Parameterized Approximation Schemes using Graph Widths

Michael Lampis*

Research Institute for Mathematical Sciences (RIMS), Kyoto University
mlampis@kurims.kyoto-u.ac.jp

Abstract. Combining the techniques of approximation algorithms and parameterized complexity has long been considered a promising research area, but relatively few results are currently known. In this paper we study the parameterized approximability of a number of problems which are known to be hard to solve exactly when parameterized by treewidth or clique-width. Our main contribution is to present a natural randomized rounding technique that extends well-known ideas and can be used for both of these widths. Applying this very generic technique we obtain approximation schemes for a number of problems, evading both polynomial-time inapproximability and parameterized intractability bounds.

1 Introduction

Approximation algorithms and parameterized complexity are two of the most popular ways of dealing with NP-hard optimization problems. Nevertheless, the two sets of techniques are usually treated independently. It's therefore a very natural question whether combining the techniques of both theories can be used to obtain algorithmic results which are out of reach for each one of them separately. This has often been identified as a promising research field (see [30] for a survey), but its development has so far been somewhat limited. The goal of this paper is to add some results in this area by designing parameterized approximation schemes for problems which are both parameterized intractable (W-hard) and hard to approximate in polynomial time (APX-hard).

The problems we will focus on are optimization problems on graphs of bounded treewidth or clique-width. These two graph widths are of central importance to parameterized complexity theory. At the same time, they play a significant role in the design of approximation algorithms, since subroutines employing them are often used as building blocks of larger algorithms. Therefore, understanding the extent to which we can efficiently approximate problems which remain W-hard for these widths is of potentially great importance from several points of view.

In this paper we want to show that many such hard problems actually turn out to be well-approximable in FPT time. Perhaps the easiest way to explain our aim is to state a representative example of the type of results we will establish.

Theorem 1 (partial statement).
There exists a randomized $(1+\epsilon)$*-approximation algorithm for* Max Cut *running in time* $\left(\frac{\log n}{\epsilon}\right)^{O(w)} n^{O(1)}$*, where* w *is the input graph's clique-width.*

Max Cut is of course a problem of central importance in the contexts of both approximability and parameterized complexity. It is APX-hard (so an approximation ratio of $1+\epsilon$ is probably impossible in polynomial time) and W-hard parameterized by clique-width (so the fastest exact algorithm probably needs time roughly n^w). Our main point here is that using a *parameterized*

* Research partially supported by a Scientific Grant-in-Aid from Ministry of Education, Culture, Sports, Science and Technology of Japan.

approximation approach we can evade these lower bounds, leading to a $(1+\epsilon)$-approximation running in time only $(\log n)^{O(w)}$, that is, an FPT approximation scheme. More generally, the goal of this paper is to provide, in a *uniform* way, similar approximation (or bicriteria approximation) schemes for a diverse set of W-hard and APX-hard graph problems. The problems for which we will provide algorithms are the following: Max Cut, Edge Dominating Set, Bounded Degree Deletion, Capacitated Dominating Set, Capacitated Vertex Cover, Equitable Coloring and Graph Balancing. For most of these problems we are able to provide equally efficient algorithms for both treewidth and clique-width (a detailed description of all results is given further below).

Paper overview: In this paper we adopt a generic technique that is a variation of the standard dynamic programming used for treewidth and clique-width. We observe that for a number of problems which are parameterized intractable for these widths, the hardness intuitively stems from the fact that large *integers* need to be stored in the dynamic programming table. These integers are usually calculated simply by *adding* previously calculated entries (all the problems listed above fall into this general category, though for some this is not obvious). We want to shrink the table, and thus speed up the algorithms, by storing these integers approximately.

The basic idea we use is very natural. We fix a parameter $\delta > 0$ and represent all integers in $\{1, \ldots, n\}$ by rounding them to the closest integer power of $(1+\delta)$. If δ is not too small $(\delta = \Omega(\frac{1}{\log^c n}))$ the natural dynamic programming table's size is dramatically reduced from n^w to $(\log n)^{O(w)}$. The obvious obstacle to this approach, however, is that during the process of running a dynamic programming algorithm on the approximate values the rounding errors will propagate and potentially pile up to a large error. How can we keep the errors under control?

In this paper we suggest a very natural randomized rounding approach to this problem. The main contribution is to show that this rounding idea can be seamlessly incorporated into the standard dynamic programming techniques of treewidth and clique-width to give efficient approximation schemes. In order to make the transition from exact to approximate algorithms as cleanly as possible we separate the analysis into two parts. First, we introduce an abstract model of computation, called Approximate Addition Trees, which captures the essence of the rounding ideas we described. We fully analyze the approximation performance of these Trees and prove some general approximation theorems. Then, relying on this analysis we give a series of approximation algorithms using clique-width and treewidth. It's worth stressing at this point that all the algorithms we will present follow the standard dynamic programming mold that should be very familiar to readers accustomed to graph widths. The important difference is that their analysis, in addition to standard methods, also crucially relies on our results on Approximate Addition Trees (which we can use as a black box). Thus, by abstracting away the Addition Trees, our technique can be viewed as a natural extension of well-known ideas. The hope is that this modularization will allow our technique to be easily reused and eventually become part of the standard graph width toolkit.

Thus, what is left to describe is the workings of Approximate Addition Trees. This is, expectedly, the most technical part of the paper. As we will see, there do exist some important special cases where a complicated analysis can be avoided (notably, when the input tree is balanced) and there is some value in these cases since they can help make some algorithms deterministic. However, in order to obtain the more interesting results of this paper we need an analysis of full generality. In other words, we need to establish an approximation theorem that works for all Addition Trees without making any special assumptions about their structure. Our main technical contribution is

that we do establish such a result and this allows us to analyze all the algorithms of this paper in terms of Addition Trees. We thus present a robust, unified technique that works for both treewidth and clique-width (and potentially other similar graph widths), without relying on any non-trivial width-specific properties.

Summary of results: Let us now formally state the algorithmic results presented in this paper. Full problem definitions are given further below.

In the following theorems, $n^{O(1)}$ factors are omitted from the running times.

Theorem 1. *Given an n-vertex graph $G(V, E)$, a clique-width expression with w labels, and an error parameter $\epsilon > 0$, there exist randomized algorithms which, with high probability, achieve the following:*

- *Produce an approximate solution to* Max Cut *with size at least $\frac{\text{OPT}}{1+\epsilon}$ in time $(\log n/\epsilon)^{O(w)}$.*
- *Produce an approximate solution to* Edge Dominating Set *with size at most $(1+\epsilon)$OPT in time $(\log n/\epsilon)^{O(w)}$.*
- *Given an integer k, either decide (correctly) that G does not admit an* Equitable Coloring *with k colors or produce a valid k-coloring where the ratio of the sizes of any two color classes is at most $(1+\epsilon)$ in time $(\log n/\epsilon)^{O(k)}2^{kw}$.*
- *Given an integer Δ find a set of vertices that is at most as large as the optimal solution for* Bounded Degree Deletion *to degree Δ and whose deletion makes the maximum degree at most $(1+\epsilon)\Delta$, in time $(\log n/\epsilon)^{O(w)}$.*
- *Given a capacity for each vertex, find a* Capacitated Dominating Set *of size at most* OPT*, such that all but at most ϵn vertices are dominated, in time $(\log n/\epsilon)^{O(w)}$.*

In addition, if instead of a clique-width expression we are given a tree decomposition of width w, there exist deterministic algorithms, with the same running times, achieving all the above.

Furthermore, we also have the following results for treewidth.

Theorem 2. *Given an n-vertex graph $G(V, E)$, a tree decomposition of width w, and an error parameter $\epsilon > 0$, there exist deterministic algorithms which achieve the following:*

- *Produce an approximate solution with cost at most $(1+\epsilon)$OPT for* Graph Balancing*, in time $(\log n)^{O(w)}$.*
- *Given a capacity for each vertex, find a* Capacitated Dominating Set *(or* Capacitated Vertex Cover*) of size at most* OPT*, such that no capacity is violated by a factor of $(1+\epsilon)$ or more, in time $(\log n)^{O(w)}$.*

The algorithms achieving the results of Theorems 1 and 2 are given in Section 4.

Previous work: Let us first review previous work for the specific problems we will focus on. In Max Cut we want to partition the vertices of a graph into two sets so that the number of edges with one endpoint in each set is maximized. Max Cut was shown to be W-hard when parameterized by clique-width in [23]. The problem is known to be APX-hard in general [33]. In Edge Dominating Set we want to select the smallest possible set of edges such that all edges

share an endpoint with a selected edge. This problem is also APX-hard and W-hard for clique-width [23]. Both problems are FPT parameterized by treewidth. Let us remark that, in addition to these two problems, very few other problems are known to be W-hard for clique-width but FPT for treewidth. The set of problems that behave this way are sometimes considered part of "the price of generality" that clique-width affords, compared to treewidth. Investigating this price has been an interesting research topic in parameterized complexity. One interpretation of the results of this paper is therefore that this "price" is not as high as previously believed, since two of the most prominent problems from this family can be well-approximated for clique-width.

In Equitable Coloring we want to find a proper k-coloring of a graph so that all color classes have the same size. Equitable Coloring is known to be W-hard by the results of Fellows et al. [19] even when parameterized by both the treewidth of the input graph w and the number of colors k. An exact XP algorithm parameterized by $w+k$ can be easily obtained with standard techniques, but a more general XP algorithm parameterized by w only is shown in [6]. The problem is easily shown to generalize Graph Coloring (if we add a sufficiently large independent set to a graph G then G admits an equitable coloring with k colors if and only if its chromatic number is at most k).

In Graph Balancing we are given an edge-weighted graph and need to find an orientation of the edges so that the maximum weighted out-degree of any vertex is minimized. Graph Balancing is sometimes also called Minimum Maximum Outdegree in the literature. One motivation for the study of this problem is that it is a special case of min-makespan scheduling (if vertices represent machines and edges represent jobs that can be processed only by their endpoints). Graph Balancing was shown to be W[1]-hard parameterized by treewidth by Szeider [38]. In [1] it was shown to be in P when all edge weights are equal, while in [37] an XP algorithm was shown when the problem is parameterized by treewidth. Regarding polynomial-time approximations, in [17] the problem was shown to be hard to approximate with a ratio better than 1.5 even if all weights are 1 or 2. In the same paper a 1.75-approximation was given.

In Bounded Degree Deletion we want to delete as few vertices as possible to make the maximum degree of a graph Δ. Bounded Degree Deletion was shown to be W[1]-hard parameterized by treewidth by Betzler et al. [4]. The problem generalizes Vertex Cover and is therefore APX-hard to solve for general graphs. In Capacitated Dominating Set (Capacitated Vertex Cover) we are given a graph where each vertex v has a capacity $c(v)$ and we want to find a dominating set (resp. vertex cover) such that each selected vertex v is used to dominate at most $c(v)$ other vertices (resp. cover $c(v)$ edges). Capacitated Dominating Set and Capacitated Vertex Cover were shown W[1]-hard when parameterized by the size of a minimum feedback vertex set (and therefore, also when parameterized by treewidth) in [16]. Both problems are at least APX-hard to approximate in polynomial time, since they generalize Dominating Set and Vertex Cover. Since the algorithms we give approximate the capacity (or degree) constraints they can be described as $(1, 1+\epsilon)$-bicriteria approximations for these problems.

Let us also recall some more general related work. Very few FPT approximation schemes are currently known. For an overview of the most important results see the survey by Marx [30]. The same paper gives an FPT approximation scheme for Max Vertex Cover parameterized by the size of the cover. This is extended in [36] to an FPT approximation scheme for Max Cover. See also [10] for FPT approximation schemes for related covering problems. Sum Edge Multicoloring is a rare example of a problem currently known to admit an FPT approximation scheme parameterized by treewidth [29].

Let us also mention that the notion of FPT approximation also makes sense when one is trying to obtain constant factor approximations (instead of approximation schemes) (see e.g. [24]). It's also interesting to approximate problems which are FPT, if the approximation algorithm can run in significantly improved time (see [11] or [5]). The complexity of parameterized approximations for naturally parameterized problems (that is, parameterized by the size of the solution) has also been considered. Unfortunately, the evidence so far seems to suggest that standard problems, such as Clique and Dominating Set are hard to approximate even in a parameterized setting [9],[13].

In this paper we focus on problems parameterized by treewidth or clique-width. For an introduction to these notions see [8, 15, 18]. It was initially believed that problems solvable on trees are almost always FPT parameterized by treewidth. Gradually, many exceptions were discovered. This includes problems such as Edge-disjoint Paths and L(2,1)-coloring, which are NP-hard for graphs of treewidth 2 [32, 20] and Steiner Forest which is NP-hard for graphs of treewidth 3 ([2] gives a PTAS for this problem using treewidth). More relevant to our purposes are problems which are solvable in polynomial time for constant treewidth, but not FPT. Some examples of such problems when parameterized by treewidth (in addition to the problems we consider in this paper) are the following: Target Set Selection [3], Maximum Path Coloring [28], List Hamiltonian Cycle [31], List Coloring (even if parameterized by vertex cover) [19], General Factor [34], Generalized Satisfiability parameterized by the treewidth of the dual or incidence graph [35], Generalized Domination [12], Bounded Edge-Disjoint Paths [25]. All problems which are W-hard for treewidth are of course also W-hard for the more general clique-width. Additionally, in [22] it is shown that Graph Coloring, Hamiltonicity, Max Cut and Edge Dominating Set are W-hard parameterized by clique-width.

2 Definitions and Preliminaries

We assume that we are using the real-word RAM model. We use $\log(n)$ to denote the base-2 logarithm of n and $\ln(n)$ to denote the natural logarithm. In addition $\log_{(1+\delta)}(n)$ is the logarithm base-$(1+\delta)$, for $\delta > 0$. We have $\log_{(1+\delta)}(n) = \ln(n)/\ln(1+\delta)$. The function $\lfloor x \rfloor$, for $x \in \mathbb{R}$ denotes the maximum integer that is not larger than x.

We use boldface to denote vectors, for example $\boldsymbol{d}$. Sometimes a vector $\boldsymbol{s} \in S^k$ for S a set and $k \in \mathbb{N}$ will also be viewed as a function from $\{1, \ldots, k\}$ (or some other convenient set of size k) to S, and vice-versa. For a function $f : S_1 \to S_2$ we use $f^{-1}(u), u \in S_2$ to denote the set $\{v \in S_1 \mid f(v) = u\}$.

We will also use the following standard facts.

Lemma 1. *Let $x, \delta \in \mathbb{R}$. Then the following hold:*

1. $1 + x \leq e^x$
2. *If $x \in (0, \frac{1}{2})$ then $e^{x/2} \leq 1 + x$*
3. *If $x \in (-\frac{1}{2}, \frac{1}{2})$ then $e^x \leq 1 + x + x^2$*
4. *If $x \in (-\frac{1}{2}, \frac{1}{2})$ then $|\ln(1+x)| \geq \frac{|x|}{2}$*
5. *If $\delta \in (0, \frac{1}{2})$ and $\delta|x| \leq \frac{1}{2}$ then $|(1+\delta)^x - 1| \geq \frac{1}{4}\delta|x|$*

Proof. For item 1 one can consider the function $f(x) = e^x - x - 1$. This function has a global minimum at $x = 0$ (this can be established by looking at its derivative), thus $f(x) \geq f(0) = 0$. For

item 2 we can use the Taylor expansion $e^{x/2} = \sum_{i=0}^{\infty} \frac{(x/2)^i}{i!} \leq 1 + \frac{x}{2} + \sum_{i=2}^{\infty} \frac{x^i}{8} \leq 1 + \frac{x}{2} + \frac{x^2}{4} \leq 1 + x$, where we have used the fact that $x \leq \frac{1}{2}$.

For item 3 we again use the Taylor expansion $e^x = 1 + x + \sum_{i=2}^{\infty} \frac{x^i}{i!} \leq 1 + x + \sum_{i=2}^{\infty} \frac{|x|^i}{2} \leq 1 + x + x^2$, where in the last inequality we used the fact that $|x| \leq \frac{1}{2}$.

For item 4 if $x \geq 0$ this follows from item 2 by taking the natural logarithm of both sides. For $x \leq 0$ we can use the fact that since $|x| < 1$ we have $\ln(1 - |x|) = -\sum_{i=1}^{\infty} \frac{|x|^i}{i}$, therefore $|\ln(1 + x)| = |\ln(1 - |x|)| = \sum_{i=1}^{\infty} \frac{|x|^i}{i} \geq |x|$.

For item 5 first consider the case $x \geq 0$. We have $\ln(1 + \delta) \geq \frac{\delta}{2}$ (by item 2) so since $x \geq 0$ we have $x \ln(1 + \delta) \geq \frac{\delta x}{2} \Rightarrow (1 + \delta)^x \geq e^{\frac{\delta x}{2}} \geq 1 + \frac{\delta x}{2}$, where we used item 1. Thus, $(1 + \delta)^x - 1 \geq \frac{\delta x}{2} \Rightarrow |(1 + \delta)^x - 1| \geq \frac{\delta |x|}{2}$, since $x \geq 0$ and the item is proved in this case. For $x < 0$ we have $|(1 + \delta)^x - 1| = 1 - (1 + \delta)^x = \frac{(1+\delta)^{|x|} - 1}{(1+\delta)^{|x|}}$. Using our calculations from the $x \geq 0$ case we have $(1 + \delta)^{|x|} - 1 \geq \frac{\delta |x|}{2}$ so we get $|(1 + \delta)^x - 1| \geq \frac{\delta |x|}{2(1+\delta)^{|x|}} \geq \frac{\delta |x|}{2e^{\delta |x|}} \geq \frac{\delta |x|}{2\sqrt{e}} \geq \frac{\delta |x|}{4}$ where we have used that $1 + \delta \leq e^{\delta}$ (item 1) and $\delta |x| \leq \frac{1}{2}$. □

We assume the reader has some familiarity with standard definitions from parameterized complexity, such as the classes FPT and XP (see [21]). For a parameterized problem with input size n and parameter k an FPT Approximation Scheme (FPT-AS) is an algorithm which, given an error parameter $\epsilon > 0$ runs in time $f(k, \frac{1}{\epsilon})$ (that is, FPT time when parameterized by $k + \frac{1}{\epsilon}$) and produces a solution that is at most a multiplicative factor $(1 + \epsilon)$ from the optimal (see [30]). The problems we consider in this paper are parameterized by some graph width. We design (randomized) algorithms running in time $\left(\frac{\log n}{\epsilon}\right)^{O(k)} n^{O(1)}$. By standard facts in parameterized complexity theory such running times imply FPT algorithms.

Graph widths We use standard graph theoretic notation. For an undirected graph $G(V, E)$ and $S \subseteq V$ we denote by $G[X]$ the graph induced by X. We will use the standard notion of tree decomposition (for an introduction to this notion see the survey by Bodlaender and Koster [8]). Given a graph $G(V, E)$ a tree decomposition of G is a tree $T(I, F)$ such that every node $i \in I$ has associated with it a set $X_i \subseteq V$, called the bag of i. In addition, the following are satisfied: $\bigcup_{i \in I} X_i = V$; for all $(u, v) \in E$ there exists $i \in I$ such that $\{u, v\} \subseteq X_i$; and finally for all $u \in V$ the set $\{i \in I \mid u \in X_i\}$ is a connected sub-tree of T. The width of a tree decomposition is defined as $\max_{i \in I} |X_i| - 1$. The treewidth of a graph G is the minimum treewidth of a tree decomposition of G.

As is standard, when dealing with problems on graphs of bounded treewidth we will assume that a "nice" tree decomposition of the input graph is supplied with the input. In a nice tree decomposition the tree T is a rooted binary tree and each node i of the tree is of one four special types (see [8] for a definition).

We will also use the notion of clique-width (see [15, 18]). The set of graphs of clique-width w is the set of vertex-labelled graphs which can be constructed inductively using the following operations:

1. Introduce: $i(l)$, for $l \in \{1, \ldots, w\}$ is the graph consisting of a single vertex with label l.
2. Union: $\cup(G_1, G_2)$, for G_1, G_2 having clique-width w is the disjoint union of these two graphs.
3. Join: $\sigma(G, l_1, l_2)$, for G having clique-width w and $l_1, l_2 \in \{1, \ldots, w\}$ is the graph obtained from G by adding all possible edges from vertices with label l_1 to vertices with label l_2.

4. Rename: $\rho(G, l_1, l_2)$, for G having clique-width w and $l_1, l_2 \in \{1, \ldots, w\}$ is the graph obtained from G by changing the label of all vertices having label l_1 to l_2.

A clique-width expression of width w for $G(V, E)$ is a recipe for constructing a w-labelled graph isomorphic to G. More formally, a clique-width expression is a rooted binary tree such that each node has one of four possible types, corresponding to the operations described above. In addition, all leaves are Introduce nodes, each Introduce node has a label $\in \{1, \ldots, w\}$ associated with it, and each Join or Rename node has two labels in $\{1, \ldots, w\}$ associated with it. For each node i the graph G_i is defined as the graph obtained by applying the operation of node i to the graph (or graphs) associated with its child (or children). All graphs G_i are subgraphs of G and for all leaves with label l we define their associated graph to be $\eta(l)$.

Again, as is customary, when dealing with a problem parameterized by clique-width we will assume that a clique-width expression of the input graph is supplied with the input. We can also assume without loss of generality that the given clique-width expression has some nice properties. In particular, whenever the operation $\sigma(G_i, l_1, l_2)$ is used we can assume that there are no edges between vertices with labels l_1, l_2, since otherwise we can edit the clique-width expression up to this point to remove the Join operations that produced such edges and this does not affect the final graph.

3 Approximate Addition Trees

In this section we describe an abstract model of computation which one may naturally call Addition Trees. In such a Tree each node calculates a value that is the sum of the values of its children. We also define an Approximate version of these trees, where each node *probabilistically* rounds calculated values to integer powers of $(1 + \delta)$, for some parameter $\delta > 0$. These trees capture the rounding scheme that will be the heart of the algorithms of the next section. Our goal is to prove that the values of Approximate and Exact Addition Trees are almost always very close, even if δ is not too small (we want $\delta = \Omega(1/\log^c n)$). We require δ to be in this range, because in the end the algorithms of the next section will run in time roughly $(\log n/\delta)^w$. Thus, if we allow δ to become polynomial in n (which would make this section easy), we will get algorithms as slow as the trivial exact ones.

Intuitively, there are two extreme cases to consider here. First, if a tree is balanced (that is, it has logarithmic height), it is not hard to establish that rounding errors cannot pile up too badly (Theorem 3). Somewhat surprisingly, this easy case is already sufficient to obtain several non-trivial algorithmic results, because tree decompositions can always be balanced reasonably well (more details are given in the next section). However, to obtain the more interesting results of this paper we need to deal with clique-width, where the input decomposition cannot in general be balanced. Therefore, we have to deal with general Addition Trees.

Our proof strategy then is to move on to a second extreme case: caterpillars. Here the height of the tree is large, but we know that one operand of each addition has no previously accumulated error. Despite this, this is actually a pretty hard case. To see why, intuitively one can think of the accumulated error at each level of the tree as a random variable, since the rounding performed on each step is randomized. The error has some probability of increasing and some of decreasing, depending on how we round, but it changes by at most a factor of $(1 + \delta)$ in each step. So, if we look at its logarithm (with base $(1 + \delta)$) it can (randomly) increase or decrease by at most 1. Thus, the process we are trying to analyze is akin to a memoryless random walk on the real line. We

want to prove that the end result of the walk after n steps is with high probability contained in an area of size only roughly $1/\delta = \text{poly}(\log n)$. Such a statement would be, however, false if the walk was completely unbiased, so we cannot rely on standard tools, such as Chernoff bounds or the Azuma inequality, because the result they give is too weak (they give concentration in an area of size roughly $\sqrt{n}$). Instead, we need to use moment-generating functions to derive a problem-specific concentration bound that takes into account our algorithm's tendency to "self-correct". Because of this special tendency (Lemma 3), our random walk is much more strongly concentrated around its expectation than usual random walks.

Thus, eventually we establish (in Lemma 5) that the approximation error is small in the caterpillar case, with high probability. Once this has been shown we can extend the same ideas to prove a sufficiently good approximation theorem for general trees by performing induction on the "balanced height" of the tree (Theorem 4). Roughly speaking, the idea is that in any node of an arbitrary tree either both children have roughly the same accumulated error (in which case the node is balanced) or one has potentially much higher error (in which case the proof is similar to that for caterpillars). So, combining the ideas of the two cases we can handle arbitrary trees.

We remark that the only parts of this section needed to follow the results of the next one are Definitions 1,2 and Theorems 3,4. Let us now proceed to give full details.

For the following definition recall that a rooted binary tree is full if all non-leaf nodes have exactly two children.

Definition 1. *An Addition Tree is a full rooted binary tree T where we associate with each leaf l a non-negative integer* input x_l *and with each node v a non-negative integer* value y_v*. The inputs are given with T. The value of each node is calculated as follows:*

1. *For each leaf l we set $y_l := x_l$*
2. *For each internal node v with two children u_1, u_2 whose values have already been calculated we set $y_v := y_{u_1} + y_{u_2}$.*

Let us also define an approximate version of this model.

Definition 2. *An Approximate Addition Tree with parameter δ is a full rooted binary tree T where we associate with each leaf l a non-negative integer* input x_l *and with each node v a non-negative* approximate value z_v*. The approximate value of each node is calculated as follows:*

1. *For each leaf l we set $z_l := x_l$*
2. *For each internal node v with two children u_1, u_2 we set $z_v := z_{u_1} \oplus z_{u_2}$, where the $\oplus$ operation is defined below.*

Let $a_v := z_{u_1} + z_{u_2}$. We will call a_v the initial approximate value *of v.*

We use $\oplus$ to denote the following operation: for two non-negative numbers x_1, x_2 we define $x_1 \oplus x_2 := 0$ if $x_1 = x_2 = 0$. Otherwise, select a real number $r \in (0, 1)$ uniformly at random and set $x_1 \oplus x_2 := (1+\delta)^{\lfloor \log_{(1+\delta)}(x_1+x_2)+r \rfloor}$.

The motivation behind this definition is that the number of possible values that can be stored in a node is much smaller than in an exact tree. In particular, note that whenever z_v is non-zero, it must be an integer power of $(1 + \delta)$. If the maximum value calculated by the exact tree is at most polynomial in n and $\delta = \Omega(1/\log^c n)$ then there are at most $\text{poly}(\log n)$ many different values

that may appear in an approximate tree. Using this we will be able to obtain smaller dynamic programming tables in the next section. First though, we have to show that the approximate values are close to the correct ones.

Let us now give the definition of approximation error by which we will measure the progress of our algorithm. Since it is not hard to see that for any node v for which $y_v = 0$ an Approximate Addition Tree will also have $z_v = 0$, we are only concerned with the approximation error for nodes where $y_v \neq 0$. Therefore, in the remainder we will implicitly assume that we are talking about a tree where for all v, $y_v > 0$, because sub-trees with value 0 can be ignored.

Definition 3. *Let v be a node of an Addition Tree, y_v its (positive) value and z_v its approximate value calculated if we view the tree as an Approximate Addition Tree. Then the error λ_v is defined as $\lambda_v := \log_{(1+\delta)} \frac{z_v}{y_v}$.*

Note that λ_v and z_v are random variables (they depend on the random rounding choices made during the computation), while y_v is fully specified once the inputs of the tree are fixed. Informally, λ_v measures how many "$(1+\delta)$ intervals" off our approximation is from the correct interval.

Before we go on, let us make an easy observation that will be sufficient to handle an important special case.

Theorem 3. *If an Approximate Addition Tree has maximum depth h then for all nodes v we always have $|\lambda_v| \leq h + 1$. Therefore, if $\delta < \frac{\epsilon}{2h}$ then for all v we have $\max\{\frac{z_v}{y_v}, \frac{y_v}{z_v}\} < 1 + \epsilon$.*

Proof. The proof is simple and proceeds by induction on h. For trees of height 0 (that is, isolated nodes), z_v has the correct interval, so $|\lambda_v| \leq 1$. For the inductive step observe that when adding two values the maximum absolute relative error cannot increase by more than a factor of $1 + \delta$. □

As a consequence of Theorem 3 we get that in trees of height $O(\log n)$ we can set $\delta = \Theta(1/\log n)$ and get error at most $1 + \epsilon$ everywhere *with probability 1*. Thus, such balanced trees are an easy case which is already solved, even without the use of randomization. As mentioned though, we also need to handle the much more complicated general case.

The main intuitive observation that we will use to give an approximation guarantee can be summarized as follows: the process by which the initial approximation a_v is calculated is "self-correcting", while the rounding step that follows it is unbiased. Therefore, the whole process tends to self-correct. More precisely, we will show that if $\lambda_{u_1}, \lambda_{u_2}$ are the errors of the two children of a node, then the error for a_v is somewhere between the two. This means that unless the two errors are exactly equal, the maximum of the two errors will have a tendency to decrease. This intuition will be made more precise in Lemma 3.

First, we need to introduce a definition and an auxiliary lemma.

Definition 4. *Let v be an internal node of an Approximate Addition Tree, and a_v its initial approximate value as in Definition 2. We define $p_v := \log_{(1+\delta)}(a_v) - \lfloor \log_{(1+\delta)}(a_v) \rfloor$.*

Of course p_v is again a random variable, since it depends on a_v. However, note that p_v, as defined, is exactly equal to the probability of "rounding up", that is, the probability (over the choice of $r \in (0, 1)$) that $\lfloor \log_{(1+\delta)}(a_v) + r \rfloor > \lfloor \log_{(1+\delta)}(a_v) \rfloor$. To see this, observe that we round up if and only if $\log_{(1+\delta)}(a_v) + r \geq \lfloor \log_{(1+\delta)}(a_v) \rfloor + 1 \Leftrightarrow r \geq 1 - p_v$ which is an event that has probability p_v, since r is selected uniformly at random and $p_v \in [0, 1]$.

Lemma 2. *Consider an Approximate Addition Tree with parameter $\delta < \frac{1}{2}$ and let v be an internal node with two children u_1, u_2. Then $z_{u_2} \geq \frac{1}{2}\delta p_v z_{u_1}$.*

Proof. We have $p_v = \log_{(1+\delta)}(a_v) - \lfloor \log_{(1+\delta)}(a_v) \rfloor$, but $a_v = z_{u_1} + z_{u_2}$ therefore $\lfloor \log_{(1+\delta)}(a_v) \rfloor \geq \lfloor \log_{(1+\delta)}(z_{u_1}) \rfloor = \log_{(1+\delta)}(z_{u_1})$ where we used the fact that z_{u_1} is an integer power of $(1+\delta)$.

We now have $p_v \leq \log_{(1+\delta)}(z_{u_1}+z_{u_2}) - \log_{(1+\delta)}(z_{u_1}) = \log_{(1+\delta)}\left(1 + \frac{z_{u_2}}{z_{u_1}}\right)$. Therefore, to establish the lemma it is sufficient to show that $\frac{z_{u_2}}{z_{u_1}} \geq \frac{1}{2}\delta \log_{(1+\delta)}\left(1 + \frac{z_{u_2}}{z_{u_1}}\right)$. To ease notation let $\gamma = \frac{z_{u_2}}{z_{u_1}}$.

We have $\gamma \geq \frac{\delta}{2}\log_{(1+\delta)}(1+\gamma) \Leftrightarrow \gamma \ln(1+\delta) \geq \frac{\delta}{2}\ln(1+\gamma) \Leftrightarrow (1+\delta)^{\gamma} \geq (1+\gamma)^{\frac{\delta}{2}}$. Now, from Lemma 1-(2) and $\delta < \frac{1}{2}$ we have $(1+\delta)^{\gamma} \geq e^{\frac{\delta\gamma}{2}}$, while from Lemma 1-(1) we have $(1+\gamma)^{\frac{\delta}{2}} \leq e^{\frac{\delta\gamma}{2}}$. The result follows. □

We are now ready to give the main self-correction lemma.

Lemma 3. *Consider an Approximate Addition Tree with parameter $\delta < \frac{1}{2}$. Let v be an internal node with two children u_1, u_2. If $\max\{|\lambda_{u_1}|, |\lambda_{u_2}|\} < \frac{1}{4\delta}$ then $|\log_{(1+\delta)} \frac{a_v}{y_{u_1}+y_{u_2}}| \leq \max\{|\lambda_{u_1}|, |\lambda_{u_2}|\} - \frac{1}{20}\delta p_v |\lambda_{u_1} - \lambda_{u_2}|$.*

Proof. Informally, what this lemma states is that, before the rounding step is applied, the absolute value of the error will decrease when compared with the maximum error up to this point. In fact, the decrease will be proportional to the absolute difference of the two previous errors.

Since we have not assumed an ordering on the children we can assume without loss of generality that $|\lambda_{u_1}| \geq |\lambda_{u_2}|$. From the definitions we have $a_v = (1+\delta)^{\lambda_{u_1}} y_{u_1} + (1+\delta)^{\lambda_{u_2}} y_{u_2}$. If $\lambda_{u_1} = \lambda_{u_2}$ then the inequality of the lemma is true, so assume that $|\lambda_{u_1}| > |\lambda_{u_2}|$.

We now have $\log_{(1+\delta)} \frac{a_v}{y_{u_1}+y_{u_2}} = \lambda_{u_1} + \log_{(1+\delta)} \frac{y_{u_1} + (1+\delta)^{\lambda_{u_2}-\lambda_{u_1}} y_{u_2}}{y_{u_1}+y_{u_2}}$. Taking into account that $|\lambda_{u_1}| > |\lambda_{u_2}|$ it is not hard to see that the second term is positive if and only if the first term is negative. Therefore, $\left|\log_{(1+\delta)} \frac{a_v}{y_{u_1}+y_{u_2}}\right| = |\lambda_{u_1}| - \left|\log_{(1+\delta)} \frac{y_{u_1} + (1+\delta)^{\lambda_{u_2}-\lambda_{u_1}} y_{u_2}}{y_{u_1}+y_{u_2}}\right|$.

Now consider the term $\left|\log_{(1+\delta)} \frac{y_{u_1} + (1+\delta)^{\lambda_{u_2}-\lambda_{u_1}} y_{u_2}}{y_{u_1}+y_{u_2}}\right|$. We only need to show that this term is at least as large as $\frac{1}{20}\delta p_v |\lambda_{u_1} - \lambda_{u_2}|$ to establish the lemma. Observe that this term is an increasing function of y_{u_2}. Therefore, we can use a lower bound on y_{u_2} to give a lower bound on this term. But by Lemma 2 we have $z_{u_2} \geq \frac{1}{2}\delta p_v z_{u_1} \Leftrightarrow y_{u_2} \geq \frac{1}{2}\delta p_v (1+\delta)^{\lambda_{u_1}-\lambda_{u_2}} y_{u_1}$. We thus get:

$$\left|\log_{(1+\delta)} \frac{y_{u_1} + (1+\delta)^{\lambda_{u_2}-\lambda_{u_1}} y_{u_2}}{y_{u_1}+y_{u_2}}\right| \geq \left|\log_{(1+\delta)} \frac{y_{u_1} + \frac{1}{2}\delta p_v y_{u_1}}{y_{u_1} + \frac{1}{2}\delta p_v (1+\delta)^{\lambda_{u_1}-\lambda_{u_2}} y_{u_1}}\right| = \tag{1}$$

$$= \left|\log_{(1+\delta)} \frac{1 + \frac{1}{2}(1+\delta)^{\lambda_{u_1}-\lambda_{u_2}}\delta p_v}{1 + \frac{1}{2}\delta p_v}\right| = \tag{2}$$

$$= \frac{\left|\ln\left(1 + \frac{\frac{1}{2}\delta p_v\left((1+\delta)^{\lambda_{u_1}-\lambda_{u_2}} - 1\right)}{1+\frac{1}{2}\delta p_v}\right)\right|}{\ln(1+\delta)} \geq \tag{3}$$

$$\geq \frac{1}{\delta}\left|\ln\left(1 + \frac{\frac{1}{2}\delta p_v\left((1+\delta)^{\lambda_{u_1}-\lambda_{u_2}} - 1\right)}{1 + \frac{1}{2}\delta p_v}\right)\right| \geq \tag{4}$$

$$\geq \frac{p_v \left|(1+\delta)^{\lambda_{u_1}-\lambda_{u_2}} - 1\right|}{4 + 2\delta p_v} \geq \tag{5}$$

$$\geq \frac{1}{4}\frac{\delta p_v |\lambda_{u_1} - \lambda_{u_2}|}{4 + 2\delta p_v} \geq \tag{6}$$

$$\geq \frac{1}{20}\delta p_v |\lambda_{u_1} - \lambda_{u_2}| \tag{7}$$

To go from (1) to (2) we used the fact that $|\log(x)| = |\log(1/x)|$, while for (3) we simply changed the base of the logarithm and performed some calculations. For (4) we use the fact that $\ln(1+\delta) \leq \delta$ which can be inferred from Lemma 1-(1). To get (5) we use Lemma 1-(4). To see that Lemma 1-(4) applies observe that $(1+\delta)^{\lambda_{u_1}-\lambda_{u_2}} \leq (1+\delta)^{\frac{1}{2\delta}} \leq \sqrt{e} \leq 2$. To go from (5) to (6) we use Lemma 1-(5) and the fact that $|\lambda_{u_1} - \lambda_{u_2}| \in (-\frac{1}{2\delta}, \frac{1}{2\delta})$. Finally, (7) follows from the fact that $\delta < \frac{1}{2}$ and $p_v \leq 1$. □

Lemma 3 will be our main tool in proving that with high probability the values of an Approximate Addition Tree are not too far from those of the corresponding exact tree. We will proceed by induction, starting from a simple case of path-like trees (caterpillars). We need the following definition:

Definition 5. *Let T be a rooted full binary tree. The* balanced height *of a node v, denoted $\mathrm{h_b}(v)$ is defined as follows:*

1. *If v is a leaf then $\mathrm{h_b}(v) = 0$.*
2. *If v is an internal node with children u_1, u_2 and $\mathrm{h_b}(u_1) > \mathrm{h_b}(u_2)$ then $\mathrm{h_b}(v) = \mathrm{h_b}(u_1)$.*
3. *If v is an internal node with children u_1, u_2 and $\mathrm{h_b}(u_1) = \mathrm{h_b}(u_2)$ then $\mathrm{h_b}(v) = \mathrm{h_b}(v_1) + 1$.*

Lemma 4. *In any full binary tree with n nodes and root r we have $n \geq 2^{\mathrm{h_b}(r)}$.*

Proof. The proof proceeds by induction on n. The lemma is trivial for $n = 1$. For a larger tree, consider the two trees rooted at children of the root. If they have the same balanced height then by induction they both have at least $2^{\mathrm{h_b}(r)-1}$ nodes, which means the whole tree has at least $2^{\mathrm{h_b}(r)}$ nodes. Otherwise, one has balanced height $\mathrm{h_b}(r)$ and by induction that sub-tree contains at least $2^{\mathrm{h_b}(r)}$ nodes. □

We will proceed inductively to prove a bound on the probability of a large error occurring during the computation of an Approximate Addition Tree. Our bound will depend on δ, n and the balanced height of the root of the tree.

To begin, observe that the case $\mathrm{h_b}(r) = 0$ is trivial, as this can only occur if the tree contains only one node. So the first interesting case is $\mathrm{h_b}(r) = 1$, which happens if the rooted tree is made up of a single path with a leaf attached to each internal node and the root.

In the remainder we will assume we are dealing with an Approximate Addition Tree with n nodes, root node r and parameter $\delta \in (0, \frac{1}{2})$. We first establish the following lemma:

Lemma 5. *If $\mathrm{h_b}(r) \leq 1$ then for all $\lambda \in (\frac{2}{\sqrt{\delta}}, \frac{1}{4\delta})$ we have $\mathbf{Pr}\left[\exists v \in T : \ |\lambda_v| > \lambda\right] \leq 2n^2 e^{-\frac{\lambda\sqrt{\delta}}{20}}$.*

Proof. We will first bound the probability that a certain node is the first to have absolute error greater than λ, that is, the probability that it has such an error even though all its descendants

did not. Then we will take a union bound over all nodes. Observe that it is sufficient to consider internal nodes, since the errors for leaves are by definition 0.

Consider an arbitrary internal node v. We will bound the conditional probability that its absolute error is larger than λ, given the fact that all its descendants have smaller absolute error. (To ease presentation we will omit this conditioning, which should be read implicitly in the remainder of this proof). Let $t \in (0, \frac{1}{2})$ be a parameter we will set later. We have

$$\mathbf{Pr}\left[|\lambda_v| > \lambda\right] = \mathbf{Pr}\left[e^{t|\lambda_v|} > e^{t\lambda}\right] \leq \frac{\mathbf{E}\left[e^{t|\lambda_v|}\right]}{e^{t\lambda}}$$

where for the last step we applied Markov's inequality, since we are dealing with a non-negative random variable.

Let us now try to bound the expectation which appears on the right-hand side. The node v has two children u_1, u_2, one of which must be a leaf, since the tree has balanced height 1. Without loss of generality let u_2 be the leaf, and therefore $\lambda_{u_2} = 0$. By applying Lemma 3 we get that $|\log_{(1+\delta)}(a_v/y_v)| \leq |\lambda_{u_1}| - \frac{1}{10}\delta p_v |\lambda_{u_1}|$, where we are using the fact that we are conditioning under the event that $|\lambda_{u_1}| < \lambda \leq \frac{1}{4\delta}$.

Let us now look at the rounding step. Recall that we denote by p_v the probability of "rounding up". We have that $\log_{(1+\delta)}(z_v) = \log_{(1+\delta)}(a_v) + r_v$, where r_v is a random variable that depends on the choice of $r \in (0,1)$. In particular, r_v has the following distribution: with probability p_v we have $\log_{(1+\delta)}(z_v) = \lfloor \log_{(1+\delta)}(a_v) \rfloor + 1$ and therefore $r_v = 1 - p_v$. On the other hand, with probability $1 - p_v$ we have $\log_{(1+\delta)}(z_v) = \lfloor \log_{(1+\delta)}(a_v) \rfloor$ and therefore $r_v = -p_v$.

Rewriting we get $\lambda_v = \log_{(1+\delta)} \frac{z_v}{y_v} = \log_{(1+\delta)} \frac{a_v}{y_v} + r_v \Rightarrow |\lambda_v| = |\log_{(1+\delta)} \frac{a_v}{y_v} + r_v| = |\log_{(1+\delta)} \frac{a_v}{y_v}| + r'_v \leq |\lambda_{u_1}| - \frac{1}{10}\delta p_v |\lambda_{u_1}| + r'_v$ where we define r'_v as follows: $r'_v := r_v$ if $a_v \geq y_v$ and $r'_v := -r_v$ otherwise. Let us now bound the change caused in the expectation by the rounding step.

Claim. For $t > 0$ we have $\mathbf{E}\left[e^{tr'_v} \mid p_v\right] \leq e^{t^2 p_v}$

Proof. In case $r'_v = r_v$ we have $\mathbf{E}\left[e^{tr_v} | p_v\right] = p_v e^{t(1-p_v)} + (1-p_v)e^{-tp_v} = e^{-tp_v}(1 - p_v + p_v e^t) \leq e^{-tp_v} e^{-p_v + p_v e^t} \leq e^{-tp_v - p_v + p_v + tp_v + t^2 p_v} = e^{t^2 p_v}$, where we used Lemma 1-(1) and Lemma 1-(3).

In case $r'_v = -r_v$ we have $\mathbf{E}\left[e^{-tr_v} | p_v\right] = p_v e^{-t(1-p_v)} + (1-p_v)e^{tp_v} = e^{tp_v}(1 - p_v + p_v e^{-t}) \leq e^{tp_v} e^{-p_v + p_v e^{-t}} \leq e^{tp_v - p_v + p_v - tp_v + t^2 p_v} = e^{t^2 p_v}$ where again we used Lemma 1. □

We now have:

$$\mathbf{E}\left[e^{t|\lambda_v|}\right] = \mathbf{E}\left[e^{t|\lambda_v|} \mid |\lambda_{u_1}| \geq \frac{1}{\sqrt{\delta}}\right] \mathbf{Pr}\left[|\lambda_{u_1}| \geq \frac{1}{\sqrt{\delta}}\right] + \mathbf{E}\left[e^{t|\lambda_v|} \mid |\lambda_{u_1}| < \frac{1}{\sqrt{\delta}}\right] \mathbf{Pr}\left[|\lambda_{u_1}| < \frac{1}{\sqrt{\delta}}\right] \quad (8)$$

The second term of the right-hand-side of (8) can be upper-bounded by $e^{t + t\sqrt{1/\delta}}$, since $|\lambda_v| \leq |\lambda_{u_1}| + 1$. We now seek to find a value of t such that the first term is upper-bounded by $\mathbf{E}\left[e^{t|\lambda_{u_1}|}\right]$.

$$\mathbf{E}\left[e^{t|\lambda_v|} \mid |\lambda_{u_1}| \geq \sqrt{1/\delta}\right] \leq \mathbf{E}\left[e^{t|\lambda_{u_1}| - t\frac{\delta}{20}p_v|\lambda_{u_1}| + tr'_v} \mid |\lambda_{u_1}| \geq \sqrt{1/\delta}\right] \leq \quad (9)$$

$$\leq \mathbf{E}\left[\mathbf{E}\left[e^{t|\lambda_{u_1}| - t\frac{\sqrt{\delta}}{20}p_v + tr'_v} \mid p_v\right] \mid |\lambda_{u_1}| \geq \sqrt{1/\delta}\right] \leq \quad (10)$$

$$\leq \mathbf{E}\left[e^{t|\lambda_{u_1}| - t\frac{\sqrt{\delta}}{20}p_v + t^2 p_v} \mid |\lambda_{u_1}| \geq \sqrt{1/\delta}\right] \quad (11)$$

To obtain (9) we used Lemma 3. Then to obtain (10) we used the fact that we are conditioning on $|\lambda_{u_1}| \geq \sqrt{1/\delta}$ to replace λ_{u_1} in one term of the exponent. We also used standard properties of conditional expectations to remove the dependence on r'_v, which using the claim gives (11). We can now set $t = \frac{\sqrt{\delta}}{20}$ (so we do indeed have $t \in (0, \frac{1}{2})$ as promised) and we get

$$\begin{aligned}\mathbf{E}\left[e^{\frac{\sqrt{\delta}}{20}|\lambda_v|}\right] &\leq \mathbf{E}\left[e^{\frac{\sqrt{\delta}}{20}|\lambda_{u_1}|} \mid |\lambda_{u_1}| \geq \sqrt{1/\delta}\right] \mathbf{Pr}\left[|\lambda_{u_1}| \geq \sqrt{1/\delta}\right] + e^{\frac{\sqrt{\delta}}{20}+\frac{1}{20}} \leq \\ &\leq \mathbf{E}\left[e^{\frac{\sqrt{\delta}}{20}|\lambda_{u_1}|}\right] + 2\end{aligned}$$

Since we have established the above for any internal node v, it now follows by induction than for any node v we have

$$\mathbf{E}\left[e^{\frac{\sqrt{\delta}}{20}|\lambda_v|}\right] \leq 2n$$

Using this fact, we can now conclude that the probability that an arbitrary internal node v is the first to have absolute error greater than λ is at most $2ne^{-\frac{\lambda\sqrt{\delta}}{20}}$. Therefore, by union bound, the probability that some node has absolute error at least λ is at most $2n^2e^{-\frac{\lambda\sqrt{\delta}}{20}}$, as claimed. □

We are now ready to extend the previous lemma to trees of arbitrary balanced height.

Lemma 6. *Let T be an Approximate Addition Tree with root r and $\mathrm{h_b}(r) = h$. Then for all $\lambda \in (\frac{2}{\sqrt{\delta}}, \frac{1}{4\delta})$ we have $\mathbf{Pr}\left[\exists v \in T \ : \ |\lambda_v| > \lambda h\right] \leq (h+1)n^{h+1}e^{-\frac{\lambda\sqrt{\delta}}{20}}$.*

Proof. The proof proceeds by induction on h. The base case for $h = 1$ is exactly Lemma 5, which we have already established. Now, assume that the result has been proved for trees of balanced height up to $h-1$. Let B_{h-1} be the event that there exists some node u with $\mathrm{h_b}(u) \leq h-1$ such that $|\lambda_u| > (h-1)\lambda$. Then we have:

$$\mathbf{Pr}\left[\exists v \in T \ : \ |\lambda_v| > \lambda h\right] \leq \mathbf{Pr}\left[B_{h-1}\right] + \mathbf{Pr}\left[\exists v \in T \ : \ |\lambda_v| > \lambda h \mid \neg B_{h-1}\right]$$

Let us bound the two terms separately. For the first term, for every node $u \in T$ with $\mathrm{h_b}(u) \leq h-1$ we consider the sub-tree rooted at u and containing all its descendants. By the inductive hypothesis in a rooted tree of balanced height at most $h-1$ the probability that there exists a vertex u with $|\lambda_u| > (h-1)\lambda$ is at most $hn^h e^{-\frac{\lambda\sqrt{\delta}}{20}}$. There are at most n such trees considered, so by union bound $\mathbf{Pr}\left[B_{h-1}\right] \leq hn^{h+1}e^{-\frac{\lambda\sqrt{\delta}}{20}}$.

For the second term, the argument is similar to that of the proof of Lemma 5. Consider an arbitrary node v of the tree such that $\mathrm{h_b}(v) = h$. We will bound the probability that this is the first node with absolute error greater than λh, given that the absolute errors of all its descendants with the same balanced height are strictly smaller than λh. This time, we are also conditioning on the event $\neg B_{h-1}$, so its descendants with smaller balanced height have error at most $(h-1)\lambda$.

Observe that any node v of balanced height h has at most one child of balanced height h (otherwise the balanced height of the tree would be at least $h+1$). If both children of v have balanced height $h-1$ and the event $\neg B_{h-1}$ is true, then $|\lambda_v| \leq (h-1)\lambda + 1 < \lambda h$ with probability 1. So the interesting case is when exactly one child of v has balanced height h.

Let u_1, u_2 be the two children of v with $\mathrm{h_b}(u_1) = h$ and $\mathrm{h_b}(u_2) < h$. Again, $t \in (0, \frac{1}{2})$ is a parameter to be fixed later and we are conditioning over the events that all descendants of v have absolute error at most λh and $\neg B_{h+1}$ is true.

$$\mathbf{Pr}\left[|\lambda_v| > \lambda h\right] = \mathbf{Pr}\left[e^{t|\lambda_v|} > e^{t\lambda h}\right] \leq \frac{\mathbf{E}\left[e^{t|\lambda_v|}\right]}{e^{t\lambda h}}$$

We now need to bound the expectation which appears on the right-hand side. Similarly to Equation (8) of Lemma 5 we have:

$$\begin{aligned}\mathbf{E}\left[e^{t|\lambda_v|}\right] = \mathbf{E}\left[e^{t|\lambda_v|} \mid |\lambda_{u_1}| \geq (h-1)\lambda + \sqrt{1/\delta}\right]\mathbf{Pr}\left[|\lambda_{u_1}| \geq (h-1)\lambda + \sqrt{1/\delta}\right] \\ + \mathbf{E}\left[e^{t|\lambda_v|} \mid |\lambda_{u_1}| < (h-1)\lambda + \sqrt{1/\delta}\right]\mathbf{Pr}\left[|\lambda_{u_1}| < (h-1)\lambda + \sqrt{1/\delta}\right] \qquad (12)\end{aligned}$$

By using the fact that $\neg B_{h-1}$ is true we know that if $|\lambda_{u_1}| > (h-1)\lambda + \frac{1}{\sqrt{\delta}}$ then the difference $|\lambda_{u_1} - \lambda_{u_2}| \geq \frac{1}{\sqrt{\delta}}$. Thus, we can upper-bound the first term by $\mathbf{E}\left[e^{t|\lambda_{u_1}|}\right]$ in exactly the same way as in Lemma 5 by setting $t = \frac{\sqrt{\delta}}{20}$. The second term is at most $e^{t((h-1)\lambda + \frac{1}{\sqrt{\delta}} + 1)}$. We can now use the same argument as in Lemma 5 to get

$$\mathbf{E}\left[e^{t|\lambda_v|}\right] \leq n e^{\frac{\sqrt{\delta}}{20}((h-1)\lambda + \frac{1}{\sqrt{\delta}} + 1)} \Rightarrow \mathbf{Pr}\left[|\lambda_v| > \lambda h\right] \leq 2ne^{-\frac{\lambda\sqrt{\delta}}{20}}$$

The above is an upper bound on the probability that an arbitrary node v of balanced height h is the first to have absolute error more than λh, assuming all nodes of balanced height at most $h-1$ have error at most $(h-1)\lambda$. By union bound, the probability that any node of balanced height h has error more than λh (again conditioned on $\neg B_{h-1}$) is at most $2n^2 e^{-\frac{\lambda\sqrt{\delta}}{20}}$. Summing with the upper bound we have on the probability of B_{h-1} we have that the probability of any node having absolute error more than λh is at most $hn^{h+1}e^{-\frac{\lambda\sqrt{\delta}}{20}} + 2n^2 e^{-\frac{\lambda\sqrt{\delta}}{20}} < (h+1)n^{h+1}e^{-\frac{\lambda\sqrt{\delta}}{20}}$ and the result follows. □

We are now ready to prove the main theorem of this section

Theorem 4. *Let T be an Approximate Addition Tree on n nodes with parameter $\delta \in (0, \frac{1}{2})$. There exists a fixed constant $C > 0$ such that for all $\epsilon \in (0, \frac{1}{8})$ and sufficiently large n, if $\delta < \frac{\epsilon^2}{C \log^6 n}$ we have*

$$\mathbf{Pr}\left[\exists v \in T : \max\{\frac{z_v}{y_v}, \frac{y_v}{z_v}\} > 1 + \epsilon\right] \leq n^{-\log n}$$

Proof. The theorem follows from Lemma 6 as follows. First, note that $|\lambda_v| > \lambda h \Leftrightarrow \max\{\frac{z_v}{y_v}, \frac{y_v}{z_v}\} > (1+\delta)^{\lambda h}$. By Lemma 1 $(1+\delta)^{\lambda h} \geq e^{\frac{\delta\lambda h}{2}} \geq 1 + \delta\lambda h/2$.

Now we set $\lambda = \frac{2\epsilon}{\delta h}$. Notice that $\frac{2\epsilon}{\delta h} < \frac{1}{4\delta}$ because $h \geq 1$ (otherwise the tree is trivial). Also, $\frac{2\epsilon}{\delta h} > \frac{2}{\sqrt{\delta}} \Leftrightarrow \epsilon > h\sqrt{\delta}$ which holds because by Lemma 4 we have $h \leq \log n$. Therefore, we can apply Lemma 6 for the chosen λ. We get

$$\mathbf{Pr}\left[\exists v \in T : \max\{\frac{z_v}{y_v}, \frac{y_v}{z_v}\} > 1+\epsilon\right] \leq (h+1)n^{h+1}e^{-\frac{\epsilon}{10h\sqrt{\delta}}}$$

The result follows by noting that the bound on the right is an increasing function of h and $h \leq \log n$. □

4 Approximation Schemes

We are now ready to use the results of the previous section to design some approximation algorithms. As mentioned, we only need Definitions 1,2 (including the definition of the $\oplus$ operation) and Theorems 3, 4. Let us first describe the general technique.

In all problems we will assume we are supplied either a tree decomposition of width w or a clique-width expression with w labels. An important property of all the problems we consider is that they admit an exact dynamic program that takes time (roughly) n^w. These dynamic programs calculate a set of w integers on each node of the decomposition by adding previously calculated values or values read from the graph. The idea is to reuse this dynamic program, but round all values to integer powers of $(1+\delta)$ and perform all additions with the $\oplus$ operation. Note that, we are in fact able to also handle some other auxiliary operations besides addition (such as comparisons). It will, however, be important that our dynamic programs avoid using subtractions (because then we lose the approximation guarantee) and part of our effort is devoted to achieving this.

How do we analyze the approximation ratio of such an algorithm? As usual, each node of the tree decomposition or the clique-width expression represents a subgraph of the original graph. For each problem, we have a notion of partial solution confined to a subgraph. We want to show that for each partial solution there exists (with high probability) an entry in the approximate dynamic programming table that matches its value and vice-versa. This is where Addition Trees become useful. For each partial solution value we define inductively an Addition Tree T whose root calculates that value. We then inductively establish that the approximate dynamic programming table contains some value that *follows the same distribution* as the result of T, if T is viewed as an Approximate Addition Tree. Thus, the approximate dynamic programming table contains (with high probability) an almost correct value. To simplify things, by tweaking the parameters appropriately we can make the probability high enough that *all* the values of the approximate table are close to being correct, assuming w is not too large. Observe that this allows the analysis to be performed using essentially the same inductive reasoning as for standard (exact) dynamic programming algorithms.

What remains to say is what values we select for the parameter δ. Here we have two choices. In the general case, we set δ to the value dictated by Theorem 4. This works quite well essentially all the time, as we can guarantee that with high probability the Addition Trees we consider during the analysis of our algorithm will have almost correct values. We can thus obtain randomized approximation schemes for both clique-width and treewidth with the promised running time of roughly $(\log n/\epsilon)^{O(w)}$. This is the general technique we consider to be the main contribution of this paper.

Nevertheless, as mentioned there exists an important special case where things can become simpler. It is known that for any graph of treewidth w there exists a tree decomposition of width $O(w)$ and only logarithmic (in n) height [7]. Thus, another approach available to us is to use this theorem to first balance the decomposition and then rely on Theorem 3, instead of the more general

Theorem 4. Unfortunately, this does not speed up our algorithms significantly (because the larger δ given in Theorem 3 is almost out-weighed by the blow-up in the width of the decomposition). Importantly, it is impossible to apply this approach to clique-width, because similar balancing results blow up the number of labels used exponentially ([14]), which would result in an unreasonable running time of roughly $(\log n)^{2^w}$ (and this is likely to be unavoidable). More generally, relying on a treewidth-specific balancing theorem detracts from the bigger point of developing a general technique, since there are many graph widths for which balancing would not even make sense (e.g. pathwidth). Despite these shortcomings, we still believe there may be some value in this treewidth-specific balancing trick and mention it as an alternative approach because it allows us to get rid of randomization in this case. To the best of our knowledge, this is the first application of tree decomposition balancing theorems to approximation algorithms (the main motivation for their study so far was parallel and distributed algorithms). It would be interesting to see if in the future a combination of a balancing preprocessing step with the ideas of Theorem 4 could lead to better running times or more derandomized algorithms.

In the rest of this section we first, for the sake of completeness, give a complete proof of one algorithmic result, namely the approximation scheme for Max Cut. We then simply sketch the dynamic programs for all other problems, since the analysis is similar, highlighting some interesting details. We also give some brief remarks explaining why some algorithms cannot (easily) be extended to clique-width and why for some problems we only get bi-criteria approximations.

4.1 Max Cut

In this section we attack the Max Cut problem parameterized by clique-width. In Max Cut we are given a graph $G(V, E)$ and are asked to find a partition of V into $L, R \subseteq V,\ L = V \setminus R$ such that the number of edges with exactly one endpoint in L is maximized. We assume that a clique-width expression for G with w labels is supplied as part of the input.

We will follow the straightforward dynamic program for this problem, as used for example in [23]. The idea is to describe a partial solution by keeping track of the intersection of each label set with L. The first difference is of course that we will round all values to save time. The second (somewhat counter-intuitive) difference is that for each label set we will keep track of the size of its intersection with *both* L and R. In the exact program this would be wasteful, since one of these two numbers can be calculated from the other, using the (known) size of the whole label set. However, here we are trying to implement a dynamic program that relies only on additions. As a side effect, our dynamic program will likely include solutions which are clearly incorrect (the sum of the sizes of the two intersections being different from the size of the set) but this will not be a major problem if we can guarantee that all values are at most $(1 + \epsilon)$ far from a valid solution.

Dynamic Program As mentioned, we view the clique-width expression as a rooted binary tree. First, define the set $B := \{0\} \cup \{(1 + \delta)^j \mid j \in \mathbb{N}\}$. Informally, B is the set of rounded values that may appear in our table. Even though B is infinite, whenever the algorithm produces an entry with value larger than $(1 + \epsilon)n^2$ we simply drop that entry. It will not be hard to see that this will not affect the analysis if all entries are within a $(1 + \epsilon)$ factor of being correct (then nothing will be dropped). Observe that the size of B then becomes $\log_{(1+\delta)}(n^2) = \text{poly}(\log n/\epsilon)$, if we set δ according to Theorem 4.

The dynamic programming table D_i for a node i is a subset of $B^w \times B^w \times B$. The informal meaning is that an entry $(\boldsymbol{l}, \boldsymbol{r}, c) \in D_i$ if and only if there exists a partition of the subgraph of G represented by the sub-tree rooted at i into L_i, R_i such that:

1. for all $l \in \{1, \ldots, w\}$ the number of vertices in L_i (respectively R_i) with label l is roughly $\boldsymbol{l}(l)$ (respectively $\boldsymbol{r}(l)$)
2. the number of edges with exactly one endpoint in L_i is roughly c

A dynamic programming algorithm can now be formulated in a straightforward way. It is easy to fill the table for initial nodes. For Rename and Union nodes the exact dynamic program would perform some addition, which we now replace with the $\oplus$ operation. For example, consider a Rename node with labels $l_1 \to l_2$. For each entry $(\boldsymbol{l}, \boldsymbol{r}, c)$ in the child's table we create an entry $(\boldsymbol{l'}, \boldsymbol{r'}, c)$ in the current node as follows: we set $\boldsymbol{l'}(l_1) := 0$, $\boldsymbol{l'}(l_2) := \boldsymbol{l}(l_1) \oplus \boldsymbol{l}(l_2)$ and $\boldsymbol{l'}$ the same as $\boldsymbol{l}$ for other labels to make the vector $\boldsymbol{l'}$ of a new entry (similarly for $\boldsymbol{r'}$). In the same way, for a Union node, for each entry $(\boldsymbol{l_1}, \boldsymbol{r_1}, c_1)$ in the first child's table and for each entry $(\boldsymbol{l_2}, \boldsymbol{r_2}, c_2)$ in the second child's table we construct an entry $(\boldsymbol{l_1} \oplus \boldsymbol{l_2}, \boldsymbol{r_1} \oplus \boldsymbol{r_2}, c_1 \oplus c_2)$, where $\oplus$ is applied component-wise for vectors.

For Join nodes with labels l_1, l_2 we do something similar. For each entry $(\boldsymbol{l}, \boldsymbol{r}, c)$ in the child's table construct a new vector with the same $\boldsymbol{l}, \boldsymbol{r}$ and $c' := c \oplus (\boldsymbol{l}(l_1) \cdot \boldsymbol{r}(l_2) + \boldsymbol{l}(l_2) \cdot \boldsymbol{r}(l_1))$. Note that if the elements of $\boldsymbol{l}, \boldsymbol{r}$ are known with error at most $(1 + \epsilon)$ then the second term of this addition is known with error at most $(1 + \epsilon)^2 \approx 1 + 2\epsilon$.

Analysis First, observe that because the running time of the algorithm is polynomial in the size of the tables, the algorithm clearly achieves the running time stated in Theorem 1. We only have to prove its approximation guarantee.

Any node i of the clique-width expression defines a subgraph G_i of G (the graph produced by the sub-expression rooted at i). A partial solution is simply a restriction of a solution L, R for G to the vertices of G_i. The *signature* of a partial solution is a vector of $2w + 1$ integers that would represent this solution in an exact dynamic programming table. In other words, the signature is the entry we would expect to see in the table D_i if we were not rounding. In this case, it contains the exact size of the intersections of L, R with each label set and the size of the cut in G_i. To keep things simpler, we will only be concerned with the $2w$ values that represent the intersection. As mentioned, if we can get these right, the algorithm also gets the size of the cut right within a factor of roughly $(1 + 2\epsilon)$.

Consider a partial solution and its signature. Our strategy is to inductively define a mapping that gives for each such signature a collection of $2w$ Addition Trees. The exact results of these trees will be the values of the signature (that is, the sizes of the intersections of the two sets with each label). We will then establish (by induction) that there exists an entry in our algorithm's table whose values follow the same distribution as those Trees, if they are viewed as Approximate Addition Trees. It will follow that for every partial solution there exists, with high probability, an entry with roughly the same values. The converse statement can be shown with essentially the same ideas.

The above can clearly be done for initial nodes, where all values stored are 0 or 1, so our algorithm stores them exactly. The relevant Addition Trees are of height 0. Assume inductively that we have established the correspondence up to a certain height in the clique-width expression. Let us then consider a Rename node i with labels $l_1 \to l_2$ and a child j. Consider a partial solution to G_i.

This partial solution has some corresponding partial solution in G_j. By the inductive hypothesis, there exists an entry in $(\boldsymbol{l}, \boldsymbol{r}, c) \in D_j$ corresponding to this solution. In particular, there exist Approximate Addition Trees whose results have the same distribution as $\boldsymbol{l}(l_1)$ and $\boldsymbol{l}(l_2)$ and whose exact values are the same as the corresponding values in the partial solution to G_j. Consider the Tree obtained from these by adding a new root and making the old roots its children. This Tree now follows the same distribution as the value $\boldsymbol{l'}(l_2)$ calculated by our algorithm. Its exact value is the value we would get in the exact signature (since the same was true for the sub-trees). Reasoning in the same way about the other coordinates we have completed the inductive step in this case.

The cases of Join and Union nodes can be handled with similar inductive arguments as above. We can thus establish that for any valid partial solution signature there exists an entry of the table that approximately corresponds to it, in the sense that their respective values are connected through Addition Trees. It is also not hard to use inductive reasoning to also establish the converse (for every approximate entry there exists a corresponding partial solution). We now select the entry in the root's table that has maximum c. With high probability, it must correspond to a solution with approximately the same cut size (otherwise we can find an Approximate Addition Tree with large error). Retracing the steps of the dynamic programming we can turn this entry into an actual cut.

4.2 Edge Dominating Set

Let us now give a dynamic programming algorithm for Edge Dominating Set. Here, things are a little trickier, because it's not immediately obvious that using subtractions can be avoided. We will use the following equivalent version of the problem: we are looking for a minimum-size set of vertices S such that S is a vertex cover of G and $G[S]$ has a perfect matching. It is not hard to see that this is the same problem (intuitively, S is the set of vertices incident on an edge of the edge dominating set) because an optimal edge dominating set is also a maximal matching.

We define the dynamic programming table D_i for a node i as a subset of $(B \cup \{F\})^w \times B^w$, where F is a special symbol. Let G_i be the corresponding subgraph. Fix a solution to the problem in G, that is a vertex cover S and a matching of its vertices M. The intended meaning is that an entry $(\boldsymbol{s}, \boldsymbol{c}) \in D_i$ represents this solution if

1. for all $l \in \{1, \ldots, w\}$ the number of vertices of G_i with label l included in S is roughly $\boldsymbol{s}(l)$. If $\boldsymbol{s}(l) = F$ then *all* vertices with label l are in S.
2. for all $l \in \{1, \ldots, w\}$ the number of vertices of G_i with label l incident to an edge of $M \cap G_i$ is roughly $\boldsymbol{c}(l)$

Informally, $\boldsymbol{s}$ tells us how many vertices we have selected in S from each label set, while $\boldsymbol{c}$ tells us how many of the selected vertices have already been matched. Clearly, the intended meaning implies that $\boldsymbol{c}(j) \leq \boldsymbol{s}(j)$ for all j, but this may not always be the case. We will still be happy if this is true up to an error factor $(1 + \epsilon)$. In the calculations below, when we perform arithmetic operations on a value $\boldsymbol{s}(l)$ that is equal to F we substitute it with the size of the set V_l of vertices with label l.

Let us now describe the algorithm. Initial nodes are easy ($\boldsymbol{c}$ is all-zero, $\boldsymbol{s}$ can have a single coordinate that is 0 or F). For Rename and Union nodes we just have to add (component-wise) appropriate entries, taking care to maintain F values if possible. The interesting case is Join nodes.

Let i be a join node with labels $l_1 \leftrightarrow l_2$ and child j. For each entry $(\boldsymbol{s}, \boldsymbol{c}) \in D_j$ do the following. Let V_{l_1}, V_{l_2} be the set of vertices with label l_1, l_2 respectively in G_i. If $\boldsymbol{s}(l_1) \neq F$ and

$\boldsymbol{s}(l_2) \neq F$ then ignore this entry because this partial solution is not a vertex cover of G_i. Otherwise, for each $m \in \{0, \ldots, \min(|V_{l_1}|, |V_{l_2}|)\}$ calculate a vector $\boldsymbol{c_m}$ which is identical to $\boldsymbol{c}$ except that $\boldsymbol{c_m}(l_1) = \boldsymbol{c}(l_1) \oplus m$ and $\boldsymbol{c_m}(l_2) = \boldsymbol{c}(l_2) \oplus m$. Informally, we are selecting how many of the join edges will eventually be used in the matching M. If we have $\boldsymbol{c_m}(l_1) > (1+\epsilon)\boldsymbol{s}(l_1)$ or $\boldsymbol{c_m}(l_2) > (1+\epsilon)\boldsymbol{s}(l_2)$ ignore $\boldsymbol{c_m}$. Otherwise, add $(\boldsymbol{s}, \boldsymbol{c_m})$ to D_i.

Once the root's table has been calculated we select the entry $(\boldsymbol{s}, \boldsymbol{c})$ such that $\sum_l \boldsymbol{s}(l)$ is minimized, among entries where $\boldsymbol{c}(l) \geq \boldsymbol{s}(l)/(1+\epsilon)$. Retracing the steps of the dynamic programming we then obtain a vertex cover S. In polynomial time we can calculate a maximum matching in $G[S]$. This is our initial candidate solution. If some vertex of S is unmatched we add one of the edges connecting it to $V \setminus S$. We now have an edge dominating set.

We are now faced with two problems. First, we need to prove that S has roughly the same number of vertices as an optimal solution. Second, we need to prove that $G[S]$ contains an almost perfect matching (more precisely, it contains a set of edges M such that almost all vertices are incident on one edge of M). We can again rely on an inductive analysis using Approximate Addition Trees, as in the case of Max Cut. The main difference is that our algorithm now occasionally drops some partial solutions. This happens in the case of Join nodes when we have not selected enough vertices of V_{l_1}, V_{l_2} to produce a proper vertex cover. This step is easily seen to be correct, as there is no approximation involved. Alternatively, solutions are dropped when we are trying to select too many of the new edges in M. In this case, we leave enough "slack" in our comparisons so that if a solution is erroneously dropped we can extract an Approximate Addition Tree with high error, something that can only happen with low probability. Barring this, we can assume that all partial solutions are represented in the root's table and each table entry approximately corresponds to a solution. Thus, the solution we select in the end will have an almost optimal size for S and contain an almost perfect matching.

4.3 Equitable Coloring

This problem admits a very simple additive dynamic program. Let us briefly describe it. In the case of clique-width, for each node the table is a subset of $(B^k)^w$. Informally, the signature of a partial coloring is the number of vertices of each color contained in each label set. We can guarantee to produce a valid coloring if we make sure that in all Join nodes we drop partial solutions that use the same color somewhere in both label sets. This can be done without a problem because 0 values are stored exactly in our table. In the end, we pick the table entry appearing to give the most equitable coloring and extract a coloring by retracing the dynamic program. The running time is $(\log n/\epsilon)^{O(kw)}$.

Let us also note that we can give a faster algorithm for treewidth. Here the dynamic program needs to remember the size of each color class and the coloring of the w vertices of a bag. Thus, the table is a subset of $B^w \times \{1, \ldots, k\}^w$, leading to a running time of $(k \log n/\epsilon)^{O(w)}$.

4.4 Capacitated Dominating Set and Vertex Cover

Let us first describe the algorithm for Capacitated Dominating Set on clique-width promised in Theorem 1. The dynamic programming table D_i is a subset of $B^w \times B^w \times B^w \times \{0, \ldots, n\}$. Informally, $(\boldsymbol{a}, \boldsymbol{u}, \boldsymbol{d}, c) \in D_i$ if there exists a set S of *exactly* c vertices and a partial mapping of vertices of $V \setminus S$ to S such that

1. The total capacity of vertices of S with label l is roughly $\boldsymbol{a}(l)$

2. The number of vertices with label l in $V \setminus S$ mapped to S (i.e. dominated) is roughly $\boldsymbol{d}(l)$
3. The number of vertices of $V \setminus S$ mapped to vertices in S with label l is roughly $\boldsymbol{u}(l)$.

More simply, $\boldsymbol{a}$ is the total available capacity in a label set, $\boldsymbol{u}$ is the capacity already used in our partial solution and $\boldsymbol{d}$ is the number of vertices of each label set that we have already covered.

It is not hard to see how to fill this table for Initial, Rename and Union nodes (only additions are needed). For Join nodes, the interesting point is that we need to decide how many of the new edges are used. Similarly to the case of Edge Dominating Set, for each possible number m we add m to the *used* capacity $\boldsymbol{u}$ on one side and the number of *dominated* vertices $\boldsymbol{d}$ on the other. We drop solutions which are clearly (by more than a factor $1+\epsilon$) invalid. Note that, because for each label set we only care about its *total* capacity, not the number of selected vertices, we can afford to keep track of the total size of the dominating set exactly. This only adds a polynomial factor to the algorithm's running time.

In the end we extract a solution from the root's table and argue about its correctness as usual. One complication is that, since we do not know $\boldsymbol{a}$ and $\boldsymbol{d}$ exactly, the solution may be violating some capacities and it may not be a complete dominating set. We repair the solution by allowing some more vertices to be undominated. If the total capacity of each label set was violated by no more than $(1+\epsilon)$ this drops at most ϵn vertices. Similarly, if the solution was not a complete dominating set (and one exists), at most ϵn vertices are not dominated. We thus get the promised bi-criteria guarantee.

Let us also discuss the case of treewidth (Theorem 2). Here the dynamic program for Capacitated Dominating Set is easier. We just need to remember which of the vertices of each bag have been selected and how much of their capacity has already been used. The result is a set that dominates the whole graph, has size at most OPT, but may violate some capacities by $(1+\epsilon)$. Observe that this is stronger than the bi-criteria approximation we get for clique-width since, again, we could drop the coverage for some vertices to fix the capacities (but in general, it's not clear if we can trade in the other direction).

It's worthy of note that for Capacitated Dominating Set we can probably not hope to obtain anything better than a bi-criteria approximation, such as the one we gave here. The reason is that in [16] it is shown that Capacitated Dominating Set is W-hard even if parameterized by *both* the treewidth and OPT. Thus, if we could obtain an FPT approximation scheme for OPT without violating constraints, we would be able to set ϵ to an appropriate small value and obtain an FPT algorithm for a W-hard problem.

Finally, let us mention the Capacitated Vertex Cover algorithm promised in Theorem 2. This can be obtained by reducing Capacitated Vertex Cover to a vertex-weighted version of Capacitated Dominating Set. First, observe that if we associate with each vertex of an instance of Capacitated Dominating Set a non-negative integer cost and ask for a solution of minimum total cost the algorithm we gave still works with minimal modification. Consider now a Capacitated Vertex Cover instance. First, sub-divide each edge (this does not increase the treewidth w) and set the capacities of new vertices to 0. Then, add a new vertex u and connect it to all of the original vertices of the instance (this increases w by 1). Set the capacity of u to be equal to its degree. Finally, set the cost of each original vertex to 1, the cost of u to 0 and the cost of vertices constructed in the sub-divisions to n. If we view this as an instance of weighted Capacitated Dominating Set it is straightforward that a solution of cost $s < n$ exists if and only if the original Capacitated Vertex Cover instance had a solution of size s.

It would be interesting to see if an algorithm for Capacitated Vertex Cover on clique-width can be given. At the moment this seems more challenging than for Capacitated Dominating Set, because it is not sufficient to remember the total available capacity of a whole label set.

4.5 Bounded Degree Deletion

A dynamic program for clique-width can be given as follows: each table is a subset of $B^w \times B^w \times \{0, \ldots, n\}$. Informally, a partial solution is a set of "active" vertices which will *not* be deleted. We remember how many of these we have in each label set. We also remember what is the maximum degree of any active vertex in each label set. Finally, we remember the total number of vertices we have deleted. The only interesting case is Join nodes, where the new maximum degree is calculated by adding to the previous maximum degree the number of active vertices in the other label set.

A dynamic program for treewidth can be formulated by keeping track of the active vertices inside a bag. For each one of these we remember (approximately) the number of active neighbors it has in the set of vertices that appear in bags lower in the decomposition. The running time for both clique-width and treewidth is $(\log n/\epsilon)^{O(w)}$.

Let us remark that, even though we only present a bi-criteria approximation algorithm here, to the best of our knowledge there is no complexity barrier ruling out the existence of an approximation scheme for the natural objective of this problem (the size of the deletion set).

4.6 Graph Balancing

We only deal with this problem for treewidth. We assume that edge weights are written in unary (this is the interesting case that is usually studied to avoid unnecessary complications). For a bag of the tree decomposition, the signature of a partial solution is the weighted out-degree each vertex of the bag has already accumulated due to edges whose other endpoint appears lower in the decomposition. We keep track of this information and the maximum seen so far, making the table be a subset of B^{w+1}. When a vertex is forgotten (that is, we move to the first bag that does not contain it) we go through every orientation of the edges connecting it to the rest of the bag and update the out-degrees of other vertices accordingly. The running time is $(\log n/\epsilon)^{O(w)}$ and we can extract a solution with out-degree at most $(1+\epsilon)$OPT.

Let us remark that for the case of clique-width, it's not possible to achieve a similar result. It is known that even for graphs with edge weights 1 and 2 it is NP-hard to tell if OPT is 2 or 3 [17]. Take an arbitrary such instance, multiply all edge weights with n^2 and then replace every non-edge with an edge of weight 1. It is now NP-hard to tell if OPT is at most $2n^2 + n$ or at least $3n^2$, so a better than 3/2 approximation is NP-hard even for cliques.

5 Conclusions

We have presented a generic technique which can be applied with minor modifications to a number of hard problems. The question now becomes to which other problems we can apply this technique. The most prominent next target is Hamiltonicity parameterized by clique-width. This is another example that separates clique-width from treewidth in term of parameterized tractability, and its natural dynamic program uses integers. Unfortunately, the natural program also uses subtractions, so it does not immediately yield to our methods. In several of the problems of this paper we managed

to work around such obstacles, rewriting the natural program to only use additions. Is this possible here, or are completely new ideas needed?

Let us also mention that in this paper we have focused on establishing that our algorithms run in $(\log n/\epsilon)^{O(w)}$, but we have not been paying much attention to the constant hidden in the exponent. Another good direction would therefore be to improve the analysis of Approximate Addition Trees in order to obtain better running times for our schemes. Perhaps combining this with the idea of (partially) balancing the given decomposition could also help speed up the overall algorithms.

Acknowledgment I am grateful to an anonumous reviewer for pointing out that treewidth balancing theorems can be used to obtain some of the results of this paper in a simpler way.

References

1. Yuichi Asahiro, Eiji Miyano, and Hirotaka Ono. Graph classes and the complexity of the graph orientation minimizing the maximum weighted outdegree. In Harland and Manyem [27], pages 97–106.
2. MohammadHossein Bateni, MohammadTaghi Hajiaghayi, and Dániel Marx. Approximation schemes for steiner forest on planar graphs and graphs of bounded treewidth. In Leonard J. Schulman, editor, *STOC*, pages 211–220. ACM, 2010.
3. Oren Ben-Zwi, Danny Hermelin, Daniel Lokshtanov, and Ilan Newman. Treewidth governs the complexity of target set selection. *Discrete Optimization*, 8(1):87–96, 2011.
4. Nadja Betzler, Robert Bredereck, Rolf Niedermeier, and Johannes Uhlmann. On bounded-degree vertex deletion parameterized by treewidth. *Discrete Applied Mathematics*, 160(1):53–60, 2012.
5. Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michal Pilipczuk. An $O(c^k n)$ 5-Approximation Algorithm for Treewidth. In *FOCS*, pages 499–508. IEEE Computer Society, 2013.
6. Hans L Bodlaender and Fedor V Fomin. Equitable colorings of bounded treewidth graphs. *Theoretical Computer Science*, 349(1):22–30, 2005.
7. Hans L. Bodlaender and Torben Hagerup. Parallel algorithms with optimal speedup for bounded treewidth. *SIAM J. Comput.*, 27(6):1725–1746, 1998.
8. Hans L Bodlaender and Arie MCA Koster. Combinatorial optimization on graphs of bounded treewidth. *The Computer Journal*, 51(3):255–269, 2008.
9. Edouard Bonnet, Bruno Escoffier, Eun Jung Kim, and Vangelis Th. Paschos. On subexponential and fpt-time inapproximability. In Gutin and Szeider [26], pages 54–65.
10. Edouard Bonnet and Vangelis Th. Paschos. Parameterized (in)approximability of subset problems. *CoRR*, abs/1310.5576, 2013.
11. Ljiljana Brankovic and Henning Fernau. Combining two worlds: Parameterised approximation for vertex cover. In Otfried Cheong, Kyung-Yong Chwa, and Kunsoo Park, editors, *ISAAC (1)*, volume 6506 of *Lecture Notes in Computer Science*, pages 390–402. Springer, 2010.
12. Mathieu Chapelle. Parameterized complexity of generalized domination problems on bounded tree-width graphs. *CoRR*, abs/1004.2642, 2010.
13. Rajesh Hemant Chitnis, MohammadTaghi Hajiaghayi, and Guy Kortsarz. Fixed-parameter and approximation algorithms: A new look. In Gutin and Szeider [26], pages 110–122.
14. Bruno Courcelle and Mamadou Moustapha Kanté. Graph operations characterizing rank-width and balanced graph expressions. In Andreas Brandstädt, Dieter Kratsch, and Haiko Müller, editors, *WG*, volume 4769 of *Lecture Notes in Computer Science*, pages 66–75. Springer, 2007.
15. Bruno Courcelle, Johann A Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems*, 33(2):125–150, 2000.
16. Michael Dom, Daniel Lokshtanov, Saket Saurabh, and Yngve Villanger. Capacitated domination and covering: A parameterized perspective. In Martin Grohe and Rolf Niedermeier, editors, *IWPEC*, volume 5018 of *Lecture Notes in Computer Science*, pages 78–90. Springer, 2008.
17. Tomás Ebenlendr, Marek Krcál, and Jiri Sgall. Graph balancing: a special case of scheduling unrelated parallel machines. In Shang-Hua Teng, editor, *SODA*, pages 483–490. SIAM, 2008.

18. Wolfgang Espelage, Frank Gurski, and Egon Wanke. How to solve NP-hard graph problems on clique-width bounded graphs in polynomial time. In Andreas Brandstädt and Van Bang Le, editors, *WG*, volume 2204 of *Lecture Notes in Computer Science*, pages 117–128. Springer, 2001.
19. Michael R Fellows, Fedor V Fomin, Daniel Lokshtanov, Frances Rosamond, Saket Saurabh, Stefan Szeider, and Carsten Thomassen. On the complexity of some colorful problems parameterized by treewidth. *Information and Computation*, 209(2):143–153, 2011.
20. Jirí Fiala, Petr A. Golovach, and Jan Kratochvíl. Distance constrained labelings of graphs of bounded treewidth. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *ICALP*, volume 3580 of *Lecture Notes in Computer Science*, pages 360–372. Springer, 2005.
21. Jörg Flum and Martin Grohe. *Parameterized complexity theory*, volume 3. Springer Heidelberg, 2006.
22. Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Clique-width: on the price of generality. In Claire Mathieu, editor, *SODA*, pages 825–834. SIAM, 2009.
23. Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Algorithmic lower bounds for problems parameterized with clique-width. In Moses Charikar, editor, *SODA*, pages 493–502. SIAM, 2010.
24. Fabrizio Frati, Serge Gaspers, Joachim Gudmundsson, and Luke Mathieson. Augmenting graphs to minimize the diameter. In Leizhen Cai, Siu-Wing Cheng, and Tak Wah Lam, editors, *ISAAC*, volume 8283 of *Lecture Notes in Computer Science*, pages 383–393. Springer, 2013.
25. Petr A. Golovach and Dimitrios M. Thilikos. Paths of bounded length and their cuts: Parameterized complexity and algorithms. *Discrete Optimization*, 8(1):72–86, 2011.
26. Gregory Gutin and Stefan Szeider, editors. *Parameterized and Exact Computation - 8th International Symposium, IPEC 2013, Sophia Antipolis, France, September 4-6, 2013, Revised Selected Papers*, volume 8246 of *Lecture Notes in Computer Science*. Springer, 2013.
27. James Harland and Prabhu Manyem, editors. *Theory of Computing 2008. Proc. Fourteenth Computing: The Australasian Theory Symposium (CATS 2008), Wollongong, NSW, Australia, January 22-25, 2008. Proceedings*, volume 77 of *CRPIT*. Australian Computer Society, 2008.
28. Michael Lampis. Parameterized maximum path coloring. In Dániel Marx and Peter Rossmanith, editors, *IPEC*, volume 7112 of *Lecture Notes in Computer Science*, pages 232–245. Springer, 2011.
29. Dániel Marx. Minimum sum multicoloring on the edges of planar graphs and partial k-trees. In Giuseppe Persiano and Roberto Solis-Oba, editors, *WAOA*, volume 3351 of *Lecture Notes in Computer Science*, pages 9–22. Springer, 2004.
30. Dániel Marx. Parameterized complexity and approximation algorithms. *Comput. J.*, 51(1):60–78, 2008.
31. Kitty Meeks and Alexander Scott. The parameterised complexity of list problems on graphs of bounded treewidth. *CoRR*, abs/1110.4077, 2011.
32. Takao Nishizeki, Jens Vygen, and Xiao Zhou. The edge-disjoint paths problem is np-complete for series–parallel graphs. *Discrete Applied Mathematics*, 115(1):177–186, 2001.
33. Christos H Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *Journal of computer and system sciences*, 43(3):425–440, 1991.
34. Marko Samer and Stefan Szeider. Tractable cases of the extended global cardinality constraint. In Harland and Manyem [27], pages 67–74.
35. Marko Samer and Stefan Szeider. Constraint satisfaction with bounded treewidth revisited. *J. Comput. Syst. Sci.*, 76(2):103–114, 2010.
36. Piotr Skowron and Piotr Faliszewski. Approximating the MaxCover problem with bounded frequencies in FPT time. *CoRR*, abs/1309.4405, 2013.
37. Stefan Szeider. Monadic second order logic on graphs with local cardinality constraints. *ACM Trans. Comput. Log.*, 12(2):12, 2011.
38. Stefan Szeider. Not so easy problems for tree decomposable graphs. *CoRR*, abs/1107.1177, 2011.