq \frac{4}{3}\gamma k \log k$.

On the other hand, if M contains a vertex of $S_i \setminus W_i$, then it does not contain any vertex of S_j, for $j \neq i$. This is because X vertex-separates S_1, S_2, S_3 (any two separated vertices cannot have an edge connecting them). Hence, M is a clique in the triangulation of $G[S_i \cup X]$. By induction we know that $|M| \leq \gamma k \log k$. This concludes the proof of the lemma. □

THEOREM 3.5. *Procedure lgk-triang(G, ∅, k) finds a triangulation of G of clique number* $\leq (\frac{4}{3}\gamma \log k) \cdot k$, *for* $\gamma = c\beta$, *and* $c = 12 + 3\log_k \beta$, *if the treewidth of G is at most* $k - 1$. *This procedure takes time* $O(n^4 \cdot k \log k)$.

Proof. Lemma 3.4 guarantees the correctness of the procedure. We prove the time bound below (it is important to notice that the normal recursive argument for time computation does not work here, as we end up with at least two graphs with sizes that sum up to more than the original size, and each may be of size $n - 1$ (e.g., when $k = n - 2$)).

We will bound the number of times that Step 3 is called in procedure *lgk-triang* by $O(|V|)$. Together with the time bound in Corollary 3.2 and Corollary 2.2 this provides the time bound promised in the theorem.

We distinguish three cases for Step 3 in procedure *lgk-triang*:

(1) For at least two of $i = 1, 2, 3$, $|S_i \cup X| > \beta k \log k$;
(2) Exactly one of $i = 1, 2, 3$ has $|S_i \cup X| > \beta k \log k$; and
(3) For all $i = 1, 2, 3$, $|S_i \cup X| \leq \beta k \log k$.

Let $\Phi(i)$ be the number of vertices that may participate in a future invocation of Step 3 after the i-th call to Step 3.

Each invocation of case *(1)* on a set of vertices, S, increases Φ by at most $2\beta k \log k$. To see that, notice that the three sets generated from case *(1)* are $S_1 \cup X$, $S_2 \cup X$ and $S_3 \cup X$ (the latter one is included only if $S_3 \neq \emptyset$). Their joint size is at most $|S_1 \cup X| + |S_2 \cup X| + |S_3 \cup X| = |S_1| + |S_2| + |S_3| + |X| + |X| + |X| = |S_1 \cup X \cup S_2 \cup S_3| + |X| + |X| = |S| + 2|X| \leq |S| + 2\beta k \log k$. This is larger from $|S|$ by at most $2\beta k \log k$.

Each invocation of case *(2)* on a set of vertices, S, decreases Φ by at least 1. This is seen as follows. First, notice that $|S_i| \geq 1$ for $i = 1, 2$. Now, assume $|S_1 \cup X| \leq \beta k \log k$ (otherwise, $|S_2 \cup X| \leq \beta k \log k$, and the treatment is identical). The vertices of S_1 will not participate in any future invocation of Step 3 because $|S_1 \cup X| \leq \beta k \log k$ is our stopping condition (step 2 of Algorithm *lgk-triang*). Thus, there are only the vertices of $S_2 \cup X$ left from those of S, and $|S_2 \cup X| = |S| - |S_1| - |S_3| \leq |S| - 1$.

Each invocation of case *(3)* on a set of vertices, S, decreases Φ by at least $\beta k \log k$. This is because $|S| > \beta k \log k$ or we would not have executed Step 3 for S. After executing Step 3 with this case, none of the vertices of S will participate in any future invocation of Step 3 (both partitions are of size $\leq \beta k \log k$).

Now we show that there are are no more than $max(0, \frac{|V|}{\beta k \log k} - 3)$ invocations of Step 3 in *lgk-triang* that are of case *(1)*, for $G(V, E)$ and $k > 0$. We show this by induction on $|V|$. For every graph G, let $invoke_1(G)$ denote the number of invocations of Step 3 for subsets of G that are of case *(1)*.

Assume that the claim was proved for V's of sizes smaller than n, and that $|V| = n$. If $|V| \leq \beta k \log k$, we are done (no invocations at all) If $|V| > \beta k \log k$, then

we will apply Step 3 once, generating three subgraphs, $G[S_1 \cup X], G[S_2 \cup X], G[S_3 \cup X]$ (the last one may be discarded if $S_3 = \emptyset$). If the invocation was of type *(1)*, then at least two of the subgraphs have more than $\beta k \log k$ vertices and for those $max(0, \frac{|S_i \cup X|}{\beta k \log k} - 3) = \frac{|S_i \cup X|}{\beta k \log k} - 3$. Assume for a moment that the last one holds for all i (we discuss the case of only two subgraphs with more than $\beta k \log k$ vertices later). Using the induction hypothesis,

$$
\begin{aligned}
&invoke_1(G) \leq \\
&\quad 1 + invoke_1(G[S_1 \cup X]) + invoke_1(G[S_2 \cup X]) + invoke_1(G[S_3 \cup X]) \leq \\
&\quad 1 + \tfrac{|S_1|+|X|}{\beta k \log k} - 3 + \tfrac{|S_2|+|X|}{\beta k \log k} - 3 + \tfrac{|S_3|+|X|}{\beta k \log k} - 3 \leq \\
&\quad \tfrac{|V|+2|X|}{\beta k \log k} - 8 \leq \tfrac{|V|+2\beta k \log k}{\beta k \log k} - 8 = \tfrac{|V|}{\beta k \log k} - 6 \leq \tfrac{|V|}{\beta k \log k} - 3
\end{aligned}
$$

which concludes the induction step for this case. If instead there are only two subgraphs with more than $\beta k \log k$ vertices, then assume without loss of generality that those subgraphs are $G[S_1 \cup X]$ and $G[S_2 \cup X]$. Using a similar argument as above, and noticing that $invoke_1(G[S_3 \cup X]) = 0$ (or $S_3 = \emptyset$, which is equivalent):

$$
\begin{aligned}
&invoke_1(G) \leq \\
&\quad 1 + invoke_1(G[S_1 \cup X]) + invoke_1(G[S_2 \cup X]) + invoke_1(G[S_3 \cup X]) \leq \\
&\quad 1 + \tfrac{|S_1|+|X|}{\beta k \log k} - 3 + \tfrac{|S_2|+|X|}{\beta k \log k} - 3 + 0 \leq \\
&\quad \tfrac{|V|+2|X|}{\beta k \log k} - 5 \leq \tfrac{|V|+\beta k \log k}{\beta k \log k} - 5 = \tfrac{|V|}{\beta k \log k} - 4 \leq \tfrac{|V|}{\beta k \log k} - 3
\end{aligned}
$$

which concludes the induction step.

If this invocation was not of type *(1)*, and using the induction hypothesis (assuming without loss of generality that $|S_1 \cup X| \geq \beta k \log k$),

$$
\begin{aligned}
&invoke_1(G) \leq \\
&\quad 1 + invoke_1(G[S_1 \cup X]) + invoke_1(G[S_2 \cup X]) + invoke_1(G[S_3 \cup X]) \leq \\
&\quad max(0, \tfrac{|S_1 \cup X|}{\beta k \log k} - 3) + 0 + 0 \leq max(0, \tfrac{|V|}{\beta k \log k} - 3)
\end{aligned}
$$

which concludes the induction step for this case as well.

Thus, there are no more than $\frac{|V|}{\beta k \log k}$ invocations of Step 3 that are of case *(1)* (and if $|V| \geq \beta k \log k$, then there are no more than $\frac{|V|}{\beta k \log k} - 3$). Each of those invocations adds at most $2\beta k \log k$ to $\Phi(i)$. This means that there are at most $\frac{|V|}{\beta k \log k} \cdot 2\beta k \log k$ invocations of Step 3 that are of cases *(2)* or *(3)*, since our potential function $\Phi(i)$ is never negative. Thus, there are no more than $2\frac{|V|}{2\beta k \log k} \cdot (2\beta k \log k)$ invocations of Step 3 altogether. This is in $\Theta(|V|)$. □

The theorem guarantees a triangulation that is within a factor of $\frac{4}{3}\gamma \log k$ of the optimal, with $\gamma = c\beta$, when c can be chosen according to k. To see how the constant factor guaranteed by Theorem 3.5 behaves with respect to k, we notice the following cases. For $k \geq 4$, we can take $c = 30$, and get $\gamma = 30 \cdot 2764.8$. For $k \geq 5$ we can take $c = 27$, for $k \geq 8$ we can take $c = 24$, and for $k \geq 16$ we can take $c = 21$. For $k \geq \beta$ we can take $c = 12$.

Thus, from a theoretical viewpoint, we can use an algorithm such as one of those presented in the rest of this paper (which take time that is exponential in k) to test if the treewidth is more than β, getting a constant factor approximation to the actual treewidth if it does. If this algorithm fails, then we can use *lgk-triang* to find the treewidth up to a factor of $\frac{4}{3} \cdot 12 \cdot \beta \log OPT$ from the actual treewidth, OPT.

This means that the combined algorithm gives an approximation factor of at most $16 \cdot \beta \cdot \log OPT$.

From a practical point of view, bounding k by a (possibly large) constant for algorithms that take time that is exponential in k is not of real use. However, as our analysis in Theorem 4.8 (see Section 4.3) shows, there is an algorithm that provides a constant factor approximation (in fact, a factor of $(4 + \frac{1}{2})$) that takes time roughly proportional to $2^{3k}n^2$, that allows us (both theoretically and practically (see Section 7)) to assume that $k \geq 16$ in those graphs that are given to *lgk-triang*. In this case, the constant in our approximation factor is $\frac{4}{3} \cdot 21 \cdot \beta$.

4. Constant-Factor Approximations in Time Exponential in OPT. In this section we present three algorithms for finding constant-factor approximations to treewidth. The first two algorithms use two-way separators recursively. They differ on their choice of actual separator: 2/3 versus 1/2. The last algorithm uses a three-way separator in a similar manner.

4.1. Minimum Vertex Separators. We briefly describe the notion of a vertex separator. Let $G = (V, E)$ be an undirected graph. A set S of vertices is called an (a, b)-*vertex-separator* if $\{a, b\} \subset V \setminus S$ and every path connecting a and b in G passes through at least one vertex contained in S. An (a, b)-vertex-separator of minimum cardinality is said to be a *minimum* (a, b)-*vertex-separator*. The weaker property of a vertex separator being *minimal* requires that no subset of the (a, b)-vertex-separator is an (a, b)-vertex-separator.

Algorithms for finding minimum vertex separators typically reduce the problem to a maximum flow problem in a directed graph. The algorithm of Even and Tarjan reported in [25] for finding minimum vertex separators uses Dinitz's algorithm [23] with time complexity $O(|V|^{\frac{1}{2}}|E|)$.

Another possibility is to use the Ford-Fulkerson flow algorithm [28] (alternatively, see [18]), for computing maximum flow. For an original graph of treewidth $< k$ this involves finding at most k augmenting paths of capacity 1. Thus, the combined algorithm using the Ford-Fulkerson maximum flow algorithm finds a minimum (a, b)-vertex-separator in time $O(k(|V| + |E|))$.

Finally, to compute the vertex connectivity of a graph and a minimum separator, without being given a pair (a, b), we check the connectivity of any c vertices (c being the connectivity of the graph) to all other vertices. When Dinitz's algorithm is used as above, this procedure takes time $O(c \cdot |V|^{\frac{3}{2}} \cdot |E|)$, where $c \geq 1$ is the connectivity of G. For the cases of $c = 0, 1$ there are well known linear time algorithms. Even [24] also showed a way to test for k connectivity of a graph using only $n + k^2$ pairs of vertices.

4.2. 4-Approximation Algorithm. Procedure *2way-2/3-triang*, displayed in Figure 4.1, finds factor-4 approximations. For a graph G and a parameter k, running 2way-2/3-triang(G, $\emptyset$, k), either returns a valid answer that the treewidth of G is of size $> k - 1$ or it returns a triangulation of G of clique number at most $4k + 1$.

This algorithm is very similar to that of [49], as presented in [47]. The main difference is the more efficient algorithm that we use for exact vertex separation, which we provide below. The addition of elements to W' in step 2 ensures completeness of our separator (see Lemma 4.2's proof).

It is important to notice that there is no benefit in finding a $\frac{2}{3}$-vertex-separator for V instead. The approximation factor will still be 4, and the time saved by dividing

PROCEDURE 2way-2/3-triang(G, W, k)
$G = (V, E)$ with $|V| = n$, $W \subseteq V$, k integer.

1. If $n \leq 4k$, then make a clique of G. Return.
2. Let $W' \leftarrow W$. Add to W' vertices from V such that $|W'| = 3k + 2$.
3. Find X, a minimum $\frac{2}{3}$-vertex-separator of W' in G, with S_1, S_2 two nonempty parts separated by X ($S_1 \cup S_2 \cup X = V$) and $|X| \leq k$. If there is no such separator, then output "*the treewidth exceeds* $k - 1$" and exit.
4. For $i \leftarrow 1$ to 2 do
   (a) $W_i \leftarrow S_i \cap W$.
   (b) call 2way-2/3-triang($G[S_i \cup X]$, $W_i \cup X$, k).
5. Add edges between vertices of $W \cup X$, making a clique of $G[W \cup X]$.

FIG. 4.1. *A 4-approximation triangulation algorithm.*

W into proportional subsets is negligible compared to the time we would spend in dividing V.

LEMMA 4.1. *If $G(V, E)$ is a graph, k an integer and $W \subseteq V$ such that $|W| \leq 3k + 2$, then* 2way-2/3-triang*(G,W,k) either outputs correctly that the treewidth of G is more than k or it triangulates G such that the vertices of W form a clique and the clique number of the result is at most $4k + 1$.*

The proof is identical to that presented in [48, 47].

Figure 4.2 presents the algorithm we will use for finding a $\frac{2}{3}$-vertex-separator of W' in G (step 3 in procedure *2way-2/3-triang*). It checks choices of sets of vertices to be separated until a solution is found or the choices are exhausted. The intuition behind making a clique from each selected set, W^i, is that doing so prevents any element from that clique from becoming an element in the separated subset of the other side. Given an arbitrary vertex separator of v_{W^1}, v_{W^2}, any vertex in the clique of W^1 must be either in the separator itself or in S_1.

PROCEDURE $\frac{2}{3}$-vtx-sep(W, G, k)
$G = (V, E)$ with $|V| = n$, $W \subseteq V$, k integer.

1. Nondeterministically take a set W^1 of $\lceil \frac{|W|}{2} \rceil$ vertices from W and a set W^2 of $\lceil \frac{|W|}{3} \rceil$ vertices from $W \setminus W^1$.
2. Let $G' \leftarrow G$. Add edges to G' so that W^1 is a clique and W^2 is a clique. Create new vertices v_{W^1}, v_{W^2} in G' and connect them to all the vertices of W^1, W^2, respectively.
3. Find a minimum (v_{W^1}, v_{W^2})-vertex-separator, X. If $|X| \leq k$, return $|X|$ and two separated subsets S_1, S_2, discarding v_{W^1}, v_{W^2}. Otherwise, return "*failure*".

FIG. 4.2. *Find a $\frac{2}{3}$-vertex-separator of W in G.*

LEMMA 4.2. *Let $G(V, E)$ be a graph, $k \geq 0$ an integer, and $W \subseteq V$ of size $3k+2$. Algorithm $\frac{2}{3}$-vtx-sep(W, G, k) finds a $\frac{2}{3}$-separator of W in G of size $\leq k$, if it exists, returning* failure *otherwise. It does so in time $O(\frac{2^{4.38k}}{k} f(|V|, |E| + k^2, k))$, given a* min-$(a, b)$-vertex-separator *algorithm taking time $f(n, m, k)$.*

Proof. We prove the correctness of the algorithm first. Assume that the algorithm finds a separator X of S_1, S_2 in G'. X is also a separator of S_1, S_2 in G, by the way

we constructed G' from G. Also, X separates $W^1 \setminus X$ and $W^2 \setminus X$ in G' because $W^1 \cup \{v_{W^1}\}$ and $W^2 \cup \{v_{W^2}\}$ are cliques in G' and X separates v_{W^1}, v_{W^2} (if X does not separate $W^1 \setminus X$ and $W^2 \setminus X$ in G', then there is a path between v_{W^1}, v_{W^2} that does not go through X). Finally, X is a $\frac{2}{3}$-vertex-separator of W because $|W^1|, |W^2| \geq \frac{|W|}{3}$, $W^1 \setminus X \subset S_1$ and $W^2 \setminus X \subseteq S_2$, so $|S_i \cap W| \leq \frac{2}{3}|W|$, for $i = 1, 2$. Notice that S_1, S_2 are never empty because $|X| \leq k$ and $|S_i| \geq |W^i| - |X| \geq 1$ for $i = 1, 2$ ($|W^i| \geq \lceil \frac{|W|}{3} \rceil = k + 1$ because $|W| = 3k + 2$).

For the reverse direction, assume that the treewidth of G is $k - 1$ and we show that the algorithm will find a suitable separator. Assume first that there are two sets of vertices S_1, S_2 separated by X in G such that $S_1 \cup S_2 \cup X = V$ and $|S_i \cap W| \leq \frac{|W|}{2}$, for $i = 1, 2$. Let $W^i_{\text{n\_sep}} = W \cap S_i$, for $i = 1, 2$. Let $W^i_{\text{sep}} \subseteq W \cap X$ such that $W^1_{\text{sep}} \cup W^2_{\text{sep}} = W \cap X$ and $|W^i_{\text{sep}} \cup W^i_{\text{n\_sep}}| = \frac{|W|}{2}$. Let $W^i = W^i_{\text{sep}} \cup W^i_{\text{n\_sep}}$, for $i = 1, 2$. Then, X separates $W^1 \setminus X, W^2 \setminus X$, as $W^i \setminus X = W^i_{\text{n\_sep}}$, for $i = 1, 2$. Thus, running steps 2,3 in our algorithm using this selection of W^1, W^2 will find a separator of size $\leq |X| \leq k$. By the previous paragraph, this separator is a $\frac{2}{3}$-vertex-separator of W in G.

Now we show that if there are no such sets S_1, S_2, X, then our algorithm still finds a suitable separator. By Corollary 2.6, there are three sets, A, B, C, of vertices separated by X in G such that $|X| \leq k$ and $|W \cap C| \leq |W \cap B| \leq |W \cap A| \leq \frac{1}{2}|W|$. Let $S_1 = A$, $S_2 = B \cup C$. If $|S_2 \cap W| \leq \frac{|W|}{2}$, then the first selection case would cover this W (the previous paragraph). Thus, $|S_2 \cap W| > \frac{|W|}{2}$. Take $W^1 \subset (S_2 \cap W)$ of size $\frac{|W|}{2}$ and $W^2 \subset ((S_1 \cup X) \cap W) \setminus W^1$ of size $\frac{|W|}{3}$. The selection of W^2 is possible because $|(S_1 \cup X) \cap W| = |S_1 \cap W| + |X \cap W| \geq \frac{1}{3}|W \setminus X| + |X \cap W| = \frac{1}{3}|W|$. For this selection of W^1, W^2 our algorithm will find a separator of size $\leq |X| \leq k$ because X is already a separator of $W^1, W^2 \setminus X$. By the first paragraph in this proof, this separator is a $\frac{2}{3}$-vertex-separator of W in G.

Finally, each choice of W^1 takes $O(f(|V|, |E|+k^2, k))$ time to check, for $f(n, m, k)$ the time taken by a *min-(a, b)-vertex-separator* algorithm over a graph with n vertices, m edges and treewidth $k - 1$. There are $\binom{3k+2}{1.5k+1}$ ways to choose $1.5k + 1$ elements (W^1) from a set of $3k + 2$ elements (W). Also, there are $\binom{1.5k+1}{k+1}$ ways to choose $k + 1$ elements (W^2) from a set of $1.5k + 1$ elements ($W \setminus W^1$). Since $\binom{3k+2}{1.5k+1} = O(\frac{2^{3k}}{\sqrt{k}})$ and $\binom{1.5k+1}{k+1} = O(\frac{2^{1.3776k}}{\sqrt{k}})$ (using Stirling's approximation), we get the time bound of $O(\frac{2^{4.3776k}}{k} f(|V|, |E| + k^2, k))$. □

PROPOSITION 4.3 (cf [47]). *If the treewidth of $G(V, E)$ is $k-1$, then $|E| \leq |V|k$.*

THEOREM 4.4. *Procedure 2way-2/3-triang(G, ∅, k) finds a triangulation of G of clique number $\leq 4k + 1$, if the treewidth of G is at most $k - 1$, in time $O(2^{4.38k}|V|^{\frac{5}{2}})$ or $O(2^{4.38k}|V|^2 k)$ if we use the minimum (a, b)-vertex-separator algorithm of [25] or [28], respectively.*

Proof. Lemmas 4.1 and 4.2 prove the correctness. For the time bound, an analysis that is very similar to that provided in the proof of Theorem 3.5 shows that there are at most $O(|V|)$ calls to *2way-2/3-triang*. Since each recursive step runs $\frac{2}{3}$*-vtx-sep* once and makes a clique of size $\leq 4k + 2$, we get that the combined procedure using [25]'s algorithm for min-(a, b)-vertex-separator (time $O(|V|^{\frac{1}{2}}|E|)$) takes time $O(\frac{2^{4.38k}}{k}|V|^{\frac{1}{2}}(|E| + k^2)|V|)$. Using Proposition 4.3 we get the bound $O(\frac{2^{4.38k}}{k}|V|^{\frac{3}{2}}(|V|k + k^2)) = O(2^{4.38k}|V|^{\frac{5}{2}})$. Similarly, using the algorithm given by [28] for finding a minimum (a, b)-vertex-separator in time $O(k(n + m))$ we get time

$O(2^{4.38k}|V|^2k)$. □

4.3. $(4+\frac{1}{2})$-Approximation Algorithm. We can avoid many choices examined in procedure $\frac{2}{3}$-vtx-sep if we allow the resulting separator to be slightly larger. Procedure 2way-half-vtx-sep, presented in Figure 4.4 does that, returning a close-to-minimum two-way $\frac{1}{2}$-vertex separator. The combined procedure, called *2way-half-triang*, is identical to procedure 2way-2/3-triang besides replacing step 3. It is presented in Figure 4.3.

PROCEDURE 2way-half-triang(G, W, k)
$G=(V,E)$ with $|V|=n$, $W\subseteq V$, k integer.

1. If $n\leq 4k$, then make a clique of G. Return.
2. Let $W'\leftarrow W$. Add to W' vertices from V such that $|W'|=3k+2$.
3. Find X, a two-way $\frac{1}{2}$-vertex-separator of W' in G, with S_1,S_2 the two nonempty parts separated by X ($S_1\cup S_2\cup X=V$) and $|X|\leq\frac{3}{2}k$. If there is no such separator, then output "*the treewidth exceeds* $k-1$" and exit.
4. For $i\leftarrow 1$ to 2 do
   (a) $W_i\leftarrow S_i\cap W$.
   (b) call 2way-half-triang($G[S_i\cup X]$, $W_i\cup X$, k).
5. Add edges between vertices of $W\cup X$, making a clique of $G[W\cup X]$.

FIG. 4.3. *A* $(4+\frac{1}{2})$*-approximation triangulation algorithm.*

LEMMA 4.5. *If* $G(V,E)$ *is a graph with treewidth* $<k$ *and* $W\subseteq V$*, then there is a two-way* $\frac{1}{2}$*-vertex-separator of* W *in* G *with size at most* $k+\frac{1}{6}|W|$

Proof. By Corollary 2.6 there are $A,B,C\subset V$ separated by X such that $A\cup B\cup C\cup X=V$, $|X|\leq k$ and $|W\cap C|\leq|W\cap B|\leq|W\cap A|\leq\frac{1}{2}|W|$. If $|(B\cup C)\cap W|\leq\frac{1}{2}|W|$, then $A,(B\cup C)$ and X satisfy our desired conditions.

Thus, assume that $|(B\cup C)\cap W|>\frac{1}{2}|W|$. Take $X_C\subset W\cap C$ of size $|(B\cup C)\cap W|-\frac{1}{2}|W|$. Then $|X_C|=|(B\cup C)\cap W|-\frac{1}{2}|W|\leq\frac{2}{3}|W|-\frac{1}{2}|W|=\frac{1}{6}|W|$. Let $X'=X\cup X_C$, $S_1=A$ and $S_2=(B\cup C)\setminus X_C$. This X',S_1,S_2 satisfy the desired conditions because $|S_2\cap W|\leq\frac{1}{2}|W|$, $|S_1|\leq\frac{1}{2}|W|$, $|X'|\leq|X|+|X_C|\leq k+\frac{1}{6}|W|$ and X' separates S_1,S_2 (because X separates S_1,S_2). □

LEMMA 4.6. *If* $G(V,E)$ *is a graph with* n *vertices,* k *an integer and* $W\subseteq V$ *such that* $|W|\leq 3k+2$*, then* 2way-half-triang*(G,W,k) either outputs correctly that the treewidth of* G *is more than* $k-1$ *or it triangulates* G *such that the vertices of* W *form a clique and the clique number of the resulting graph is at most* $(4+\frac{1}{2})k+2$.

Proof. If the algorithm outputs that the treewidth is more than $k-1$, then it did not find a decomposition of W as needed. If the treewidth is at most $k-1$, then Lemma 4.5 guarantees the existence of a two-way $\frac{1}{2}$-vertex-separator of W in G with size at most $k+\frac{1}{6}|W|$. Thus, this separator is of size at most $k+\frac{1}{6}|W|\leq k+\frac{1}{6}(3k+2)=\frac{3}{2}k+\frac{1}{3}$ (and because the size cannot be fractional, it is at most $\frac{3}{2}k$). If we did not find such a separator, then the treewidth is indeed at most $k-1$.

The same argument used for the proof of Lemma 4.1 shows that the algorithm always terminates and, if it is successful, then it returns a graph that is triangulated.

We show that the clique number of this triangulation is at most $(4+\frac{1}{2})k+2$. First, notice that always $|W|\leq 3k+2$. Initially, $|W|\leq 3k+2$ by our assumption in the statement of the lemma. As the algorithm is called recursively, $|X|\leq\frac{3}{2}k$ and $|W_i|\leq\frac{1}{2}|W'|=\frac{3}{2}k+1$. Thus, $|W_i\cup X|\leq\frac{3}{2}k+1+\frac{3}{2}k=3k+1$, which concludes the induction step (W in the recursive call to the algorithm is $W_i\cup X$).

Now, let M be a maximal clique. If M contains no vertex of $S_i \setminus W_i$, for $i = 1, 2$, then M contains only vertices of $W \cup X$. Thus, $|M| \leq 3k+2+\frac{3}{2}k = (4+\frac{1}{2})k+2$. On the other hand, if M contains a vertex of $S_i \setminus W_i$, then it does not contain any vertex of S_j, for $j \neq i$. This is because X vertex-separates S_1, S_2 (any two separated vertices cannot have an edge connecting them). Hence, M is a clique in the triangulation of $G[S_i \cup X]$. By induction we know that $|M| \leq (4+\frac{1}{2})k+2$. This proves the lemma. □

Procedure 2way-half-vtx-sep is very similar to procedure $\frac{2}{3}$-vtx-sep with one main difference. While $\frac{2}{3}$-vtx-sep selects two sets of sizes $\frac{1}{2}|W|$ and $\frac{1}{3}|W|$, procedure 2way-half-vtx-sep selects two sets of size $\frac{1}{2}|W|$. This precludes finding two-way separators in which one of the sets is of size $\frac{2}{3}|W|$ (as we did before).

PROCEDURE 2way-half-vtx-sep(W, G, k)
$G = (V, E)$ with $|V| = n$, $W \subseteq V$, k integer.

1. Nondeterministically choose a set W^1 of $\frac{|W|}{2}$ vertices from W. Let W^2 be $W \setminus W^1$.
2. Let $G' \leftarrow G$. Add edges to G' so that W^1 is a clique and W^2 is a clique. Create new vertices v_{W^1}, v_{W^2} in G' and connect them to all the vertices of W^1, W^2, respectively.
3. Find a minimum (v_{W^1}, v_{W^2})-vertex-separator, X. If $|X| \leq \frac{3}{2}k$, return $|X|$ and two separated subsets S_1, S_2, discarding v_{W^1}, v_{W^2}. Otherwise, return "*failure*".

FIG. 4.4. *Find a two-way $\frac{1}{2}$-vertex-separator of W in G.*

LEMMA 4.7. *Let $G(V, E)$ be a graph, $k \geq 0$ an integer, and $W \subseteq V$ of size $3k+2$. Algorithm $\frac{1}{2}$-vtx-sep(W, G, k) finds a two-way $\frac{1}{2}$-separator of W in G of size $\leq \frac{3}{2}k$, if it exists, returning* failure *otherwise. It does so in time $O(\frac{2^{3k}}{\sqrt{k}} f(|V|, |E| + k^2, k))$, given a* min-$(a, b)$-vertex-separator *algorithm taking time $f(n, m, k)$.*

Proof. We prove the correctness of the algorithm. First, assume that the algorithm finds a separator X of S_1, S_2 in G'. X is also a separator of S_1, S_2 in G, by the way we constructed G' from G. Also, X separates $W^1 \setminus X$ and $W^2 \setminus X$ in G' because $W^1 \cup \{v_{W^1}\}$ and $W^2 \cup \{v_{W^2}\}$ are cliques in G' and X separates v_{W^1}, v_{W^2} (if X does not separate $W^1 \setminus X$ and $W^2 \setminus X$ in G', then there is a path between v_{W^1}, v_{W^2} that does not go through X). Finally, X is a $\frac{1}{2}$-vertex-separator of W because $|W^1| = |W^2| = \frac{|W|}{2}$, $W^1 \setminus X \subset S_1$ and $W^2 \setminus X \subseteq S_2$. S_1, S_2 are non-empty because $|S_1| \geq |W| - |X| - |W^2| \geq 3k + 2 - \frac{3}{2}k - (\frac{1}{2}k + 1) = 1$ (similarly for S_2).

Now, assume that there is a two-way $\frac{1}{2}$-vertex-separator X of W in G with $|X| \leq \frac{3}{2}k$. Let S_1, S_2 be two separated sets of vertices in G such that $S_1 \cup S_2 \cup X = V$ and $|S_i \cap W| \leq \frac{|W|}{2}$, for $i = 1, 2$. Let $W^i_{\text{n\_sep}} = W \cap S_i$, for $i = 1, 2$. Let $W^i_{\text{sep}} \subseteq W \cap X$ such that $W^1_{\text{sep}} \cup W^2_{\text{sep}} = W \cap X$ and $|W^i_{\text{sep}} \cup W^i_{\text{n\_sep}}| = \frac{|W|}{2}$. Let $W^i = W^i_{\text{sep}} \cup W^i_{\text{n\_sep}}$, for $i = 1, 2$. Then, X separates $W^1 \setminus X, W^2 \setminus X$, as $W^i \setminus X = W^i_{\text{n\_sep}}$, for $i = 1, 2$. Thus, running steps 2,3 in our algorithm using this selection of W^1, W^2 will find a separator of size $\leq |X| \leq \frac{3}{2}k$. By the previous paragraph, this separator is a $\frac{1}{2}$-vertex-separator of W in G.

Finally, each choice of W^1 takes $O(f(|V|, |E|+k^2, k))$ time to check, for $f(n, m, k)$ the time taken by a *min-(a, b)-vertex-separator* algorithm over a graph with n vertices, m edges and treewidth $k-1$. There are $\binom{3k+2}{\frac{3}{2}k+1}$ ways to choose $\frac{3}{2}k+1$ elements (W^1) from a set of $3k+2$ elements (W). Since $\binom{2n}{n} = \frac{2^{2n}}{\sqrt{\pi n}}(1 + O(\frac{1}{n}))$, we get the time

bound of $O(\frac{2^{3k}}{\sqrt{k}} f(|V|, |E| + k^2, k))$. □

THEOREM 4.8. *Procedure 2way-half-triang(G, ∅, k) finds a triangulation of G of clique number* $\leq 3k+2$*, if the treewidth of G is at most* $k-1$*, in time* $O(2^{3k} n^{\frac{5}{2}} k^{\frac{1}{2}})$ *or* $O(2^{3k} n^2 k^{\frac{3}{2}})$ *if we use the minimum* (a, b)*-vertex-separator algorithm of [25] or [28], respectively.*

The proof of this theorem is similar to that of Theorem 4.4.

4.4. $(3 + \frac{2}{3})$-Approximation Algorithm. The last two algorithms recursively divide the set of vertices into two sets. Doing so we give up some of the separators guaranteed by Lemma 2.5. In this section we present a different angle on the tradeoff between the size of the separator, the size of each of the separated sides and the computational complexity of finding the separator. We find approximate *three-way separators*, and use them in a similar way to the one used above.

A generalization of the minimum (a, b)-vertex-cut problem is the problem of finding minimum multiway-vertex-cut. Given an undirected graph, $G(V, E)$, and a set of nodes, $v_1, ..., v_l \in V$, a minimum multiway cut is a minimum-cardinality set of nodes $S \in V$ such that $v_1, ..., v_l$ are in different connected components in $V \setminus S$. The weighted version requires a minimum-weight set of nodes.

Unlike the minimum (a, b)-vertex-cut problem, the problem of finding a minimum multiway-vertex-cut is NP-hard and MAXSNP-hard for $l \geq 3$ [19, 29] (i.e., there is $\epsilon > 0$ such that approximating the problem within a factor of $(1+\epsilon)$ is NP-hard). For the (a, b)-vertex-cut problem the maximum flow is equal to the minimum capacity cut in both directed and undirected graphs. This is not the case for multiway-vertex-cut. Nevertheless, [29] showed that by solving a maximum multicommodity flow problem, one can find an l-way vertex cut (in an undirected graph) that is of size within a factor $(2 - \frac{2}{l})$ to the optimal (*multicommodity flow* is a generalization of maximum-flow for multiple sources, sinks and commodities sent between them [41]).

This algorithm was used subsequently by [7] to offer an algorithm for minimum-treewidth triangulation. This algorithm takes time $O(2^{4.66k} n\, poly(n))$, for $poly(n)$ the time required to solve a linear program of size n.

Figure 4.5 recalls the main loop of the algorithm of [7]. The algorithm differs from that of [49] in using a 3-way separator instead of a 2-way separator. The separator, X of W, is required to satisfy $|(S_i \cap W) \cup X| \leq (1+\alpha)k$, for all three sets S_i, $i = 1, 2, 3$, for a given $\alpha \geq 1$. Let us call such a separator a α-sum-separator.

PROCEDURE 3way-triang(G, W, k)
$G = (V, E)$ with $|V| = n$, $W \subseteq V$, k integer.

1. If $n \leq (2\alpha + 1)k$, then make a clique of G. Return.
2. Let $W' \leftarrow W$. Add to W' vertices from V such that $|W'| = (1+\alpha)k + 1$.
3. Find X, a minimum α-sum-separator of W' in G, with S_1, S_2, S_3 three parts separated by X (at least two are nonempty) and $S_1 \cup S_2 \cup S_3 \cup X = V$. If there is none, then output "*the treewidth exceeds* $k-1$" and exit.
4. For $i \leftarrow 1$ to 3 do
   (a) $W_i \leftarrow S_i \cap W$.
   (b) call 3way-triang($G[S_i \cup X]$, $W_i \cup X$, k).
5. Add edges between vertices of $W \cup X$, making a clique of $G[W \cup X]$.

FIG. 4.5. *A* $(3 + \frac{2}{3})$*-approximation triangulation algorithm.*

Figure 4.6 presents a new procedure for producing an α-sum-separator. It calls

a procedure for 3-way vertex separation $3^{|W|}$ times instead of $4^{|W|}$ times as in the algorithm of [7].

PROCEDURE α-sum-sep(W, G, k)
$G = (V, E)$ with $|V| = n$, $W \subseteq V$, k integer.

1. Nondeterministically divide $|W|$ into three sets, W^1, W^2, W^3, such that $\frac{|W|}{2} \geq |W^1| \geq |W^2| \geq |W^3|$.
2. If $|W^1| > k$, then set $W^2 \leftarrow W^2 \cup W^3$ and return the result of steps 2–3 of algorithm $\frac{2}{3}$-vtx-sep (Figure 4.2).
3. Let $G' \leftarrow G$. Add edges to G' so that each of W^1, W^2, W^3 is a clique. Create new vertices, $v_{W^1}, v_{W^2}, v_{W^3}$ in G' and connect them to all the vertices of W^1, W^2, W^3, respectively.
4. Find an α-approximation to a minimum $(v_{W^1}, v_{W^2}, v_{W^3})$-vertex-separator, X. If $|X| \leq \alpha k$, return X and the three separated sets, S_1, S_2, S_3, discarding $v_{W^1}, v_{W^2}, v_{W^3}$. Otherwise, return "*failure*".

FIG. 4.6. *Find an α-sum-separator in G of size at most k.*

LEMMA 4.9. *Let $G(V, E)$ be a graph, $k \geq 0$ an integer, and $W \subseteq V$ of size $(1+\alpha)k+1$. Algorithm α-sum-sep(W, G, k) finds a α-sum-separator of W in G, if it exists, returning* failure *otherwise. It does so in time $O(2^{3.6982k} f(|V|, |E| + k^2, k))$, for $\alpha = \frac{4}{3}$, with $f(n, m, k)$ the time taken by an algorithm for α approximation to* min-(a, b, c)-vertex-separator.

Proof. We prove the correctness of the algorithm first. Assume that the algorithm finds a α-sum-separator X of S_1, S_2, S_3 in G'. X is also a separator of S_1, S_2 in G, by the way we constructed G' from G. Also, X separates $W^i \setminus X$ and $W^j \setminus X$, $i \neq j \leq 3$, in G' because $W^i \cup \{v_{W^i}\}$ and $W^j \cup \{v_{W^j}\}$ are cliques in G' and X separates v_{W^i}, v_{W^j}.

To see that X is an α-sum-separator of W we examine two cases. In the first, $|W^1| \leq k$. Thus, $|X| \leq \alpha k$ (otherwise we return "*failure*"). $|(S_i \cap W) \cup X| \leq |W^i| + |X| \leq (1+\alpha)k$, for $i = 1, 2, 3$, because $S_i \cap W \subseteq W^i$. Thus, this is an α-sum-decomposition. In the second case, $|W^1| > k$. Thus, $|W^2 \cup W^3| < \alpha k$ because $|W| = (1+\alpha)k$. Also, $|X| \leq k$ because it was returned by step 3 of algorithm $\frac{2}{3}$-vtx-sep (Figure 4.2). Thus, $|(S_i \cap W) \cup X| \leq |W^i| + |X| \leq (1+\alpha)k$. Notice that $|S_i \cap W| \geq 1$, for at least two of $i = 1, 2, 3$ (i.e., X does not contain at least two of the W^i's). In the first case this is because $|W^2 \cup W^3| = |W| - |W^1| \geq \alpha k + 1 > |X|$ (we set $|W| = (1+\alpha)k + 1$). In the second case this is because $|X| \leq k$, $|W^1| > k$ and $|W^2| = |W| - |W^1| \geq |W| - \frac{|W|}{2} > k$.

For the reverse direction, assume that the treewidth of G is $k-1$. We show that the algorithm finds a suitable separator. Let S_1, S_2, S_3 be three sets as guaranteed by Corollary 2.6, separated by X in G such that $S_1 \cup S_2 \cup S_3 \cup X = V$, $|S_3 \cap W| \leq |S_2 \cap W| \leq |S_1 \cap W| \leq \frac{|W|}{2}$ and $|X| \leq k$.

If $|W \cap S_1| \leq k$, then let $W^i_{\text{n\_sep}} = W \cap S_i$, for $i = 1, 2, 3$. Let $W^i_{\text{sep}} \subseteq W \cap X$ such that $W^1_{\text{sep}} \cup W^2_{\text{sep}} \cup W^3_{\text{sep}} = W \cap X$ and $|W^i_{\text{sep}} \cup W^i_{\text{n\_sep}}| \leq \frac{|W|}{2}$. Let $W^i = W^i_{\text{sep}} \cup W^i_{\text{n\_sep}}$, for $i = 1, 2, 3$. Then, X separates $W^1 \setminus X, W^2 \setminus X, W^3 \setminus X$ because $W^i \setminus X = W^i_{\text{n\_sep}}$, for $i = 1, 2, 3$. Thus, running steps 3,4 in our algorithm using this selection of W^1, W^2, W^3 will find a separator of size $\leq \alpha |X| \leq \alpha k$. By the first part of the proof, this separator is a α-sum-separator of W in G.

If $|W \cap S_1| > k$, then let $W^1_{\text{n\_sep}} = W \cap S_1$ and $W^2_{\text{n\_sep}} = W \cap (S_2 \cup S_3)$. Let $W^i_{\text{sep}} \subseteq W \cap X$, $i = 1, 2$ such that $W^1_{\text{sep}} \cup W^2_{\text{sep}} = W \cap X$ and $|W^i_{\text{sep}} \cup W^i_{\text{n\_sep}}| \leq \frac{|W|}{2}$.

Let $W^i = W^i_{\text{sep}} \cup W^i_{\text{n\_sep}}$, for $i = 1, 2$. Then, X separates $W^1 \setminus X, W^2 \setminus X$ because $W^i \setminus X = W^i_{\text{n\_sep}}$, for $i = 1, 2$. Thus, running steps 2,3 of algorithm $\frac{2}{3}$-vtx-sep (Figure 4.2) using this selection of W^1, W^2 will find a separator of size $\leq |X| \leq k$. By the first part of the proof, this separator is a α-sum-separator of W in G.

Finally, each choice of W^1 takes $O(f(|V|, |E|+k^2, k))$ time to check, for $f(n, m, k)$ the time taken by a α-approximating *3-way-vertex-separator* algorithm (or a minimum vertex separator algorithm, if it takes more time than the approximate 3-way-vertex-separator) over a graph with n vertices, m edges and treewidth $k-1$. There are at most $3^{|W|}$ ways to divide W into three sets. Since $|W| \leq (1+\alpha)k+1$, we run a vertex separation algorithm at most $3^{(1+\alpha)k+1} = O(3^{2\frac{1}{3}k}) = O(2^{3.6982k})$ times, for $\alpha = \frac{4}{3}$. Thus, the total time is $O(2^{3.6982k} f(|V|, |E| + k^2, k))$. □

THEOREM 4.10 ([7]). *If $G(V, E)$ is a graph with n vertices, $k \geq 1$ an integer, $\alpha \geq 1$ a real number, and $W \subset V$ such that $|W| \leq (\alpha+1)k+1$, then 3way-triang(G,W,k) triangulates G such that the vertices of W form a clique and such that the size of a largest clique of the triangulated graph $\leq (2\alpha+1)k$ or the algorithm correctly outputs that the cliquewidth of G is larger than k.*

Solutions for linear programs of multicommodity flow problems are typically slow. The linear programming subroutine used by the procedure of [7] for the subroutine of [29] can be replaced by the multicommodity flow algorithm of [39] (that work presents algorithms for directed, weighted networks). This combined algorithm finds a factor-$(1+\epsilon)\frac{4}{3}$ approximation to the optimal 3-way separator in time $O(\epsilon^{-2} nm \log^4 n)$, given $\epsilon > 0$. Selecting $\epsilon = \frac{1}{8k}$ guarantees that the separator is in fact a factor-$\frac{4}{3}$ approximation to the optimal (because the separator size is integral).

Using this procedure, the complexity of running the algorithm with $\alpha = \frac{4}{3}$ is $O(2^{3.6982k} n\, f(n, m+k^2, k)) = O(2^{3.6982k} nk^3 n^2 \log^4 n) = O(2^{3.6982k} n^3 k^3 \log^4 n)$. This is an improvement over the $O(2^{4.66k} n\, poly(n))$ of [7], especially because we have reduced the exponential dependency on k by a factor of about 2^k.

5. Applications to Other Graph Problems. The problem of finding the *pathwidth* of a graph is closely related to that of treewidth.

DEFINITION 5.1 ([48]). *A* path decomposition *of a graph $G(V, E)$ is a tree decomposition of G that is a path (chain).*

The *width* of a *path decomposition* is its width as a tree decomposition. The *pathwidth* of a graph G equals the minimum width over all path decompositions of G.

LEMMA 5.2 ([35]). *Let $G(V, E)$ be a graph with n vertices and treewidth $k \geq 1$. Then the pathwidth of G is at most $(k+1)\log(n-1)$.*

Using this lemma we can see that any of the algorithms we presented provides an approximation to pathwidth as well, with a somewhat larger factor. Kloks [35] in fact provides an algorithm for transforming a tree decomposition to a path decomposition of width at most $\log(n-1)$ times the width of the original tree decomposition. This procedure is fairly simple, and we do not bring it here.

Recall from Section 3.3 that $\beta = \frac{4\beta_1}{b-b'} = 2764.8$, with $\beta_1 = 115.2$ the constant factor given by the implementation of sparse cut in directed graphs of [41].

COROLLARY 5.3. *Let $G(V, E)$ be a graph with n vertices and treewidth $k \geq 1$. Then, using Procedure lgk-triang(G, $\emptyset$, k) and then applying the procedure provided by [35] for turning a tree decomposition to a path decomposition, finds a path-decomposition of G of width $\leq (\frac{4}{3}\gamma \log k) \cdot k \cdot \log(n-1)$, for $\gamma = c\beta$, and $c = 12 + 3\log_k \beta$. This procedure takes time $O(n^4 k \log k)$.*

COROLLARY 5.4. *The procedure of [35] together with* lgk-triang *output a path decomposition that has width within a factor of $O(\log(OPT) \cdot \log n)$ from the actual*

pathwidth (OPT) of G.

The last corollary follows immediately from the observation that every path decomposition is also a tree decomposition, so the treewidth of G must be at most OPT.

6. Allowing Arbitrary Node Weights. In many problems it is important to pay attention to the weight of each node in the graph. For example, while reasoning with a Markov Network N, if we create a clique from three nodes A, B, C that have 2 values each (binary nodes), then the clique has 2^3 possible values. However, if A, B, C have $2, 4, 8$ possible values, respectively, then the clique has $2 \times 4 \times 8 = 64 = 2^6$ values. Thus, we convert the problem associated with a network of m nodes $A_1, ..., A_m$, taking $n_1, ..., n_m$ values, respectively, into a graph problem where the weight on node A_i is $log_2 n_i$, and the treewidth problem becomes a *weighted treewidth* problem (each partition in an optimal tree decomposition should have *weight* that is no more than $k + 1$).

The notion of *treewidth* was originally defined for unweighted graphs. It was extended to weighted graphs in [8]. We recall this notion in what follows, and define *weighted treewidth* for weighted graphs. In this section we extend our algorithms to the weighted case, and show that these extensions are correct.

6.1. Weighted Treewidth. DEFINITION 6.1. *For a weighted graph $G(V, E)$, the* weighted width *of a* tree-decomposition *of G, $T = \langle \mathbb{X}, \mathbb{E} \rangle$ is* $\max_{X \in \mathbb{X}}(w(X) - 1)$. *The* weighted treewidth *of G equals the minimum weighted width over all tree-decompositions of G.*

LEMMA 6.2. *The weighted treewidth of G is the minimum $k \geq 0$ such that G is a subgraph of a triangulated graph with all cliques of weight at most $k + 1$.*

Proof. For a tree decomposition $T = \langle \mathbb{X}, \mathbb{E} \rangle$ we can define a triangulation G^t of G by making each partition in this tree decomposition into a clique in G^t. This graph is triangulated because it is a tree of cliques. Also, there are no cliques with larger weight than the weight of the partition from $\mathbb{X}$ of largest weight. This is because the cliques we created from the tree decomposition in G^t are the only maximal cliques in G^t (to see this, notice that every clique in G^t must appear together in at least one partition in every tree decomposition of G^t (proved in [35])). Finally, for the partition with the largest weight, there is a corresponding clique in G^t of the same weight.

For the other direction, let G^t be a triangulation of G of minimum maximal clique-weight. Since it is triangulated, it is the intersection graph of a family of subtrees of a tree $T = \langle \mathbb{X}, \mathbb{E} \rangle$ [53] (given a family of subtrees of a tree, a graph is constructed as follows: the vertices of the graph are the subtrees and two vertices are adjacent if the corresponding subtrees have at least one node in common). For each node i in this tree, define a subset X_i consisting of vertices for which the corresponding subtree contains i. Let $\mathbb{X} = \{X_i \mid i \in I\}$. It is easy to check that T is a tree decomposition for G^t (and thus also for G). Furthermore, each subset corresponds to a clique in G^t and thus has weight at most $k + 1$. This shows that the weighted width of T is at most $k + 1$. □

LEMMA 6.3. *Let $G(V, E)$ be a graph with n vertices and weighted treewidth k. There exists a set X with weight $k+1$ such that every connected component of $G[V \setminus X]$ has at most weight $\frac{1}{2}w(V)$.*

Proof. Let $T = \langle \mathbb{X}, \mathbb{E} \rangle$ be a tree decomposition of G of minimum weighted-width. Using a simple traversal of the tree we can find a partition that will serve as the required X. Here is how this traversal works: Select a node $X_i \in \mathbb{X}$ at random. If X_i satisfies the properties we seek for X, then stop. Otherwise, from among i's neighbors,

select the neighbor j such that the set of vertices from V that appear in the subtree of T rooted in j but are not in X_i has the largest weight.

This process terminates because we never go back-and-forth between nodes. To see this, notice that when we move from X_i to X_j, the subtree rooted in X_j (after the nodes of X_i are removed from it) has more than $\frac{1}{2}w(V)$ weight. Assume that after moving to X_j, the subtree rooted in X_i (after the vertices of X_j are removed from it) has weight that is more than $\frac{1}{2}w(V)$. Then, $w(Subtree_j \setminus X_i) > \frac{1}{2}w(V)$ and $w(Subtree_i \setminus X_j) > \frac{1}{2}w(V)$. However, this implies that $w(V) = w(Subtree_j \setminus X_i) + w(Subtree_i \setminus X_j) > \frac{1}{2}w(V) + \frac{1}{2}w(V) = w(V)$. Contradiction. □

The following lemma has a similar proof that we omit here.

LEMMA 6.4. *Let $G(V, E)$ be a graph with n vertices and weighted treewidth k. For every set $W \subseteq V$ there exists a set X with weight $k+1$ such that every connected component S of $G[V \setminus X]$ satisfies that $w(S \cap W) \leq \frac{1}{2}w(W)$.*

COROLLARY 6.5. *Let $G(V, E)$ be a graph with $n \geq k+1$ vertices and treewidth k. For every $W \subseteq V$, $|W| > 1$, there is a vertex separator X and sets $A, B, C \subset V$ such that $A \cup B \cup C \cup X = V$, A, B, C are separated by X, $w(X) \leq k+1$ and $w(W \cap C) \leq w(W \cap B) \leq w(W \cap A) \leq \frac{1}{2}w(W)$.*

We also make the following simple observation.

PROPOSITION 6.6. *If the weighted treewidth of G is k, then every vertex in V has weight at most k.*

6.2. Polynomial-time $O(\log OPT)$-Approximation Algorithm. In this section we modify our triangulation and treewidth algorithms earlier in this paper for the notion of weighted treewidth. Procedure *lgk-triang* needs a single change, replacing Steps 2,3 with the following:

1. If $w(V) \leq \beta \cdot k \cdot \log k$, then make a clique of G. Return.
2. Find X, an approximate minimum 3-way *weighted* $\frac{2}{3}$-vertex-separator of W in G, with S_1, S_2, S_3 the three parts separated by X (with $S_1, S_2 \neq \emptyset$, but possibly $S_3 = \emptyset$). If $w(X) > \beta \cdot w(W) \cdot \log k$, then output "*the weighted treewidth exceeds k*" and exit.

This amounts to replacing comparisons in which we counted vertices with similar comparisons in which we weigh vertices, and also using a weighted version of the same vertex separator algorithm. For the weighted version of Procedure *2-3-vsep-lgk* (Section 3.2) we need only to modify the call to procedure *bal-node-cut* as follows: Define $\pi_1(v) = w(v)$ for all $v \in V$; Define $\pi_2(v) = w(v)$ for all $v \in W$ and $\pi_2(v) = 0$ for all $v \in V \setminus W$. We call the new overall procedure *wlgk-triang*.

The proof of the following theorem is identical to that of Theorem 3.5, but uses Lemma 6.3 instead of Lemma 2.5.

THEOREM 6.7. *Procedure wlgk-triang(G, $\emptyset$, k) finds a triangulation of G of clique weight $\leq (\frac{4}{3}\gamma \log k) \cdot k$, for $\gamma = c\beta$, if the weighted treewidth of G is at most $k-1$, and $c = 12 + 3\log_k \beta$. This procedure takes time $O(n^4\, k \log k)$.*

6.3. Constant-factor Approximations to Weighted Treewidth. For algorithms *2way-2/3-triang*, *2way-half-triang* and *3way-triang* we perform similar replacement of steps. There, we need to show that our specialized vertex separator algorithms have the same performance time and approximation guarantee for the weighted vertices case. Since those procedures rely on subroutines for finding minimum (a, b)-vertex-separators, they are naturally extended to the case of weighted vertices. We modify the selection of subsets $W^1, W^2 \subset W$ so that they take maximal and minimal sets such that $w(W^1), w(W^2)$ are close to the limits taken in the original subroutines.

Intuitively, the time taken by each of the procedures is still bounded by a function of the number of vertices is W. Since $|W| \leq w(W)$ (because each vertex in V has weight at least 1 (we are interested in those graphs that correspond to Bayesian Networks or other cases that produce weighted graphs with $w(v) \geq 1$ for every $v \in V$)), the time bounds we established for our original algorithms hold here as well. The precise argument is a little more involved and is presented below.

We treat procedure *2way-2/3-triang* first. In $\frac{2}{3} - vtx - sep$ (Figure 4.2) we take W^2 minimal subset of W such that $w(W^2)$ is at least $\frac{w(W)}{3}$. We also take W^1 maximal subset of $W \setminus W^2$ of weight at most $\frac{w(W)}{2}$. The proof of the correctness follows in a similar way to the simple case, with several delicate modifications. Call the resulting procedure *weighted-$\frac{2}{3}$-vtx-sep*.

LEMMA 6.8. *Let $G(V, E)$ be a graph, $k \geq 0$ an integer, and $W \subseteq V$ of weight at least $3k + 3$. Algorithm weighted-$\frac{2}{3}$-vtx-sep(W, G, k) finds a vertex separator of W in G of weight $\leq k$ and parts S_1, S_2 such that $w(S_1 \cap W), w(S_2 \cap W) \leq \frac{2}{3}w(W)$, if such a separator exists, returning* failure *otherwise. It does so in time $O(2^{1.459147|W|} f(|V|, |E|+|W|^2, k))$, given a* min-$(a, b)$-vertex-separator *algorithm taking time $f(n, m, k)$.*

Proof. We prove the correctness of the algorithm first. Assume that the algorithm finds a separator X of S_1, S_2 in G'. X is also a separator of S_1, S_2 in G, by the way we constructed G' from G. Also, X separates $W^1 \setminus X$ and $W^2 \setminus X$ in G' (similar to the proof of Lemma 4.2). Finally, we show that $w(S_1 \cap W), w(S_2 \cap W) \leq \frac{2}{3}w(W)$. We notice that $w(W^2) \geq \frac{w(W)}{3}$ by the way we chose W^2. Also, $w(W^2) \leq \frac{w(W)}{3} + k$ because every vertex in V has weight at most k and we chose W^2 minimal such that $w(W^2) \geq w(W)/3$. Since $w(W) \geq 3k + 3$ we get that $w(W^2) \leq \frac{2}{3}w(W)$. Also, $w(W^1) \leq w(W) - w(W^2) \leq w(W) - \frac{w(W)}{3} = \frac{2}{3}w(W)$. Since $W^i \setminus X \subseteq S_i$ for $i = 1, 2$, also $|S_i \cap W| \leq \frac{2}{3}w(W)$. Notice that S_1, S_2 are never empty because $w(X) \leq k$ and $w(S_i) \geq w(W^i) - w(X) \geq 1$ for $i = 1, 2$ ($w(W^i) \geq \frac{w(W)}{3} \geq k + 1$ because $w(W) \geq 3k + 3$).

For the reverse direction, assume that the weighted treewidth of G is $k - 1$ and we show that the algorithm will find a suitable separator. Assume first that there are two sets of vertices S_1, S_2 separated by X in G such that $S_1 \cup S_2 \cup X = V$ and $w(S_i \cap W) \leq \frac{w(W)}{2}$, for $i = 1, 2$. Let $W^i_{\text{n\_sep}} = W \cap S_i$, for $i = 1, 2$. Let $W^i_{\text{sep}} \subseteq W \cap X$ such that $W^1_{\text{sep}} \cup W^2_{\text{sep}} = W \cap X$ and $w(W^i_{\text{sep}} \cup W^i_{\text{n\_sep}}) \leq \frac{2}{3}w(W)$. Let $W^i = W^i_{\text{sep}} \cup W^i_{\text{n\_sep}}$, for $i = 1, 2$. Then, X separates $W^1 \setminus X, W^2 \setminus X$, as $W^i \setminus X = W^i_{\text{n\_sep}}$, for $i = 1, 2$. Thus, running steps 2,3 in our algorithm using this selection of W^1, W^2 will find a separator of size $\leq w(X) \leq k$. By the previous paragraph, this separator is a $\frac{2}{3}$-vertex-separator of W in G.

Now we show that if there are no such sets S_1, S_2, X, then our algorithm still finds a suitable separator (assuming that it exists at all). By Corollary 6.5, there are three sets, A, B, C, of vertices separated by X in G such that $w(X) \leq k$ and $w(W \cap C) \leq w(W \cap B) \leq w(W \cap A) \leq \frac{1}{2}w(W)$. Let $S_1 = A$, $S_2 = B \cup C$. If $w(S_2 \cap W) \leq \frac{w(W)}{2}$, then the first selection case would cover this W (the previous paragraph). Thus, $w(S_2 \cap W) > \frac{w(W)}{2}$. Take $W^1 \subset (S_2 \cap W)$ of weight at most $\frac{w(W)}{2}$ and $W^2 \subset ((S_1 \cup X) \cap W) \setminus W^1$ of weight at least $\frac{w(W)}{3}$. The selection of W^2 is possible because $w((S_1 \cup X) \cap W) = w(S_1 \cap W) + w(X \cap W) \geq \frac{1}{3}w(W \setminus X) + w(X \cap W) \geq \frac{1}{3}w(W)$. For this selection of W^1, W^2 our algorithm will find a separator of size $\leq w(X) \leq k$ because X is already a separator of $W^1, W^2 \setminus X$. By the first paragraph in this proof,

this separator is a $\frac{2}{3}$-vertex-separator of W in G.

Finally, each choice of W^1 takes $O(f(|V|, |E| + |W|^2, k))$ time to check, for $f(n, m, k)$ the time taken by a *min-(a, b)-vertex-separator* algorithm over a graph with n vertices, m edges and weighted treewidth $k-1$. There are $\binom{|W|}{|W|/3}$ ways to choose $|W|/3$ elements (W^2) from a set of $|W|$ elements (W). Also, there are $\binom{2|W|/3}{|W|/2}$ ways to choose $|W|/2$ elements (W^1) from a set of $2|W|/3$ elements ($W \setminus W^2$). Since $\binom{|W|}{|W|/3} \leq O(2^{0.918295|W|})$ and $\binom{2|W|/3}{|W|/2} \leq O(2^{0.540852|W|})$, we get the time bound of $O(2^{1.459147|W|} f(|V|, |E| + |W|^2, k))$. □

We are left to present the modifications that we make to the recursive part of each of the triangulation procedure. For *2way-2/3-triang* (Figure 4.1) we replace Steps 1,2, 3 with

1. If $w(V) \leq 4k$, then make a clique of G. Return.
2. Let $W' \leftarrow W$. Add to W' a minimal (not necessarily minimum) number of vertices from V such that $w(W') \geq 3k+2$ (not necessarily of minimum weight among such W').
3. Find X, a minimum weighted $\frac{2}{3}$-vertex-separator of W' in G, with S_1, S_2 two nonempty parts separated by X ($S_1 \cup S_2 \cup X = V$) and $w(X) \leq k$. If there is no such separator, then output "*the weighted treewidth exceeds $k-1$*" and exit.

Notice that for Step (2) above we simply need to add a minimal number of vertices to W', and not necessarily try to reach a minimum number of additional vertices or a minimum weight for W'. This requirement is needed to ensure that $|W'| \leq 3k+2$ (which has an impact on the running time of our *weighted-$\frac{2}{3}$-vtx-sep*), and we do not care how large $w(W')$ turns out to be (in fact, since every vertex in V has weight at most k, this weight is at most $4k+2$).

THEOREM 6.9. *Procedure weighted-2way-2/3-triang(G, $\emptyset$, k) finds a triangulation of G of maximum weighted clique $\leq 4k+1$, if the weighted treewidth of G is at most $k-1$, in time $O(2^{4.38k}|V|^{\frac{5}{2}})$ or $O(2^{4.38k}|V|^2 k)$ if we use the minimum (a, b)-vertex-separator algorithm of [25] or [28], respectively.*

The modifications for procedures *2way-half-triang* (Figure 4.3) and *3way-triang* (Figure 4.5) are similar. We bring them here without their proofs which are similar to those brought above.

For a weighted version of *2way-half-vtx-sep* we replace the original choice of W^1, W^2 in Figure 4.4 with a choice of W^1 to be a minimal set such that $w(W^1) \geq \frac{w(W)}{2}$ and $W^2 = W \setminus W^1$. We also replace Steps 1,2,3 in procedure *2way-half-triang* with

1. If $w(V) \leq 4k$, then make a clique of G. Return.
2. Let $W' \leftarrow W$. Add to W' a minimal (not necessarily minimum) number of vertices from V such that $w(W') \geq 3k+2$.
3. Find X, a two-way weighted $\frac{1}{2}$-vertex-separator of W' in G, with S_1, S_2 the two nonempty parts separated by X ($S_1 \cup S_2 \cup X = V$) and $w(X) \leq \frac{3}{2}k$. If there is no such separator, then output "*the treewidth exceeds $k-1$*" and exit.

We call the resulting procedure *weighted-2way-half-triang*.

THEOREM 6.10. *Procedure weighted-2way-half-triang(G, $\emptyset$, k) finds a triangulation of G of clique weight $\leq 3k+2$, if the treewidth of G is at most $k-1$, in time $O(2^{3k} n^{\frac{5}{2}} k^{\frac{1}{2}})$ or $O(2^{3k} n^2 k^{\frac{3}{2}})$ if we use the minimum (a, b)-vertex-separator algorithm of [25] or [28], respectively.*

Finally, the modifications for procedures *3way-triang* (Figure 4.5 for a weighted

version include replacing the original choice of W^1, W^2, W^3 in procedure *α-sum-sep* (Figure 4.6) with a choice of W^1, W^2, W^3 such that $\frac{w(W)}{2} \geq w(W^1) \geq w(W^2) \geq w(W^3)$. We also replace Steps 1,2,3 in procedure *3way-triang* with

1. If $w(V) \leq (2\alpha + 1)k$, then make a clique of G. Return.
2. Let $W' \leftarrow W$. Add to W' a minimal (not necessarily minimum) number of vertices from V such that $w(W') \geq (1 + \alpha)k + 1$.
3. Find X, a minimum weighted α-sum-separator of W' in G, with S_1, S_2, S_3 three parts separated by X (at least two are nonempty) and $S_1 \cup S_2 \cup S_3 \cup X = V$. If there is none, then output "*the treewidth exceeds* $k - 1$" and exit.

where *weighted α-sum-separator* is defined to satisfy $w((S_i \cap W) \cup X) \leq (1 + \alpha)k$ for every $i = 1, 2, 3$.

THEOREM 6.11. *If $G(V, E)$ is a graph with n vertices, $k \geq 1$ an integer, $\alpha \geq 1$ a real number, and $W \subset V$ such that $w(W) \leq (\alpha + 1)k + 1$, then weighted-3way-triang(G,W,k) triangulates G such that the vertices of W form a clique and such that the size of a heaviest clique of the triangulated graph $\leq (2\alpha + 1)k$ or the algorithm correctly outputs that the weighted treewidth of G is larger than $k - 1$.*

7. Experimental Results. We implemented constructive variants of our algorithms *2way-2/3-triang* and *2way-half-triang*, i.e., given a graph, G, they return a tree decomposition of G. The main difference between the description given above and our implementation is that we do not increase the size of W' to be $3k + 2$ in step 2) of Figure 4.1 (we do not know what k is, a priori). Instead, we gradually increase W''s size during the execution of $\frac{2}{3}$-vtx-sep, until we find a cardinality of $|W'|$ for which a minimum separator has both separated sets non-empty. This is particularly useful when only some of the partitions of the tree decomposition are of size close to the limit.

Graph	$\|V\|$	$\|E\|$	Time $(4 + \frac{1}{2})$-apx	Time 4-apx	$(4 + \frac{1}{2})$-apx W+1	4-apx W+1	*min-deg.* W+1
CYC1	142	469	1m 2s	6m 34s	21	21	14
CPCS1	360	1036	8m 50s	1hr 11m	28	26	21
CPCS2	421	1704	15m 40s	3hr 55m	33	33	24
HPKB1	446	2637	2hr 7m	14hr 13m	58	45	37
HPKB2	570	3840	7hr 52m	5dy 23hr	70	60	41

FIG. 7.1. *Graphs, their processing time and the resulting width of the decomposition.*

We use an implementation of Chekassky and Goldberg [16] for Dinitz's max-flow algorithm. We have experimented with several graphs of various sizes and treewidths that are associated with real-world problems. The graphs and implementations are available at [2]. The results are depicted in Figure 7.1. They were achieved on a Sun SuperSparc 60. For comparison[5] we ran the implementation of the algorithm of [52]. Unfortunately, that algorithm did not return answers for any of these graphs after more than three days. This is not surprising if we compare our theoretical results to those reported in [7, 52]. These algorithms have been tested with graphs of treewidths ≤ 6, $n \leq 50$, $m \leq 110$ (real-world graphs) and treewidth ≤ 10, $n \leq 100$ (artificially generated), respectively, an order of magnitude lower than those used here.

It is important and interesting to notice that the *min-degree* heuristic [50, 33], which iteratively selects a node that has as few neighbors as possible, makes a clique

[5] We could not get the implementation of [7].

from the neighbors and removes the node, achieved better tree decompositions than our approximation-guaranteed algorithms on these samples. This heuristic takes between 1 second and 2 minutes on our sample graphs with a sub-optimal implementation, but is not guaranteed to approximate the optimal by a constant factor (examples exist in which this heuristic performs arbitrarily bad as compared to the optimum).

The results that we achieved with these implementations were used successfully in [43] for inference with large knowledge bases in First-Order Logic.

8. Conclusions. We presented four related approximation algorithms for triangulation of minimum treewidth. The first algorithm, *lgk-triang*, is the first polynomial time algorithm for an approximation factor that does not depend on the size of the graph. It gives rise to the best approximation factor known for treewidth and pathwidth in polynomial time. Two of the others, *2way-2/3-triang* and *3way-triang*, are modifications of previous algorithms that improve their running speed by a factor exponential in k and polynomial in n. The fourth algorithm, *2way-half-triang*, has the best combined n, k time bound known for any constant-factor approximation algorithm. In this paper we also introduced the notion of *weighted treewidth*, which is better suited for applications in artificial intelligence. We extended these algorithms to this more general problem, and derived similar approximation guarantees and time bounds.

We showed that our algorithms are efficient enough to solve large problems of practical importance. The results of some of the tree decompositions that we produced are currently being used in reasoning with the HPKB and CYC knowledge bases [17, 42] using algorithms of [4].

Between the time of submission of this manuscript and its publication several related results appeared in conferences and deserve mentioning here. In particular, Feige, Hajiaghayi, and Lee [26] present algorithms for $O(\sqrt{OPT})$ approximation to treewidth, thus improving on the bound given in this paper. Also, Fomin, Kratsch, and Todinca [27] give bounds on the number of minimal separators and maximal cliques in a graph, yielding a bound of $O(1.9601^n)$ on time for computing the exact treewidth.

Acknowledgements. Work with Sheila McIlraith has raised my interest in the methods that iteratively divided graphs in two/three parts, interest for which she deserves thanks. Kirill Shoikhet allowed me to use his code for optimal triangulation. Daphne Koller and Ben Tasker have given me access to their copy of the CPCS Bayesian networks. Daphne Koller and Irina Rish have pointed out the heuristic I compared with. I have used an implementation of Dinitz flow algorithm that is due to Andrew Goldberg and Boris Chekasski. This research was supported in part by DARPA grant N66001-00-C-8018 (RKF program).

Appendix A. Balanced Edge Cuts in Directed Graphs.

We bring an outline of the algorithm of [41] for finding a b'-balanced edge cut that is within $O(\log p)$ from the optimal b-balanced edge cut, for p the number of vertices in G^* of non-zero weight.

The procedure is based on a solution to a multicommodity flow problem in a directed graph. We first recount the procedure for finding sparse cuts in uniformly weighted graphs. We then extend this procedure in several steps and get a procedure for balanced edge-cuts in directed graph.

Given a directed graph G, a *multicommodity flow* problem with edge costs and node weights has $k \geq 1$ commodities, each with its own source node s_i, sink node t_i,

and demand $D_i > 0$. The objective is to simultaneously route D_i units of commodity i from s_i to t_i for each i so that the total amount of all commodities passing through any edge is no greater than its capacity (cost).

Given a procedure that solves multicommodity flow, we find *sparse cuts* as follows.

1. Solve (accurately or approximately) the dual linear program of the given k-commodity flow problem in a directed graph, finding a distance function, d. There are efficient algorithms for that (e.g., [39] does that in deterministic time $O(k^2nm \log k)$ for k commodities).
2. Partition the graph using d from the solution of Step '1' and $\Delta = \frac{1}{2n^2}$: First, create a graph G^+ from G that has unit distance for each edge (make every edge into a path proportional to that distance). Then, randomly select $v \in V + \cap V$ and find a subset of G^+ that is of some proper distance from v (the distance j is selected such that the difference between the set of radius j and the set of radius $j+1$ is less than a threshold that depends on Δ). Remove this subset from G^+ and repeat the process until no vertices of G in G^+ are left. The intersection of the sets that we got with V are the sets in the partitioning.
3. Aggregate partitions from Step '2' into two sets forming a cut that is of proper size. We are guaranteed to get a cut of ratio $36 \cdot M \cdot \log n$ (M is the maximum flow, which is an upper bound on the minimum ratio cut).

To find a sparse cut in a directed, weighted graph G, we change the above algorithm as follows.

1. We allow vertices to have weights, $\pi(v)$, and have $\mathcal{P}$ the set of nodes for which π is nonzero. We change the algorithm by using a slightly different linear program (instead of the simple multi-commodity flow program), use $\Delta = \frac{1}{2p^2}$, for $p = |\mathcal{P}|$, and aggregate the sets of Step '2' above in a way that matches $\mathcal{P}$. To compute the sparsest cut we use the last algorithm with all capacities set to the corresponding edge weights.
2. We allow direction of edges by setting the same linear program as before, but with demand from u to v being $\pi(u)\pi(v)$ (same demand on both directions). Step '2' changes to include the direction of the edges. In particular, now we check two balls or radius i from v: one for incoming distance and one for outgoing distance. The smaller among the two sets is chosen (after not passing the threshold for increase) and the final sets are divided into in-sets and out-sets. The last stage is similar to the original one, but now the constant factor is somewhat worse ($115.2 \cdot M \cdot \log p$). We use $\Delta = \frac{1}{4p^2}$.

Finally, to find a b'-balanced edge cut for $b' \leq 1/3$ and $b' < b$, iteratively find a sparsest cut in the largest set from the previous iteration. We stop when the resulting subgraph is no larger than $(1 - b') \cdot w(V)$. This subgraph and the union of the sets we removed from G define the cut.

To find a 3-way b'-balanced *directed edge cut* for $b' \leq 1/3$ and $b' < b$, we use the same algorithm as for the undirected one, but divide the subgraphs that are *pealed off* from G into out-edges and in-edges. This gives a three-way decomposition (the third subgraph is the one left over from the *pealing*). It keeps the same constant factor as before.

REFERENCES

[1] Karl A. Abrahamson and Michael R. Fellows. Finite automata, bounded treewidth and well-

quasiordering. In *Graph structure theory*, number 147 in Contemporary Mathematics, pages 539–564. American Mathematical Society, 1993. Proc. Conf. on Graph Minors (Seattle, 1991).

[2] Eyal Amir. Partitioning and reasoning project website. http://www.cs.uiuc.edu/~eyal/decomp.

[3] Eyal Amir, Robert Krauthgamer, and Satish Rao. Constant factor approximation of vertex-cuts in planar graphs. In *Proc. 35rd ACM Symp. on Theory of Computing*, pages 90–99. ACM Press, 2003.

[4] Eyal Amir and Sheila McIlraith. Partition-based logical reasoning. In *Principles of Knowledge Representation and Reasoning: Proc. Seventh Int'l Conference (KR '2000)*, pages 389–400. Morgan Kaufmann, 2000.

[5] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embebdings in a K-tree. *SIAM J. Alg. and Discr. Meth.*, 8:277–284, 1987.

[6] Stefan Arnborg, Jens Lagergren, and Detlef Seese. Problems easy for tree-decomposable graphs. *Journal of Algorithms*, 12:308–340, 1991.

[7] Ann Becker and Dan Geiger. A sufficiently fast algorithm for finding close to optimal junction trees. In *Proc. Twelfth Conference on Uncertainty in Artificial Intelligence (UAI '96)*, pages 81–89. Morgan Kaufmann, 1996.

[8] Ann Becker and Dan Geiger. A sufficiently fast algorithm for finding close to optimal clique trees. *Artificial Intelligence*, 125(1-2):3–17, 2001.

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

[10] Hans L. Bodlaender. Treewidth: Algorithmic techniques and results. In Igor Prívara and Peter Ruzicka, editors, *Mathematical Foundations of Computer Science 1997*, volume 1295 of *lncs*, pages 19–36. Springer, 1997.

[11] Hans L. Bodlaender. Personal communication. September 2000.

[12] Hans L. Bodlaender, John R. Gilbert, Hjálmtýr Hafsteinsson, and Ton Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *Journal of Algorithms*, 18(2):238–255, March 1995.

[13] Hans L. Bodlaender and Ton Kloks. Better algorithms for the pathwidth and treewidth of graphs. In *Automata, Languages and Programming, 18th International Colloquium*, volume 510 of *LNCS*, pages 544–555. Springer-Verlag, 1991.

[14] Vincent Bouchitte and Ioan Todinca. Treewidth and minimum fill-in: grouping the minimal separators. *SIAM Journal on Computing*, 31(1):212–232, 2001.

[15] Hajo Broersma, Ton Kloks, Dieter Kratsch, and Haiko Müller. A generalization of AT-free graphs and a generic algorithm for solving treewidth, minimum fill-in and vertex ranking. In *WG: Graph-Theoretic Concepts in Computer Science, International Workshop WG*, pages 88–99, 1998. LNCS 1517.

[16] Boris V. Chekassky and Andrew V. Goldberg. On implementing the push-relabel method for the maximum flow problem. *Algorithmica*, 19(4):390–410, 1997.

[17] Paul Cohen, Vinay Chaudhri, Adam Pease, and Robert Schrag. Does prior knowledge facilitate the development of knowledge-based systems. In *Proc. National Conference on Artificial Intelligence (AAAI '99)*, pages 221–226. AAAI Press/MIT Press, 1999.

[18] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. McGraw Hill, 1989.

[19] William H. Cunningham. The optimal multiterminal cut problem. In *DIMACS Series in Disc. Math. and Theor. Comput. Sci.*, volume 5, pages 105–120. American Mathematical Society, 1991.

[20] Rina Dechter. Bucket elimination: A unifying framework for probabilistic inference. In *Proc. Twelfth Conference on Uncertainty in Artificial Intelligence (UAI '96)*, pages 211–219. Morgan Kaufmann, 1996.

[21] Rina Dechter and Judea Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38:353–366, 1989.

[22] Erik D. Demaine, Mohammad Taghi Hajiaghayi, Naomi Nishimura, Prabhakar Ragde, and Dimitrios M. Thilikos. Approximation algorithms for classes of graphs excluding single-crossing graphs as minors. *Journal of Computer and System Sciences*, 69(2):166–195, 2004.

[23] E. A. Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Mathematics Doklady*, 11:1277–1280, 1970.

[24] Shimon Even. An algorithm for determining whether the connectivity of a graph is at least k. *SIAM Journal on Computing*, 4(3):393–396, September 1975.

[25] Shimon Even. *Graph Algorithms*. Computer Science Press, 1979.

[26] Uriel Feige, Mohammad Taghi Hajiaghayi, and James R. Lee. Improved approximation algo-

rithms for minimum-weight vertex separators. In *Proc. 37rd ACM Symp. on Theory of Computing*, pages 563–572. ACM Press, 2005.

[27] Fedor V. Fomin, Dieter Kratsch, and Ioan Todinca. Exact (exponential) algorithms for treewidth and minimum fill-in. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP 2004)*, pages 568–580. Springer Verlag, 2004.

[28] L. R. Ford Jr. and D. R. Fulkerson. *Flows in networks*. Princeton University Press, 1962.

[29] Naveen Garg, Vijay V. Vazirani, and Mihalis Yannakakis. Multiway cuts in directed and node weighted graphs. In *Automata, Languages and Programming, 21st ICALP*, volume 820 of *Lecture Notes in Computer Science*, pages 487–498. Springer-Verlag, 1994.

[30] Roy Goldman, Narayanan Shivakumar, Suresh Venkatasubramanian, and Hector Garcia-Molina. Proximity search in databases. In *Proceedings of the 24th Intl' Conf. on Very Large Databases (VLDB 1998)*, 1998.

[31] Holger H. Hoos and Thomas Stützle. SATLIB - the satisfiability library. Canadian SATLIB site, hosteb by the Laboratory for Computational Intelligence at the computer science department of the University of British Columbia in Vancouver, Canada, 2001. Can be found at http://www.satlib.org.

[32] Finn V. Jensen, Steffen L. Lauritzen, and Kristian G. Olesen. Bayesian updating in recursive graphical models by local computation. *Computational Statistics Quarterly*, 4:269–282, 1990.

[33] Uffe Kjaerulff. *Aspects of efficiency imporvement in bayesian networks*. PhD thesis, Aalborg University, Department of Mathematics and Computer Science, Fredrik Bajers Vej 7E, DK-9220 Aalborg, Denmark, 1993.

[34] P. Klein, A. Agrawal, R. Ravi, and S. Rao. Approximation through multicommodity flow. In *Proc. 31st IEEE Symp. on Foundations of Computer Science (FOCS'90)*, pages 726–739. IEEE Press, October 1990.

[35] Ton Kloks. *Treewidth: computations and approximations*, volume 842 of *LNCS*. Springer-Verlag, 1994.

[36] Jens Lagergren. Efficient parallel algorithms for tree-decomposition and related problems. In *Proc. 31st IEEE Symp. on Foundations of Computer Science (FOCS'90)*, pages 173–182. IEEE Press, October 1990.

[37] Jens Lagergren and Stefan Arnborg. Finding minimal forbidden minors using a finite congruence. In *Proc. 18th Int. Coll. Automata, Languages and Programming*, number 510 in Lecture Notes in Computer Science, pages 532–543. Springer-Verlag, 1991.

[38] Steffen L. Lauritzen and David J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society B*, 50(2):157–224, 1988.

[39] Tom Leighton, Fillia Makedon, Serge Plotkin, Clifford Stein, Eva Tardos, and Spyros Tragoudas. Fast approximation algorithms for multicommodity flow problems. *Journal of Computer and System Sciences*, 50(2):228–243, 1995.

[40] Tom Leighton and Satish Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In *Proc. 29th IEEE Symp. on Foundations of Computer Science (FOCS'88)*, pages 422–431, 1988.

[41] Tom Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46(6):787–832, 1999.

[42] Douglas B. Lenat. Cyc: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38, 1995.

[43] Bill MacCartney, Sheila McIlraith, Eyal Amir, and Tomas Uribe. Practical partition-based theorem proving for large knowledge bases. In *Proc. Eighteenth International Joint Conference on Artificial Intelligence (IJCAI '03)*, pages 89–96. Morgan Kaufmann, 2003.

[44] Sheila McIlraith and Eyal Amir. Theorem proving with structured theories. In *Proc. Seventeenth International Joint Conference on Artificial Intelligence (IJCAI '01)*, pages 624–631. Morgan Kaufmann, 2001.

[45] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[46] Malcolm Pradhan, Gregory Provan, Blackford Middleton, and Max Henrion. Knowledge engineering for large belief networks. In *Proc. Tenth Conference on Uncertainty in Artificial Intelligence (UAI '94)*, pages 484–490. Morgan Kaufmann, 1994.

[47] Bruce A. Reed. Finding approximate separators and computing tree width quickly. In *Proc. 24th ACM Symp. on Theory of Computing*, pages 221–228. ACM Press, 1992.

[48] Neil Robertson and Paul D. Seymour. Graph minors. II: algorithmic aspects of treewidth. *Journal of Algorithms*, 7:309–322, 1986.

[49] Neil Robertson and Paul D. Seymour. Graph minors XIII. the disjoint paths problem. *J. Comb. Theory, Series B*, 63:65–110, 1995.

[50] D. J. Rose. Triangulated graphs and the elimination process. *J. of Discrete Mathematics*, 7:317–322, 1974.

[51] Paul D. Seymour and Robin Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994.

[52] Kirill Shoikhet and Dan Geiger. A practical algorithm for finding optimal triangulations. In *Proc. National Conference on Artificial Intelligence (AAAI '97)*, pages 185–190. Morgan Kaufmann, 1997.

[53] J. R. Walter. Representations of chordal graphs as subtrees of a tree. *Journal of Graph Theory*, 2:265–267, 1978.