Toward a Model for Backtracking and Dynamic Programming

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

January 2, 2005

Abstract

We consider a model (BT) for backtracking 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

Proving unconditional lower bounds for computing explicit functions remains one of the most challenging problems in computational complexity. Since 1949, when Shannon showed that a random function has large circuit complexity [25], little progress has been made toward proving lower bounds for the size of unrestricted Boolean circuits that compute *explicit* functions. One explanation for this phenomenon was given by the Natural Proofs approach of Razborov and Rudich [24] who showed that most of the existing lower bound techniques are incapable of proving such lower bounds. One way to investigate the complexity of explicit functions in spite of these difficulties is to study reductions between problems, e.g. to identify a canonical problem (like an NP-complete problem) and show that a given computational task is "not easier" than solving this problem. Another approach would be to restrict the model to avoid the inherent Natural Proof limitations, while preserving a model strong enough to incorporate many "natural" algorithms.

This second direction has recently attracted the attention of many researchers. Khanna, Motwani, Sudan and Vazirani [22] formalize various types of local search paradigms, and in doing so, provide a more precise understanding of local search algorithms. Woeginger [26] 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. Borodin, Nielsen and Rackoff [8] introduce priority algorithms as a model of greedy-like algorithms. Arora, Bollobás and Lovász [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

*Institute for Advanced Study, Princeton, US. Supported by CCR grant $\mathcal{N}$CCR-0324906.
†Department of Computer Science, University of Toronto, bor,avner,toni@cs.toronto.edu
‡CSE Department, University of California San Diego, bureshop, russell@cs.ucsd.edu.
§Research partially supported by NSF Award CCR-0098197.
¶Some of this research was performed at the Institute for Advanced Study in Princeton, NJ supported by the State of New Jersey.

(see for example [12, 13, 10, 18]). Finally, Chvátal [9] proves a lower bound for Knapsack in an algorithmic model that involves elements of branch-and-bound and dynamic programming.

We continue in this direction by presenting a hierarchy of models for backtracking algorithms for general search and optimization problems. Many well-known algorithms and algorithmic techniques can be simulated within these models, both those that are usually considered backtracking and some that would normally be classified as greedy or dynamic programming. We prove several upper and lower bounds on the capabilities of algorithms in this model, in some cases proving that the known algorithms are essentially the best possible within the model.

The starting point for the BT model is the priority algorithm model [8]. 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 with all clauses containing it in a CNF. 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 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 recently studied in [16].) Rather than imposing complexity constraints on the allowable orders, we require it to be localized By introducing branching, a *backtracking algorithm* (BT) can pursue a number of different partial solutions. Given a specific input, a BT 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 BT-algorithm. We view the maximum width of a BT 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 [26]. This branching extension can be applied to either the fixed or adaptive order (fixed BT and adaptive BT) 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 BT algorithms. In order to model the power of backtracking programs (say as in DPLL algorithms for SAT)[1] we need to extend the model further. In a fully adaptive BT 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).

Our results The problems we consider are all well-studied; namely, knapsack, satisfiability and interval scheduling. For knapsack, we show an exponential lower bound (for optimality) on the width of adaptive BT algorithms, and for achieving an FPTAS in the adaptive model we show upper and lower bounds polynomial in $1/\epsilon$. Knapsack also exhibits a separation between width-1 and width-2 adaptive BT: a width-1 BT cannot approximate knapsack better than a factor of $n^{-1/4}$, while a standard 1/2-approximation falls into width-2 BT. For SAT, we show that 2-SAT

[1]The BT 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 $P \neq NP$.

is solved by a linear-width adaptive BT, but needs exponential width for any fixed order BT, and also that MAX2SAT cannot be efficiently approximated by any fixed BT algorithm. We then show that 3-SAT requires exponential width and exponential depth first size in the fully adaptive BT model. This lower bound in turn gives us an exponential bound on the width of fully adaptive BT algorithms for knapsack by "BT reduction." For m-machine interval scheduling we show an $\Omega(n^m)$-width lower bound (for optimality) in the adaptive BT model, an inapproximability result in the fixed BT model, and an approximability separation between width-1 BT and width-2 BT in the adaptive model.

2 The Backtracking Model and its Relatives

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 **Backtracking 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 x_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 a backtracking algorithm for a backtracking problem.

Definition 1. *A* backtracking algorithm $\mathcal{A}$ *for problem* $P = (\mathcal{D}, \{f^n\})$ *consists of the* ordering *functions*

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

and the choice *functions*

$$c_{\mathcal{A}}^k : \mathcal{D}^{k+1} \times H^k \mapsto \mathcal{O}(H \cup \{\perp\}).^2$$

We separate the following three classes of BT algorithms

- **Fixed** *algorithms:* $r_{\mathcal{A}}^k$ *does not depend upon any of its arguments.*
- **Adaptive** *algorithms:* $r_{\mathcal{A}}^k$ *depends on* $D_1, D_2, ..., D_k$ *but not on* $a_1, ..., a_k$.
- **Fully adaptive** *algorithms:* $r_{\mathcal{A}}^k$ *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 $x_1, \ldots, x_k$ 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 $r_{\mathcal{A}}^k$. 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 $r_{\mathcal{A}}^k$, 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 $c_{\mathcal{A}}^k$.

[2] All of our lower bound results will apply to non-uniform BT 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 [8].

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 BT problem and $\mathcal{A}$ is a BT 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 $< D_1^v, ...D_k^v, a_1^v, ...a_k^v >$.*
- *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_{k+1}^v be the data item in $I \setminus \{D_1^v, ..., D_k^v\}$ that goes first in the list $r_{\mathcal{A}}^k(\vec{D}^v, \vec{a}^v)$. Assume that the output $c_{\mathcal{A}}^k(\vec{D}^v, D_{k+1}^v, \vec{a}^v)$ has the form $(c_1, ..., c_d, \perp, c_{d+1}, \dots)$, 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 $(D_1^{v_i}, ..., D_{k+1}^{v_i}, a_1^{v_i}, ..., a_{k+1}^{v_i}) = (D_1^v, ..., D_k^v, D_{k+1}^v, a_1^v, ..., a_k^v, c_i)$ resp.*

Each leaf node t of depth n contains a permuted sequence of the data items I (permuted by the ordering functions $r_{\mathcal{A}}^k$ 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 $f_P(D_1^t, ..., D_n^t, a_1^t, ..., a_n^t) = 1$ where a_k^t is the decision for D_k^t. 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 0 for a maximization problem and ∞ for a minimization problem if the solution is not feasible.)*

Definition 3. *We say that $\mathcal{A}$ is a* **correct algorithm** *for a BT 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** *$S_{\mathcal{A}}^{df}(I)$ 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 $S_{\mathcal{A}}^{df}(I) \leq nW_{\mathcal{A}}(I)$.*

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 $S_{\mathcal{A}}^{df}(n)$ analogously.*

The size $S_{\mathcal{A}}^{df}(I)$ 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 in parallel during the execution. Second, it gives a universal upper bound on the running time of any search style.

While the Fixed and Adaptive classes are ostensibly less powerful than Fully Adaptive algorithms, they remain quite powerful. For example, the width 1 algorithms in these classes are precisely the fixed and adaptive 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 BT algorithms can simulate Woeginger's DP-simple algorithms ([26]).

The reader may notice some similarities between the BT 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, BT algorithms are allowed to branch, or try more than one possibility.[3]

A note on computational complexity: We do not impose any restrictions on the functions $r^k_{\mathcal{A}}$ and $c^k_{\mathcal{A}}$ like 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 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 BT 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 input models that will render the model useless; for example if a node in a graph contans the information about its neighbours *and* their neighbours, then it contains enough information that the ordering function can summon the largest clique as its first items, making an NP-hard problem solvable by a width-1 BT algorithm. In our discussion, we use input representations which seem to us the most natural.

2.1 BT as an Extension of Dynamic Programming and other Algorithm Models

How does our model compare to other models? As noted above, the width 1 BT 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 [26] as *simple dynamic-programming* or *DP-simple*. Many 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 efficient DP-simple algorithms.

The simulation of these algorithms by a fixed-BT algorithm is straightforward once one makes the following observation. Since all parallel runs of a fixed or adaptive BT algorithm view the same input, and the computational power of the function f is unlimited, 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, we consider the simulation of the DP algorithm to solve interval scheduling on one machine. Recall, 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

[3]Recently, a version of the online model in which many partial solutions may be constructed was studied by Halldorsson, et al [19]. Their online model is a special case of a fixed order BT algorithm.

j'th interval; of course $T[j]$ extends $T[i]$ for some $i < j$. We can now think of a BT 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 $j+1^{th}$ interval.

Note that for most 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 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.

2.2 General Lower bound strategy

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

Below is a 2-player game for proving these lower bounds for adaptive BT. This is similar to the lower bound techniques for priority algorithms from [8, 15]. 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 BT.

The game is between the Solver and the Adversary. Initially, the Adversary presents to the algorithm 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.

This continues until P_i is empty. The Solver wins if PI_i is not a valid instance, or if $|T_i| \leq w(|PI|)$ for all $1 \leq i \leq n$, and PS_i is a valid solution, optimal solution, or approximately optimal solution for PI (if we are trying for a search algorithm, exact optimization algorithm, or approximation algorithm, respectively). Otherwise, the Adversary wins.

Any BT 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 BT 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 any partial solution 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 to A, ensuring victory. Since all partial solutions are indispensable, either the above strategy works, or the Solver keeps all partial solutions in T_i, which grows exponentially and eventually exceeds $w(n)$.

For the fixed BT 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 move a. Other than that, the game is identical.

3 The Knapsack problem

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 thepositive side, has an FPTAS. In this section we study the width-approximation tradeoff for BT algorithms for the problem.

Narrow BT algorithms As a warmup, we start by observing that width-1 and width-2 BT algorithms for Knapsack behave dramtically different. 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 a fixed width 2 BT that orders the higest profit items first, and orders the rest by their profit to weight ratio as above.

We next show an $n^{1/4}$ inapproximability result for knapsack for width-1 BT (ie, priority algorithms), where n is the number of items. 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 items. The algorithm then achieves profit 1, while the optimum is $n^{3/4} \cdot n^{-1/2} = n^{1/4}$. If a medium or small item was chosen, leave 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 1.

Wide BT 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 BT.

We prove that it is not possible to solve the problem with an adaptive BT 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 BT 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 [21] and Lawler [23] that uses Dynamic program and a lower bound of $\Omega((1/\epsilon)^{1/3.17})$. We notice that both our lower bounds in this section already hold for the simple variant of the knapsack problem, where for each item the profit is equal to the weight.

Theorem 6. *The width of an optimal adaptive BT for the simple knapsack 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 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 considered 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 $N - \sum Q$, 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 $N - \sum Q$, then the average of the numbers in R should be $a = \frac{2}{n} \cdot (N - \sum Q)$. Since $0 \leq \sum Q \leq (n/4)(8N/3n) = 2N/3$, 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 $|w - a(n/2 - 2)| \leq U$. 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 7. *Knapsack can be $(1 - \epsilon)$-approximated by a width $(1/\epsilon)^2$ adaptive-BT. At least width $(1/\epsilon)^{1/3.17}$ is needed for such an approximation, even for the simple knapsack 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 6), 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 BT 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 [23] 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 BT algorithm can achieve this balance of parameters using width $8/\epsilon^2$: Start with $\tau' = \Pi$; 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 there 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. $\square$

Remark 1. *Certain known algorithms (see [17, 20]), which should intuitively be called greedy, allow* semi-revocable decisions. *We can consider this additional strength in the context of BT algorithms. This means that at any point we can revoke previous accept decisions. We only insist that any partial solution is feasible (eg for the knapsack problem, we do not exceed the knapsack capacity N). This extension only applies to packing problems; that is, where changing accept decisions to rejections does not make a feasible solution infeasible. We can extend the proof of theorems 6 and 7 so as to allow revocable acceptances with slightly worse parameters. Recall that in Theorem 6 we look at $n/2$ elements of the range $[0, N/2]$ and then show that all $n/4$ 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 6, 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 $|\mathcal{F}| \geq \binom{n/2}{n/(2b)}/\binom{n/a}{n/(2b)} = 2^{\omega(n)}$.*

4 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.

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 BT model, but has a simple linear time algorithm in the adaptive BT model (for the weak data type). Thus, we obtain an exponential separation between the fixed and adaptive BT models. Next, we give exponential lower bounds in the fully adaptive model for 3SAT (strong data type).

4.1 2-Satisfiability in the Fixed Model

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

Theorem 8. *For sufficiently large n, any fixed BT 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 describes 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 any 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 any 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) 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 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. $\square$

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. The above argument gives the following constant-factor inapproximation lower bound for this problem:

Theorem 9. *For any $\epsilon > 0$, there exists a $\delta > 0$ such that for all sufficiently large n, any fixed BT 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. (of Theorem 9) The game is played exactly as in the proof of Theorem 8. 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 $x_2^i \stackrel{*}{=} x_3^i$ and one with $x_9^i \stackrel{*}{=} x_{10}^i$). 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. $\square$

A similar idea to the 2SAT inapproximation can be used to show an inapproximation result for Vertex Cover (where the items are nodes with their adjacency lists).

Theorem 10. *For any $\epsilon > 0$, there exists a $\delta > 0$ such that for all sufficiently large n, any fixed BT algorithm for solving Vertex Cover on n vertices requires width $2^{\delta n}$ to achieve a $\frac{17}{16} - \epsilon$ approximation.*

Proof. (sketch) Each node x_i gives rise to three sets of items
(1) For $j \neq k \in [n] \setminus \{i\}$, $x_j - x_i - x_k$ (that is, x_i has neighbors x_j and x_k).
(2) For any $j \in [n] \setminus \{i\}$, $x_j - x_i$.
Let M be the first $m = \lfloor n/17 \rfloor$ disjoint (1)-items in the ordering: $\{y_8^i - y_9^i - y_{10}^i\}_{i=1}^m$. Partition the vertices into m groups of size 17 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:

$$y_1^i - y_2^i - y_3^i - y_6^i - y_7^i - y_8^i - y_9^i - y_{10}^i - y_{11}^i - y_{12}^i - y_{13}^i - y_{14}^i - y_{15}^i \quad \text{and} \quad y_4^i - y_5^i - y_{16}^i - y_{17}^i,$$

or

$$y_4^i - y_5^i - y_6^i - y_7^i - y_8^i - y_9^i - y_{10}^i - y_{11}^i - y_{12}^i - y_{16}^i - y_{17}^i \quad \text{and} \quad y_1^i - y_2^i - y_3^i - y_{13}^i - y_{14}^i - y_{15}^i.$$

If the algorithm has included y_9^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 (so the whole group takes 9 instead of 8). On the other hand, if the algorithm has excluded y_9^i and the adversary chooses the second configuration, then the algorithm needs 6 for the 11-chain instead of 5 (or 9 instead of 8 for the group). Note again that the algorithm cannot predict the adversary's choices when it is deciding about M because vertices $y_3^i, y_5^i, y_6^i, y_1^i 2, y_1^i 3, y_1^i 6$ 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 $\{y_6^i\}_{i=1}^m$ for δ a constant less than $\lfloor n/17 \rfloor$, 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 $(1/2+\epsilon')1 + (1/2-\epsilon')9/8 = 17/16 - \epsilon$. □

4.2 2-Satisfiability in the Adaptive Model

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

Theorem 11. *There is a width-$O(n)$ adaptive BT algorithm for 2SAT on n variables. Further, this upper bound holds for the weak data type for SAT.*

Proof. (sketch) 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 BT 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 in the graph, such as $l_1 \to l_2 \to l_3 \to \ldots \to l_m$ has only linearly many satisfying assignments; namely the variables 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.

Using an adaptive ordering we can "grow" such a path as follows. Suppose we start with x_1. The algorithm then chooses a variable new x_2 that appears in a clause $\neg x_1 \vee x_2$ if there is one (that is, look at an edge $x_1 \to x_2$). Then, it continue to look for a path $x_1 \to x_2 \to x_3$ and so on. As long as this is possible we only need to maintain a linear number of solutions. When the path is not extendable in this fashion, few different cases are possible. We sketch two. If we get a path

$x_1 \to x_2 \to x_3 \to \neg x_2$ we can safely set x_2 to 1, 'prune' it from the path and continue. If the path is $x_1 \to x_2 \to x_3 \to x_2$ then we know that $x_2 = x_3$ and we introduce a new variable x_{23} that must be set to this common value, and continue with the path $x_1 \to x_{23}$. □

5 3-Satisfiability in the Fully Adaptive Model

We prove an exponential lower bound for 3SAT using 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 local search paradigms from algebraic methods [4].

Theorem 12. *Any fully adaptive BT 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.*

5.1 Linear systems over expanders

Below we present the machinery developed in [3] and [2]. These concepts provide a convenient way to analyze a linear system when the underlying matrix is a "good" expander.

Definition 13. *Let A be an $m \times n$ 0/1-matrix. We denote the i-th row of A by A_i and identify it with the set $\{j \mid A_{ij} = 1\}$. The cardinality of this set is denoted by $|A_i|$. We extend the notation A_i to $A_I = \bigcup_{i \in I} A_i$ for a set of rows $I \subseteq [m]$.*

There are 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 14. *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 $i \in I$ that contains j. We say that A is an (r, s, c)-*boundary expander *if*

1. *$|A_i| \le s$ for all $i \in [m]$, and*
2. *$\forall I \subseteq [m]$ $(|I| \le r \Rightarrow |\partial I| \ge c \cdot |I|)$.*

*Matrix A is an (r, s, c)-*expander *if condition 2 is replaced by*

2'. *$\forall I \subseteq [m]$ $(|I| \le r \Rightarrow |A_I| \ge c \cdot |I|)$.*

It is easy to check that any (r, s, c)-expander is an $(r, s, 2c - s)$-boundary expander.

Definition 15 ([4]). *Let $A \in \{0,1\}^{m \times n}$. For a set of columns $J \subseteq [n]$ define the following inference relation $\vdash_J$ on the set $[m]$ of rows of A:*

$$I \vdash_J I_1 \equiv \ |I_1| \le r/2 \ \wedge \ \partial_A(I_1) \subseteq A_I \cup J. \tag{1}$$

Let the closure $\mathrm{Cl}(J)$ *of J be the set of all rows which can be inferred via $\vdash_J$ from the empty set.*

Lemma 16 ([4]). *Let A be $(r, 3, c')$-boundary expander. For any set J with $|J| \le (c'r/2)$, $|\mathrm{Cl}(J)| \le |J|/c'$.*

[4]We would like to note that if one adds a little bit of random noise into the linear mapping (and the goal would be to find a solution that satisfies almost all equations) then there is no efficient algorithm known for this task. It was conjectured by the first author [1] that the resulting mapping may be a pseudorandom generator against P.

Restrictions and consistency. In what follows the behavior of BT algorithms will be analyzed with pairs of restrictions (σ_x, σ_y). Intuitively σ_x corresponds to the current assignment on x and σ_y corresponds to the known partial information about b. We regard a set of assigned variables $Vars(\sigma_x)$ as a set of columns $J = \{j | \sigma_x(x_j) \in \{0,1\}\}$ and $Vars(\sigma_y)$ as a set of rows $I = \{i | \sigma_y(y_i) \in \{0,1\}\}$.

Definition 17. *For a fixed matrix $A \in \{0,1\}^{n\times n}$ we call a pair of partial assignments (σ_x, σ_y)* consistent *w.r.t. A if σ_x can be extended to a global assignment $x \in \{0,1\}^n$ and σ_y may be extended to $b \in \{0,1\}^n$ such that $Ax = b$.*

Finally we need the following lemma from [2].

Lemma 18. *Let $A \in \{0,1\}^{m\times n}$ be $(r,3,c)$-boundary expander, $b \in \{0,1\}^m$ and σ a restriction on x and denote $I = \mathrm{Cl}(Vars(\sigma))$. Assume that there exists a solution for the system $(Ax)_I = b_I$ that extends σ. Then, for any set of rows I' of size $< r/2$ there exists a solution of the system $(Ax)_{I'} = b_{I'}$ that extends σ.*

The following theorem 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]. The proof appears in the appendix.

Theorem 19. *For any constant $\delta < 2$ there exist a constant $\epsilon > 0$ and $K > 0$ and a family A_n of $n \times n$ matrices s.t.*

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

5.2 Reduction to the backtrack game on a linear system

Our lower bound for BT algorithms comes from a SAT instance corresponding to a linear system $Ax = b$. Surprisingly, the lower bound uses only the facts that the algorithm does not know b and that A is expanding and of full rank; in other words, we will analyze algorithms which know A in advance. In this setting, a successful BT algorithm has a rather limited structure, which we use when analyzing its behavior. We view the algorithm as playing a game with the object of finding a solution to $Ax = b$ for unknown b. While there are good strategies to win this game quickly, we show that any BT algorithm is a very bad player.

Definition 20 (backtrack game on a linear system). *A* strategy *$\mathcal{P}$ is a function that maps a pair of partial assignments to a decision to reveal a bit b_i or to guess bit x_i ($i \in [m]$). For a given strategy $\mathcal{P}$ and a linear system $Ax = b$ we define a* backtrack game tree *$T_{\mathcal{P}}(Ax = b)$ in the following recursive way.*

- *Every node of $T_{\mathcal{P}}(Ax = b)$ contains a pair of partial assignments (σ_x^v, σ_y^v). The assignment σ_y^v is always consistent with the value of b. The assignment (σ_x^v, σ_y^v) is always consistent w.r.t. A.*
- *The root node contains the empty partial assignments.*
- *(Revealing move). If v is a node containing the pair (σ_x^v, σ_y^v) and $\mathcal{P}(\sigma_x^v, \sigma_y^v) =$ <u>reveal i</u> then v has the unique child v' with $\sigma_x^{v'} = \sigma_x^v$, $\sigma_y^{v'} = \sigma_y^v \cup \{y_i = b_i\}$.*

- *(Guessing move). If v is a node containing pair (σ_x^v, σ_y^v) and $\mathcal{P}(\sigma_x^v, \sigma_y^v) = \underline{\text{guess } j}$ then denote $\sigma_x^{v(left)} = \sigma_x^v \cup \{x_j = 0\}$, $\sigma_x^{v(right)} = \sigma_x^v \cup \{x_j = 1\}$. v will have 0,1 or 2 successors in accordance with the number of consistent pairs.*

Hereafter, let us fix an $n \times n$ nondegenerate matrix A which is $(r, 3, c)$-expander that contains at most K ones in each column with $r = \Omega(n)$, $K = O(1)$, $c > 17/9$. The existence of such matrix is provided by Theorem 19. Thus A is $(r, 3, c')$-boundary expander for $c' = 2c - 3$. For b a vector we will treat a linear system $Ax = b$ as a 3CNF that results after expressing every linear equation with 4 clauses in the standard way.

Theorem 21. *Let $b \in_U \{0,1\}^n$. For any BT algorithm $\mathcal{A}$ with probability* $1 - o(1)$

$$W_{\mathcal{A}}(Ax = b) = 2^{\Omega(n)}.$$

Proof. Assume that b is a uniformly chosen random vector. Then $T_{\mathcal{A}}(Ax = b)$ is a random tree the size of which we need to estimate.

Given a backtrack algorithm $\mathcal{A}$ we construct the strategy $\mathcal{P}_{\mathcal{A}}$ for the BT game. While it is difficult to estimate exactly what the algorithm knows about b at any point (mainly because it may have inferred that some items are *not* part of the input), the corresponding strategy will stay ahead of the algorithm in terms of knowledge of b and it will be very easy to track this knowledge. Every node of $\mathcal{A}$ is translated into a sequence of steps of $\mathcal{P}_{\mathcal{A}}$, and we start with the root node of $T_{\mathcal{A}}(Ax = b)$ that is mapped to the root node of $T_{\mathcal{P}_{\mathcal{A}}}(Ax = b)$.

Now, let u be a node in $T_{\mathcal{A}}(Ax = b)$ s.t. the ordering at this point $\mathcal{A}(\vec{D^u}, \sigma_x^u,)$ is a sequence of items $(D_1^u, D_2^u, ...)$. Assume that we have defined a mapping of u into a node v in $T_{\mathcal{P}_{\mathcal{A}}}(Ax = b)$. Possibly D_1^u is already contradictory with (σ_x^v, σ_y^v) in that it contradicts σ_y^v or it represents a variable already set in σ_x^v; in this case we know that D_1^u is not in the input[5]. We therefore assume wlog that D_1^u is not contradictory. Now, let I be the set of bits associated with the clauses of D_1^u. Notice that the bits $\{b_i\}_{i \in I}$ are the ones that determine whether D_1^u is one of the input items associated with $Ax = b$. Also, since every variable may participate in at most K linear equations, $|I| \le K$. The BT game now reveals all the currently unrevealed bits $\{b_i\}_{i \in I}$. Let v' be a new node with $\sigma_x^{v'} = \sigma_x^v$ and $\sigma_y^{v'}$ extends σ_y^v on I. At this point we make a *data-check* that can have the following two possible outcomes.

- **A negative item-check** D_1^u contradicts $\sigma_y^{v'}$ and hence D_1^u is not in the input. In this case the next data item in the list is considered using the above set of rules.
- **A positive item-check** D_1^u does not contradict $\sigma_y^{v'}$, in which case it might be one of the input items. The strategy will proceed regardless. Suppose $D_1^u = (x_j, C_1, ..., C_t)$, and denote $I' = \mathrm{Cl}(Vars(\sigma_x^{v'}) \cup \{j\})$. If $b_{I'}$ contains yet unknown bits that are not set in $\sigma_y^{v'}$ then the corresponding revealing steps take place so that I' is exposed in the vertex v''.

  After that a guessing move takes place w.r.t. the variable x_j. This results in a splitting of the vertex v'' into two vertices v^{left} and v^{right} according to the value of x_j. The translation of one step of $\mathcal{A}$ in the node u is finished and v^{left}, v^{right} are growing according to the two children of u in $T_{\mathcal{A}}(Ax = b)$. If either v^{left} or v^{right} contains an inconsistent pair (σ_x, σ_y) then the corresponding branch(es) are terminated.

Lemma 22. *$P_{\mathcal{A}}$ is a well-defined BT game strategy and there exists an injective function μ that maps guessing nodes of $T_{P_{\mathcal{A}}(Ax=b)}$ to nodes of $T_{\mathcal{A}}(Ax = b)$.*

[5] Note, that in this case it is unreasonable for $\mathcal{A}$ to

Proof. The map μ is defined recursively in the natural way w.r.t. the execution of $\mathcal{A}$. Further, one can inductively show the following semantics : at any node u the amount of information known about b is no more than σ_v^y, where v is the corresponding node in the game. This immediately shows that whenever $\mathcal{A}$ aborts one or two possible extensions of the node u, the corresponding node v in $T_{\mathcal{P}_\mathcal{A}}$ contains an inconsistent pair (σ_v^x, σ_v^y), since otherwise any consistent extension of σ_v^x, σ_v^y could be the unique solution and this will lead to a failure of $\mathcal{A}$ to solve the problem. $\square$

Lemma 22 says that it is enough to bound the size or width of the tree resulting from the strategy $P_\mathcal{A}$ in order to get a simiar bound to the size/width of $\mathcal{A}$.

Lemma 23. *For the strategy $\mathcal{P}_\mathcal{A}$, all leaf nodes in $T_{\mathcal{P}_\mathcal{A}}(Ax = b)$ have depth at least $r/2$.*

Proof. Assume for the sake of contradiction that there exists a path of length less than $r/2$, let v be the last node on this path. By the definition of BT game strategy v contains a pair of restrictions (σ_x^v, σ_y^v) that are consistent w.r.t. A. The only possibility of a path being terminated is when a new bit b_i is revealed and the new pair of restrictions $\sigma_x^v, \sigma_y^v \cup \{y_i = b_i\}$ is inconsistent with A. Let $I = \mathrm{Cl}(Vars(\sigma_x^v))$. By the construction of $\mathcal{P}_\mathcal{A}$, σ_y^v gives values to the bits b_I. This implies that σ_x^v may be extended to satisfy $(Ax)_I = b_I$. By Lemma 18 σ_x^v may be extended to satisfy $(Ax)_{I'} = b_{I'}$ for $I' = Vars(\sigma_y^v) \cup \{i\}$. We have a contradiction. $\square$

Definition 24. *Assume that T is a tree. A random path $\pi(T)$ goes from the root to one of the leaves of T in the following way. The first node of π is the root of T. At every step the next node in the path is chosen uniformly at random from the set of all children of the current node.*

For a strategy $P_\mathcal{A}$ denote by $T_{P_\mathcal{A}}^{r/2}(Ax = b)$ a subtree of $T_{P_\mathcal{A}}(Ax = b)$ that consists of all nodes of depth less than or equal $r/2$. We now consider a random path in the random tree $T_{\mathcal{P}_\mathcal{A}}^{r/2}(Ax = b)$. Our plan is to prove that w.h.p. this path has a lot of branching nodes (i.e. nodes that have more than one child).

Lemma 25. *With probability $1-2^{-\Omega(r)}$ (over random choices of b and π), a random path $\pi(T_{\mathcal{P}_\mathcal{A}}^{r/2}(Ax = b))$ contains $\Omega(r)$ guessing nodes with both children present.*

Proof. Consider a random path π in $T_{\mathcal{P}_\mathcal{A}}^{r/2}(Ax = b)$. By Lemma 23 it has at least $r/2$ edges. We first prove that there are $\Omega(r)$ guessing nodes along this path with high probability. Assume that there are at most $r/4$, otherwise we're done. Let v be the final node of π at depth $r/2$, and let $\hat{t}$ be the number of *item-checks*. Our first goal is to show $\hat{t} = \Omega(r)$. It is enough to show that on average there are at most a constant number of revealing nodes per item-check. There are two types of bits that are revealed. The ones that correspond to the $\leq K$ equations associated with the data items, and the ones that are in the closure of the current partial assignment. Notice that all the revealed bits of the second type are a subset of $\mathrm{Cl}(Vars(\sigma_x^v))$, as $\mathrm{Cl}(\cdot)$ is monotone. Notice also that $|Vars(\sigma_x^v))| \leq \hat{t}$. These two observations, together with Lemma 16, show that for $\hat{t} < c'r/2$, $|\mathrm{Cl}(Vars(\sigma_x^v))| \leq |Vars(\sigma_x^v)|/c' \leq \hat{t}/c'$. Putting it all together, there are (amortized) at most $K + 1/c'$ revealed bits per each *item-check* node (unless $\hat{t} \geq c'r/2$ in which case $\hat{t} = \Omega(r)$ anyway), and so $\hat{t} = \Omega(r)$. We have established that there are many item checks. Next, we need to bound the number of guessing-nodes; those are exactly the nodes associated with positive item-checks. Denote by M_t the number of items that have been identified as present in the formulas after t item checks. Notice that since an *item-check* is positive with probability at least $1/2^K$, the sequence $M_t - 2^{-K}t$ is a submartingale. Applying Azuma's inequality gives that $M_{\hat{t}} \geq \frac{1}{2} \cdot 2^{-K}\hat{t}$ with probability $1 - 2^{-\Omega(t)}$. This implies that w.h.p. the random path contains $2^{-K}\hat{t}/2$ guessing points. Now we need to show that the number of forced choices (i.e. guessing nodes with one child) is not too big.

Lemma 26 ([3]). *Assume that an $m \times n$ matrix A is an $(r,3,c)$-expander, $X = \{x_1, \dots, x_n\}$ is a set of variables, $\hat{X} \subseteq X$, $b \in \{0,1\}^m$, and $\mathcal{L} = \{\ell_1, \dots, \ell_k\}$ is a tuple of linear equations from the system $Ax = b$. 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 $|\hat{X}|\left(\frac{1}{2} - \frac{14-7c}{2(2c-3)}\right)$.*

Denote $\hat{X} = Vars(\sigma_v^x)$ and by $\mathcal{L}$ the set of linear equations in the system $(Ax)_{Vars(\sigma_v^y)} = b_{Vars(\sigma_v^y)}$. By Lemma 26 the space of those partial assignments on $\hat{X}$ that are consistent with σ_v^y has dimension $\Omega(|\hat{X}|) = M_{\hat{t}} \geq 2^{-K}\hat{t}/2 = \Omega(r)$, this implies that there are at least that many guessing points with both children present in $T_{\mathcal{A}_\mathcal{P}}^{r/2}$. ☐

Lemma 27. *With probability $1 - 2^{-\Omega(r)}$, $T_{P_\mathcal{A}}^{r/2}$ contains $2^{\Omega(r)}$ nodes.*

Proof. By Lemma 25 with probability $1-2^{-\delta r}$ a random path in $T_{\mathcal{P}_\mathcal{A}}^{r/2}$ contains at least ϵr branching points. By averaging, this implies that

$$\Pr_b\left[\Pr_\pi[\pi(T_{\mathcal{A}_\mathcal{P}}(Ax=b)) \text{ contains at most } \epsilon r \text{ branching points}] > 2^{-\delta r/2}\right] \leq 2^{-\delta r/2}.$$

Lemma 28. *If a random path in tree T has ϵr branching points with probability $1/2$ then T has at least $2^{\epsilon r}/2$ nodes.*

Proof. Let us give each node v in T an *identifier* which is $0-1$ string of length ϵr that contains the direction on the first ϵr branching point on the way from the root to v. An identifier is *good* iff the correponding path that it defines has at least ϵr branching nodes. There is an injective map from the set of good identifiers to the set of nodes in T and a randomly chosen identifier is good with probability $1/2$. ☐

It follows that $T_{P_\mathcal{A}}^{r/2}(Ax = b)$ has exponential size w.h.p. Lemma 27 is proved. ☐

Theorem 21 now follows by Lemmas 22,27. ☐

Theorem 21 says that for any SAT BT algorithm $\mathcal{A}$ and choosing b uniformly at random, $W_\mathcal{A}(Ax = b) = 2^{\Omega(n)}$ w.h.p. We do not need much additional work to estimate the depth first size. Consider the first branching point of $T_\mathcal{A}(Ax = b)$; it corresponds to the choice of the value of the first variable x_j. Assume that $\mathcal{A}$ recommends considering the assignment $x_j = 0$ first. However with probability $1/2$, $x_j = 1$, but the proof of Theorem 21 implies that the subtree corresponding to $x_j = 0$ has exponential size. Thus with probability $1/2 - o(1)$, $S_\mathcal{A}^{df} = 2^{\Omega(n)}$.

5.3 Free Branching

Here we show that the 3SAT lower bound extends to an even stronger model than fully adaptive BT. We will need this stronger lower bound below where we prove a lower bound for knapsack in the fully adaptive model by a reduction from 3SAT.

For any BT problem P over domain D, augment D by including an infinite number of *dummy* data items $\{R_1, R_2, ..., R_n, ...\}$ and let these dummy items be implicitly included in every instance I. Decisions about these items do not affect the semantics of a solution (they are ignored by f_P).

They do, however, allow any fully-adaptive algorithm $\mathcal{A}$ for P to branch without making a decision about the real data items so that it can, in particular, consider multiple orderings on the remainder of the items (this ability is not useful in fixed or adaptive BT). One can modify and get the following strengthening of Theorem 12.

Theorem 29. *For any fully adaptive BT algorithm with free-branching, $\mathcal{A}$, there exists at least one b for which $W_{\mathcal{A}}(Ax = b) = 2^{\Omega(n)}$.*

The proof idea is to reduce the free-branching case to the original one. Note, that after introducing the dummy variables $R_1, R_2, \ldots$ the solution is not unique anymore, thus we cannot apply Lemma 22 to a free-branching algorithm. Let us consider a class of *possibly incorrect* BT algorithms for SAT that do not necessarily find a solution. For a fixed matrix A we say that $\mathcal{A}$ *finds* a given vector $\hat{x} \in \{0,1\}^n$ iff it solves the system $Ax = (A\hat{x})$.

Definition 30. *Given a free-branching algorithm $\mathcal{A}$ define a possibly incorrect free-branching algorithm $\mathcal{A}^{<t}$ in the following way. Whenever a computational branch of $\mathcal{A}$ makes t free queries cut this branch.*

Let $\ell = \Omega(r)$ be an even integer that we assign later. We may assume that for any b $T_{\mathcal{A}}(Ax = b)$ has size less than $2^{\ell/10}$ (otherwise the theorem follows).

Lemma 31.

$$\Pr_{\hat{x}}[\mathcal{A}^{<\ell/2} \text{ finds } \hat{x}] \geq 1/2.$$

Proof. By our assumption, the tree $T_{\mathcal{A}}(Ax = b)$ has at most $2^{\ell/10}$ nodes for every b. Assume that $\hat{x}$ is fixed, $b = A\hat{x}$. Choose the values $\vec{r}$ of free bits at random. Consider the path that goes from the root to the unique (for any specific $\vec{r}$) solution. This path is a random path in the subtree T of $T_{\mathcal{A}}(Ax = b)$ that contains all nodes v for which σ_x^v is consistent with $\hat{x}$, clearly the size of T is less than or equal $2^{\ell/10}$. By lemma 28 this path has at least $\ell/2$ branching nodes with probability less than 1/2. This means that $A^{<\ell/2}$ finds x with probability at least 1/2. ☐

Lemma 32. *There exists a non-free-branching, possibly incorrect BT algorithm $\mathcal{A}'$ s.t.*

$$\Pr[\mathcal{A}' \text{ finds } x] \geq (1/2)2^{-(\ell/2-1)}.$$

Proof. Consider the free-branching algorithm $\mathcal{A}^{<\ell/2}$, which finds the solution with probability 1/2 and uses $\ell/2 - 1$ random bits. We may deterministically assign these bits so that the resulting algorithm finds a solution with probability $(1/2)2^{-(\ell/2-1)}$. ☐

A possibly incorrect BT algorithm may abort a branch leading to the satisfying assignment. One may extend it to the correct BT algorithm with the following properties.

Proposition 33. *For every possibly incorrect, non-free-branching BT algorithm $\mathcal{A}'$ there exists a correct, non-free-branching algorithm $\hat{\mathcal{A}}$, s.t. for all b $T_{\mathcal{A}'}(Ax = b)$ is a subtree of $T_{\hat{\mathcal{A}}}(Ax = b)$.*

Under our assumption there exists a possibly incorrect, non-free-branching algorithm $\mathcal{A}'$ that solves the problem with probability at least $2^{-\ell}$ according to a randomly chosen b. Consider the corresponding correct BT algorithm $\hat{\mathcal{A}}$ that extends $\mathcal{A}'$ provided by Proposition 33.

Definition 34. *For a BT strategy $P_{\hat{\mathcal{A}}}$, $\hat{x} \in \{0,1\}^n$ and $b \in \{0,1\}^n$ we define a path $\pi_{\hat{x}}(T^{r/2}_{P_{\hat{\mathcal{A}}}}(Ax = b))$ in the following way. Whenever $P_{\hat{\mathcal{A}}}$ makes a revealing move and reads b_i if $b_i = (A\hat{x})_i$ then continue the path, otherwise abort it. If $P_{\hat{\mathcal{A}}}$ makes a guessing move then the path $\pi_{\hat{x}}$ goes according to the value of $\hat{x}$.*

Proposition 35. *Assume that $\pi_{\hat{x}}(T^{r/2}_{P_{\hat{\mathcal{A}}}}(Ax = b))$ has length $r/2$ (i.e. it doesnot abort). Then $\pi_{\hat{x}}$ does not depend on b.*

Proof. The proposition becomes trivial after expanding the definitions. As long as the path $\pi_{\hat{x}}(T_{P_{\mathcal{A}}}(Ax = b))$ does not abort one may define it without knowing b. Namely, instead of revealing b_i one always reads $(A\hat{x})_i$ as a subsitute for b_i, on every guessing point the path follows according to $\hat{x}$. □

Thus for an arbitrary x we may study the path π_x without knowing the specific b.

Proposition 36. *Let $\hat{x} \in_U \{0,1\}^n$. The distribution of $\pi_{\hat{x}}$ coincides with the random path $\pi(T^{r/2}_{P_{\mathcal{A}}}(Ax = b))$ for $b \in_U \{0,1\}^n$.*

Proof. Assume that the bits of b as well as $\hat{x}$ are hidden. We generate paths $\pi_{\hat{x}}$ and $\pi(T^{r/2}_{P_{\mathcal{A}}}(Ax = b))$ in parallel. At every point revealing b_i along the latter path the probability that $\pi_{\hat{x}}$ goes in the same direction is $\Pr[(A\hat{x})_i = b_i] = 1/2$. The local consistency condition provides that this event is independent of the information revealed about $\hat{x}$. At every guessing point x_j along $\pi(T^{r/2}_{P_{\mathcal{A}}}(Ax = b))$ a random direction is chosen, in $\pi_{\hat{x}}$ a random bit $\hat{x}_j$ is revealed. Once again local consistency provides the independence. □

We have almost finished the proof. Recall that $\hat{\mathcal{A}}$ extends $\mathcal{A}'$, in particular we may regard $\mathcal{A}'$ a heuristic that aborts some branches of $\hat{\mathcal{A}}$ in some predefined way. Let us call x good iff

- $\mathcal{A}'$ finds x.
- π_x contains ϵr guessing nodes with both children present, where ϵ is the constant provided by Lemma 25.

By our assumption on $\mathcal{A}'$, it does not abort with probability greater than 2^{-l}, hence there are at least $2^{n-\ell}$ xs that satisfy the first condition. By Lemma 25 at most $2^{n-\delta r}$ xs violate the second condition. If we assume that $\ell + 1 < \delta r$ we infer that there are at least $2^{n-\ell-1}$ good xs. We say that a given b *invokes* $\hat{x}$ iff $\pi_{\hat{x}}$ belongs to $T^{r/2}_{P_{\hat{\mathcal{A}}}}(Ax = b)$.

Proposition 37. *Assume that $\hat{x}$ is good. Then for $b \in_U \{0,1\}^n$, $\hat{x}$ is invoked by b with probability $2^{\epsilon r - r/2}$.*

By averaging argument there exists some b that invokes at least $2^{n-\ell-1}2^{\epsilon r - r/2}$ good xs. Let us check how they *occupy* the leaves of $T_{P_{\hat{\mathcal{A}}}}$ (i.e. to what leaf π_x comes for good x). Every leaf may accomodate at most $2^{n-r/2}$ good xs, thus there should be at least $2^{\epsilon r - \ell - 1}$ leaves occupied. $\mathcal{A}'$ cannot terminate a branch that leads to an occupied leaf. Choosing $\ell = \epsilon r/2$ we infer that $T_{P_{\hat{\mathcal{A}}}}$ contains $2^{\Omega(r)}$ branches not terminated by $\mathcal{A}'$. This proves that $W_{\mathcal{A}}(Ax = b) > 2^{\Omega(r)}$.

5.4 BT Reductions

Given the 3SAT lower bounds in Theorem 29 and utilizing an appropriate reduction from 3SAT to the SUBSET-SUM search problem, we obtain the same exponential lower bounds for SUBSET-SUM and hence for the simple knapsack problem.

Theorem 38. *Any fully adaptive BT algorithm for simple knapsack on n items requires width and depth-first size $2^{\Omega(n)}$.*

Proof. The theorem follows from the standard polytime reduction (see, for example, [11]) from 3SAT to SUBSET-SUM which turns out to be a "BT-reduction". Let $(x_i, C_1, \ldots, C_k)$ be a variable item in the weak or strong data type. Given m 3-clauses on n variables, the reduction creates a set of $n' = 2n + 2m$ base-6 numbers each with $n + m$ digits. Namely, for each propositional variable, the reduction creates two knapack items (one correspnding to positive occurences and one to negative occurences of the variable) and two knapsack items corresponding to each clause. (These clause items will correspond to dummy variables in the SAT domain.) Any BT algorithm (with or without free choices) for solving the knapsack problem will induce a BT algorithm (with free choices) for solving 3SAT. Namely, any decision on either of the literal knapsack items for a particular variable gives us a decision for that variable because an optimal solution to knapsack is one in which exactly one of the items corresponding to positive and negative literals must be taken. For a node in a path of a BT tree that represents the first time an item of that variable appears, we convert the decision to a corresponding truth value for that variable. For the second occurrence in a branch of an item corresponding to a certain variable, we extend the branch (if possible) only where the decision is consistent. Decisions for the clause items will correspond to free decisions. We emphasize that the ordering for the 3SAT instance is well defined and any optimal solution to knapsack (ie one that achieves the target) is a branch that is converted to a branch representing a solution for the 3SAT formula. □

Given the specific example above, it is not difficult to define the concept of a BT-reduction between problems which will preserve the property of being efficienlty computable by a BT algorithm. Intuitively, the main criterion of such a reduction is that it establish a correspondence between items in an instance of the first problem and items in the induced instance of the second problem. In other words, it should be very local. We then need a way to map the decisions of the second problem back to decisions for the first problem. One could formalize this notion at various levels of generality. In order to avoid unnecessary details, we simply say that a problem P_1 reduces to problem P_2 if there is a mapping g from the items of P_1 to sets of items in P_2, and a mapping f from decisions on an item in P_2 to decisions in P_1. We can order $\mathcal{D}_1$ given an ordering on $\mathcal{D}_2$ provided that the $\mathcal{D}_2$-subsets corresponding to different items for P_1 are disjoint. Then, given any decision on the first item in one of these subsets, we use f to decide about the corresponding $\mathcal{D}_1$ item. Given a tree in P_1, the above setting allows us to define an associated tree in P_1. Finally, for the reduction to work, we have to require that those branches of the tree of P_2 that represent a solution (optimal solution in the case of optimization problem) can be associated with solution branches in the corresponding tree in P_1. The one additional point is that, if the induced instance of P_2 contains items that don't correspond to any particular P_1-item (as is the case with the clause items above), then we need to introduce dummy items into P_1's computation.

The following definition and proposition formalize the above intuition.

Definition 39. *Let $P_1 = (\mathcal{D}_1, \{f_1^n\})$ and $P_2 = (\mathcal{D}_2, \{f_2^n\})$ be two BT problems that use choices K_1 and K_2, respectively. A BT-reduction from P_1 to P_2 consists of the following:*

- *A function $g_1^n : \mathcal{D}_1 \to \mathcal{P}(\mathcal{D}_2)$, such that for any two distinct $D, D' \in \mathcal{D}_1$, $g_1^n(D) \cap g_1^n(D') = \emptyset$.*
- *A function $g_2^n : \mathcal{D}_2 \times K_2 \to K_1$.*
- *A set of items $\mathcal{E}^n = \{E_1^n, \ldots, E_{m=m(n)}^n \in \mathcal{D}_2\}$ that are not in the image of g_1^n.*

These objects must satisfy the following property: for any $I = \{D_1, \ldots, D_n\} \subset \mathcal{D}_1$, let $g_1^n(I) \subset \mathcal{D}_2$ denote the union of all images of the items in I under g_1^n. Let $\alpha : g_1^n(I) \to K_2$ be any assignment *of decisions to these items. Finally, for each D_i, let D'_i be any item in $g_1^n(D_i)$. Then,*

$$f_1^n(D_1, \ldots, D_n, g_2^n(D'_1, \alpha(D'_1)), \ldots, g_2^n(D'_n, \alpha(D'_n))) = 1$$

if and only if

$$\exists b_1, \ldots, b_m \in K_2 \; f_2^{n'}(g_1^n(I), E_1^n, \ldots, E_m^n, \alpha(I), b_1, \ldots, b_m) = 1.$$

(We abuse notation slightly in the last line by making $g_1^n(I)$ and $\alpha(I)$ vectors). The number n' is the number of items in the P_2 instance: $|g_1^n(I)| + m(n)$.

Often we drop the n superscript for ease of notation.

Proposition 40. *Given two BT problems $P_1 = (\mathcal{D}_1, \{f_1^n\})$ and $P_2 = (\mathcal{D}_2, \{f_2^n\})$, if there is a fully-adaptive-with-free-branching BT-algorithm $\mathcal{A}$ for P_2 and a BT-reduction from P_1 to P_2, then there is a fully-adaptive-with-free-branching BT-algorithm $\mathcal{B}$ for P_1 such that $W_{\mathcal{B}}(n) \leq W_{\mathcal{A}}(n')$, $S_{\mathcal{B}}^{df}(n) \leq S_{\mathcal{A}}^{df}(n')$ and $S_{\mathcal{B}}^{bf}(n) \leq S_{\mathcal{A}}^{bf}(n')$.*

Proof. Given any instance $I = (D_1, \ldots, D_n)$ of P_1, we construct a computation tree $T_{\mathcal{B}}(I)$ that is isomorphic to the tree $T_{\mathcal{A}}(I')$, where $I' = g_1(I) \cup \mathcal{E}$. This will partially define $r_{\mathcal{B}}$ and $c_{\mathcal{B}}$. For the remainder of the domains of these functions, they may return anything.

We will introduce dummy items into the computation of $\mathcal{B}$ so that there will be a bijection between $\mathcal{B}$'s items and $\mathcal{A}$'s items. Let g'_1 be a function from $\mathcal{D}_1 \cup \{R_1, R_2, \ldots\}$ to $\mathcal{D}_2 \cup \{R'_1, R'_2, \ldots\}$ as follows: for each $Z \in \mathcal{D}_1$ pick an arbitrary $Z' \in g_1(D_i)$ and let $g'_1(Z) = Z'$. For every other item in $g_1(Z)$ pick a unique dummy element to be its preimage under g'_1. Then, extend g'_1 so that it includes a 1-1 correspondence from some of the remaining unused dummy items to $\mathcal{E}$. Finally, extend g'_1 so that it includes a 1-1 correspondence between the remaining dummy items of P_1 to the dummy items of P_2.

The root of $T_{\mathcal{B}}$ corresponds to the root of $T_{\mathcal{A}}$. In general, assume we have a node v of $T_{\mathcal{B}}$ that corresponds to a node u of $T_{\mathcal{A}}$. Let $\langle \vec{D}^u, \vec{a}^u \rangle$ be the label of u and let $\langle \vec{D}^v, \vec{a}^v \rangle$ be the label of v. Then, since we have an injection $g'_1 : \mathcal{D}_1 \cup \{R_1, \ldots\} \hookrightarrow \mathcal{D}_2 \cup \{R'_1, \ldots\}$, the ordering $r_{\mathcal{A}}(\vec{D}^u, \vec{a}^u)$ induces an ordering on $\mathcal{D}_1 \cup \{R_1, \ldots\}$. Let $r_{\mathcal{B}}(\vec{D}^v, \vec{a}^v)$ output this induced ordering. Now v sees the first item, Z, in $I \cup \{R_1, \ldots\}$ minus the elements of $\vec{D}^v$ according to the ordering. Let $(b_1, \ldots, b_d, \perp, \ldots)$ be the ordering returned by $c_{\mathcal{A}}(\vec{D}^u, Z' = g'_1(Z), \vec{a}^u)$. Define $c_{\mathcal{B}}(\vec{D}^v, Z, \vec{a}^v)$ as the tuple $(g_2(Z', b_1), \ldots, g_2(Z', b_d))$. This establishes a correspondence between the children of u and the children of v.

The definition of reduction guarantees that any path in $T_{\mathcal{B}}$ that corresponds to a path constituting a solution in $T_{\mathcal{A}}$ is itself a solution. Hence $W_{\mathcal{B}}(n) \leq W_{\mathcal{A}}(m)$, $S_{\mathcal{B}}^{df}(n) \leq S_{\mathcal{A}}^{df}(m)$ and $S_{\mathcal{B}}^{bf}(n) \leq S_{\mathcal{A}}^{bf}(m)$. □

6 Interval scheduling

Interval selection is the classical problem of selecting, among a set of intervals associated with profits, 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.

When all the profits are the same, a straight-forward greedy algorithm (in the sense of [8]) 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 BT algorithm. A completely different approach that uses 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 DP approach is inferior here.

It has been shown in [8] 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 BT, even for the special case of proportional profit 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 model where we have constant inapproximability results, and then we will prove lower bounds for the adaptive case, which is considerably more involved.

6.1 Interval Scheduling in the Fixed Model

Theorem 41. *A width $\gamma < \binom{\frac{n-1}{2}}{m}$ fixed-ordering BT for interval scheduling with proportional profit on m machines and n intervals cannot achieve a better approximation ratio than $1+\delta-\frac{1}{2m(2\gamma^{1/m}+1)}$, for any $\delta > 0$.*

Proof. We begin with the special case of $m = 1$. The set of possible inputs are intervals in $[0, 1]$ of the form $[a/W, b/W]$ where a, b are integers and W is a function of γ which will be fixed later. We start with some definitions.

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 $[b, 1]$ is called a *one interval*. Let L be the set of zero-intervals whose right endpoint is at most $\frac{1}{2}$, and let R be the set of one-intervals whose left endpoint is at least $\frac{1}{2}$. 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 notice 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 $2t - 1$. If we look at the first $2t - 1$ elements of S, the majority of them are (wlog) 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 in any way such that if $[0, \frac{1}{2}]$ and $[\frac{1}{2}, 1]$ are both selected, then they are matched to each other. This matching, along with the t distinct middle intervals (one of which may be empty) needed to connect each pair of the matching, constitutes a set of t unsettled complete triplets.

Now, consider a BT 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 wlog that all of the zero-intervals come before the one-intervals. Since no two 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. 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. This is not quite enough, however, since here we have only $2(\gamma+1)+1$, rather than n, input items. To handle this we include in the set of input items $n-(2(\gamma+1)+1)$ v"dummy" intervals of length δ/n for arbitrarily small δ. These dummy intervals can contribute at most δ to the non optimal solution, which gives an inapproximation bound of $1+\delta-1/W = 1+\delta-1/2(2\gamma+1)$ for any $\delta > 0$.

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 $m-1/W$ (or $m+\delta-1/W$ if we add $n-(2t+1)$ dummy intervals) and our approximation ratio is at most $\frac{m+\delta-1/W}{m} \leq 1+\delta-1/(2m(2t+1)) \sim 1+\delta-1/2m(2\gamma^{\frac{1}{m}}+1)$. ☐

Remark 2. *As in Theorem 7, the proof of Theorem 41 can also be extended so as to allow revocable acceptances. Setting $\gamma = 1$ and $m = 1$ in Theorem 41 slightly improves an inapproximation bound of [20] although the bound in [20] applies to adaptive orderings.*

6.2 Interval Scheduling in the Adaptive Model

The following theorem shows that adaptivity still does not allow a BT algorithm to get a substantial reduction in width compared to the Simple DP setting. Specifically we show that for a constant number of machines m, $\Omega(n^m)$ width is required to solve the problem exactly. However, it is quite clear that the way to achieve a lower bound must change dramatically. We cannot hope, for example, to get a superconstant lower bound if the adversary offers an input which contains a set of three intervals covering $[0,1]$, as is the case with the lower bound of the Theorem 41. This intuitively leads to a setting in which the intervals are small, so as to necessitate a large set of intervals in any optimal solution.

Before we turn to the lower bound we show that an adaptive BT algorithm of width 2 supplies a better approximation than priority algorithms (width 1); namely we get an approximation ratio of 1/2, which beats the tight approximation result for priority algorithms of 1/3 was shown in [8].

Theorem 42. *In interval scheduling with* 1 *machine, a width* 2 *adaptive BT guarantees an approximation ratio of* 1/2.

Proof. We take an ordering resembling the 2-CHAIN ordering of intervals discussed in [8]. We start with the left-most interval in terms of the start time. Next, among the intervals intersecting the most recent interval, $[s,t]$, we take an interval $[s',t']$ with $t' > t$ and t' maximal. If there is no such interval, we choose the interval with smallest start point $s' > t$. If there is no such interval we take an arbitrary ordering of the remaining intervals. The resulting ordering is depicted in Figure 1. It is clear that up to the point where we take the arbitrary ordering, the odd intervals are disjoint from each other, and so are the even. Moreover the remaining intervals are contained in the union of the intervals already seen. Therefore, our algorithm considers exactly two possible solutions: the set of even intervals and the set of odd intervals. One of these must achieve approximation ratio at least 1/2. ☐

Theorem 43. *The width of an optimal adaptive BT for interval scheduling with proportional profits on m machines and n intervals is $\binom{\Omega(n/m)}{m}$.*

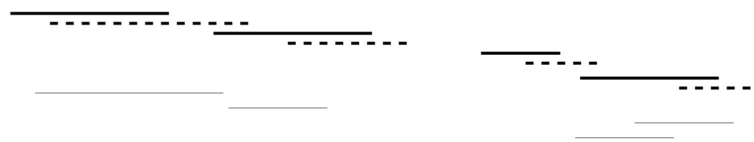

Figure 1: ordering adaptively to get 2 approximation factor for width-2 BT. Only bold and dashed intervals are credited to the BT algorithm.

Proof. The set of data items are the intervals of size at most $1/n$ in $[0, 1]$ with endpoints of the form i/W for any $W > mn^2$. We will associate a graph with the set of revealed intervals, where the endpoints of the intervals are vertices in the graph, and the intervals themselves are the edges. We also define a point r to be *zero connected* (*one connected*) if r is connected to 0 (respectively, 1) in the graph by edges going left (right). An interval is zero (one) connected if both its endpoints are zero (one) connected; it is *generic* if neither of its endpoints are.

The adversary applies the following two rules for eliminating intervals for $n - 1$ phases (until $n - 1$ intervals have been revealed). Initially, we have a set $V = \{0, 1\}$. Throughout the algorithm's progress, we will add to V the endpoints of the revealed intervals. At each stage,

1. Cancel all unseen intervals both of whose endpoints are in V.

2. Cancel all intervals ending (starting) in r if r is zero connected (one connected) and we have already seen 2 intervals ending (starting) in r.

We denote by P_i the set of revealed intervals after i phases of the game. Clearly $|P_i| = i$. We are interested in $P = P_{n-1}$, the set of $n - 1$ revealed intervals at the end of the game. The union of all intervals in P is strictly contained in $[0, 1]$ as all intervals are of length at most $1/n$. Notice that the graph associated with P is acyclic because each interval contributes at least one endpoint not previously seen. Further, when an interval is revealed, it is either generic and stays that way, or it is zero (one) connected.

Let C_0 be the set of zero connected intervals in P and let C_1 be the set of all one connected intervals in P and let G be the remaining intervals in P. Note that since P does not cover the entire interval, no point/interval can be both zero and one connected, and thus every interval in P lies in at most one of the sets G, C_0, C_1.

We will now define three types of indispensable partial solutions, those that involve only generic intervals, those that involve only zero connected intervals, and those that involve only one connected intervals. We will argue that there must be a large number of indispensable solutions of at least one of the three types.

A set of m generic intervals $S = \{I_1, I_2, \ldots, I_m\} \subset G$ is indispensable (with respect to P) if there is a valid set of future inputs F_S that extends S to a complete solution, but does not extend any other $S' \subset P$ to a complete solution.

Claim 44. *All choices of $S = \{I_1, I_2, \ldots, I_m\} \subset G$ of generic intervals are indispensable. Moreover, $|F_S| = O(mn)$.*

Proof. (of Claim 44) Let $I_j = [s_j, r_j]$, $j = 1, \ldots m$ be the generic intervals. For each j, we take F_j to be a set of intervals each of length at least $\frac{1}{2n}$ that together with I_j cover (with no overlaps) the interval $[0, 1]$. Further, they avoid all edge points underlying P and $F_1, \ldots, F_{j-1}$ except for s_j and

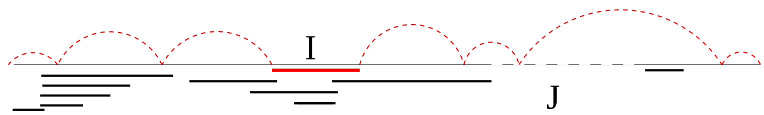

Figure 2: $\{I\}$ is indispensable

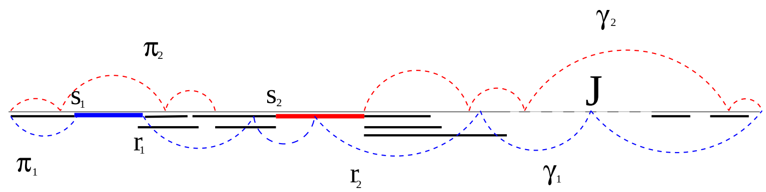

Figure 3: (r_1, r_2) are indispensable

r_j (it's not hard to see that this is possible given that $W > 5mn^2$). Let $F_S = \cup_j F_j$. Clearly F_S has size at most $2mn$. If we accept exactly I_j and F_j on machine j, we get a complete solution. We argue that there is no complete solution other than this one (modulo permutations of the machines) over $P \cup F_S$. Since P involves $n-1$ intervals, each of length at most $1/n$, there must be some gap J that is not covered by anything in P. To get a complete solution, each machine must accept an interval overlapping J; let's say some machine schedules $a \in F_i$. How can we get a complete solution for this machine? Assume wlog that I_i lies to the left of J. Clearly, the machine must schedule all intervals of F_i lying to the right of I_i. Further, since r_i is not zero-connected, the only way to connect r_i to zero is via I_i and the left intervals of F_i. The same holds for each of the machines. □

We will now discuss the zero connected case. The one connected case is analogous. A partial solution of intervals to machines is *projected* to a multiset of m points by taking the rightmost points covered by each machine. For example, if intervals $\{[.2, .4], [.5, .6]\}$ are scheduled on machine 1, and interval $\{[0, .45]\}$ is scheduled on machine 2, then this solution projects to $R = \{.6, .45\}$ A multiset R of zero connected points/intervals is *indispensable* if for some partial solution ρ defined on P that projects to R, there is a set of future inputs, F_R, that extends ρ to an optimal solution, but does not extend any solution that does not project to R. We think of the projection operation as defining an equivalence relation over all partial solutions, and we will show that each equivalence class induced by this relation is indispensable. This implies that at least one partial solution from each class must be maintained by the algorithm, and thus we will show a lower bound equal to the number of equivalence classes.

Claim 45. *Every independent set $R = \{r_1, r_2, .., r_m\} \subset C_0$ of zero connected points/intervals is indispensable. Moreover $|F_R| = O(mn)$. Similarly, every independent set $L = \{l_1, l_2, \ldots, l_m\} \subset C_1$ of one connected points/intervals is indispensable, and $|F_L| = O(mn)$.*

Proof. (of Claim 45) Let $R = \{r_1, r_2, \ldots, r_m\} \subset C_0$ and assume wlog that $r_1 < r_2 < r_3 \ldots$. Also assume that R is an independent set, namely no interval $[r_i, r_j]$ appears in P. To define F_R, we first take paths [6] γ_j starting from 1 and ending at r_j while avoiding all existing edge-points of P except 1 and r_j (see figure 3). This is easy to do, as there are no restrictions as a result from the adversary rules at the first phase (the only thing to notice is that r_j cannot be one-connected). As before we can make sure that there are only $O(mn)$ intervals in those paths by insisting that the path contain edges of size $\Omega(1/n)$.

We now present a set of edge disjoint paths $\pi_1, \ldots, \pi_m$ from r_j to 0 as follows. We will start by defining π_m. Its first edge is the unique zero-connected interval ending at r_m, call it $[s_m, r_m]$. Now consider the set of edges in P going left from s_m. There are at most two of them, one of them, call it $[s_{m1}, s_m]$ which is part of the zero-connected chain containing the point r_m, and possibly another *free* interval with right endpoint s_m, call it $[s_{m2}, s_m]$ which is not zero connected. We know that it's not zero connected because there are no cycles in the underlying graph. If the free interval is in P, then we use it in extending π_m to the left, and otherwise, we add a new interval $[s_{m_2}, s_m]$ to the future set of inputs. In either case, the interval $[s_{m_2}, s_m]$ extends π_m the left. Now we proceed as in Claim 44, and extend π_m to the left all the way to zero, by adding future intervals each of length at least $1/2n$, avoiding all edge points of P along the way. The other π_i's, $i = 1, \ldots, m-1$ are defined similarly. Let F_i denote the set of intervals added to P to obtain γ_i and π_i, and let Q_i denote the intervals in $\gamma_i \cup \pi_i$ that are in P. Let $F_R = \cup_i F_i$ denote the set of all future intervals added.

Having constructed γ_i and π_i we need to show that R is indispensable with respect to F_R. For each i, we defined a complete coverage of the $[0, 1]$ interval by the path $\gamma_i \cup \pi_i$, made up of intervals from $Q_i \subset P$, and F_i. It is easy to see that the partial solution obtained by accepting the intervals in Q_i on machine i, and rejecting all other intervals in P is contained in the equivalence class defined by R. Further, this partial solution can be extended to a complete solution within F_R (by the γ_i's and the π_i's).

It remains to show the converse; namely, that any partial solution in P that can be extended to a complete solution projects to R. We must take (wlog) the intervals of γ_i on machine i. But since there are no future intervals extending from r_i to the left, the only way to get to a complete solution is to take intervals from P which shows the desired property of the solution. □

We can now complete the proof of our theorem. Let $|G| = g$, $|C_0| = c_0$ and $|C_1| = c_1$. We will first argue that one of g, c_0 or c_1 is at least $(n-1)/6$. Whenever an interval $[s, r]$ in P is revealed, it must be in G, C_0 or C_1 unless one of the following two cases occurs: (i) r is zero connected but s is not, or (ii) s is one connected, but r is not. Case (i) can occur only once for each revealed member of C_0 (since each zero connected point can have at most two intervals going left due to the elimination rules), and case (ii) can occur only once for each revealed member of C_1. Hence, $g + 2c_0 + 2c_1 \geq n$, so one of them is at least $(n-1)/6$. If $g \geq (n-1)/6$, then from Claim 44, we need $\binom{g}{m} = \binom{\Omega(n)}{m}$ active solutions after the first $n-1$ rounds. Otherwise, if $c_0 \geq (n-1)/3$, then since we are dealing with acyclic graphs, we can extract an independent set from C_0 of size at least $|C_0|/2 \geq (n-1)/12$. Thus by Claim 45, we need to maintain all the possible projections of this set onto the m machines. There are $\binom{\Omega(n)}{m}$ of these. Again the one connected case is analogous. Since the number of input intervals altogether is $N \leq O(mn)$, we get a lower bound of $\binom{\Omega(N/m)}{m}$, expressed in terms of the input size N. □

[6] taking a path means taking the intervals induced by the path

7 Open Questions

There are many open questions regarding our BT 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 BT algorithm? This is particularly interesting due to the lack of tight complexity-bound inapproximation results for vertex cover (see [7]). For interval scheduling, can the adaptive BT 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 BT 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 BT? We also do not know if a $O(1)$-width adaptive BT can achieve an $O(1)$-approximation ratio for interval scheduling with arbitrary profits. Also for any of the problems already studied in the priority framework (eg [8, 15, 5]) it would be interesting to consider constant width BT algorithms.

Will the BT 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.

Finally, while we have shown that the BT model has strong connections to dynamic programming and backtracking, can it be extended to capture other common algorithms? For example, we show that BT captures simple dynamic programming (where we consider the input items one-by-one), but what about other dynamic programming algorithms, such as the longest-common-subsequence or string alignment algorithm? Also, it seems natural to augment BT with randomness: each decision would be taken with a certain probability and one would study tradeoffs between the expected width and the probability of obtaining a solution. Finally, [14] recently defined a model that enhances BT by making use of memoizing. How much can this improve on the complexity of BT algorithms?

8 Acknowledgments

We thank Spyros Angelopoulos, Paul Beame, Sashka Davis, Jeff Edmonds, and Charles Rackoff for their very helpful comments. A special thanks goes to Marchetti-Spaccamela for contibuting the upper bound of theorem 7.

References

[1] M. Alekhnovich. More on average case vs approximation complexity. In *Proc. 44th Ann. Symp. on Foundations of Computer Science*, 2003.

[2] M. Alekhnovich. Lower bounds for k-DNF resolution on random 3CNF. Manuscript, 2004.

[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.

[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. In *Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization*, volume 2462 of *Lecture Notes in Computer Science*, pages 26–39. Springer-Verlag, 2002.

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

[7] S. Arora, B. Bollobás, and L. Lovász. Proving integrality gaps without knowing the linear program. In *Proceedings of the 43rd Annual IEEE Conference on Foundations of Computer Science*, pages 313–322, 2002.

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

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

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

[11] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition (Page 1015).* MIT Press, Cambridge, Mass., 2001.

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

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

[14] S. Davis and R. Impagliazzo. A formal model of dynamic programming algorithms. Manuscript in preparation, 2004.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

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

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

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

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

[23] 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.

[24] A. Razborov and S. Rudich. Natural proofs. *J. Comput. Syst. Sci.*, 55(1):24–35, 1997.

[25] C. E. Shannon. The synthesis of two-terminal switching circuits. *Bell Systems Tech. J.*, 28:59–98, 1949.

[26] 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 Theorem 19) We use the probabilistic method that gives a good source of expanders. There are two subtle points here, the maximal number of ones in a column and the rank of the resulting matrix. We handle the former issue using the trick from [2] and the latter by the techniques from [3].

Lemma 46 ([3]). *Let L be a linear subspace of $\{0,1\}^n$ of codimension k. Let the vector v be random vector in $\{0,1\}^n$ with weight 3. Then* $\Pr[v \notin L] = \Omega(\frac{k}{n})$.

Assume that n is an integer and c is a constant slightly bigger than δ, say $c = \delta + (2-\delta)/2$. Let K be a large constant the exact value of which will be determined later. Consider the random matrix A_n that contains Kn rows, each of which is chosen uniformly at random from the set of all rows of weight 3.

Proposition 47. *With probability* $1 - o(1)$ $\mathrm{rk}(\mathrm{A_n}) > (1 - \mathrm{e}^{-\Omega(\mathrm{K})})\mathrm{n}$.

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 46 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

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

□

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

Lemma 48. *With probability* $1 - o(1)$ A_n *is* $(r, 3, c)$*-expander for* $r = o\left(\frac{n}{K^{1/(2-c)}}\right)$.

Denote by $\hat{n} = \mathrm{rk}(\mathrm{A_n})$. By Proposition 47 $\hat{n} > (1 - e^{-\Omega(K)})n$. One may remove at most $2^{-\Omega(K)}n$ columns from A_n so that the resulting $(Kn) \times \hat{n}$ matrix has rank $\hat{n}$. Denote by J_1 the index set of these columns. Denote by J_2 the index set of the columns that contain at least $\frac{3K}{c-\delta}(n/r)$ ones. Since A_n has overall Kn ones $|J_2| \le (c-\delta)r/3$. Denote by $\hat{K} = 2Kn/r$, note that it is still a constant. If we remove from A all the columns corresponding to $J_1 \cup J_2$ then the resulting matrix has the 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 the peeling procedure developed in [3].

Definition 49 ([3]). *For an* $A \in \{0,1\}^{m\times n}$ *and a subset of its columns* $J \subseteq [n]$ *we define an inference relation* $\vdash_J^\delta$ *on the set of* $[m]$ *rows of* A:

$$I \vdash_J^\delta I_1 \equiv |I_1| \le r/2 \;\wedge\; \left|\bigcup_{i\in I_1} A_i \setminus \left[\bigcup_{i\in I} J_i(A) \cup J(t)\right]\right| < \delta|I_1| \tag{2}$$

Given a set of rows I and a set of columns J consider the following cleaning *step:*

- *If there exists a nonempty subset of rows* I_1 *such that* $I \vdash_J^\delta I_1$*, then*
  - *Add* I_1 *to* I*.*
  - *Remove all rows corresponding to* I_1 *from* A*.*

Repeat the cleaning step as long as it is applicable. Fix any particular order on the sets to exclude ambiguity, initialize $I = \emptyset$ *and denote the resulting content of* I *at the end by* $\mathrm{Cl}^{\mathrm{e}}(J)$*.*

Proposition 50. *For* $A \in \{0,1\}^{m\times n}$ *a matrix with* 3 *ones in each row and* $J \subseteq [n]$ *denote* $\hat{J} = \bigcup_{\mathrm{Cl}^{\mathrm{e}}(J)} A_i$ *and* $\hat{I} = \mathrm{Cl}^{\mathrm{e}}(J)$*. Let* $\hat{A}$ *be the matrix that results from* A *after removing the columns* $\hat{J}$ *and the rows* $\hat{I}$*. Then* $\hat{A}$ *is* $(r/2, 3, \delta)$*-expander.*

Lemma 51. *If* $|J| < (c-\delta)r/2$*, then* $|\mathrm{Cl}^{\mathrm{e}}(J)| < 2(c-\delta)^{-1}|J|$ *no matter in what order have cleaning steps taken place.*

Proof. Assume that $|\mathrm{Cl}^{\mathrm{e}}(J)| \geq 2(c-\delta)^{-1}|J|$. Consider the sequence $I_1, I_2, \ldots, I_t$ appearing in the cleaning procedure; i.e.,

$$J \cup \bigcup_{i \in I_1 \cup I_2 \cup \ldots \cup I_k} A_i \vdash_A I_{k+1}.$$

Note that $I_i \cap I_{i'} = \emptyset$ for $i \neq i'$, because we remove the implied set of rows from A at each cleaning step. Denote $C_t = \bigcup_{k=1}^t I_k$. Let T be the first t such that $|C_t| > (c-\delta)^{-1}|J|$. Note that $|C_T| \leq r$. Because of the expansion properties of A, $|\bigcup_{i\in C_T} A_i| \geq c|C_T|$, which implies

$$|\bigcup_{i\in C_T} A_i \setminus J| > c|C_T| - |J| \geq c|C_T|/2. \tag{3}$$

On the other hand, every time we add some I_{t+1} to C_t during the cleaning procedure, only $\delta|I_{t+1}|$ new elements can be added to $\bigcup_{i\in C_t} A_i \setminus J$ (of those elements that have never been there before). This implies

$$|\bigcup_{i\in C_T} A_i \setminus J| \leq \delta|C_T|,$$

which contradicts (3). □

We are ready to finish the theorem. Choose K large enough so that $|J_1 \cup J_2| < (c-\delta)r/2$. Denote by

$$\hat{J} = \bigcup_{i\in \mathrm{Cl}^{\mathrm{e}}(J_1\cup J_2)} A_i, \qquad \hat{I} = \mathrm{Cl}^{\mathrm{e}}(J_1 \cup J_2).$$

By Lemma 51 $|\hat{J}| < r$. If we remove all columns corresponding to $\hat{J}$ from A then the resulting matrix A' has the full rank. This is because after we remove the columns corresponding to J_1 we get the matrix in which all columns are linearly independent, thus after the removal of $\hat{J}$ the matrix still has the full rank. Let us choose any $\hat{n}$ linearly independent rows of A' and denote the resulting matrix by $\hat{A}$. The latter is a nondegenerate $\hat{n}\times\hat{n}$ matrix moreover it doesnot contain any rows from $\hat{I}$ (because the latter are empty). By Proposition 50 this implies that $\hat{A}$ is $(r/2, 3, \delta)$-expander. □