Toward a Model for Backtracking and Dynamic Programming

Michael Alekhnovich* Allan Borodin† Joshua Buresh-Oppenheim‡
Russell Impagliazzo§¶ Avner Magen† Toniann Pitassi†¶

October 6, 2007

Abstract

We propose a model called *priority branching trees (pBT)* for backtracking and dynamic programming algorithms. Our model generalizes both the priority model of Borodin, Nielson and Rackoff, as well as a simple dynamic programming model due to Woeginger, and hence spans a wide spectrum of algorithms. After witnessing the strength of the model, we then show its limitations by providing lower bounds for algorithms in this model for several classical problems such as Interval Scheduling, Knapsack and Satisfiability.

1 Introduction

The "Design and Analysis of Algorithms" is a basic component of the Computer Scienc Curriculum. Courses and texts for this topic are often organized around a toolkit of algorithmic paradigms or meta-algorithms such as greedy algorithms, divide and conquer, dynamic programming, local search, etc. Surprisingly (as this is often the main "theory course"), these algorithmic paradigms are rarely, if ever, precisely defined. Instead, we provide informal definitional statements followed by (hopefully) well chosen illustrative examples. Our informality in algorithm design should be compared to computability theory where we have a well accepted formalization for the concept of an algorithm, namely that provided by Turing machines and its many equivalent computational models (i.e. consider the almost universal acceptance of the Church-Turing thesis). While quantum computation may challenge the concept of "efficient algorithm", the benefit of having a well defined concept of an algorithm and a computational step is well appreciated.

In contrast, consider the following repesentative informal definition of a greedy algorithm in one of the standard texts [14]: "Algorithms for optimization problems typically go through a sequence of steps, with a set of choices at each step. ... A *greedy algorithm* always makes the choice that looks best at the moment." For pedagogical purposes this informal approach allows most students to understand the relevant concepts and it may well be that any attempt to provide precise definitions would be counter-productive. But what if we wanted to provably understand the extent to which (say) greedy algorithms or dynamic programming can efficiently and optimally solve problems such as weighted interval scheduling and maximum matching in a bipartite graph? Clearly to prove

*Institute for Advanced Study, Princeton, US. Supported by CCR grant $\mathcal{N}$CCR-0324906.

†Department of Computer Science, University of Toronto, bo r,avner,toni@cs.toronto.edu.

‡Computing Science Department, Simon Fraser Univ ersity, jburesho@cs.sfu.ca. Research partially supported by a grant from the PIMS Institute.

§CSE Department, University of California, San Diego, russell@cs.ucsd.edu.

¶Some of this research was performed at the Institute for Advanced Study in Princeton, NJ supported by the State of New Jersey.

any limitations of a particular algorithmic approach we must have a precise definition. While it is probably not possible to achieve the universality of a Church-Turing thesis for say greedy algorithms, we can certainly try to provide models that capture many if not most examples that we have in mind.

This reserach direction has a substantial (albeit perhaps not that influential) history as well as being the subject of some recent attention. We will review some of the relevant work in sections 2 and 3. We continue this line of research by presenting a model for backtracking algorithms as applied to combinatorial search and optimization problems. Informally, in the *priority branching tree (pBT) model* an algorithm creates a tree of solutions, where each branch of the tree gradually builds a solution one item at a time. We classify such algorithms according to the manner in which items are considered resulting in fixed, adaptive and strongly adaptive pBT algorithms. For fixed and adaptive order pBT algorithms, the item order is the same on each branch, where for adaptive pBT the item order depends on the actual set of input items. In strongly adaptive pBT algorithms the ordering can be different on each branch. We formally define the pBT model and its variants in section 3. Many well-known algorithms and algorithmic techniques can be simulated within these models, both those that are usually considered backtracking (using the strongly adaptive model) and some that would normally be classified as greedy or "simple" dynamic programming (using even the fixed order model) as in the teminology of Woeginger [36]. We prove several upper and lower bounds on the capabilities of algorithms in our models in some cases proving that the known algorithms are essentially the best possible within the model.

Our results The computational problems we consider are all well-studied; namely, Interval Scheduling, Knapsack and Satisfiability. For m-machine Interval Scheduling we show a tight $\Omega(n^m)$-width lower bound (for optimality) in the adaptive-order pBT model, an inapproximability result in the fixed-order pBT model, and an approximability separation between width-1 pBT and width-2 pBT in the adaptive-order model. For Knapsack, we show an exponential lower bound (for optimality) on the width of adaptive-order pBT algorithms, and for achieving an FPTAS in the adaptive-order model we show upper and lower bounds polynomial in $1/\epsilon$. Knapsack also exhibits a separation between width-1 and width-2 adaptive-order pBT: a width-1 pBT cannot approximate Knapsack better than a factor of $n^{-1/4}$, while a standard 1/2-approximation falls into width-2 pBT. For SAT, we show that 2-SAT is solved by a linear-width adaptive-order pBT, but needs exponential width for any fixed-order pBT, and also that MAX2SAT cannot be efficiently approximated by any fixed-order pBT algorithm. (Using a similar argument we show an inapproximation result for Vertex Cover with respect to fixed-order pBT algorithms.) We then show that 3-SAT requires exponential width and exponential depth first size in the fully-adaptive-order pBT model. A small extension to this width lower bound in turn gives us an exponential bound on the width of fully-adaptive-order pBT algorithms for Knapsack using a "pBT reduction."

2 Brief History of Related Work

We briefly mention some relevant work in the spirit of our results. Our pBT model and the negative results we derive for the knapsack problem are very similar to the results of Chvátal [12] who proves an exponential lower bound for Knapsack in an algorithmic model that involves elements of branch-and-bound and dynamic programming (DP). In section 3.1 we will relate the branch and bound model to our model. Karp and Held [29] introduced a formal language approach for defining "decomposable" combinatorial optimization problems and derived a formalism (based on finite automata) for dynamic programming within this context. Helman and Rosenthal [25] and Helman

[24] extended the Karp and Held approach to a non assocative analogue of regular expressions. In this approach, the concantenation operation represents the concatenation of partial solutions while the + operation represents the union of partial solutions. A main goal of [25] was to model "non-serial" DP applications for computing optimal parenthesizations as in the matrix chain problem and constructing binary search trees. (See, for example, the text by Cormen, et al [14] for the definition of these problems and the well-known DP algorithms that achieve optimal solutions.) Moreover, by separating the syntax and the semantics (for a particular application), the formalism in [25] exposes an equivalence between these two well known applications. In terms of complexity results, Helman and Rosenthal show that the well known algorithms minimize complexity for oblvious programs (in terms of both the concatenation and + operations) where oblivious programs are essentially circuits where the opertations executed do not depend on the data. Hence such oblivious programs do not distinguish between the $\Theta(n^3)$ complexity bound for the basic DP algorithms in contrast to Knuth's [31] $\Theta(n^2)$ DP algorithm for computing an optimal binary search tree. Helman [24] extends the formalism in [25] so as to be able to provide a formalism capable of modelling both branch and bound and DP algorithms. The complexity results here are also restricted to oblivious programs (applied to the acyclic stage graph problem). A further distinction is made in terms of "data-based" programs which are meant to model more "realistic" programs. These formal language approaches provide very expressive formalisms but it is not clear to what extent one can derive significant lower bounds within these formalisms. More recently, Khanna, Motwani, Sudan and Vazirani [30] formalize various types of local search paradigms, and in doing so, provide a more precise understanding of local search algorithms. Woeginger [36] defines a class of simple dynamic programming algorithms and provides conditions for when a dynamic programming solution can be used to derive a FPTAS for an optimization problem. As stated befofe, these simple dynamic programming algorithms can be modelled within our fixed order pBT model. Achlioptas and Sorkin [1] define myopic algorithms for the purpose of analyzing the satisfiability of random 3CNF formulas. Borodin, Nielsen and Rackoff [9] introduce priority algorithms as a model of greedy-like algorithms. We will swee that myopic SAT algorithms are (for the most part) priority algorithms or small width pBT algorithms. Arora, Bollobás, Lovász and Tourlakis [7] study wide classes of LP formulations, and prove integrality gaps for Vertex Cover within these classes. The most popular methods for solving SAT are DPLL algorithms—a family of backtracking algorithms whose complexity has been characterized in terms of resolution proof complexity (see for example [15, 16, 13, 21]). The pBT model encompasses DPLL in many situations where access to the input is limited to some extent.

3 The pBT Model and its Relatives

We begin with some motivation and an informal description of priority branching trees (pBT). The starting point for the pBT model is the priority algorithm model of [9]. We assume that the input is represented as a set of data items, where each data item is a small piece of information about the problem; it may be a time interval representing a job to be scheduled, a vertex with its list of the neighbours in a graph, a propositional variable [1] with all clauses containing it in a CNF formula. Priority algorithms consider one item at a time and maintain a single partial solution (based on the items considered thus far) that it continues to extend. What is the order in which items are considered? A fixed-order algorithm initially orders the items according to some criteria (e.g., in

[1] We note that by using this input representation for a CNF formula, myopic algorithms [1] can be viewed as priority algorithms (when only one variable is set) or as constant-width pBT algorithms when a constant number of variables are set in a "free move" of the algorithm. In hindsight, we see that the myopic requirement of iteratively and irrevocably setting propositional variables (in their context of satisfying low density CNF formulas) further motivates the general priority framework for making myopic decisions about input items.

the case of Knapsack, sort the items by their weight to value ratio). A more general (adaptive order) approach would be to change the ordering according to the items seen so far. For example, in the greedy set cover algorithm, in every iteration we order the sets according to the number of yet uncovered elements. (The distinction between fixed and adaptive orderings has also been studied in [19].) Rather than imposing complexity constraints on the allowable orders, we require them to be "localized".[2] By introducing branching, a *priority branching tree* (pBT) can pursue a number of different partial solutions. Given a specific input, a pBT algorithm then induces a computation tree. Of course, it is possible to solve any properly formulated search or optimization problem in this manner: simply branch on every possible decision for every input item. In other words, there is a tradeoff between the quality of a solution and the complexity of the pBT-algorithm. We view the maximum width of a pBT program as the number of partial solutions that need to be maintained in parallel in the worst case. As we will see, this extension allows us to model the simple dynamic programming framework of Woeginger [36]. This branching extension can be applied to either the fixed or adaptive order (fixed-order pBT and adaptive-order pBT) and in either case each branch (corresponding to a partial solution) considers the items in the same order. For example, various DP based optimal and approximate algorithms for the Knapsack problem can be seen as fixed- or adaptive-order pBT algorithms. In order to model the power of backtracking programs (say as in DPLL algorithms for SAT)[3] we need to extend the model further. In a fully-adaptive-order pBT we allow each branch to choose its own ordering of input items. Furthermore, we need to allow algorithms to prioritize (using a depth first traversal of the induced computation tree) the order in which the different partial solutions are pursued. In this setting, we can consider the number of nodes traversed in the computation tree before a solution is found (which may be smaller than the tree's width).

We now formalize these concepts. Let $\mathcal{D}$ be an arbitrary data domain that contains objects D_i called **data items**. Let H be a set, representing the set of allowable decisions for a data item. For example, for the Knapsack problem, a natural choice for $\mathcal{D}$ would be the set of all pairs (x, p) where x is a weight and p is a profit; the natural choice for H is $\{0, 1\}$ where 0 is the decision to reject an item and 1 is the decision to accept an item.

A **search/optimization problem** P is specified by a pair $(\mathcal{D}_P, f_P)$ where $\mathcal{D}_P$ is the underlying data domain, and f_P is a family of objective functions, $f_P^n : (D_1, \ldots, D_n, a_1, \ldots, a_n) \mapsto \mathbb{R}$, where $a_1, ..., a_n$ is a set of variables that range over H, and $D_1, \ldots, D_n$ is a set of variables that range over $\mathcal{D}$. On input $I = D_1, \ldots, D_n \in \mathcal{D}$, the goal is to assign each a_i a value in H so as to maximize (or minimize) f_P^n. A search problem is a special case where f_P^n outputs either 1 or 0.

For any domain S we write $\mathcal{O}(S)$ for the set of all orderings of elements of S. We are now ready to define pBT algorithms.

Definition 1. *A* priority branching tree (pBT) algorithm $\mathcal{A}$ *for problem* $P = (\mathcal{D}, \{f_P^n\})$ *consists of the* ordering *functions*

$$r_{\mathcal{A}}^k : \mathcal{D}^k \times H^k \mapsto \mathcal{O}(\mathcal{D})$$

and the choice *functions*

$$c_{\mathcal{A}}^k : \mathcal{D}^{k+1} \times H^k \mapsto \mathcal{O}(H \cup \{\perp\}).^4$$

[2]The precise sense in which we restrict the allowable orders to be localized will be formalized in Definition 1. We note that, in hindsight, our fixed orders are those satisfying Arrow's "independence of irrelevant alternatives" axiom, as used in social choice theory [8].

[3]The pBT model encompasses DPLL in many situations where access to the input is limited. If access is unlimited, then proving superpolynomial lower bounds for DPLL amounts to proving $P \neq NP$.

[4]All of our lower bound results will apply to non-uniform pBT algorithms that know n, the number of input items, and hence more formally, the ordering and choice functions should be denoted as $r_{\mathcal{A}}^{n,k}$ and $c_{\mathcal{A}}^{n,k}$. A discussion regarding "precomputed" information can be found in [9].

We separate the following three classes of pBT algorithms

- **Fixed-Order** *algorithms:* $r^k_{\mathcal{A}}$ *does not depend upon* k *or any of its arguments.*
- **Adaptive-Order** *algorithms:* $r^k_{\mathcal{A}}$ *depends on* $D_1, D_2, ..., D_k$ *but not on* $a_1, ..., a_k$*.*
- **Fully-Adaptive-Order** *algorithms:* $r^k_{\mathcal{A}}$ *depends on both* $D_1, D_2, ..., D_k$ *and* $a_1, ..., a_k$*.*

The idea of the above specification of $\mathcal{A}$ is as follows. Initially, the set of actual data items is some unknown set I of items from $\mathcal{D}$. At each point in time, a subset of actual data items, $D_1, \ldots, D_k \subseteq S$ has been *revealed*, and decisions $a_1, \ldots, a_k$ have been made about each of these items in turn. At the next step, the backtrack algorithm (possibly) re-orders the set of all possible data items as specified by $r^k_{\mathcal{A}}$. Then as long as there are still items from I left to be discovered, another data item from I is revealed with the property that the one revealed next will be the first item in I, according to the ordering $r^k_{\mathcal{A}}$, that has not already been revealed. When this new item, $D_{k+1} \in I$ has been revealed, a set of possibilities are explored on this item, as specified by $c^k_{\mathcal{A}}$. Namely, the algorithm can try any subset of choices from H on this new data item, including the choice to abort ($\perp$). This is described more formally by the notion of a computation tree of program $\mathcal{A}$ on input I, as defined below. We say that a rooted tree is **oriented** if it has an ordering on its leaves from the **left** to the **right**.

Definition 2. *Assume that* P *is a search/optimization problem and* $\mathcal{A}$ *is a pBT algorithm for* P*. For any instance* $I = (D_1, ..., D_n)$*,* $D_i \in \mathcal{D}_P$ *we define the* **computation tree** $T_{\mathcal{A}}(I)$ *as an oriented rooted tree in the following recursive way.*

- *Each node* v *of depth* k *in the tree is labelled by a tuple* $< D^v_1, ...D^v_k, a^v_1, ...a^v_k >$*.*
- *The root node has the empty label.*
- *For every node* v *of depth* $k < n$ *with a label* $< \vec{D}^v, \vec{a}^v >$*, let* D^v_{k+1} *be the data item in* $I \setminus \{D^v_1, ..., D^v_k\}$ *that goes first in the list* $r^k_{\mathcal{A}}(\vec{D}^v, \vec{a}^v)$*. Assume that the output* $c^k_{\mathcal{A}}(\vec{D}^v, D^v_{k+1}, \vec{a}^v)$ *has the form* $(c_1, ..., c_d, \perp, c_{d+1}, \ldots)$*, where* $c_i \in H$*. If* $d = 0$ *then* v *has no children. Otherwise it has* d *child nodes* $v_1, ..., v_d$ *that go from left to right and have labels* $(D^{v_i}_1, ..., D^{v_i}_{k+1}, a^{v_i}_1, ..., a^{v_i}_{k+1}) = (D^v_1, ..., D^v_k, D^v_{k+1}, a^v_1, ..., a^v_k, c_i)$ *resp.*

Each leaf node t *of depth* n *contains a permuted sequence of the data items* I *(permuted by the ordering functions* $r^k_{\mathcal{A}}$ *used on the path ending at* t*) with the corresponding decisions in* H *(determined by the choice functions on this path). For a search problem we say that a leaf is a* **solution** *for* $I = (D_1, ..., D_n)$ *iff* $f_P(D^t_1, ..., D^t_n, a^t_1, ..., a^t_n) = 1$ *where* a^t_k *is the decision for* D^t_k*. For an optimization problem every leaf determines a solution and a value for the objective function on the instance* I*.*

We can define the semantics so that the value of the objective function is $-\infty$ for a maximization problem and ∞ for a minimization problem if the solution is not feasible. Similarly, if I is not a well-formed instance of the problem, then every solution should attain the same value in the objective function.

Definition 3. *We say that* $\mathcal{A}$ *is a* **correct algorithm** *for a pBT search problem* P *iff for any YES instance* I*,* $T_{\mathcal{A}}(I)$ *contains at least one solution. For an optimization problem, the value of* $\mathcal{A}(I)$ *is the value of the leaf that optimally or best approximates the value of the objective function on the instance* I*.*

- *For an algorithm $\mathcal{A}$ we define the* **width of the computation** $W_{\mathcal{A}}(I)$ *as the maximum of the number of nodes over all depth levels of* $T_{\mathcal{A}}(I)$.
- *We define the* **depth first search size** $S^{df}_{\mathcal{A}}(I)$ *as the number of tree nodes that lie to the left of the leftmost solution of* $T_{\mathcal{A}}(I)$.

Proposition 4. *For any* $\mathcal{A}$ *and* I $S^{df}_{\mathcal{A}}(I) \leq nW_{\mathcal{A}}(I)$.

Definition 5. *For any* $\mathcal{A}$ *and any* n, *define* $W_{\mathcal{A}}(n)$, *the width of* $\mathcal{A}$ *on instances of size* n *as* $\max\{W_{\mathcal{A}}(I) \ : \ |I| = n\}$. *Define* $S^{df}_{\mathcal{A}}(n)$ *analogously.*

The size $S^{df}_{\mathcal{A}}(I)$ corresponds to the running time of the depth first search algorithm on $T_{\mathcal{A}}(I)$. We will be mainly interested in the width of $T_{\mathcal{A}}(I)$ for two reasons. First, it has a natural combinatorial meaning: the maximum number of partial solutions that we maintain simultaneously during the execution. As such, the width is a measure of space complexity (for a level by level implementation of the algorithm). Second, the width provides a universal upper bound on the running time of any search style.

While the fixed- and adaptive-order models are ostensibly less powerful than fully-adaptive-order algorithms, they remain quite powerful. For example, the width 1 algorithms in these classes are precisely the fixed- and adaptive-order priority algorithms, respectively, that capture many well known greedy algorithms. In addition, we will see that they can simulate large classes of dynamic programming algorithms; for example, fixed-order pBT algorithms can simulate Woeginger's DP-simple algorithms ([36]).

The reader may notice some similarities between the pBT model and the online setting. Like online algorithms, the input is not known to the algorithm in advance, but is viewed as an input stream. However, there are two notable differences: First, the ordering is given here by the algorithm and not by the adversary, and secondly, pBT algorithms are allowed to branch, or try more than one possibility.[5]

A note on computational complexity: We do not impose any restrictions on the functions $r^k_{\mathcal{A}}$ and $c^k_{\mathcal{A}}$ such as computability in polynomial time. This is because all lower bounds in this model come from information theoretic constraints and hold for any (even non-computable) $r^k_{\mathcal{A}}$ and $c^k_{\mathcal{A}}$. However, if these functions are polytime computable then there exists an efficient algorithm $\mathcal{B}$ that solves the problem[6] in time $S^{df}_{\mathcal{A}}(I)n^{O(1)}$. In particular, all upper bounds presented in this paper correspond to algorithms which are efficiently computable. Another curious aspect is that one has to choose the representation model carefully in order to limit the information in each data item, because once a pBT algorithm has seen all of the input (or can infer it), it can immediately solve the problem. Hence, we should emphasize that there are unreasonable (or at least non-standard) input models that will render the model useless; for example if a node in a graph contains the information about its neighbours *and* their neighbours, then it contains enough information that (using exponential time) the ordering function can summon the largest clique as its first items, making an NP-hard problem solvable by a width-1 pBT algorithm. In our discussion, we use input representations which seem to us the most natural.

[5] A version of the online model in which many partial solutions may be constructed was studied by Halldorsson, et al [22]. Their online model is a special case of a fixed-order pBT algorithm.

[6] In this regard, the depth first search measure has to be further clarified for the approximation of optimization problems. Namely, in contrast to a search problem, it may not be known that (say) a c-approximate solution has been found. One way to retain the claim that polynomial time functions $r^k_{\mathcal{A}}$ and $c^k_{\mathcal{A}}$ provide a polynomial time algorithm is by imposing a polynomial time constructible complexity bound. That is, let $T(n)$ be a constructible polynomial time complexity bound. We can then define the output of a $T(n)$ time bounded pBT algorithm to be the best solution found within the first $T(n)$ nodes of the depth first search of the pBT tree.

3.1 pBT as an Extension of Dynamic Programming and other Algorithm Models

How does our model compare to other models? As noted above, the width 1 pBT algorithms are exactly the priority algorithms, so many greedy algorithms fit within the framework. Examples include Kruskal or Prim's algorithms for spanning tree, Dijkstra's shortest path algorithm, and Johnson's greedy 2-approximation for Vertex Cover.

Secondly, which is also one of the main motivations of this work, the fixed-order model captures an important class of dynamic programming algorithms defined by Woeginger [36] as *simple dynamic-programming* or *DP-simple*. Many (but certainly not all) algorithms we call "DP algorithms" follow the schema formalized by Woeginger: Given an ordering of the input items, in the k-th phase the algorithm considers the k-th input item X_k, and produces a set $\mathcal{S}_k$ of solutions to the problem with input $\{X_1, \ldots, X_k\}$. Every solution in $\mathcal{S}_k$ *must extend* a solution in $\mathcal{S}_{k-1}$. Knapsack (with small integer input parameters), and Interval Scheduling with m machines, are two well studied problems which have well known DP-simple algorithms. The standard DP algorithm for the string edit distance problem can also be viewed as a DP-simple algorithm.

The simulation of these algorithms by a fixed-order pBT algorithm is straightforward once one makes the following observation. Since all parallel runs of a fixed- or adaptive-order pBT algorithm view the same input, each run can simulate all other runs. Thus, width $w(n)$-algorithms in both of these models are equivalent to *sequential* algorithms that maintain a set T_k of at most $w(n)$ partial solutions for the partial instance (representing each of the up to $w(n)$ active runs at this level) with the following restriction. Since the solution found must extend one of the partial solutions for the runs, any solution in T_{k+1} must extend a solution in $\mathcal{S}_k$. For concreteness, consider the simulation of a well known DP algorithm to solve Interval Scheduling on one machine. This algorithm orders intervals by their ending time (earliest first). It then calculates $T[j]$ = the intervals among the first j which give maximal profit and which schedule the j'th interval; of course $T[j]$ extends $T[i]$ for some $i < j$. We can now think of a pBT algorithm which in the j-th level has partial-solutions corresponding to $T[0], T[1], \ldots, T[j]$. To calculate the partial solutions for the first $j+1$ intervals we take $T[j+1]$ extending one of the $T[i]$'s and also take $T[0], T[1], \ldots, T[j]$ so as to extend the corresponding partial solutions with a 'reject' decision on the $j+1^{st}$ interval.

Note that for many dynamic programming algorithms, the size of the number of solutions maintained is determined by an array where each axis has length at most n. Thus, the size of T_k typically grows as some polynomial n^d. In this case, we call d the *dimension* of the algorithm. Note that we have $d = \log w(n)/\log n$, so a lower bound on width yields a lower bound on this dimension.

While powerful, there are also some restrictions of the model that seem to indicate that we cannot simulate all (intuitively understood as) back-tracking or branch-and-bound algorithms. That is, our decision to abort a run can only depend (although in an arbitrary way) on the partial instance, whereas many branch-and-bound methods use a global prunning criterion such as the value of an LP relaxation. These types of algorithms are incomparable with our model. Since locality is the only restriction we put on computation, it seems difficult to come up with a meaningful relaxation to include branch and bound that does not trivialize our model.

3.2 General Lower bound strategy

Since most of our lower bounds are for the fixed- and adaptive-order models, we present a general framework for achieving these lower bounds. The fully-adaptive-order lower bound for SAT (which yields the fully-adaptive-order Knapsack lower bound by reduction) is more specialized.

Below is a 2-player game for proving these lower bounds for adaptive-order pBT. This is similar to the lower bound techniques for priority algorithms from [9, 17]. The main difference is that there is a set of partial solutions rather than a single partial solution. We will later describe how to simplify the game for fixed-order pBT.

The game is between the Solver and the Adversary. Initially, the Adversary presents to the Solver some finite set of possible input items, P_0. Initially, partial instance PI_0 is empty, and T_0 is the set consisting of the null partial solution. The game consists of a series of phases. At any phase i, there is a set of possible data items P_i , a partial instance PI_i and a set T_i of partial solutions for PI_i. In phase i, $i \geq 1$, the Solver picks any data item $a \in P_{i-1}$, adds a to obtain $PI_i = PI_{i-1} \cup \{a\}$, and chooses a set T_i of partial solutions, each of which must extend a solution in T_{i-1}. The Adversary then removes a and some further items to obtain P_i.

Let n be the first point where P_n is empty. The Solver wins if $|T_i| \leq w(n)$ for all $1 \leq i \leq n$, and there is a $PS_n \in T_n$ that is a valid solution, optimal solution, or approximately optimal solution for PI_n (if we are trying for a search algorithm, exact optimization algorithm, or approximation algorithm, respectively). Otherwise, the Adversary wins. Note that if PI_n is not a well-formed instance for any reason (for example, if we are dealing with a node model for graphs and the item for node j claims that k is a neighbor, but the item for node k does not list j as a neighbor), then it is easy for the Solver to achieve a good solution since the optimization function will always return some default value. Any pBT algorithm of width $w(n)$ gives a strategy for the Solver in the above game. Thus, a strategy for the Adversary gives a lower bound on pBT algorithms.

Our Adversary strategies will usually have the following form. The number of rounds, n, will be fixed in advance. We will maintain the following invariant: For many partial solutions PS to PI_i, there is an extension of PI_i to an instance $A \subseteq PI_i \cup P_i$ so that all valid/optimal/ approximately optimal solutions to A contain PS. We'll call such a partial solution *indispensable*, since if $PS \notin T_i$, the Adversary can set P_i so that $PI_i \cup P_i = A$, ensuring victory. Hence the Solver must keep all indispensable partial solutions in T_i, which results in large width.

For the fixed-order pBT game, the Solver must order all items before the game starts. The Solver must pick the first item in P_i in this ordering as its next data item. Other than that, the game is identical.

4 Interval Scheduling

Interval selection is the classical problem of selecting, among a set of intervals each associated with a profit, a subset of pairwise disjoint intervals so as to maximize their total profits. This can be thought of as scheduling a set of jobs with time-intervals on one machine. When there is more than one machine the task is to schedule jobs to machines so that the jobs scheduled on any particular machine are disjoint; here too, the goal is to maximize the overall profit of the scheduled jobs.

In terms of our formal pBT definitions in section 3, the m machine weighted intervlal selection problem can be represented as follows. A domain or input item $D_i = (s_i, f_i, w_i)$ where s_i (respectively, f_i, w_i) is the start time (respectively, finishing time, profit) of the i^{th} input item. A decision $a_i \in H = \{0, 1, 2, \ldots, m\} \cup \{0\}$ indicates the machine $j \geq 1$ on which interval i is to be scheduled or that the interval is not to be schedule denoted by $a_i = 0$. The objective function f_P^n sums the profits of scheduled intrevals; that is, $f_P^n = \sum_{i:a_i \neq 0} w_i$ if the scheduled jobs constitute a feasible schedule and $-\infty$ if not feasible where a solution is feasible if $\forall j \geq 1[a_i = a_k = j$ implies $i = k$ or $[s_i, f_i) \cap [s_k, f_k) = \emptyset$. [7]

[7]For $a < b < c$, we do not consider $[a, b)$ to intersect with $[b, c)$.

When all the profits are the same, a straight-forward greedy algorithm (in the sense of [9]) solves the problem. For arbitrary profits the problem is solvable by a simple dynamic programming algorithm of dimension m, and hence runtime $O(n^m)$. The way to do this is to order intervals in increasing order of their finishing points, and then compute an m-dimensional table T where $T[i_1, i_2, i_3, \ldots, i_m]$ is the maximum profit possible when no intervals later (in the ordering) than i_j are scheduled to machine j; it is not hard to see that entries in this table can be computed by a simple function of the previous entries.

As mentioned earlier, such an algorithm gives rise to an $O(n^m)$-width, fixed-order pBT algorithm. A completely different approach that uses min cost flows achieves a running time of $O(n^2 \log n)$ ([6]). An obvious question, then, is whether Dynamic Programming, which might seem like the natural approach, is really inferior to other approaches. Perhaps it is the case that there is a more sophisticated way to get a Dynamic Programming algorithm that achieves a running time which is at least as good as the flow algorithm. In this section we prove that there is no better simple Dynamic Programming algorithm than the obvious one, and, however elegant, the simple DP approach is inferior here.

It has been shown in [9] that there is no constant approximation ratio to the general problem using priority algorithms. Our main result in this section is proving that any adaptive-order pBT, even for the special case of proportional profit (i.e. profit = length of interval) Interval Scheduling, requires width $\Omega(n^m)$; thus in particular any simple-DP algorithm requires at least m dimensions. We will first present lower bounds in the fixed-order model where we have constant inapproximability results, and then we will prove a lower bound for the adaptive case, which is considerably more involved.

4.1 Interval Scheduling in the Fixed-Order Model

Theorem 6. *A width $\gamma < \binom{\frac{n-3}{2}}{m}$ fixed-order pBT for interval scheduling with proportional profit on m machines and n intervals cannot achieve a better approximation ratio than $1 - \frac{1}{2m(2\gamma^{1/m}+1)}$.*

Proof. We begin with the special case of $m = 1$. The set of possible inputs are intervals in $[0, 1]$ of the form $[a/W, b/W)$ or $[b/W, 1]$ where $a < b < W$ are integers and W is a function of γ which will be fixed later. More specifically, the set of intervals will be the union $L \cup M \cup R$ where $L = \{[0, q) | q < \frac{1}{2}\}, M = \{[q, s) | q < \frac{1}{2} < s\}$ and $R = \{[s, , 1)\}$ where q, s are of the form a/W as above.

A set of three intervals of the form $[0, q), [q, s), [s, 1]$, $0 < q \leq s < 1$, is called a *complete triplet*. An interval of the form $[0, q)$ is called a *zero interval*, and an interval of the form $[s, 1]$ is called a *one interval*. We say that a set of complete triplets is *unsettled* with respect to an ordering of all of the underlying intervals if either all zero-intervals are before all one-intervals, or vice versa.

We claim that for any ordering of the above intervals and for every t such that $W \geq 2(2t - 1)$, there is a set of t complete triplets which is unsettled. Let S be the sequence induced by the ordering on $L \cup R$. Each of L and R has size at least $2t - 1$. If we look at the first $2t - 1$ elements of S, say the majority of them are (without loss of generality) from L. Select t of these L-intervals and select t R-intervals from the last $2t - 1$ elements of S and match the two sets. This matching, along with the t distinct middle intervals needed to connect each pair of the matching, constitutes a set of t unsettled complete triplets.

Now, consider a pBT program of width $\gamma < (n - 1)/2$ and let $W = 2(2\gamma + 1)$ so as to guarantee there are $\gamma + 1$ unsettled complete triplets. Throw out all intervals not involved in these triplets. Assume, without loss of generality, that all of the zero-intervals come before the one-intervals. Since no two zero-intervals can be accepted simultaneously, and since the width is γ, there is

a zero-interval that is not accepted on any path. The adversary will remove all one intervals except the one belonging to the same triplet as the missing zero-interval. We then have exactly $n = 2(\gamma + 1) + 1$ intervals in the actual set of inputs; that is $\gamma = \frac{n-3}{2}$. With this input it is easy to get a solution with profit 1 by simply picking the complete triplet. But with the decisions made thus far it is obviously impossible to get such a profit, and since the next best solution has profit at most $1 - 1/W$, we can bound the approximation ratio.

The same approach as above works for $m > 1$ machines. That is, if W is large enough so that we have t unsettled triplets, then γ must be at least $\binom{t}{m}$ in order to get optimality. Therefore, given width γ, let t be minimal such that $\gamma < \binom{t}{m}$. Then we achieve profit at most $m - 1/W$ and our approximation ratio is at most $\frac{m-1/W}{m} \leq 1 - 1/(2m(2t+1)) \sim 1 - 1/2m(2\gamma^{\frac{1}{m}} + 1)$. □

Remark 1. *Certain known algorithms (see [20, 26]), which could intuitively be called greedy, allow* semi-revocable decisions*. We can consider this additional strength in the context of pBT algorithms. This means that at any point we can revoke previous accept decisions. We insist only that any partial solution is feasible (e.g. for the Interval Scheduling problem, we do not accept overlapping intervals). This extension applies only to packing problems; that is, where changing accept decisions to rejections does not make a feasible solution infeasible. In contrast to the priority model (with irrevocable decisions), there is a $\frac{1}{4}$-approximation width 1 algorithm when acceptances can be revoked. The proof of Theorem 6 immediately applies to the model with revocable acceptances. Setting $\gamma = 1$ and $m = 1$ in Theorem 6 slightly improves an inapproximation bound of [26] although the bound in [26] applies to adaptive orderings.*

4.2 Interval Scheduling in the Adaptive Model

Theorem 7. *The width of an optimal adaptive pBT for interval scheduling with proportional profits on m machines and n intervals is $\binom{\Omega(n/m^2)}{m}$.*

Proof. We set a parameter $N = 4n/m$. The initial set of data items are the intervals [8] of size less than $1/2N$ in $[0, 1]$ with endpoints i/W where $W = 5mN^2$. We associate a directed acyclic graph $G(\mathcal{I})$ with a set of intervals $\mathcal{I}$ in the following way: the vertices of the graph are the endpoints of $I \in \mathcal{I}$ and there is an edge from node s to node t if $[s, t) \in \mathcal{I}$ for $t < 1$ or $[s, t] \in \mathcal{I}$ for $t = 1$. We say that a point s is *zero connected* if there is a path from 0 to s, and similarly s is *one connected* if there is a path from s to 1. Notice that s is zero connected if and only if there is a set of disjoint intervals whose union is $[0, s]$. An interval with endpoints in $(0, 1)$ is called an *internal* interval.

In the first n phases the adversary applies the following two elimination rules (for eliminating future intervals).

1. Cancel all internal intervals both of whose endpoints are endpoints of previous intervals. (For example, if $[0, .3), [.2, .3)$ and $[.21, .23)$ were revealed, then the intervals $[.2, .21)$, $[.2, .23)$, $[.21, .3)$ and $[.23, .3)$ must be cancelled, but $[0, .2)$ and $[.2, .35)$ should not. Note that this rule guarantees that the graph $G(\mathcal{I})$ associated with the intervals after n phases has the property that any *undirected* cycle must contain either 0 or 1.

2. Cancel all intervals ending (starting) at r if $r \in (0, 1/3)$ ($r \in (2/3, 1)$) and there are m intervals ending (starting) in r.

Our goal now is to show that after N intervals are observed, there must be many indispensable classes of solutions. We let $\mathcal{P}$ be the set of N intervals that have been revealed so far. Let J be

[8] As in Theorem 6 our intervals are of the form $[s, t)$ for $t < 1$ or of the form $[s, 1]$.

an interval that is not overlapping any of the intervals in $\mathcal{P}$. Since there are N intervals of length at most $1/2N$ in $\mathcal{P}$, there must be such intervals. We now claim that there are at least half of the intervals in $\mathcal{P}$ that are contained in $[0, 2/3)$ and are left of J or that are contained in $(1/3, 1]$ and are right of J. If J is contained in $(1/3, 2/3)$ then taking (by majority) either the intervals to the left or to the right of J will do. Otherwise there must be uncovered intervals in both $[0, 1/3]$ as well as $[2/3, 1]$ (recall their total lengths amount to $1/2$). Now, there are at least as many as $N/2$ intervals contained in $[0, 2/3)$ or that are contained in $(1/3, 1]$, and these intervals will do. We now may assume, without loss of generality, the existance of a subset $\mathcal{P}'$ of $\mathcal{P}$ with $N/2$ intervals that are contained in $[0, 2/3)$ and that are to the left of an uncovered interval J. The following lemma guarantees certain useful combinatorial guarantees about $G(\mathcal{P}')$.

Lemma 8. *At least one of the following holds*

I. There are $N/5m$ vertices of $G(\mathcal{P}')$ in $(0, 2/3)$ with left dgeree (i.e. indegree) at least 1.

II. There are $\Omega(N)$ vertices of $G(\mathcal{P}')$ with indegree 0 and that have an immediate successor which is not 0 connected.

Proof. Assume that the first case doesn't hold. Let G be the set of intervals in $\mathcal{P}'$ whose right endpoints are in $(0, 1/3)$, and let $|G| = g$. Similarly let H be the set of intervals in $\mathcal{P}'$ whose right endpoints are in $[1/3, 2/3)$, and let $|H| = h$. Denote by $\deg_L(v)$ the left degree of v. Then because of elimination rule 2, we have:

$$g = \sum_{r \in (0,1/3]} \deg_L(r) \leq m \cdot |\{r \in (0, 1/3] : \deg_L(r) > 0\}| \leq m \cdot N/5m = N/5.$$

Now, since $g + h = |\mathcal{P}'| \geq N/2$ it follows that $h \geq N/2 - N/5 = 3N/10$. Now because of elimination rule 1, there are at least $3N/10$ distinct vertices mentioned in the intervals in H. Call these vertices $V(H)$.

Let $V_L(H) \subset V(H)$ be those vertices in $V(H)$ that have indegree 0. First, we observe that the size of $V_L(H)$ is at least $\Omega(n)$, since if is not, then there are too many vertices in $G(\mathcal{P}')$ with indegree at least 1, violating our assumption that the first case doesn't hold. Notice that all vertices in $V_L(H)$ must be left endpoints of some interval in H. It is left to argue that for each $v \in V_L(H)$, $right(v)$ cannot be zero connected, where $right(v)$ is the matching right endpoint of v in H. To see this, recall that for each $v \in V_L(H)$, $right(v)$ is in $(1/3, 2/3)$. Thus in order for $right(v)$ to be 0 connected, there would need to be a path from 0 to $right(v)$; but this would involve more than n intervals that have indegree at least 1, thus again violating the fact that the first case doesn't hold.

Hence we have shown that whenever case (I) doesn't hold, case (II) must. □

We now show that in any of the cases guaranteed by the lemma, the algorithm must maintain a large set of solutions in order to allow for an optimal (complete) solution.

Case I. We define a projection function π from partial solutions (namely, an assignment of the first n intervals to the different machines or to none) into subsets of $\mathbb{R}$ as follows.

$$r \in \pi(PS) \text{ iff there is a machine } M \text{ such that } r \text{ is the rightmost location assigned to } M$$

(Notice that by definition $|\pi(PS)| \leq m$). Call the set of points with positive indegree S. Let $\mathcal{PS}$ be the set of all partial solutions after the first N intervals are revealed. We claim that any algorithm must maintain $\binom{S}{m}$ partial solutions, one corresponding to every subset of S of size m. Specifically,

for every subset $\{u_1, u_2, \ldots, u_m\}$ of S, a partial solution in which the rightmost points covered by the m machines are $\{u_1, u_2, \ldots, u_m\}$ must be considered.

To prove this, fix a particular subset $\{u_1, u_2, \ldots, u_m\}$ of S. We create the following remaining input $\mathcal{Q}$. For each j we create a path γ_j connecting u_j to 1 using intervals of length $\geq 1/3N$. Further, the points p that are endpoints in the paths are disjoint except for the point 1. (It is important to note that the fact that $u_j \notin (2/3, 1)$ is used here as otherwise elimination rule 2 may have removed intervals that are essential to the construction.)

For each u_j, $j \in [1, m]$, let v_j be a left neighbour of u_j. Our goal is now to connect each v_j to 0 by m disjoint paths $\delta_1, \ldots, \delta_m$, possibly using additional valid intervals that will be part of $\mathcal{Q}$. We now need the simple fact that there is a way to add intervals to P so that the following two conditions hold.

- First, all v_j will be connected to 0 (in order to have a complete solution).
- Secondly, if a point had m left neighbours in P, then no left neighbours will be added by the new intervals. This second condition is necessary in order to be consistent with elimination rule 2.

To see that the above two conditions can be satisfied, we simply consider an extension of all v_j to the left, using existing intervals from P as much as possible. This is done until we managed to connect all v_j to 0. Notice that if a point v_j has left degree m in $G(\mathcal{P}')$, then we never need to add more intervals ending in v_j even if all paths need to go through this point. This explains why we should insist on having degree bound m and not less.

It is left to show that there is no complete solution other than this one (modulo permutations of the machines). First, notice that $\gamma_1, \gamma_2, \ldots, \gamma_m$ must all be used in order to get a complete solution as otherwise the interval J would not be covered m times.

It remains to show that in any complete solution that extends $\gamma_1, \ldots, \gamma_m$, the intervals $[v_j, u_j]$, $j \in [1, m]$, together with $\delta_1, \ldots, \delta_m$, must be used. If it were the case that for each point u_i, no new intervals of the form $[r, u_i]$ were added, then it would be clear that the intervals $[v_j, u_j]$ must be used, and this would then imply that the intervals δ_j must be used as well, in order to form a complete solution. However, it may be the case that for some $u_j > u_i$, such an interval $[r, u_i]$ was added in order to connect u_j to 0. (For example, it might be that $[u_i, u_j]$ is part of $\mathcal{P}'$, and then some interval $[r, u_i]$ would have to be added in order to connect u_j to 0.) When this happens, we say that u_j *dominates* u_i. Recall that in order to get a complete solution u_i must be connected to zero and no intervals in $\mathcal{P}'$ to the right of u_i may be used. As mentioned above, if no u_i is dominated, then there are no new intervals ending at u_i that were added, and so any solution in which $u_i \notin \pi(PS)$ cannot be made complete. For the general case we consider u_m first (assuming without loss of generality that $u_1 < u_2 \ldots < u_m$). Since u_m is not dominated it follows that a complete solution must connect u_m to zero, so that the interval ending at u_m is in $\mathcal{P}'$. Therefore if $\mathcal{P}'S$ is to be completed to a complete solution it must satisfy $u_m \in \pi(PS)$. An important observation is that if u_m dominates some u_i, then the interval $[r, u_i] \in \delta_m$ must be used to connect u_m to zero. Thus, we may consider only $u_1, \ldots, u_{m-1}$ and we may ignore the dominance relation involving u_m. Therefore we may continue arguing this way by downward induction, to show that $u_i \in \pi(PS)$ for all i.

Case II. Let L be the set of points as is guaranteed in case 2. For each $p_i \in L$ pick some interval with p_i as the left endpoint, and q_i as the right endpoint, where q_i is not zero connected. Call the produced set of intervals $\mathcal{I}$. We now argue that for any subset $\mathcal{J} = \{[p_j, q_j]\}_{j=1,\ldots m}$, the solution containing these intervals on the m machines (one per machine) and nothing else, is indispensible. For each j we connect p_j to zero and q_j to 1 using intervals of length at least $1/3N$; we call this

set of intervals F_j. We additionally require that the endpoints of F_j avoid all edgepoints of $\mathcal{I}$ and of $F_1, \ldots, F_{j-1}$ (except for p_j and q_j). This is possible as long as W as large enough. Notice that there are at most $3mN$ intervals used in $F = \cup_j F_j$ and so if $W = 5mN$ this requirement can be satisfied.

If we accept exactly $[p_j, q_j]$ and F_j on machine j, we get a complete solution. We next show that there is no complete solution other than this one (modulo permutations of the machines) over $\mathcal{P}' \cup F$. Consider the graph $G(\mathcal{P}' \cup F)$. Then there is a complete solution to all m machines exactly when there are m edge-disjoint paths from 0 to 1. Our goal is therefore to show that the existence of such paths implies that all edges $[p_j, q_j]$ are used. As we observed in the previous case, the only way to get m disjoint paths crossing over the gap J is to pick all edges corresponding to the connections in F from the q_is to 1. Therefore a complete solution must contain m disjoint paths from 0 to $q_1, \ldots, q_m$.

None of the q_j's is connected to 0 in $G(\mathcal{P}')$, hence in connecting all of them to 0 we must use the only points that were made zero connected when adding F_j's, namely $p_1, \ldots, p_m$. It is left to show that this requires using the intervals $[p_j, q_j]$. There is one subtle point that is left to argue. Suppose that the intervals I are $[p_1, q_1]$ and $[p_2, q_2]$, but that in the graph $G(\mathcal{P}')$, there are edges (p_1, q_1), (p_1, q_2), (p_2, q_1), (p_2, q_2). We need to argue that any complete solution must use the intervals $[p_1, q_1], [p_2, q_2]$ (and not the intervals $[p_2, q_1], [p_1, q_2]$.) In order to argue this, suppose for sake of contradiction that there is a second matching between the p_i's and the q_i's that allows us to obtain a complete solution. Then the union of the two matchings forms an undirected cycle in $G(\mathcal{P}')$. But this is not possible since it violates elimination rule 1.

□

5 The Knapsack and Subset-Sum problems

The Knapsack problem takes as input n non-negative integer pairs denoting the weight and profit of n items, $\{(x_1, p_1), \ldots, (x_n, p_n)\}$ and another number N, and returns a subset $S \subseteq [n]$ that maximizes $\sum_{i \in S} p_i$ subject to $\sum_{i \in S} x_i \leq N$. This is a well known NP-hard problem which, on the positive side, has an FPTAS. In this section we study the width-approximation tradeoff for pBT algorithms for the problem.

Narrow pBT algorithms As a warmup, we start by observing that width-1 and width-2 pBT algorithms for Knapsack behave dramtically differently. Recall the simple 1/2-approximation algorithm that either accepts or rejects the highest profit item, and then greedily chooses items when ordered by their decreasing profit to weight ratio. This algorithm can be clearly captured by an adaptive order width 2 pBT that orders the highest profit item first, and orders the rest by their profit to weight ratio as above[9] .

We next show an $n^{1/4}$ inapproximability result for Knapsack for width-1 pBT (ie, priority algorithms), where n is the number of items.[10] The initial input contains big items of weight 1 and profit 1, medium items of weight $1/n$ and profit $n^{-1/2}$ and small items of weight $1/n^2$ and profit $1/n$. Each appears n times. We let $N = 1$. The adversary waits until either an item is accepted or $n - n^{3/4}$ items are rejected. If a big item was accepted then the adversary leaves only medium

[9] At the expense of introducing yet another term, we might call such an algorithm "weakly adaptive" in that the ordering function $r^k_{\mathcal{A}}$ depends on k but not on the arguments of $r^k_{\mathcal{A}}$ as defined in Defintion 1. We could also modify the definition of fixed-order priority and pBT algorithms to allow such dependence on k but that would seem to violate the spirit of the intended definition.

[10] We note, however, that there is an adaptive-order width-1 pBT with revocable acceptances that achieves a 1/2 approximation.

items. The algorithm then achieves profit 1, while the optimum is at least $n^{3/4} \cdot n^{-1/2} = n^{1/4}$. If a medium or small item was chosen, the adversary leaves only big items. Now the algorithm achieves at most $n^{-1/2}$ profit while the optimum is 1. In the case where $n - n^{3/4}$ items were rejected, the adversary will leave only small items. The algorithm can then get at most $n^{3/4}/n = n^{-1/4}$ while optimum is attained by accepting all items totalling to a profit of at least 1.

Wide pBT algorithms We now move to the other side of the spectrum of the width-approximation tradeoff, ie, we consider the width needed for exact solution or for a very good approximation of the problem. There are well-known simple-DP algorithms solving the Knapsack problem in time polynomial in n and N, or in time polynomial in n and $\Pi = \max_{i=1}^{n} p_i$. Therefore, with that much width the problem is solvable by a pBT.

We prove that it is not possible to solve the problem with an adaptive-order pBT algorithm that is subexponential in n (and does not depend on N or Π). Further, we provide an almost tight bound for the width needed for an adaptive-order pBT that approximates the optimum to within $1-\epsilon$. We present an upper bound (due to Marchetti-Spaccamela) of $(1/\epsilon)^2$ based on a modification of the algorithms of Ibarra and Kim [27] and Lawler [32] that uses dynamic programming and a lower bound of $\Omega((1/\epsilon)^{1/3.17})$. We notice that both our lower bounds in this section hold for the Subset-Sum problem, the proportional profit variant of the Knapsack problem where for each item the profit is equal to its weight.

Theorem 9. *The width of an optimal adaptive-order pBT for the Subset-Sum problem is at least* $\binom{n/2}{n/4} = \Omega(2^{n/2}/\sqrt{n})$.

Proof. We are tempted to try to argue that having seen only part of the input, all possible subsets of the current input must be maintained as partial solutions or else an adversary has the power to present remaining input items that will lead to an optimal solution with a solution the algorithm failed to maintain. For an online algorithm, when the order is adversarial, such a simple argument can be easily made to work. However, the ordering (and more so the adaptive ordering) power of the algorithm requires a more subtle approach.

Let N be some large number which will be fixed later. (Since a simple DP of size poly(n,N) exists, it is clear that N must be exponential in n.) Our initial set of items are integers in $I = [0, \frac{8}{3} \cdot N/n]$. Take the first $n/2$ items, and following each one, apply the following "general-position" rule to remove certain items from future consideration: remove all items that are the difference of the sums of two subsets already seen; also remove all items that complete any subset to exactly N (ie all items with value $N - \sum_{i \in S} a_i$ where $a_1, a_2, \ldots$ are the numbers revealed so far, and S is any subset). These rules guarantee that at any point, no two subsets will generate the same sum, and that no subset will sum to N. Also notice that this eliminates at most $3^{n/2}$ numbers so we never exhaust the range from which we can pick the next input provided that $3^{n/2} << N$.

Call the set of numbers seen so far P and consider any subset Q contained in P of size $n/4$. Our goal is to show that Q is indispensable; that is, we want to construct a set $R = R_Q$ of size $n/2$ consisting of numbers in the feasible input with the following properties.

1. $P \cup R$ does not contain two subsets that have the same sum.
2. $\sum_{i \in Q} a_i + \sum_{i \in R} a_i = N$

The above properties indeed imply that Q is indispensable since obviously there is a unique solution with optimal value N and, in order to get it, Q is the subset that must be chosen among the elements

of P. We thus get a lower bound on the width which is the number of subsets of size $n/4$ in P; namely $\binom{n/2}{n/4} = \Omega(2^{n/2}/\sqrt{n})$.

How do we construct the set R? We need it to sum to $N - \sum_{i \in Q} a_i$, while preserving property 1. The elements in R must be among the numbers in I that were not eliminated thus far. If R is to sum to $N - \sum_{i \in Q} a_i$, then the average of the numbers in R should be $a = \frac{2}{n} \cdot (N - \sum_{i \in Q} a_i)$. Since $0 \leq \sum_{i \in Q} a_i \leq (n/4)(8N/3n) = 2N/3$, we get $\frac{2}{3}N/n \leq a \leq 2N/n$. This is good news since the average is not close to the extreme values of I, owing to the fact that the cardinality of R is bigger than that of Q. We now need to worry about avoiding all the points that were eliminated in the past and the ones that must be eliminated from now on to maintain property 1. The total number of such points, U, is at most the number of ways of choosing two disjoint subsets out of a set of n elements, namely $U \leq 3^n$.

Let $J = [a-U, a+U]$. We later make sure that $J \subset I$. We first pick $n/2-2$ elements in J that (i) avoid all points that need to be eliminated, and (ii) sum to a number w so that $|w - a \cdot (n/2-2)| \leq U$. This can be done by iteratively picking numbers bigger/smaller than a according to whether they average to below/above a. To complete we need to pick two points $b_1, b_2 \in I$ that sum to $v = \frac{n}{2}a - w$ and so that $b_1, b_2, b_1 - b_2$ are not the difference of sums of two subsets of the $n-2$ items picked so far. Assume for simplicity that $v/2$ is an integer. Of the $2U+1$ pairs $(v/2 - i, v/2 + i)$, where $i = 1 \ldots 2U+1$, at least one pair b_1, b_2 will have all the above conditions. All that is left to check is that we never violated the range condition, ie we always chose items in $[0, \frac{8}{3} \cdot N/n]$. We can see that the smallest number we could possibly pick is $a - U - (2U+1) \geq \frac{2}{3}N/n - 3U - 1$. Similarly the biggest number we might take is $a + 3U + 1 \leq 2N/n + 3U + 1$. These numbers are in the valid range as long as $\frac{2}{3}N/n \geq 3U + 1$. Since $U \leq 3^n$ we get that $N = 5n3^n$ suffices. □

A more careful analysis of the preceding proof yields the following width-approximability tradeoff.

Theorem 10. *For any ϵ, Knapsack can be $(1-\epsilon)$-approximated by a width $(1/\epsilon)^2$ adaptive-order pBT algorithm. For any $\epsilon \geq 2^{-\delta n}$ for some universal constant δ, Knapsack cannot be $(1-\epsilon)$-approximated by any such algorithm of width less than $(1/\epsilon)^{1/3.17}$. The lower bound holds even for the Subset-Sum problem.*

Proof. **Lower Bound.** We take the existing lower bound for the exact problem and convert it to a width lower bound for getting a $1-\epsilon$ approximation. Recall that the resolution parameter N in that proof had to be $5n3^n$ for getting a width lower bound of $2^{n/2}/\sqrt{n}$. For a given width γ, we might hope to lower the necessary resolution in order to achieve an inapproximability result. We consider a Knapsack instance with u items that require exponential width (as is implied by Theorem 9), and set N, the parameter for the range of the numbers to $5u3^u$. If u is such that $\gamma < 2^{u/2}/\sqrt{u}$ then this problem cannot be solved optimally by a width-γ pBT algorithm. Recall, the optimum is N, and the next best is $N-1$, and so the best possible approximation we can get is

$$(N-1)/N \sim 1 - 1/(5u3^u) \sim 1 - \tilde{O}(\gamma^{-2\log_2 3}).$$

Therefore $\Omega((1/\epsilon)^{1/3.17})$ width is required to get a $1-\epsilon$ approximation. To make the lower bound work for any number of items, we simply add $n-u$ 0-items to the adversarial input.

Upper Bound (Marchetti-Spaccamela). We first sketch Lawler's algorithm (built upon that of Ibarra and Kim) to approximate Knapsack. We call the solution that takes items by nonincreasing order of their profit/weight as long as possible "the canonical solution". Given parameters K and T: Round all items of profit at least T down to the closest multiple of K. Let τ be the optimum and $\tau(T)$ be the optimum restricted to items of profit at least T. For each

one of the possible $\tau(T)/K$ different profits, find the lowest weight bundle of large-profit items attaining it using dynamic programming. Now supplement each such solution with the canonical solution for the remaining items (with the remaining size of Knapsack). Simple calculations done in [32] show that the additive error in this solution is at most $K\tau/T + T$. This would have been enough, if only the algorithm knew a good estimator τ' to $\tau(T)$ in advance. Specifically, suppose $\tau(T)/2 \leq \tau' \leq \tau(T)$, then we can set $K = \epsilon^2\tau'/4$ and $T = \epsilon\tau'/2$ to get an additive error of at most $\epsilon\tau(T) \leq \epsilon\tau$. We now show that an adaptive-order pBT algorithm can achieve this balance of parameters using width $8/\epsilon^2$: Start with $\tau' = \max_i p_i$; set $K = \epsilon^2\tau'/4$ and $T = \epsilon\tau'/2$. As long as there are items with profit at least T take them (in any order) and keep solutions for all possible profits up to $2\tau'$ in multiples of K. If there is a solution that is at least $2\tau'$ update $\tau' = 2\tau'$. Set K, T again by the above relation to τ'. Notice that since the scaling factors double when we reset them, we are halving the resolution and removing possible items from the first stage of the algorithm. This means that whatever partial solutions we were maintaining before the parameter adjustment encompass those we want to maintain afterwards. We continue until all items have profit at most T. At this point we have maintained all solutions of the high-profit items in resolution K (notice the invariant $\tau(T) \leq 2\tau'$). From this point on, each one of the $2\tau'/K = 8/\epsilon^2$ partial solutions is completed greedily with items of profit smaller than T. □

Remark 2. *We can extend the proof of theorems 9 and 10 so as to allow revocable acceptances (see remark 1) with slightly worse parameters. Recall that in Theorem 9 we look at $n/2$ elements in the range $[0, N/2]$ and then show that all $n/4$ size subsets are indispensable. We can modify the proof so that this range is $[aN/n, bN/n]$ for suitable constants $a, b > 2$; we look at the first $n/2$ items and similar to the arguments in Theorem 9, show that all subsets of size $n/(2b)$ are indispensable. In the semi-revocable model it is no longer the case that this supplies a width lower bound of $\binom{n/2}{n/(2b)}$, but instead we should look for a family of feasible sets $\mathcal{F}$ such that any of the indespensible sets of size $n/(2b)$ is contained in some $F \in \mathcal{F}$. But, and this is the crucial point, feasible sets must be of size $\leq n/a$, and so every $f \in \mathcal{F}$ contains at most $\binom{n/a}{n/(2b)}$ sets, and a counting argument immediately shows that $|\mathcal{F}| \geq \binom{n/2}{n/(2b)} / \binom{n/a}{n/(2b)} = 2^{\Omega(n)}$.*

6 Satisfiability

The search problem associated with SAT is as follows: given a boolean conjunctive-normal-form formula, $f(x_1, \ldots, x_n)$, output a satisfying assignment if one exists. There are several ways to represent data items for the SAT problems, differing on the amount of information contained in data items. The simplest *weak* data item contains a variable name together with the names of the clauses in which it appears, and whether the variable occurs positively or negatively in the clause. For example, the data item $< x_i, (j, +), (k, -) >$ means that x_i occurs positively in clause C_j, and negatively in clause C_k, and these are the only occurrences of x_i in the formula. The decision is whether to set x_i to 0 or to 1. We also define a *strong* model in which a data item fully specifies all clauses that contain a given variable. Thus $D_i =< x_i, C_1, C_2, ..., C_k >$, where the $C_1, ..., C_k$ are a complete description of the the clauses containing x_i. Note that, unlike the Interval Scheduling and Knapsack data items, the various types of SAT data items (and the node data items for graphs which we mention later in this section) are not independent of one another. For example, in a well-formed instance of 2SAT in the weak data item model, there better not be three different variables that all assert that they appear in a given clause. Such considerations constrain a successful adversary in a lower bound argument.

In general we would like to prove upper bounds for the weak data type, and lower bounds for

the strong data type. We will show that 2SAT (for the strong data type) requires exponential time in the fixed-order pBT model, but has a simple linear time algorithm in the adaptive-order pBT model (for the weak data type). Thus, we obtain an exponential separation between the fixed- and adaptive-order pBT models. Next, we give exponential lower bounds in the fully-adaptive-order model for 3SAT (strong data type).

6.1 2-Satisfiability in the Fixed-Order Model

In this section we show that the fixed-order pBT model cannot efficiently solve 2SAT (or c-approximate MAX2SAT for $c > 21/22$).

Theorem 11. *For sufficiently large n, any fixed-order pBT algorithm for solving 2SAT on n variables requires width $2^{\Omega(n)}$. This lower bound holds for the strong data type for SAT.*

Proof. Consider a set of variables $x_1, \ldots, x_n$. Each variable x_i gives rise to many possible items, each of which will describe exactly two equations that hold for x_i. In the end, we will select one item from either (1) or (2) for each x_i:
(1) For some choice of $j \neq k \in [n] \setminus \{i\}$, $x_j = x_i = x_k$, or $x_j = x_i \neq x_k$ or $x_j \neq x_i = x_k$,
(2) For some choice of $j \in [n] \setminus \{i\}$, $0 = x_i = x_j$ or $x_j = x_i = 1$.
Of course, each of these constraints must be represented by a small constant (at most 4) number of clauses.

Call two items *disjoint* if they mention disjoint sets of variables. An r-chain is a chain of equations of the form

$$0 = y_1 = y_2 \stackrel{?}{=} \ldots \stackrel{?}{=} y_{r-1} = y_r = 1,$$

where $y_1, \ldots, y_r \in \{x_1, \ldots, x_n\}$ and $\stackrel{?}{=}$ is either $=$ or $\neq$.

Consider any ordering of the initial set of input items. Let M be the first $m = \lfloor n/11 \rfloor$ disjoint (1)-items in the ordering. Suppose these items are called y_6^i, $i \leq \lfloor n/11 \rfloor$, and let $y_5^i \stackrel{?}{=} y_6^i \stackrel{?}{=} y_7^i$ be the content of these items. This triple will form the middle of an 11-chain. For each i, choose eight remaining variables in order to extend the chain to an 11-chain. That is, partition the remaining variables into $\lfloor n/11 \rfloor$ disjoint sets (with possibly some items leftover if n is not divisible by 11), each of size 8, so that for each i, we have an 11-chain involving the sequence of variables: $y_1^i, y_2^i, \ldots, y_{11}^i$.

The adversary removes items to be consistent with the following 11-chains for each i:

$$0 = y_1^i = y_2^i \stackrel{*}{=} y_3^i = y_4^i = y_5^i \stackrel{?}{=} y_6^i \stackrel{?}{=} y_7^i = y_8^i = y_9^i \stackrel{*}{=} y_{10}^i = y_{11}^i = 1.$$

That is, the adversary specifies (by removal of items) all equations in the chain (in particular, those involving y_6^i are consistent with M) except those relating y_2^i to y_3^i and y_9^i to y_{10}^i.

The adversary stops the game after phase q, the phase where we see the last item of M. Note that at phase q, for each i: (i) the item y_6^i has been revealed (so one of the three possibilities has been revealed for the inequalities on either side of y_6^i); (ii) the items y_2^i, y_3^i, y_9^i and y_{10}^i have not yet been revealed; and (iii) all other items in the 11-chain may or may not be revealed, but if they have been revealed, they are consistent with the equalities written above. Let P denote the set of revealed items after q phases of the game.

We want to show that each of the $2^{\lfloor n/11 \rfloor}$ assignments to the y_6^i variables must be maintained by the algorithm at level q of the game. More formally, we partition the set of all decisions on P into equivalence classes, where two partial solutions ρ_1 and ρ_2 are equivalent if they are identical over the y_6^i variables. We will show that the set of all such equivalence classes is indispensable.

Consider one such equivalence class, and let α be the underlying assignment to the y_6^i variables. If the algorithm does not maintain a partial solution consistent with α, then the adversary can

further specify each of the 11-chains so that at least one chain will be left unsatisfied. Consider chain i: there are several cases depending on the actual inequalities that are in P on the left and right of y_6^i. The first case is when $y_5^i = y_6^i = y_7^i$ is in P. If $\alpha(y_6^i) = 0$, then the algorithm throws away all future inputs on the chain i except those consistent with the following picture:

$$0 = y_1^i = y_2^i = y_3^i = y_4^i = y_5^i = y_6^i = y_7^i = y_i^i = y_9^i \neq y_{10}^i = y_{11}^i = 1$$

Otherwise, if $\alpha(y_6^i) = 1$, the algorithm throws away all future inputs on chain i except those consistent with:

$$0 = y_1^i = y_2^i \neq y_3^i = y_4^i = y_5^i = y_6^i = y_7^i = y_8^i = y_9^i = y_{10}^i = y_{11}^i = 1.$$

The other two cases (when $y_5^i \neq y_6^i = y_7^i$, and when $y_5^i = y_6^i \neq y_7^i$) are handled similarly.

Thus we have shown that under this adversary strategy, the algorithm must consider at least $2^{\lfloor n/11 \rfloor}$ assignments. □

We can also consider the associated optimization problem MAXSAT: find an assignment to the variables of a CNF that maximizes the number of satisfied clauses. We remind the reader that *pBT* inapproximation results are incomparable with complexity-theoretic hardness of approximation results since *pBT* algorithms are incomparable with, say, polytime algorithms. It is a curious coincidence that the inapproximation ratio (21/22) that we establish for *pBT* algorithms matches the best known *NP*-hardness of approximation result for MAX2SAT [23]. This *NP*-hardness result is proven for instances of exact-MAX2SAT, where every clause has exactly two literals, while the hard examples we give in our lower bound contain some clauses with only one literal. We can use a similar technique to establish a slightly weaker inapproximation result for exact-MAX2SAT; namely, 27/28. On the positive side, we note that the well-known derandomization of the naive randomized algorithm (see, for example, [33, 35]) for exact-MAX2SAT (respectively, MAX2SAT) achieves approximation ratio 3/4 (respectively, 1/2) and can be implemented as a fixed-order priority algorithm (width-1 *pBT*).

Theorem 12. *For any $\epsilon > 0$, there exists a $\delta > 0$ such that for all sufficiently large n, any fixed-order pBT algorithm for solving MAX2SAT on n variables requires width $2^{\delta n}$ to achieve a $\frac{21}{22} + \epsilon$ approximation. Again, this lower bound holds for the strong data type for SAT.*

Proof. The game is played exactly as in the proof of Theorem 11. Notice that, when the algorithm does not cover a certain equivalence class with partial assignment α, the adversary forces at least one 11-chain to be unsatisfied. In particular, 2 out of the 22 clauses representing the 11-chain are unsatisfied (one associated with $y_2^i \stackrel{*}{=} y_3^i$ and one with $y_9^i \stackrel{*}{=} y_{10}^i$). Now fix $\epsilon > 0$ and let $\delta = (\log e)11\epsilon^2$. If the algorithm maintains $k < 2^{\delta n}$ partial solutions at phase q, then it can cover at most k of the α-equivalence classes. The probability that a random α-assignment agrees with a fixed α-assignment on more than a $(1/2 + 11\epsilon)$-fraction of the $m = \lfloor n/11 \rfloor$ variables that α sets is at most $e^{-(11\epsilon)^2 m} = e^{-11\epsilon^2 n}$. If the algorithm maintains fewer than k α-assignments, then the adversary can find an assignment α^* that agrees with each of the k α-assignments on at most a $(1/2 + 11\epsilon)$-fraction. Hence, in a $(1/2 - 11\epsilon)$-fraction of the 11-chains, 1/11 of the clauses are unsatisfied by any of the algorithm's partial solutions, so the algorithm leaves a $(1/22 - \epsilon)$-fraction of all the clauses unsatisfied. □

6.2 Vertex Cover

We note that Johnson's greedy 2-approximation for vertex cover can be implemented as a fixed-order priority algorithm (i.e. a width-1 *pBT*). Here the items are vertices with their adjacency lists

and (using any ordering) each vertex is accepted (included in the vertex cover) iff it is an end-point of a maximal matching that is being constructed. A similar idea to the 2SAT inapproximation can be used to show a constant inapproximation ratio for exponential-width fixed-order pBT algorithms computing Vertex Cover (with the same input representation). Again note the incomparability with NP-hardness of approximation results.

Theorem 13. *For any $\epsilon > 0$, there exists a $\delta > 0$ such that for all sufficiently large n, any fixed-order pBT algorithm for solving Vertex Cover on n vertices requires width $2^{\delta n}$ to achieve a $\frac{13}{12} - \epsilon$ approximation.*

Proof. (sketch) Each node x_i gives rise to two types of items
(1) For some $j \neq k \in [n] \setminus \{i\}$, $x_j - x_i - x_k$ (that is, x_i has neighbors x_j and x_k).
(2) For some $j \in [n] \setminus \{i\}$, $x_j - x_i$.
Let M be the first $m = \lfloor n/13 \rfloor$ disjoint (1)-items in the ordering: $\{y_6^i - y_7^i - y_8^i\}_{i=1}^m$. Partition the vertices into m groups of size 13 so that each contains an item from M. When the algorithm has seen every item in M, the adversary selects one of the two following configurations for each group:

$$y_1^i - y_2^i - y_3^i - y_4^i - y_5^i - y_6^i - y_7^i - y_8^i - y_9^i - y_{10}^i - y_{11}^i - y_{12}^i - y_{13}^i,$$

or

$$y_1^i - y_2^i - y_4^i - y_5^i - y_6^i - y_7^i - y_8^i - y_9^i - y_{10}^i - y_3^i - y_{11}^i - y_{12}^i - y_{13}^i.$$

If the algorithm has included y_7^i in the vertex cover and the adversary chooses the first configuration, then the algorithm is forced to cover the 13-chain with 7 vertices when 6 would have been enough; likewise if the algorithm has excluded y_7^i and the adversary chooses the second configuration. Note again that the algorithm cannot predict the adversary's choices when it is deciding about vertices in M because vertices $y_2^i, y_3^i, y_4^i, y_{10}^i, y_{11}^i$ are all clouded in obscurity since they are the centerpoints of (1)-items disjoint from M.

Again, if the algorithm maintains only $2^{\delta n}$ assignments to $\{y_7^i\}_{i=1}^m$ for δ being a sufficiently small constant, then the adversary can choose an assignment such that $1/2 - \epsilon'$ of the groups are non-optimal (for some ϵ'), giving an approximation no better than $(1/2 + \epsilon')1 + (1/2 - \epsilon')7/6 = 13/12 - \epsilon$. □

6.3 2-Satisfiability in the Adaptive-Order Model

In this section, we show that allowing adaptive ordering avoids the exponential blow up in the number of possible assignments that need to be maintained. Specifically, we give a linear width pBT algorithm for 2SAT in the adaptive-order model.

Theorem 14. *There is a width-$O(n)$ adaptive-order pBT algorithm for 2SAT on n variables. Further, this upper bound holds for the weak data type for SAT.*

Proof. Consider the standard digraph associated with a 2SAT instance. Recall that the standard algorithm for the problem goes via finding the strongly connected components of this graph. This does not fit immediately into the pBT model since, here, whenever we observe a variable we must extend partial solutions by determining its value. The algorithm we present uses the simple observation that a path of literals in the digraph, such as $l_1 \to l_2 \to l_3 \to \ldots \to l_m$, has only linearly many potentially satisfying assignments; namely the literals along the path must be set to 0 up to a certain point, and to 1 from that point on, which means at most $m+1$ possible valid assignments to the literals involved. Since the algorithm will adaptively explore only simple paths, essentially using DFS, it will never be maintaining more than a linear number of assignments. The pBT tree

that we generate, however, will not correspond to a DFS tree. Instead, think of each branch of the pBT tree as conducting the same DFS search in parallel, with each branch maintaining one of the linearly many assignments mentioned above.

Using an adaptive ordering we can "grow" a path of literals as follows. Start with an arbitrary variable item x_1 and let l_1 be a literal corresponding to x_1. Having viewed the item for x_1, we now know the names of the clauses that l_1 and $\neg l_1$ appear in. The algorithm then chooses a new (that is, unseen so far) variable x_2 (if there is one) such that there is an edge $l_1 \to l_2$ for some literal l_2 corresponding to x_2 (that is, build an ordering that prefers variables x_2 that appear in clauses of the form $(\neg l_1 \vee x_2)$ or $(\neg l_1 \vee \neg x_2)$). Then, it continues to look for a path $l_1 \to l_2 \to l_3$ and so on. Each time we see a literal corresponding to a new variable such as l_3, we extend each branch of the pBT tree as follows: on a branch that set $l_2 = 1$, set $l_3 = 1$; on a branch that set $l_2 = 0$, create two new branches setting l_3 to 0 and 1, respectively. As long as this is possible we need to maintain only a linear number of solutions.

When the path $l_1 \to \cdots \to l_i$ is no longer extendable in this fashion, it must mean that (i) the only outneighbors of l_i are literals corresponding to already-seen variables, or (ii) l_i has outdegree 0. Case (i), has two subcases: if there is an edge $l_i \to l$, where $l = \neg l_j$ for some $j < i$, then terminate all branches of the pBT tree that set $l_j = 1$ and continue growing the path from l (that is, each surviving branch of the pBT tree continues with the common ordering that prefers new variables corresponding to literals that are outneighbors of l). Otherwise, if the only out-edges are $l_i \to l$ for $l = l_j$, then terminate all branches of the pBT tree that don't set $l_i = l_j$ and continue growing the path from l_{i-1}. Finally, in case (ii), terminate all branches of the pBT tree that set $l_i = 0$ and continue growing the path from l_{i-1}. When we have explored all literals reachable from l_1, all such literals (and hence their underlying variables) will have a fixed value. We then start over with a new variable, if there is one (making sure, on each branch of the pBT tree, to respect the settings to the variables reachable from l_1, should we encounter them). □

6.4 3-Satisfiability in the Fully Adaptive-Order Model

So far we have proven lower bounds for fixed- and adaptive-order pBT algorithms. Here we use 3SAT to give the first width lower bound for a fully-adaptive-order pBT algorithm. The same lower bound also holds for the depth-first complexity measure and hence applies to a large class of backtracking algorithms for SAT, commonly known as DPLL algorithms. In particular, this lower bound can be seen to extend the lower bound of [3] against *myopic* DPLL algorithms to a more general model.

Theorem 15. *Any fully-adaptive-order pBT algorithm for 3SAT on n variables requires width* $2^{\Omega(n)}$ *and depth-first size* $2^{\Omega(n)}$*. This lower bound holds for the strong data type for SAT.*

The lower bound uses formulas that encode a full rank linear system $Ax = b$ over $\mathbf{GF}_2$. These formulas are efficiently solvable by Gaussian elimination, thus they separate our model of dynamic programming and backtracking from algebraic methods.

6.4.1 Linear systems over expanders

Let A be an $m \times n$ 0/1 matrix, x be an $n \times 1$ vector of variables and b an $m \times 1$ vector over 0/1. Given a fixed A, the $Ax = b$ problem on instance b is to find a 0/1 assignment (if there is one) to the variables in x such that $Ax = b$ where all arithmetic is performed modulo 2. More precisely, given that A is fixed, each item is of the form $\langle x_j, b_{j_1}, ..., b_{j_K} \rangle$, where $j_1, ..., j_K$ denote the indices of the rows of A such that there is a 1 in the jth column. The decisions about items are 0 and 1,

corresponding to the value assigned to the variable in question. If A has, say, at most three 1's in each row, then it is easy to see that a width-w fully-adaptive-order (depth-first) pBT algorithm for 3SAT gives a width-w fully-adaptive-order (depth-first) pBT algorithm for the $Ax = b$ problem. Hence, we will concentrate on the latter problem.

As usual, such a matrix A encodes a bipartite graph from m rows to n columns where the edge (i, j) is present if and only if $A_{ij} = 1$. For $I \subset [m]$, let $\Gamma_A(I)$ (or just $\Gamma(I)$) denote the set of neighbors of the rows I. We will often view A as a bipartite graph from equations in the linear system $Ax = b$ to variables in x, where each equation is connected to the variables it contains. Hence, we sometimes write $Vars(I)$ instead of $\Gamma(I)$.

We will need a matrix A such that the bipartite graph described above is a good expander. In fact, we will use two notions of expanders: expanders and boundary expanders. The latter notion is stronger as it requires the existence of unique neighbors. However, every strong expander is also a good boundary expander.

Definition 16. *We say that A is an (r, s, c)-*expander *if*

1. *Each row of A contains at most s 1's, and*
2. $\forall I \subseteq [m]$ $(|I| \leq r \Rightarrow |\Gamma(I)| \geq c \cdot |I|)$.

For a set of rows $I \subseteq [m]$ of an $m \times n$ matrix A, we define its boundary $\partial_A I$ *(or just ∂I) as the set of all $j \in [n]$ (called* boundary elements*) such that there exists exactly one row $i \in I$ where $A_{ij} = 1$. Matrix A is an (r, s, c)-*boundary expander *if condition 2 is replaced by*

2′. $\forall I \subseteq [m]$ $(|I| \leq r \Rightarrow |\partial I| \geq c \cdot |I|)$.

We will not explicitly mention the notion of boundary expansion until we prove the depth-first lower bound in Section 6.4.3, but we note that it is needed in the proof of Lemma 22. In general, it is not hard to see that very good expanders are also boundary expanders: any (r, s, c)-expander is an $(r, s, 2c - s)$-boundary expander.

The following lemma provides the existence of good expander matrices that have full rank over GF_2. It is an improvement upon the construction of full rank expanders in [3] and is likely to be of independent interest. The proof appears in the appendix.

Lemma 17. *For any constant $c < 2$ there exists a constants $\epsilon > 0$ and $K > 0$ and a family A_n of $n \times n$ matrices such that*

- *A_n has full rank.*
- *A_n is $(\epsilon n, 3, c)$-expander.*
- *Every column of A_n contains at most K ones.*

6.4.2 The Width Lower Bound

Fix n, let $\delta > 0$ be a sufficiently small constant and let A be a full-rank $n \times n$ matrix and an $(r, 3, c = 2 - \delta)$-expander that has at most K ones in each column, as guaranteed by Lemma 17. Here r is $\Omega(n)$ and K is a constant that depends on c and r/n. Hence the items in this problem can be described by the name of a variable, say, x_j, and (up to) K 0/1 values, say, $q_1, ..., q_K$ which represent values for each b_i where $A_{ij} = 1$. Let $\mathcal{A}$ be any fully-adaptive-order pBT algorithm for the $Ax = b$ problem.

Consider the following Solver-Adversary game that operates in rounds. In each round:
(1) The Solver proposes a possible item $D = \langle x_j, q_1, ..., q_K \rangle$.
(2) The Adversary *reveals* values for $b_{j_1}, ..., b_{j_K}$, where $j_1, ..., j_K$ are the rows of A that have a 1 in the jth column. The Adversary must remain consistent with its previous answers.
(3) If the q values match the b values, the Solver chooses a set $S \subseteq \{0,1\}$; otherwise the round ends and a new round begins.
(4) The Adversary selects an element from S (if there is one).
The game ends when the Solver has proposed an item and chosen a subset for every variable or when the Solver chooses $\emptyset$ in step (3).

It is not hard to see that $\mathcal{A}$ gives a strategy for the Solver in the above game against any Adversary. In fact, the Solver in the game has a major advantage over the algorithm: whenever it proposes an item that is not present in the input, it gets to see the corresponding item that is present. Let $T_{\mathcal{A}}$ be the tree that represents $\mathcal{A}$'s strategy against all adversaries. This tree will have branch points for the Adversary's decisions in step (2)—call these b-nodes—and for the Adversary's decisions in step (4)—call these x-nodes. Each b-node will have at most 2^K children and each x-node will have at most 2 children. Furthermore, each b-node will have exactly one child that is an x-node and all other children will be b-nodes.

But how does $T_{\mathcal{A}}$ relate to the pBT trees created by $\mathcal{A}$ on individual instances of the problem? Consider a fixed $b \in \{0,1\}^n$. Let $T_{\mathcal{A}}^b$ be the subtree of $T_{\mathcal{A}}$ consisting of all paths that are consistent with b at the b-nodes. Every path in $T_{\mathcal{A}}^b$ will have a corresponding path in the pBT tree of $\mathcal{A}$ on instance $Ax = b$. Hence, our strategy for the lower bound will be to analyze $T_{\mathcal{A}}$ and show that it must contain many paths that are all consistent with some fixed b.

Definition 18. *For a path π in $T_{\mathcal{A}}$, let $\chi(\pi)$ be the (partial) assignment to the variables $x = (x_1, ..., x_n)$ corresponding to the branches of x-nodes that π follows, and let $\beta(\pi)$ be the values of $b = (b_1, ..., b_n)$ that have been revealed according to the branches of b-nodes that π follows.*

Lemma 19. *For any $w \in \{0,1\}^n$, there must be a path π_w in $T_{\mathcal{A}}$ such that $\chi(\pi) = w$.*

Proof. If there weren't, set $b = Aw$ and run $\mathcal{A}$ on the instance $Ax = b$. $\mathcal{A}$ does not solve the problem on that instance. □

Definition 20. *Let $r' = r/4$. A* partial path *π is a path in $T_{\mathcal{A}}$ that starts at the root and stops at the first point such that $\beta(\pi)$ reveals at least r' components of b. Notice this means that $\beta(\pi)$ reveals less than $r' + K$ components of b. A partial path π is called* good *if $\chi(\pi)$ assigns values to at least $\gamma r'$ variables in x, for a sufficiently small constant $\gamma > 0$. Otherwise, it is called* bad*. For $w \in \{0,1\}^n$, we say that a partial path π* finds *w if there is an extension π' of π such that $\chi(\pi') = w$ and $\beta(\pi') = Aw$.*

Note that no partial path can contain more than $r' + K$ x-nodes. If one did, consider the submatrix of A consisting of the columns corresponding to the variables set on that partial path; such a submatrix has at most $r' + K$ nonzero rows, so it cannot have full column rank and therefore A would not have full rank. In what follows, for the sake of simplicity, we will disregard the extra term of K when discussing the maximum number of bits of b revealed or the maximum number of x-nodes along any partial path.

Lemma 21. *No partial path in $T_{\mathcal{A}}$ can find more than $2^{n-r'}$ assignments in $\{0,1\}^n$.*

Proof. By definition, a partial path π gives values to r' components of b. For any extension π' of π, certainly $\beta(\pi')$ is an extension of $\beta(\pi)$. There are $2^{n-r'}$ such extensions. If π finds w, then it must be the case that $w = A^{-1}b$ for some extension b to $\beta(\pi)$. Hence there are at most $2^{n-r'}$ such w's (here we are, of course, using the fact that A is full rank). □

To proceed, we will need the following technical lemma:

Lemma 22 ([3])**.** *Assume that an $m \times n$ matrix A is an $(r, 3, c)$-expander for $\frac{17}{9} < c < 2$. Let $x = \{x_1, \ldots, x_n\}$ be a set of variables, $\hat{x} \subseteq x$, $b \in \{0,1\}^m$, and let $\mathcal{L} = \{\ell_1, \ldots, \ell_k\}$ be a tuple of linear equations from the system $Ax = b$. Assume further that $|\hat{x}| \leq r$ and $|\mathcal{L}| \leq r$. Denote by L the set of assignments to the variables in $\hat{x}$ that can be extended on x to satisfy $\mathcal{L}$. If L is not empty then it is an affine subspace of $\{0,1\}^{|\hat{x}|}$ of dimension greater than $|\hat{x}| \left(\frac{1}{2} - \frac{14-7c}{2(2c-3)}\right)$.*

Moreover, because of the linear algebraic structure of L, we can say that each partial assignment in L can be extended to the same number of satisfying assignments for $\mathcal{L}$.

Lemma 23. *No good partial path in $T_{\mathcal{A}}$ can find more than $2^{n-qr'}$ assignments in $\{0,1\}^n$ where q is a constant strictly bigger than 1.*

Proof. Each good partial path π assigns values to at least $\gamma r'$ variables in x (via $\chi(\pi)$). Let this set of variables be $\hat{x}$. Also, let $\mathcal{L}$ be the set of equations corresponding to $\beta(\pi)$. We assume that $\chi(\pi)$ can be extended to satisfy $\mathcal{L}$, since otherwise π finds no assignments and we are done. There are at most $2^{n-r'}$ assignments to x that satisfy $\mathcal{L}$. We can partition this set of assignments based on the partial assignments they give to $\hat{x}$. Applying Lemma 22 with $\delta = 2 - c$, there are at least $2^{(\frac{1}{2} - \frac{7\delta}{2-4\delta})\gamma r'} \geq 2^{(\frac{1}{2} - 7\delta)\gamma r'}$ (for δ sufficiently small) partitions each of equal size. Let $q = 1 + (\frac{1}{2} - 7\delta)\gamma$. Then there are at most $2^{n-qr'}$ extensions to $\chi(\pi)$ that satisfy $\mathcal{L}$, so certainly π finds at most $2^{n-qr'}$ assignments. □

Lemma 24. *There are at most $2^{\epsilon r'}$ bad partial paths in $T_{\mathcal{A}}$, where ϵ is a constant strictly smaller than 1.*

Proof. Each bad partial path can be specified by two sequences. Let B be a sequence of r' bits, denoting the values of each new component of b revealed along the partial path in the order they are revealed (when multiple bits of b are revealed at a single b-node, put some arbitrary order on them). Let X be a sequence of $a = \gamma r'$ bits, denoting the values of the variables set along the partial path (in the order they are set). Not all such sequences are valid, however. Consider a particular placement of the bits of X (in order) among the bits of B. For each occurrence of a bit from X, the preceeding bit of B is fixed (it is possible that more than one bit before the bit from X is fixed). This is because that bit of B is used to specify the single child of a particular b-node that is an x-node. Let B_X be the bits of B immediately preceeding the bits from X. Now look at the remainder of B, $B \setminus B_X$. Each consecutive subsequence of size $2K$ in $B \setminus B_X$ has at most $2^{2K} - 1$ possible values. This is because these bits are either fixed or are used specify children of b-nodes that are not x-nodes. Let $z = (r' - a)/2K$. Given a particular X and a particular placement of X among the r' bits of B, there are at most $(2^{2K} - 1)^z$ possible values of B. Therefore, the total number of bad paths is at most

$$\begin{aligned}
\binom{r'}{a} 2^a \left(2^{2K} - 1\right)^z &= 2^a 2^{2Kz} \binom{r'}{a} \left(1 - \tfrac{1}{2^{2K}}\right)^z \\
&\leq 2^{r'} \left(\frac{er'}{a}\right)^a e^{-z/2^{2K}} \\
&\leq 2^{r'} (e/\gamma)^a \left(e^{((1/\gamma)-1)/2K2^{2K}}\right)^{-a} \\
&\leq 2^{r'-a} \\
&= 2^{r'-\gamma r'},
\end{aligned}$$

where the last inequality follows by setting γ sufficiently small compared to K. □

We are now ready to prove the width lower bound.

Theorem 25. *Every fully-adaptive-order pBT algorithm $\mathcal{A}$ for $Ax = b$ requires width $2^{\Omega(n)}$.*

Proof. We will show that there are significantly more than $2^{r'}$ good partial paths in $T_{\mathcal{A}}$. If we set b randomly, then each partial path remains in $T^b_{\mathcal{A}}$ with probability at least $2^{-r'}$, so there must be a setting of b where $T^b_{\mathcal{A}}$, and hence the pBT tree of $\mathcal{A}$, is big.

By Lemma 24, there are at most $2^{\epsilon r'}$ bad paths in $T_{\mathcal{A}}$ where $\epsilon < 1$. By Lemma 21, each such bad path finds at most $2^{n-r'}$ assignments. Therefore, all the bad paths together find at most $2^{n-(1-\epsilon)r'}$ assignments. Since the set of all partial paths must find 2^n assignments by Lemma 19, the set of good paths must find at least $2^n - 2^{n-(1-\epsilon)r'} \geq (1 - o(1))2^n$ assignments. By Lemma 23, then, there must be at least $(1 - o(1))2^{qr'}$ good paths in $T_{\mathcal{A}}$, where $q > 1$. For any (partial) path π, the probability that a random b will be consistent with π is $2^{-r'}$. Hence, the expectation over random b of the number of good paths in $T^b_{\mathcal{A}}$, is at least $2^{(q-1)r'}$. Thus there must be a setting of b that achieves this width. □

Note that Theorem 25 is proving something stronger than stated. The theorem is showing an exponential lower bound on the expected width with respect to the input distribution induced by the uniform distribution over random b vectors. If we define a randomized pBT algorithm as a distribution over deterministic pBT algorithms, then we get a lower bound on the expected width of randomized pBT algorithms by applying Yao's minmax principle.

6.4.3 The Depth-First Lower Bound

We now know that for every fully-adaptive-order pBT algorithm, there is a setting of b so that the $Ax = b$ problem requires large width. Of course, that algorithm may be able to put an orientation on its pBT tree for such a b such that the leftmost path of the pBT tree finds the corresponding solution. If the tree is generated in a depth-first manner, then the algorithm may solve the instance very quickly. Here we prove that for any depth-first pBT algorithm, there must be a choice of b such that the algorithm must traverse $2^{\Omega(n)}$ paths before it finds the solution. In order to achieve this, we need to take a closer look at what $T^b_{\mathcal{A}}$ looks like for a "typical" choice of b. In particular, we will show that for almost all b's, the tree $T^b_{\mathcal{A}}$ has exponential width.

Let $c' = 2c - 3$, so that A is an $(r, 3, c')$-boundary expander. The eventual lower bound (in Theorem 30) will begin by fixing $\mathcal{A}$ and any $b \in \{0,1\}^n$ such that $T^b_{\mathcal{A}}$ contains no bad partial paths. By Lemma 24, almost every b satisfies this. We will then implicitly describe a set of $2^{\Omega(n)}$ partial paths that must appear in $T^b_{\mathcal{A}}$. Since we have so much flexibility in our choice of b, we will choose one such that the corresponding solution appears in the right subtree of the top branching point in $T^b_{\mathcal{A}}$, and observe that there are actually an exponential number of partial paths in the left subtree alone.

Definition 26 ([4])**.** *For any set of variables $\hat{x}$ in the linear system $Ax = b$, define the following inference relation on subsets of equations:*

$$\mathcal{L}_1 \vdash_{\hat{x}} \mathcal{L}_2 \equiv \ |\mathcal{L}_1| \leq r/2 \ \wedge \ \partial\mathcal{L}_2 \subseteq \hat{x} \cup Vars(\mathcal{L}_1). \tag{1}$$

Let $\mathrm{Cl}(\hat{x})$ *(the closure of $\hat{x}$) denote the union of all sets of equations $\mathcal{L}$ that can be inferred (through the transitive closure of $\vdash_{\hat{x}}$) from $\emptyset$.*

Proposition 27. *For any set of variables $\hat{x}$ of size at most $c'r/2$,* $\mathrm{Cl}(\hat{x})$ *has size at most $|\hat{x}|/c'$.*

Proof. If not, consider unioning the sets comprising $\mathrm{Cl}(\hat{x})$ in some arbitrary order that respects the order of inference: $\mathcal{L}_1, \mathcal{L}_2, ...$ Define $C_k = \cup_{i=1}^{k} \mathcal{L}_i$, and let t be the minimum number such that C_t has size greater than $|\hat{x}|/c'$. Because of the order of the sets, $\partial C_t \subseteq \hat{x}$. Also C_t has size at most r. But then, by boundary expansion, C_t should have a boundary of size at least $c' \ |C_t| > |\hat{x}|$. □

The following lemma is fairly straightforward, but very important. It basically says that closures of sets of variables are the "hardest" subsets of equations to satisfy.

Lemma 28 ([2])**.** *Let A be an $(r, 3, c')$-boundary expander and fix any $b \in \{0,1\}^n$. Let χ be any partial assignment to x and let $\hat{x}$ be the set of variables underlying χ. Let $\mathcal{L} = \mathrm{Cl}(\hat{x})$. If there is an assignment to x satisfying $\mathcal{L}$ that is consistent with χ, then for every subset of equations $\mathcal{L}'$ of size at most $r/2$, there is an assignment to x consistent with χ satisfying $\mathcal{L}'$.*

We will often abuse notation and write $\mathrm{Cl}(\chi)$ for $\mathrm{Cl}(\hat{x})$, where χ is a partial assignment to the variables $\hat{x}$. In what follows, given a b and a node v in $T^b_{\mathcal{A}}$, we will say that v *satisfies its closure* if there is an assignment to x consistent with $\chi(v)$ that satisfies those equations of $Ax = b$ in $\mathrm{Cl}(\chi(v))$.

Lemma 29. *Let b be such that $T^b_{\mathcal{A}}$ contains no bad partial paths. Then $T^b_{\mathcal{A}}$ has at least $2^{\Omega(r')}$ good partial paths. In fact, both subtrees below the top branching point of $T^b_{\mathcal{A}}$ contain $2^{\Omega(r')}$ good partial paths.*

Proof. A sufficient set of good partial paths in $T^b_{\mathcal{A}}$ will be those partial paths π that maintain the invariant that, for all v on π, v satisfies its closure.

We first argue that any v in $T^b_{\mathcal{A}}$ of depth less than r' that satisfies its closure has a child in $T^b_{\mathcal{A}}$. The only possible violation of this statement is if v is an x-node that has no children (equivalently, if the Solver chooses $\emptyset$ in step (3)). But if v satisfies its closure, then, by Lemma 28, there is an assignment to x consistent with $\chi(v)$ that satisfies the equations underlying $\beta(v)$. Let $w \in \{0,1\}^n$ be this assignment. If v has no children, then $\mathcal{A}$ will not find w since for every node v' in $T_{\mathcal{A}}$ that is not on any path including v must have $\chi(v')$ disagreeing with $\chi(v)$ or $\beta(v')$ disagreeing with $\beta(v)$. In fact, we can even argue that there is a child of v that satisfies those equations underlying $\beta(v)$ and that satisfies its closure; this is because together these constitute a set of at most $r' + r'/c' \leq r/2$ equations (here we are applying Proposition 27), so we can still use Lemma 28 as above.

Now let π be any partial path in $T^b_{\mathcal{A}}$ that has maintained the invariant until depth r'. Let $\hat{x}$ be the variables underlying $\chi(\pi)$ and let $\mathcal{L}$ be those equations from the system $Ax = b$ that are in $\mathrm{Cl}(\hat{x})$. Since π must be a good path, $|\hat{x}| \geq \gamma r'$. By Lemma 22, there are at least $2^{(q-1)r'}$ setting (for the same q as in Lemma 23) to $\hat{x}$ that are consistent with solutions to $\mathcal{L}$. Therefore, there must be at least $(q-1)r'$ x-nodes v along π satisfying the following: let x_i be the variable set by v and assume $\chi(\pi)$ sets $x_i = 0$; then there is an assignment consistent with $\chi(v) \cup [x_i = 1]$ that satisfies $\mathcal{L}$. We will argue that each such node v has two children in $T^b_{\mathcal{A}}$ and both satisfy their closures. If there were no child of v corresponding to the partial assignment $\chi(v) \cup [x_i = 1]$, then, again, let w be the consistent assignment to x that satisfies $\mathcal{L}$; $\mathcal{A}$ would not find w. So let v' be that child of v. Since $\mathrm{Cl}(\chi(v')) \subseteq \mathcal{L}$, v' must satisfy its closure.

We have now established that there are partial paths that satisfy the invariant and that every such partial path has at least $(q-1)r'$ partial paths branching off of it that also satisfy the invariant. This means that there must be at least $2^{(q-1)r'}$ such partial paths. To see the claim about the two subtrees of $T^b_{\mathcal{A}}$, consider the first x-node in the tree and assume it corresponds to variable x_i. The closure of a single variable is empty for a good expander such as A, so that x-node must have two children both satisfying the invariant. Now simply apply the same argument to both subtrees. □

We can now prove the depth-first lower bound:

Theorem 30. *Every fully-adaptive-order pBT algorithm $\mathcal{A}$ for $Ax = b$ requires depth-first size $2^{\Omega(n)}$.*

Proof. Since $\mathcal{A}$ is a depth-first algorithm, it will impose an order (say, left-to-right) on the children of every x-node in $T_{\mathcal{A}}$. The root of $T_{\mathcal{A}}$ is a b-node *root*. Let v denote the unique x-node that is

a child of *root* and let x_i be the variable it corresponds to. Finally, assume that v sets $x_i = 0$ on its left branch and $x_i = 1$ on its right. Choose a value for $b \in \{0,1\}^n$ such that (i) b is consistent with $\beta(v)$; (ii) b's corresponding x sets $x_i = 1$; and (iii) $T^b_{\mathcal{A}}$ contains no bad partial paths. This is certainly possible since at least a $1/2^{K+1}$ fraction of b's satisfy (i) and (ii), while, by Lemma 24, all but a $2^{(\epsilon-1)r'}$ fraction of b's satisfy (iii). By Lemma 29, $T^b_{\mathcal{A}}$ contains $2^{\Omega(r')} = 2^{\Omega(n)}$ good partial paths in the left subtree of $T^b_{\mathcal{A}}$, but the solution is not found until the right subtree. □

6.5 Subset-Sum lower bound

To prove an exponential lower bound on the width of any fully-adaptive-order pBT algorithm for the Subset-Sum problem, we extend the lower bound for the $Ax = b$ problem and then use a "pBT reduction" from the extended $Ax = b$ problem to the Subset-Sum problem.

6.5.1 Extended lower bound for $Ax = b$

The extended $Ax = b$ problem is identical to the original except for the presence of n extra *equality* items, one for each row of A. The equality item e_i contains only the index i of the row it corresponds to (in particular, it contains no information about b). The possible decisions about e_i are *accept* and *reject*, where a potential solution that accepts e_i is valid only if all of the variables in row i of A are set equal (i.e. they are all 0 or all 1). Likewise, if a potential solution rejects e_i, then that solution is valid only if the variables in that row are not all equal. Notice that this extended problem is no harder for pBT than the original problem. The algorithm could simply ignore the equality items until it has set all of the variable items and then simply set the equality items accordingly. On the other hand, the presence of extra items could introduce new opportunities for the pBT algorithm to branch and/or change its ordering.

Nevertheless, we show that any fully-adaptive-order pBT algorithm for the extended problem still requires exponential width. The proof will essentially mirror the previous lower bound, with small modifications to the definitions of the game and of *good* and *bad* paths to accomodate the equality items.

Begin by fixing A as above. Each round of the Solver-Adversary game now proceeds as follows:
(1) The Solver proposes a possible item $D = \langle x_j, q_1, ..., q_K \rangle$, or an equality item e_i.
(2) In the former case, the Adversary reveals values for $b_{j_1}, ..., b_{j_K}$, where $j_1, ..., j_K$ are the rows of A that have a 1 in the jth digit. The Adversary must remain consistent with its previous answers. In the latter case, the adversary does nothing.
(3) In the former case, if the q values match the b values, the Solver chooses a set $S \subseteq \{0,1\}$; otherwise the round ends and a new round begins. In the latter case, the Solver chooses a set $S \subseteq \{accept,\ reject\}$.
(4) In both cases, the Adversary selects an element from S (if there is one).
The game ends when the Solver has proposed an item and chosen a subset for every variable and every equality item, or when the Solver reveals $\emptyset$ in step (3).

Again, let $T_{\mathcal{A}}$ be the tree that represents an algorithm $\mathcal{A}$'s strategy against all adversaries. Now there will be three kinds of nodes: the b nodes and x nodes as before, and the e-nodes (or equality nodes) which will have at most two children, corresponding to the Adversary's decision in step (4) when the Solver selects an equality item in step (1). It is still the case that each b-node will have exactly one child that is an x-node and all other children will be b-nodes. Define $T^b_{\mathcal{A}}$ as before.

Definition 31. *For a path π in $T_{\mathcal{A}}$, let $\chi(\pi)$ be the (partial) assignment to the variables $x = (x_1, ..., x_n)$ corresponding to the branches of x-nodes that π follows, and let $\beta(\pi)$ be the values of $b = (b_1, ..., b_n)$ that have been revealed according to the branches of b-nodes that π follows. Let $\eta(\pi)$*

be the (partial) assignment to the equality items corresponding to the branches of e-nodes that π follows.

Lemma 19 still holds in this new context. We define a *partial path* as in Defintion 20 and then have the following analogous defintion for good and bad parftial paths.

Definition 32. *A partial path π is called* good *if either*
(1) $\chi(\pi)$ assigns values to at least $\gamma r'$ variables in x, or
(2) there are at least $\gamma r'$ accepted equality items in π.
As in Definition 20, $\gamma > 0$ will be a sufficiently small constant. Otherwise, π is called bad. *For $w \in \{0,1\}^n$, we say that a partial path π* finds *w if there is an extension π' of π such that $\chi(\pi') = w$ and $\beta(\pi') = Aw$ and $\eta(\pi')$ is consistent with w.*

Lemma 21 still holds in exactly the same form.

Lemma 33. *No good partial path in $T_{\mathcal{A}}$ can find more than $2^{n-qr'}$ assignments in $\{0,1\}^n$ where q is a constant strictly bigger than 1.*

Proof. Let π be a good partial path. If case (1) of the definition of good holds, we proceed exactly as in Lemma 23. In case (2), let $\mathcal{L} \subseteq [n]$ denote the rows i of A such that $\beta(\pi)$ reveals bit b_i. Consider an $i \in [n]$ such that π accepts e_i and $i \in \mathcal{L}$. Any extension of π that finds a solution must set all the variables in row i to 0 if $b_i = 0$ and all the variables to 1 if $b_i = 1$. In other words, the values of the variables underlying row i are fixed. Therefore, if at least $\gamma r'/2$ of the accepted equality items correspond to equations in $\mathcal{L}$, then we can proceed as above with at least $(2-\delta)\gamma r'/2$ variables fixed (by expansion). Of course, if two such equations set a single variable in two different ways, then no extension of π will find a solution.

Now assume that at least $\gamma r'/2$ of the accepted equality items correspond to equations outside $\mathcal{L}$. Let $\hat{e} \in [n]$ denote the rows corresponding to these $\gamma r'/2$ accepted equality items. We prove an analogue of Lemma 22 that shows there are many settings to the items e_i, for $i \in \hat{e}$, that are consistent with the equations $\mathcal{L}$. More importantly, one particular partial assignment to the e_i's extends to relatively few consistent full assignments.

A *system of distinct representatives (SDR)* for $\mathcal{L}$ is an ordered pairing $((i_1, x_{j_1}), ..., (i_{r'}, x_{j_{r'}}))$ such that $\mathcal{L} = \{i_1, ..., i_{r'}\}$, x_{j_α} is a variable in the equation corresponding to row i_α and x_{j_α} does not occur in any of the equations $i_{\alpha+1}, ..., i_{r'}$. Notice that, if we have an SDR, we can set all of the variables outside the SDR to any assignment we want, and then set the variables in the SDR (in reverse order) in the unique way to satisfy $\mathcal{L}$. Given an SDR for $\mathcal{L}$, call an equation $i \in \hat{e}$ *saturated* if at least two of its underlying variables appear in the SDR. We show that there is an SDR for $\mathcal{L}$ such that at least half of the equations in $\hat{e}$ are not saturated. If this is the case, then pick a subset of $|\hat{e}|/4K$ of these unsaturated equations that are all disjoint on their two variables that are not covered by the SDR (if there are more than two such variables in an equation, just choose two arbitrarily). For each equation in this subset, we must set the two uncovered variables equal in any extension of π if we expect to find a solution. Therefore, π can find at most

$$\left(\frac{1}{2}\right)^{|\hat{e}|/4K} 2^{n-r'}$$

solutions.

Now to show that claim about the SDR. While the boundary of $\mathcal{L}$ contains elements not in $Vars(\hat{e})$, choose one and assign it to its corresponding equation in $\mathcal{L}$ for the SDR. Let $\mathcal{L}'$ denote the remaining unsassigned equations in $\mathcal{L}$. We know that $\partial\mathcal{L}'$ is contained in $Vars(\hat{e})$, so at least

$(1-2\delta)|\mathcal{L}'|$ of the variables of $\mathcal{L}'$ are in $Vars(\hat{e})$ and each of the remaining variables in $Vars(\mathcal{L}')$ is contained in at least two equations in $\mathcal{L}'$. Simple calculations show that there can be at most $(1+\delta)|\mathcal{L}'|$ elements of $Vars(\mathcal{L}')$ outside of $Vars(\hat{e})$. Due to the expansion of $\mathcal{L}' \cup \hat{e}$, we can conclude

$$(2-\delta)(|\mathcal{L}'| + |\hat{e}|) \leq (1+\delta)|\mathcal{L}'| + |Vars(\hat{e})|. \tag{2}$$

First of all, since $|Vars(\hat{e})| \leq 3|\hat{e}|$, it follows that $|\mathcal{L}'| \leq \frac{1+\delta}{1-2\delta}|\hat{e}|$. Substituting this into inequality (2), we get that

$$|Vars(\hat{e})| - |\mathcal{L}'| \geq (2-5\delta)|\hat{e}|,$$

for δ sufficiently small. There are at most $|\mathcal{L}'|$ variables in any SDR for $\mathcal{L}$ that extends the current partial SDR that are contained in $Vars(\hat{e})$. Therefore, no matter how we extend the partial SDR, there will be at least $2-5\delta$ variables per equation in $\hat{e}$ on average that are not covered by the SDR. Certainly, then, half the equations in $\hat{e}$ must not be saturated (again if δ is sufficiently small). Therefore, just extend the partial SDR in the usual way: while $\mathcal{L}'$ is not empty, choose a boundary variable and assign it to its corresponding equation. □

Lemma 34. *There are at most $2^{\epsilon r'}$ bad partial paths in $T_{\mathcal{A}}$, where ϵ is a constant strictly smaller than 1.*

Proof. Consider the proof of Lemma 24. Each bad path (by the original definition) was specified by a pair of sequences B and X. To specify a bad path by the new definition, it is sufficient to add a sequence E denoting the decisions made at e-nodes along the path. There can be potentially n such decisions, but at most $a = \gamma r'$ of them can be accepted; the others must be rejected. Therefore, we can bound the number of bad paths by taking the expression,

$$2^{r'} (e/\gamma)^a \left(e^{((1/\gamma)-1)/2K2^{2K}}\right)^{-a}$$

which we used to bound the number of bad paths by the original definition, and multiply it by $\binom{n}{\gamma r'}$. This yields a bound of

$$2^{r'} (e/\gamma)^a (f/\gamma)^a \left(e^{((1/\gamma)-1)/2K2^{2K}}\right)^{-a},$$

where f is a constant that depends on r/n. Again, by taking γ sufficiently small compared to K and f, we can make this expression at most $2^{r'-a} = 2^{r'-\gamma r'}$. □

Finally, we get the following theorem in exactly the same manner as we did Theorem 25.

Theorem 35. *Every pBT algorithm $\mathcal{A}$ for the extended $Ax = b$ problem requires width $2^{\Omega(n)}$.*

6.5.2 Reduction to Subset-Sum

To prove a lower bound for Subset-Sum, we exhibit a reduction from the extended $Ax = b$ problem to Subset-Sum that is sufficiently local that it preserves efficient fully-adaptive-order *pBT* algorithms. We will not formally define the concept of a *pBT* reduction but we believe the specific reduction we provide will illustrate the requirements for such a reduction. Informally, we need to transform items in the source problem domain to (sets of) items in the target domain in such a way that any ordering/decisions in the target domain will induce an ordering/decisions in the source domain.

We use a small modification on the standard reduction from 3SAT to Subset-Sum. Fix n and A and consider the universe of items $U_{Ax=b}$ for the extended $Ax = b$ problem. Each item in

the universe for Subset-Sum will be a decimal number with $2n$ digits. The first n digits, labelled $x_1, ..., x_n$, will correspond to the n variables of $Ax = b$. The last n digits, labelled $1, ..., n$, will correspond to the n equations in the system $Ax = b$. Given a variable item $D = \langle x_j, b_{j_1}, ..., b_{j_K} \rangle$, we create two Subset-Sum items. Assume, wlog, that $b_{j_1} = \cdots = b_{j_r} = 1$ and $b_{j_{r+1}} = \cdots = b_{j_K} = 0$. Create one item, called $SS_1(D)$, that has a 1 in digit x_j and 1's in digits $j_1, ..., j_r$ and 0's elsewhere, and another item, $SS_2(D)$, that has a 1 in digit x_j and 1's in digits $j_{r+1}, ..., j_K$ and 0's elsewhere. Let $SS(D)$ denote the set of these two items. Given an equality item e_i, create an item $SS(e_i)$ that has a 2 in digit i and 0's elsewhere. The universe for the Subset-Sum problem will be $U_{SS} = \bigcup_{D \in U_{Ax=b}} SS(D)$. The target value for Subset-Sum will be the number that has a 1 in digit x_j for each j and a 3 in digit i for each i.

An ordering σ on U_{SS} induces an ordering $Axb(\sigma)$ on $U_{Ax=b}$ in the obvious way: go through σ in order and replace each first occurrence of an item in $SS(D)$ by D. Erase any subsequent occurrences of items in $SS(D)$. Also, each decision about an item in $SS(D)$ maps to a unique decision about D: for a variable item D, $(SS_1(D), accept)$ maps to $(D, accept)$, $(SS_1(D), reject)$ to $(D, reject)$, $(SS_2(D), reject)$ to $(D, accept)$, $(SS_2(D), accept)$ to $(D, reject)$. $(SS(e_i), accept)$ maps to $(e_i, reject)$, and $(SS(e_i), reject)$ maps to $(e_i, accept)$.

Lemma 36. *If there is a width-$w(n)$ fully-adaptive-order pBT algorithm for Subset-Sum, then there is a width-$w(n)$ fully-adaptive-order pBT algorithm for the extended $Ax = b$ problem.*

Proof. Let $\mathcal{A}$ be a width-$w(n)$ algorithm for Subset-Sum. Algorithm $\mathcal{B}$ will simulate it as follows: run $\mathcal{A}$ on the universe U_{SS}. Whenever $\mathcal{A}$ specifies an order σ, the corresponding node of $\mathcal{B}$'s execution will use the related order $Axb(\sigma)$. If $\mathcal{B}$ finds that item D is the first item in the instance according to the order, then it provides $\mathcal{A}$ with whichever item in $SS(D)$ came first in σ. Whatever decisions $\mathcal{A}$ branches on for this item, $\mathcal{B}$ branches on the corresponding decisions for D. It is easy to check that $\mathcal{B}$'s execution tree will have width at most that of $\mathcal{A}$'s execution tree and a path in $\mathcal{B}$'s tree leads to a solution if and only if the corresponding path in $\mathcal{A}$'s tree leads to a solution. □

Theorem 37. *Any fully-adaptive-order pBT algorithm for Subset-Sum requires width $2^{\Omega(n)}$.*

Proof. Simply apply Lemma 36 and Theorem 35. □

7 Open Questions

There are many open questions regarding our *pBT* models. Could we show, for example, that the known greedy $2 - o(1)$ approximation algorithms for Vertex Cover are the best we can do using a polynomial width *pBT* algorithm? This is particularly interesting due to the lack of tight complexity-bound inapproximation results for Vertex Cover based on standard assumptions (see [7] for motivation of this idea). For Interval Scheduling, can the adaptive *pBT* lower bound be extended to the fully-adaptive model. For proportional profit on one machine, we are able to show that a width-2 adaptive-order *pBT* can achieve a better approximation ratio than a priority algorithm. While we know that, for one machine, an optimal solution requires width $\Omega(n)$, the tradeoff between width and the approximation ratio is not at all understood. For example, what is the best approximation ratio for a width-3 *pBT*? We also do not know if an $O(1)$-width adaptive-order *pBT* can achieve an $O(1)$-approximation ratio for interval scheduling with arbitrary profits. For any of the problems already studied in the priority framework (e.g. [9, 17, 5, 34, 10]) it would be interesting to consider constant-width *pBT* algorithms. Our only general approximation-width tradeoff results are the somewhat complementary upper and lower bounds for the knapsack problem and the fixed-order lower bound for interval scheduling for which (as noted above) we

do not have a complementary upper bound. It would be interesting to obtain (closely) matching width-approximation tradeoff results for interval scheduling and other problems whch require large width for optimality.

Although the focus of this paper has been with regard to worst case complexity, the priority and pBT models can also be studied with regard to average case (or smoothed) analysis. In particular, for finding a satisfying assignment with high probability in a random 3CNF formula, the best (with regard to the ratio of clauses per variable under which it succeeds) current algorithm [28] can be implemented as a priority algorithm. Indeed almost all random 3SAT algorithms used to study the 3SAT threshold problem have been priority algorithms (one algorithm in [1] is a width 2 pBT). Obtaining a sharp 3SAT threshold is a major open question and one can consider if it is possible to improve upon the current best density of 3.52 ([28]) by a priority or small width pBT algorithm.

Will the pBT framework lead us to new algorithms (or at least modified interpretations of old algorithms)? Small examples in this direction are the width-2 approximation for interval selection, the linear-width algorithm for 2SAT and the FPTAS for Knapsack presented in this paper.

One way to augment the pBT model would be to allow *non-deterministic branching*. That is, to allow a node in the pBT tree to branch without viewing an input item and, therefore, without assigning decisions. For example, a node could branch in this way to allow each of its children to explore different orders on the remaining items. Does this capability strengthen the pBT model (in the fully-adaptive case)? In [18], it is shown that it does not help significantly for 7SAT.

While we have shown that the pBT model has strong connections to dynamic programming and backtracking, can it be extended to capture other common algorithms? For example, we show that pBT captures simple dynamic programming but what about other dynamic programming algorithms? To capture some "non-simple" applications of dynamic programming, [11] recently defined a model that enhances pBT by making more essential use of memoization. Namely, they define a pBP (*priority branching program*) model which is a DAG analogue of our tree-based pBT programs. Amongst other results, they argue that the Bellman Ford algorithm for the least cost path problem (in a graph with no negative cycles) can be formulated within their pBP model (with non-deterministic branching), but that there is no efficient pBT algorithm (without non-deterministic branching) for the problem. It is not clear whether a pBT algorithm with non-deterministic branching can capture Bellman Ford. A further extension will be needed to capture "non serial" dynamic programming algorithms such as the well known algorithms for computing optimal matrix chain products and optimal binary search trees?

Finally, it is natural to consider randomized pBT algorithms. There are several ways to augment pBT algorithms with randomness and then to study the trade-offs between expected width (or depth first size) and the probability of obtaining a solution. One natural model is as follows. Let H be the set of allowable decisions at a given node of the pBT. Then for every subset $H' \subseteq H$, we have a probability $p_{H'}$ of choosing H' as the set of decisions to consider at that node, where $\sum_{H'} p_{H'} = 1$. In the special case that the probabilities are such that only singleton sets have positive probability, we have a randomized priority (i.e. width 1) algorithm. It is easy to see that, in general, such a randomized pBT induces a probability distribution on deterministic, but not-necessarily-correct pBT algorithms. That is, not every algorithm in this distribution succeeds in solving every instance of the problem. Therefore, the randomized lower bound alluded to after Theorem 25 does not give a lower bound for this model, unless we impose the (seemingly unreasonable) condition that the $p_{H'}$ are assigned in such a way that every resulting algorithm is correct. In other words, we do not get the desired trade-off between expected width and the probability of obtaining a solution. Can such a lower bound be proven, perhaps for SAT?

8 Acknowledgments

We thank Spyros Angelopoulos, Paul Beame, Jeff Edmonds, and Periklis Papakonstantinou for their very helpful comments. Special thanks to Alberto Marchetti-Spaccamela for contributing the upper bound of Theorem 10, and to Charles Rackoff for going above and beyond the call of helpful comments.

References

[1] Dimitris Achlioptas and Gregory B. Sorkin. Optimal myopic algorithms for random 3-SAT. In *IEEE Symposium on Foundations of Computer Science*, pages 590–600, 2000.

[2] M. Alekhnovich. Lower bounds for k-DNF resolution on random 3CNF. In *Proceedings of the 37th Symposium on Theory of Computing*, pages 251–256, 2005.

[3] M. Alekhnovich, E. Hirsch, and D. Itsykson. Exponential lower bounds for the running time of DPLL algorithms on satisfiable formulas. In *Automata, Languages and Programming: 31st International Colloquium, ICALP04*, 2004. To appear in 2005 SAT special issue of the Journal of Automated Reasoning.

[4] M. Alekhnovich and A. Razborov. Lower bounds for the polynomial calculus: non-binomial case. In *Proc. 42nd Ann. Symp. on Foundations of Computer Science*. IEEE Computer Society, 2001.

[5] S. Angelopoulos and A. Borodin. On the power of priority algorithms for facility location and set cover. *Algorithmica*, 40(4):271–291, 2004.

[6] E. M. Arkin and E. L. Silverberg. Scheduling jobs with fixed start and end times. *Disc. Appl. Math*, 18:1–8, 1987.

[7] Sanjeev Arora, Béla Bollobás, László Lovász, and Iannis Tourlakis. Proving integrality gaps without knowing the linear program. *Theory of Computing*, 2(2):19–51, 2006.

[8] Kenneth Arrow. *Social Choice and Individual Values*. Wiley, New York, 1951.

[9] A. Borodin, M. Nielsen, and C. Rackoff. (Incremental) priority algorithms. *Algorithmica*, 37:295–326, 2003.

[10] Allan Borodin, Joan Boyar, and Kim S. Larsen. Priority Algorithms for Graph Optimization Problems. In *Second Workshop on Approximation and Online Algorithms*, volume 3351 of *Lecture Notes in Computer Science*, pages 126–139. Springer-Verlag, 2005.

[11] J. Buresh-Oppenheim, S. Davis, and R. Impagliazzo. A stronger model of dynamic programming algorithms. Manuscript in preparation, 2007.

[12] V. Chvátal. Hard knapsack problems. *Operations Research*, 28(6):1402–1441, 1985.

[13] S. Cook and D. Mitchell. Finding hard instances of the satisfiability problem: A survey. In *DIMACS Series in Theoretical Computer Science*, 1997.

[14] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. MIT Press, Cambridge, Mass., 2001.

[15] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5:394–397, 1962.

[16] M. Davis and H. Putnam. A computing procedure for quantification theory. *Communications of the ACM*, 7:201–215, 1960.

[17] S. Davis and R. Impagliazzo. Models of greedy algorithms for graph problems. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2004.

[18] S. Davis and R. Impagliazzo. Randomized priority algorithms, pBT, and free-branching pBT lower bounds. Manuscript in preparation, 2007.

[19] B.C Dean, M.X. Goemans, and J. Vondrák. Approximating the stochastic knapsack problem: The benefit of adaptivity. In *Proc. 44th Ann. Symp. on Foundations of Computer Science*, 2004.

[20] T. Erlebach and F.C.R. Spieksma. Interval selection: Applications, algorithms, and lower bounds. *Technical Report 152, Computer Engineering and Networks Laboratory, ETH*, October 2002.

[21] J. Gu, P. W. Purdom, J. Franco, and B. J. Wah. Algorithms for the Satisfiability (SAT) Problem: A Survey. In *Satisfiability (SAT) Problem*, DIMACS, pages 19–151. American Mathematical Society, 1997.

[22] M. Halldorsson, K. Iwama, S. Miyazaki, and S. Taketomi. Online independent sets. *Theoretical Computer Science*, pages 953–962, 2002.

[23] J. Håstad. Some optimal inapproximability results. *JACM*, 48:798–859, 2001.

[24] P. Helman. A common schema for dynamic programming and branch and bound algorithms. *Journal of the Association of Computing Machinery*, 36(1):97–128, 1989.

[25] P. Helman and A. Rosenthal. A comprehensive model of dynamic programming. *SIAM J. on Algebraic and Discrete Methods*, 6:319–334, 1985.

[26] S.L. Horn. One-pass algorithms with revocable acceptances for job interval selection. *MSc Thesis, University of Toronto*, 2004.

[27] O. Ibarra and C. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *JACM*, 1975.

[28] A. Kaporis, L. Kirousis, and E. Lalas. Selecting complementary pairs of literals. In *Proc. LICS'03 Workshop on Typical Case Complexity and Phase Transitions*, 2003.

[29] R.M. Karp and M. Held. Finite state processes and dynamic programming. *SIAM J. Applied Mathematics*, 15:693–718, 1967.

[30] S. Khanna, R. Motwani, M. Sudan, and U. Vazirani. On syntactic versus computational views of approximability. *SIAM Journal on Computing*, 28:164–a91, 1998.

[31] D. E. Knuth. Optimum binary search trees. *Acta Informatica*, 1:14–25, 1971.

[32] E. L. Lawler. Fast approximation algorithms for knapsack problems. In *Proc. 18th Ann. Symp. on Foundations of Computer Science*, Long Beach, CA, 1977. IEEE Computer Society.

[33] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press,, 1995.

[34] Oded Regev. Priority algorithms for makespan minimization in the subset model. *Information Processing Letters*, 84(3):153–157, Septmeber 2002.

[35] Vijay V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2001.

[36] G. Woeginger. When does a dynamic programming formulation guarantee the existence of a fully polynomial time approximation scheme (FPTAS)? *INFORMS Journal on Computing*, 12:57–75, 2000.

9 Appendix

Proof. (of Lemma 17) It is well-known that the probabilistic method gives expanders with 3 ones in each row, but here there are two additional subtle points: the maximum number of ones in each column and the rank of the resulting matrix. We handle the former issue using a trick from [2] and the latter by the techniques from [3].

Lemma 38 ([3])**.** *Let L be a linear subspace of $\{0,1\}^n$ of codimension k. Let the vector v be a random vector in $\{0,1\}^n$ with weight 3. Then* $\Pr[v \notin L] = \Omega(\frac{k}{n})$.

Consider the random matrix A_n that contains Kn rows (K may depend on n), each of which is chosen uniformly at random from the set of all rows of weight 3.

Proposition 39. *With probability* $1 - o(1)$, $\mathrm{rank(A_n)} > (1 - \mathrm{e}^{-\Omega(\mathrm{K})})\mathrm{n}$.

Proof. The proof resembles the analysis of the well-known Coupon Collector puzzle. Consider the process of generating the rows of A: let A^t be the matrix consisting of the first t rows of A, thus $A = A^{[Kn]}$. By Lemma 38 it takes on average $O(n/(n-k))$ new rows to increase the rank of A^t from k to $k+1$. Thus, in order to achieve the rank $(1-e^{-K})n$ on average one has to take

$$T = O\left(\sum_{k=1}^{(1-e^{-K})n} \frac{n}{n-k}\right)$$

randomly chosen rows. It is left to notice that

$$\sum_{k=1}^{(1-e^{-K})n} 1/(n-k) = \sum_{k=(e^{-K})n}^{n-1} 1/k = K + O(1).$$

□

Assume that n is an integer and c' is a constant slightly bigger than c, say $c' = c + (2-c)/2$. From now on, K will be a large constant the exact value of which will be determined later.

The following well-known fact states that a random matrix is a good expander.

Lemma 40. *For any $c' < 2$, there is an $\alpha > 0$ such that for all $K > 0$ and n sufficiently large, A_n is an $(r, 3, c')$-expander with probability $1 - o(1)$ provided that $r \le \alpha \frac{n}{K^{1/(2-c')}}$.*

Let $n' = \mathrm{rank(A_n)}$. By Proposition 39, $n' > (1 - e^{-\Omega(K)})n$. One may remove at most $2^{-\Omega(K)}n$ columns from A_n so that the resulting $(Kn) \times n'$ matrix has rank n'. Denote by J_1 the index set of these columns. Denote by J_2 the index set of the columns that contain at least $\hat{K} = \frac{3K}{c'-c}(n/r)$ ones (note this value is constant). Since A_n has Kn ones overall, $|J_2| \le (c'-c)r/3$. If we remove from A all the columns corresponding to $J_1 \cup J_2$ then the resulting matrix has full rank and every column has at most $\hat{K}$ ones. The only problem is that it may not be an expander anymore. To fix this we use a procedure similar to one developed in [3].

Definition 41. *For an $A \in \{0,1\}^{m \times n}$ and a subset of its columns $J \subseteq [n]$ we define an inference relation $\vdash^c_J$ on the set of $[m]$ rows of A:*

$$I \vdash^c_J I_1 \equiv |I_1| \le r/2 \ \wedge \ |\Gamma(I_1) \setminus [\Gamma(I) \cup J]| < c|I_1| \tag{3}$$

Let $\mathrm{Cl^e}(J)$ denote the union of all sets of rows that can be inferred (via the transitive closure of $\vdash^c_J$) from $\emptyset$.

It is not hard to see the benefit of the $\mathrm{Cl}^{\mathrm{e}}()$ operation: namely, for $A \in \{0,1\}^{m \times n}$ an $(r, 3, c')$-expander and $J \subseteq [n]$, set $\hat{I} = \mathrm{Cl}^{\mathrm{e}}(J)$ and $\hat{J} = \Gamma(\hat{I})$. Let A' be the matrix that results from A after removing the columns $\hat{J}$ and the rows $\hat{I}$. Then A' is an $(r/2, 3, c)$-expander. We will eventually apply this transformation to A where $J = J_1 \cup J_2$, but first we need to bound the size of $\hat{I}$ in terms of $|J|$.

Lemma 42. *If* $|J| \leq (c' - c)r/2$, *then* $|\mathrm{Cl}^{\mathrm{e}}(J)| \leq (c' - c)^{-1}|J|$.

Proof. Consider unioning the sets comprising $\mathrm{Cl}^{\mathrm{e}}(J)$ in some arbitrary order that respects the order of inference: $I_1, I_2, \ldots$. Define $C_k = \cup_{i=1}^{k} I_i$ and let t be the minimum number such that C_t has size greater than $|J|/(c' - c)$. Note that $|C_t| \leq r$, so, by expansion it must be the case that

$$|\Gamma(C_t)| \geq c'|C_t|.$$

On the other hand, each new I_j in the sequence contributes at most $c|I_j|$ new elements to $\Gamma(C_{j-1})$, so

$$|\Gamma(C_t)| \leq |J| + c|C_t|.$$

The lemma follows. □

We are ready to finish the proof of Lemma 17. Let $J = J_1 \cup J_2$ and choose K large enough so that $|J| < (c' - c)r/2$. This is possible since r/n is inverse polynomial in K, but $|J_1|/n$ is inverse exponential in K. Again, let $\hat{I} = \mathrm{Cl}^{\mathrm{e}}(J)$ and let $\hat{J} = \Gamma(\hat{I})$. By Lemma 42, $|\hat{I}| < r/2$ and therefore $|\hat{J}| < r$. If we remove all columns corresponding to $\hat{J}$ from A then the resulting matrix has full column rank. This is because after we remove the columns corresponding to J_1 we get a matrix in which all columns are linearly independent, thus after the removal of $\hat{J}$ the matrix still has full column rank. At this point, all rows in $\hat{I}$ are all-zero, so we can safely remove them without decreasing the rank. Finally, let $\hat{n} = \Omega(n)$ be the number of remaining columns and throw out all but $\hat{n}$ linearly independent rows. The final matrix, $\hat{A}$, is a full-rank, $\hat{n} \times \hat{n}$, $(r/2, 3, c)$-expander that has at most $\hat{K}$ ones in each column. □