

# The Power of Free Branching in a General Model of Backtracking and Dynamic Programming Algorithms

SASHKA DAVIS  
Dept. of Computer Science  
UCSD  
sdavis@cs.ucsd.edu

JEFF EDMONDS  
Dept. of Computer Science  
York University  
jeff@cs.yorku.ca

RUSSELL IMPAGLIAZZO  
Dept. of Computer Science  
UCSD  
russell@cs.ucsd.edu

October 13, 2011

## Abstract

The *Prioritized branching trees* (pBT) model was defined in [ABBO<sup>+</sup>05] to model recursive backtracking and dynamic programming algorithms. We extend this model to the *prioritized free branching tree* (pFBT) model which allows the algorithms to branch in order to choose different priority orderings (greedy criteria). We also consider *free data items* which are useful when doing reductions. We prove surprising subexponential upper bounds in this model and matching lower bounds for 7-SAT and subset sum.

## 1 Introduction

Knowing the importance of these algorithmic techniques, it is interesting that information theoretic lower bounds have been proven for online algorithm [?], that the *priority* model extends these ideas to greedy algorithms [?] and that the *prioritized branching trees* (pBT) model extends these ideas to recursive backtracking and much of dynamic programming algorithms [ABBO<sup>+</sup>05]. Within this pBT model, [ABBO<sup>+</sup>05] was able to obtain a width  $2^{\Omega(n)}$  lower bound for 3-SAT and for subset sum (knapsack). Being able to do general reductions, however, escaped them because of dummy data items that they introduce.

The power of greedy algorithms over online algorithms is being able to choose the *priority order* (greedy criteria) specifying the order in which the data items are processed. The power of this is that when the algorithm receives the data item  $D$ , it inadvertently learns that all data items appearing before  $D$  in the priority ordering are *not* in the instance. This might be useful extra information before the algorithm is forced to irrevocably decide the part of solution related to the received data item, (for example, should the data item be included or not in the solution). The extra power that the pBT model has is allowing the algorithm to branch making different irrevocable decisions in the different branches and allowing the algorithm, later when the computational path does not seem to be working well, to prune off this branch and back track. We extend this model to the *prioritized Free Branching Tree* (pFBT) model by allowing the algorithm to also branch in order to try different priority orderings. Another hope was that the ability to free branch would solve the problem with dummy data items when doing reductions. Our set out goal was to extend the  $2^{\Omega(n)}$  width lower bound for 3-SAT and for subset sum to this new model. Towards this goal we developed a new general proof technique for this model. We, however, were only able to prove a lower bound of  $2^{\Omega(\sqrt{n})}$  for 7-SAT. This is achieved using a reduction from a slightly modified version of the *matrix inversion* ( $Mx = b$ ) problem introduced in [ABBO<sup>+</sup>05]. For completion, we also use the new technique to improve the  $2^{\Omega(n)}$  bound for these problems in the pBT model. See Figure 1 for a summary of both these and our other results.

Computational Problem	$ \mathcal{D} $	$W$ in pBT	$W$ in pFBT	$W$ with $m$ f.d.i.
General	$d$	$2^{\mathcal{O}(n)}$	$2^{\mathcal{O}(\sqrt{n} \log d)}$	$2^{\mathcal{O}\left(\frac{n \log d}{m + \sqrt{n \log d}}\right)}$
$k$ -SAT $\mathcal{O}(1)$ clauses per var	$n^{\mathcal{O}(1)}$		$2^{\mathcal{O}(\sqrt{n} \log n)}$	
$k$ -SAT $m$ clauses	$2^{n^{k-1}}$		$2^{\mathcal{O}(n^{\frac{1}{3}} m^{\frac{1}{3}} \log n)}$	
$k$ -SAT			?	
Matrix Inversion	$n^{\mathcal{O}(1)}$	$2^{\Omega(n)}$	$2^{\Omega(\sqrt{n})}$	$2^{\Omega\left(\frac{n}{m + \sqrt{n}}\right)}$
7-SAT $\mathcal{O}(1)$ clauses per var				
Matrix Parity	$n^{2^n}$		$2^{\Omega(n)}$	$2^{\Omega\left(\frac{n^2}{m+n}\right)}$
Subset Sum			$2^{\Omega(n/\log n)}$	
SS no carries or in pFBT <sup>-</sup>			$2^{\Omega(n)}$	

Figure 1: The summary of the results in the paper.

Surprisingly, our  $2^{\Omega(\sqrt{n})}$  bound was the right answer. Free branching increases the power of the algorithms significantly more than initially expected. Being able to make  $W_{\mathcal{A}}(n)$  different priority orderings, where  $W_{\mathcal{A}}(n)$  is the width of the computation tree would add some power to the algorithm, however, the real power comes when the algorithm is allowed to prune any path in which the number of data items inadvertently learned not to be in the instance is not as much as that hoped for. Effectively this allows for  $[W_{\mathcal{A}}(n)]^{\ell}$  different orderings, where  $\ell$  is the depth during which this game is played. This decreases the computation tree width needed to solve any computation problem from  $2^{\Omega(n)}$  to  $2^{\mathcal{O}(\sqrt{n} \log |\mathcal{D}|)}$  because the algorithm is able to prematurely learn what the input instance is by learning all of the data items from the domain  $\mathcal{D}$  that are not in the instance. This is  $2^{\mathcal{O}(\sqrt{n} \log n)}$  when the domain  $\mathcal{D}$  of data items is no bigger than  $n^{\mathcal{O}(1)}$  as is the case for the matrix inversion problem and for  $k$ -SAT restricted to having only  $\mathcal{O}(1)$  clauses per variable. On the other hand, this result does not directly improve the width, when the data item domain is  $|\mathcal{D}| = 2^{\Omega(n)}$ , as is the case for the general  $k$ -SAT problem. Despite the size of its data instance domain, this algorithm can surprisingly be extended to a  $2^{\mathcal{O}(n^{\frac{1}{3}} m^{\frac{1}{3}} \log n)}$  when the instance is restricted to having only  $m$  clauses. This is  $2^{\mathcal{O}(n^{1-\epsilon})}$  as long as the number of clause is  $m = o(n^2)$ . Despite the fact that our intuition is that  $k$ -SAT is easier to solve when there are lots of clauses, for pFBT we have been unable to obtain a better upper bound.

The upper bound told us that proving a  $2^{\Omega(n)}$  width lower bound would require a problem whose domain  $\mathcal{D}$  of potential data items is  $2^{\Omega(n)}$ . We also know that all these information theoretic algorithms do not perform well when they do not receive the information that they need when they need it. This motivated introducing the embarrassingly simple problem referred to as the *matrix parity* problem. The  $j^{\text{th}}$  input data items reveals the  $j^{\text{th}}$  column of a matrix and forces a decision on the parity of the  $j^{\text{th}}$  row. We give a tight lower bound of  $2^{\Omega(n)}$  on the width needed for this problem.

After proving lower bounds, one wants to be able reap more benefit using reductions. Many reductions require the introduction of *dummy* data items. For example, the obvious reduction from subset sum to the matrix parity problem introduces two data items for each variable and  $\log n$  for each of the  $n$  rows of the matrix. Without proving a lower bound directly for subset sum, we do not really know for sure (as is strongly believed) that these dummy data items are the same as the real data items for an actual subset sum algorithm. Hence, during the reduction these dummy data items are translated into what we refer to as *free data items*. These are additional data items that have no bearing on the solution of the problem and the algorithm need not make any decision about them. Surprisingly, however, we show that any computational problem can be solved in pBT with width  $W = (n \log |\mathcal{D}|)^3$  and  $m = O(n \ln |\mathcal{D}|)$  free data items.

Despite these challenges with doing reductions, we have developed two ways of doing them. For the

first way we extend our lower bounds to the tight bounds of  $2^{\Omega(\frac{n}{m+\sqrt{n}})}$  for matrix inversion and  $2^{\Omega(\frac{n^2}{m+n})}$  for matrix parity in pFBT when the problems also includes  $m$  free data items. Given the subset sum reduction requires  $m = \mathcal{O}(n \log n)$ , this translates into a  $2^{\Omega(n/\log n)}$  width bound for subset sum. Without carries, subset sum only needs  $m = n$  dummy data items giving a  $2^{\Omega(n)}$  bound. There is also an obvious reduction from 3-SAT to the matrix parity problem. There are two problems with it, however, that are worth mentioning. The first is that because the number of dummy data items needed is  $m = n^2$ , the more general result  $2^{\Omega(\frac{n^2}{m+n})}$  gives nothing. The second problem is that the version of 3-SAT arising from the reduction really can be solved with constant width in pFBT because the dummy data items do in fact reveal too much information about the instance.

The second way of getting around the challenges of doing reductions is to restrict the model pFBT to pFBT<sup>-</sup>. At any point during the current computation path, an adversary is allowed to completely reveal any data items that it worries the algorithm may use as a *free data item* making them into *known data items*. Knowing that this known data item is in the actual input instance, the algorithm no longer has any direct need to receive this data item. Besides if the algorithm did receive it, it would be forced to make an irrevocable decision about it, which it may or may not be prepared to do. The only remaining purpose of receiving a known data item is to inadvertently learn that other data items are not in the instance because they appear before these known data items in the priority order. The pFBT<sup>-</sup> model is a restriction of the pFBT model that does not allow the priority ordering to mix known and unknown data items together. Having removed their power, the  $2^{\Omega(n)}$  width lower bound for the matrix parity problem goes through in the pFBT<sup>-</sup> model no matter how many known/free data items there are. The reduction then gives the  $2^{\Omega(n)}$  width lower bound for subset sum within this model.

The paper is organized as follows. Section 2 formally defines the computation problems and the pFBT model. It also motivates this model in its ability to express online, greedy, recursive back-tracking, and dynamic programming algorithms. Section 3 provides the upper bounds arising from having free branching and/or free data items and small domain size. This section also extends this algorithm to obtain the subexponential algorithms for  $k$ -SAT. Section 4 gives the reductions from the hard problems 7-SAT and subset sum to the easy problems matrix inversion and matrix parity. Section 5 provides a general technique for proving lower bounds within pFBT and then uses this technique to obtain the results for the parity problems. After Section 2, these sections can be read in any order.

## 2 The Formal Definitions

This section defines the computational problems in Section 2.1, intuition about the model in Section 2.2, the formal definition of the pFBT model in Section 2.3, and pFBT<sup>-</sup> in Section 2.4.

### 2.1 The Computational Problems

**A General Computational Problem:** A *computational problem* with priority model  $(\mathcal{D}, \Sigma)$  and a family of objection functions  $f^n : \mathcal{D}^n \times \Sigma^n \mapsto \mathbb{R}$  is defined as follows. The input  $I = \langle D_1, D_2, \dots, D_n \rangle \in \mathcal{D}^n$  is specified by a set of  $n$  *data items*  $D_i$  chosen from the given domain  $\mathcal{D}$ . A solution  $S = \langle \sigma_1, \sigma_2, \dots, \sigma_n \rangle \in \Sigma^n$  consists of a *decision*  $\sigma_i \in \Sigma$  made about each data item  $D_i$  in the instance. For an optimization problem, this solution must be one that maximizes  $f(I, S)$ . For a search problem,  $f(I, S) \in \{0, 1\}$ .

**$k$ -SAT:** The input consists of the AND of a set of clauses, each containing at most  $k$  literals. For each variable  $x_i$ , there is a data item contains the name of this variable and the list of all clauses in which

it participates. The decision  $\sigma_i \in \{0, 1\}$  that must be made about this data item is the value of  $x_i$ . To be valid, the decided solution must form a satisfying assignment.

**Unrestricted:** The number of potential data items in the domain is  $|\mathcal{D}| = n \cdot 2^{2(2n)^{k-1}}$ , because there are  $2(2n)^{k-1}$  clauses that the given variable  $x_i$  might be in.

$m$  **clauses:** Restricting the input instance to contain only  $m$  clauses, does not restrict the domain of data items  $\mathcal{D}$ .

$\mathcal{O}(1)$  **clause per variable:** If  $k$ -SAT is restricted so that each variable is in at most  $\mathcal{O}(1)$  clauses, then  $|\mathcal{D}| = n \cdot [2(2n)^{k-1}]^{\mathcal{O}(1)} = n^{\mathcal{O}(1)}$ .

**The Matrix Inversion Problem:** This problem was introduced in [ABBO<sup>+</sup>05] and modified by us slightly. The matrix  $M \in \{0, 1\}^{n \times n}$  is fixed and known to the algorithm. The reduction to 7-SAT requires that there are at most seven ones in each row and the lower bound requires that there are at most  $K \in \mathcal{O}(1)$  ones in each column. The our lower bound requires that  $M$  is a  $(r, 7, c)$  *boundary expander*, while [ABBO<sup>+</sup>05] only required it to be a strong expander. The input to this problem is vector  $b \in \{0, 1\}^n$ . The output is  $x \in \{0, 1\}^n$  such that  $Mx = b$ . Each data item contains the name  $x_j$  of a variable and the value of the at most  $K$  bits  $b_i$  from  $b$  involving this variable, i.e. those for which  $M_{\langle i, j \rangle} = 1$ . Again, with this data item, the value of  $x_i$  must be decided. Note  $|\mathcal{D}| = n2^K$ .

**The Matrix Parity Problem (MP):** The input consists of a matrix  $M \in \{0, 1\}^{n \times n}$ . The output  $x_i$ , for each  $i \in [n]$ , is the parity of the  $i^{\text{th}}$  row, namely  $x_i = \bigoplus_{j \in [n]} M_{\langle i, j \rangle}$ . This is easy enough, but the problem is that this data is distributed in an awkward way. Each data item contains the name  $x_i$  of a variable and the  $i^{\text{th}}$  column of the matrix, namely  $\langle M_{\langle 1, i \rangle}, \dots, M_{\langle n, i \rangle} \rangle$ . Note  $|\mathcal{D}| = n2^n$ .

**Subset Sum (SS):** The input consists of a set of integers, each stored in a data item. The goal is to accept a subset of the data items that adds up to a fixed known target  $T$ . Again  $|\mathcal{D}| = n2^n$ .

**No Carries:** Another version of SS does the GF-2 sum bit-wise so that there are no carries to the next bits.

## 2.2 Expressing Online, Greedy, Recursive Back-Tracking, and Dynamic Programming Algorithms

This section provides some intuition about what features a model needs to capture online, greedy, recursive back-tracking, and much of dynamic programming. The formal definition of the pFBT model appears in Section 2.3.

To make the discussion more concrete, consider the knapsack problem specified as follows. A data item  $D_i = \langle p_i, w_i \rangle$  specifies the price and weight of the  $i^{\text{th}}$  object. The decision  $\sigma_i \in \Sigma = \{0, 1\}$  to be made is whether or not to accept the object.  $f(I, S) = \sum_i p_i \sigma_i$  as long as  $\sum_i w_i \sigma_i \leq W$ .

An *online* algorithm for such a problem would receive the data items  $D_i$  one at a time and must make an irrevocable decision  $\sigma_i$  about each as it arrives. For example, an algorithm for the knapsack problem may choose to accept the incoming data item as long as it still fits in the knapsack. We will assume that the algorithm has unbounded computational power based on what it knows from the data items it has seen already. The algorithm's lack of ability to find an optimal solution arises because it does not know the data items that it has not yet seen. After it commits to a *partial solution*  $PS^{\text{in}} = \langle \sigma_1, \sigma_2, \dots, \sigma_k \rangle$  by making decisions about the *partial instance*  $PI^{\text{in}} = \langle D_1, D_2, \dots, D_k \rangle$  seen so far, an adversary can make the future items  $PI^{\text{future}} = \langle D_{k+1}, D_{k+2}, \dots, D_n \rangle$  be such that there is no possible way to extend the solution with decisions  $PS^{\text{future}} = \langle \sigma_{k+1}, \sigma_{k+2}, \dots, \sigma_n \rangle$  so that the final solution  $S = \langle PS^{\text{in}}, PS^{\text{future}} \rangle$  is an optimal

solution for the actual instance  $I = \langle PI^{in}, PI^{future} \rangle$ . This lower bound strategy is at the core of all the lower bounds in this paper.

A *greedy* algorithm is given the additional power to be able specify a priority ordering function (greedy criteria) and is ensured that it will see the data items in this order. For example, an algorithm for the knapsack problem may choose to sort the data items by the ratio  $\frac{p_i}{w_i}$ . In order to prove lower bounds as done with the online algorithm, [?] defined the *priority* model. It allows the algorithm to specify the priority order without allowing it to see the future data items by having it specifying an ordering  $\pi \in \mathcal{O}(\mathcal{D})$  of all possible data items. The algorithm then receives the next data item  $D_i$  in the actual input according this order. A *strongly adaptive* algorithm is allowed to reorder the data items every time it sees an new data item. An added benefit of receiving data item  $D$  is that the algorithm inadvertently learns that every data item  $D'$  before  $D$  in the priority ordering is not in the input instance. The set of data items learned in this way to be not in the instance will be denoted  $PI^{out}$ .

It is huge restriction on the algorithm to require it to make a single irrevocable decisions  $\sigma_i$  about each data item before knowing the future data items. In contrast, *backtracking* algorithms search for a solution where one decision is made about a data item and then if that search fails, backs up and searches with a different decision. This forms a tree. To model this [ABBO<sup>+</sup>05] defines *prioritized Branching Trees* (pBT). A computation on an instance  $I$  dynamically builds a tree of *states* as follows. Along the path from the root state to a given state  $u$  in the tree, the algorithm has seen some partial instance  $PI^u = \langle D_1^u, D_2^u, \dots, D_k^u \rangle$  and committed to some partial solution  $PS_u = \langle \sigma_1^u, \sigma_2^u, \dots, \sigma_k^u \rangle$  about it. At this state, knowing only this information, the algorithm specifies an ordering  $\pi^u \in \mathcal{O}(\mathcal{D})$  of all possible data items. The algorithm then receives the next unseen data item  $D_{k+1}^u$  in the actual input according this order. The algorithm then is able to choose to make one decision  $\sigma_{k+1}^u$  that is irrevocable for the duration of this path, to fork making a number of such decisions, or to terminate this path of the search all together. The computation returns the solution  $S$  that is best from all the nonterminating paths.

A curious restriction of this model is that it does not allow the following completely reasonable knapsack algorithm. Return the best solution after running three parallel greedy algorithms, one with the priority function being largest  $p_i$ , another smallest  $w_i$ , and another largest  $\frac{p_i}{w_i}$ . Being fully adaptive, each state along each branch is allowed to choose a different ordering  $\pi \in \mathcal{O}(\mathcal{D})$  for its priority function. However, the computation is only allowed to fork when it is making different decisions about a newly seen data item and this only occurs after seeing a data item. We generalize their model by allowing the algorithm to fork without seeing a data item. This is facilitated by, in addition to the *input reading* states described above, having *free branching* states that do nothing except branch to some number of children. This ability introduces a surprising lot of additional power to the algorithms. We call this new model *prioritized Free Branching Tree* (pFBT) algorithms. See Section 2.3 for the formal definition.

Though it is not initially obvious how, pFBT (pBT) are also quite good at expressing *dynamic programming* algorithms. Such algorithms derive a collection of subinstances to the computational problem from the original instance such that the solution to each subinstance is either trivially obtained or can be easily computed from the solutions to the subsubinstances. Generally, one thinks of these subinstances being solved from the *smallest* to *largest*, but one can equivalently continue in the recursive backtracking model where branches are pruned when the same *subinstance* has be previously solved. Each state  $u$  in the pFBT computation tree can be thought of as being the root of the computation on the subinstance  $PI^{future} = \langle D_{k+1}, D_{k+2}, \dots, D_n \rangle$  consisting of the yet unseen data items, whose goal is to find a solution  $PS^{future} = \langle \sigma_{k+1}, \sigma_{k+2}, \dots, \sigma_n \rangle$  so that  $S = \langle PS_u, PS^{future} \rangle$  is an optimal solution for the actual instance  $I = \langle PI_u^{in}, PI^{future} \rangle$ . Here,  $PI_u^{in} = \langle D_{\langle u,1 \rangle}, \dots, D_{\langle u,k \rangle} \rangle$  is the partial instance seen along the path to this state and  $PS^u = \langle \sigma_{\langle u,1 \rangle}, \dots, \sigma_{\langle u,k \rangle} \rangle$  is the partial solution committed to. It is up to the algorithm to decide which pairs of states it considers to be representing the “same” computation. For example, in the standard dynamic programming algorithm for the knapsack problem, the data items are always viewed in the same order. Hence, all states at level  $k$  have the same “subinstance”  $PI^{future} = \langle D_{k+1}, D_{k+2}, \dots, D_n \rangle$  yet

to be seen. However, because different partial solutions  $PS^u = \langle \sigma_{\langle u,1 \rangle}, \dots, \sigma_{\langle u,k \rangle} \rangle$  have been committed to along the path to different states  $u$ , their computation tasks are different. As far as the future computation is concerned, the only thing that differentiates between these different subproblems is the amount  $W_u = W - \sum_{i \leq k} w_i \sigma_{\langle u,i \rangle}$  of the knapsack remaining. Hence, for each such  $W'$ , the algorithm should kill off all but one of the computation paths. This reduces the width from  $2^k$  to being the range of  $W'$ . For a given value  $W'$ , which of the paths should be kept is clearly the one that made the best initial partial solution  $PS_u = \langle \sigma_{\langle u,k+1 \rangle}, \dots, \sigma_{\langle u,n \rangle} \rangle$  according to maximizing  $\sum_{i \leq k} p_i \sigma_{\langle u,i \rangle}$ . An additional challenge in implementing this dynamic programming algorithm in the pBT model is the following. In the model, the algorithm does not know which path should live until it knows partial instance  $PI_u^{in}$ , however, after this partial instance is received, the pBT model does not allow cross talk between these different paths. This is where the unbounded computational power of the pBT model comes in handy. Independently, each path, knowing  $PI_u^{in}$ , knows what every other paths in the computation is doing. Hence, it knows whether or not it should be terminated. If required, it quietly commits suicide. In this way, pBT is able to model many dynamic programming algorithms.

To be fair, let us tell you that the dynamic programming algorithm for the shortest paths problem cannot be expressed in the model. Each state in the computation tree at depth  $k$  will have traversed a subpath of length  $k$  in the input graph from the source node  $s$ , and the future goal is find to a shortest path from here to the destination node  $t$ . States in the computation tree whose subpath end in the same input graph node  $v$  are considered to be computing the same subproblem. However, because the different states  $u$  have read different paths  $PI_u^{in} = \langle D_{\langle u,1 \rangle}, \dots, D_{\langle u,k \rangle} \rangle$  of the input graph, they cant know what the other computation states are doing. Without cross talk, they don't know whether they are to terminate or not. [?] formally proves that pBT requires exponential width to solve shortest paths and defines another model called *prioritized Branching Programs* (pBP) which captures this notion of memoization by merging computation states that are computing the “same” subproblem. The computation then forms a DAG instead of a tree.

### 2.3 The Formal Definition of pFBT

Formally, a pFBT algorithm  $\mathcal{A}$  is defined as follows. On instance  $I$ , a computation tree  $\mathcal{T}_{\mathcal{A}}(I)$  is constructed inductively as follows. Each state  $u$  of the tree is uniquely specified by the information learned so far about the instance, the decisions made about the solution, and the branches followed along the path  $p$  from the root state to it, namely  $u_{identifier} = \langle PI_p^{in}, PI_p^{out}, PS_p, f_p \rangle$ . Here  $PI_p^{in} = \langle D_{\langle p,1 \rangle}, \dots, D_{\langle p,k \rangle} \rangle$  and  $PS_p = \langle \sigma_{\langle p,1 \rangle}, \dots, \sigma_{\langle p,k \rangle} \rangle$ , where  $D_{\langle p,i \rangle} \in \mathcal{D}$  is the data item seen and  $\sigma_{\langle p,i \rangle} \in \Sigma$  is the decision made about it in the  $i^{th}$  input reading state in the path to  $u$ .  $PI_p^{out} \subseteq \mathcal{D}$  denotes the set of data items inadvertently learned by the algorithm to be not in the input instance.<sup>1</sup> Finally,  $f_p = \langle f_{\langle p,1 \rangle}, \dots, f_{\langle p,k' \rangle} \rangle$ , where  $f_i \in \mathbb{N}$  indicates which branch is followed in the  $i^{th}$  free branching state along the path. An input reading state  $u$  first specifies the priority function with the ordering  $\pi_{\mathcal{A}}(u_{identifier}) \in \mathcal{O}(\mathcal{D})$  of the possible data items. Suppose  $D_{\langle p,k+1 \rangle}$  is the first data item from  $I \setminus PI_p^{in}$  according to this total order. The input reading state  $u$  must then specify the decisions  $c_{\mathcal{A}}(u_{identifier}, D_{\langle p,k+1 \rangle}) \subseteq \Sigma$  to be made about  $D_{\langle p,k+1 \rangle}$ .<sup>2</sup> For each  $\sigma_{\langle p,k+1 \rangle} \in c_{\mathcal{A}}(u_{identifier}, D_{\langle p,k+1 \rangle})$ , the state  $u$  has a child  $u'$  with  $D_{\langle p,k+1 \rangle}$  added to  $PI_p^{in}$ , each data items  $D'$  appearing before  $D_{\langle p,k+1 \rangle}$  in the ordering  $\pi_{\mathcal{A}}(u_{identifier})$  added to  $PI_p^{out}$ , and  $\sigma_{\langle p,k+1 \rangle}$  added to  $PS_p$ . A free branching state  $u$  need only specify the number  $F_{\mathcal{A}}^k(u_{identifier})$  of branches it will have. For each  $f_{\langle p,k'+1 \rangle} \leq F_{\mathcal{A}}^k(u_{identifier})$ , the state  $u$  has a child  $u'$  with  $f_{\langle p,k'+1 \rangle}$  added to  $f_p$ . The width  $W_{\mathcal{A}}(n)$  of the algorithm  $\mathcal{A}$  is the maximum over instances  $I$  with  $n$  data items, of the maximum over levels  $k$ , of the

<sup>1</sup>Given one knows the priority orderings used along the path,  $PI_p^{out}$  can be deduced from the other information and hence is not actually needed to uniquely specify the state  $u$ . We, however, include it for clarity.

<sup>2</sup>If one wanted to measure the time until a solution is found, then one would want to specify the order in which these decisions  $\sigma_{\langle p,k+1 \rangle} \in c_{\mathcal{A}}(u_{identifier}, D_{\langle p,k+1 \rangle})$  were tried.

number of states  $u$  in the computation tree at level  $k$ .

## 2.4 The Formal Definition of $\text{pFBT}^-$

We now formally define the restricted model  $\text{pFBT}^-$ . Recall that a pBT algorithm with free data items is able to solve any computational problem with width  $W = (n \log d)^2$ . Though this makes the proving of lower bounds in the model even more impressive, this aspect of the model is clearly not practical. The  $\text{pFBT}^-$  model restricts the algorithm in a way that excludes the possibility of it implementing this impractical algorithm while still allowing any practical algorithm that is in pFBT to still be in  $\text{pFBT}^-$ .

Recall that *free data items* are defined to be additional data items that have no bearing on the solution of the problem and about which the algorithm need not make any decisions. *Dummy data items* are data items that are introduced to a problem when doing a reduction to another problem. A data item is said to be *known* at some point in a computation path if the algorithm knows that it is in the actual input instance even though it has not yet received it or made an irrevocable decision about it. Such items can arise in a number of ways: 1) It might be a free data item defined in the computation problem, 2) It might be a dummy data item and the algorithm “knows” that the input instance is coming from the reduction. 3) The adversary, worried that the data item may be used as free data item, may have inform the algorithm that this data item is in the input instance. In contrast, a data item is said to be *unknown* if at this point in a computation path, the algorithm still does not know if it is in the input instance.

Classify input reading states  $u$  of the computation tree into three types. It is said to be an *unknown* reading state if it puts all of the possible unknown data items before the known ones. It is said to be a *known* reading state if it puts all of the known ones before the possible unknown ones. Finally, it is said to be a *mixed* reading state if it intertwines the two types of data items together in the orderings. The model  $\text{pFBT}^-$  is defined to be the same as pFBT except that mixed reading states are not allowed.

We argue that any practical algorithm that is in pFBT is still in  $\text{pFBT}^-$  by arguing that there is no (direct) reason that an algorithm should want to receive a *known* data item. Already knowing that it is in the instance, it gains no new information, but it is still required to make an irrevocable decision about it, which it may or may not be prepared to do. After the adversary makes a data item *known*, the algorithm should readjust its strategy so that it either uses *known* reading states to get these data items out of the way, or uses *unknown* reading states to avoid receiving them until the end, at which time that algorithm knows the entire input instance and hence it is easy to make an irrevocable decisions about them.

The flaw in this argument is that *mixed* reading states are useful. When the algorithm receives this known data item  $D_{\text{known}}$ , it learns that every data item  $D_{\text{unknown}}$  before  $D_{\text{known}}$  in the priority ordering is not in the input instance. The surprising algorithm from Theorem 1 uses mixed reading states in this way to learn every data item that is not in the instance without receiving a single real unknown data item. No real poly-time algorithm, however, is able to do this. First, it requires the algorithm to keep track of an exponential amount of information. Second, it requires the algorithm to have unbounded computational power with which to instantly compute the solution for the now known instance.

Through a reduction, we are able to prove a  $2^{\Omega(n)}$  lower bound on the width of a  $\text{pFBT}^-$  algorithm to solve Subset Sum. It is curious what this means. In the unrestricted version of Subset Sum, no data item is known, hence  $\text{pFBT}^-$  seems to impose no restriction to an algorithm. However,  $\text{pFBT}^-$  does allow the adversary the extra power to make a data item *known*, if it is worried that the algorithm will use it as a *free* data item. It should be noted that even in the restricted version of Subset Sum arising from the reduction, the *known* data items are not necessarily *free*. The problem with the data item  $D_{\langle j, - \rangle}$  is that it does not become *known* until SS has already received  $D_{\langle j, + \rangle}$ . The problem with SS reading the known data items  $E_{\langle i, k \rangle}$  early is that SS does not know whether to accept or reject them. Both of these issues seem to make the dummy data items useless as free data items.

### 3 Upper Bounds

Section 3.1 provides the general surprising algorithm and then Section 3.2 extends this to the even more surprising algorithm for  $k$ -SAT.

#### 3.1 Upper Bounds with Free Branching and/or Free Data Items and Small Domain Size

**Theorem 1** (Algorithm with Free Branching and/or Free Data Items). *Consider any computation problem with  $|\mathcal{D}| = d$  potential data items. There is a pFBT algorithm for this problem with width  $W_{\mathcal{A}}(n) = 2^{\mathcal{O}(\sqrt{n \log d})}$ . When  $m$  free data items are added to the problem, the required width decreases to  $2^{\mathcal{O}(\frac{n \log d}{m + \sqrt{n \log d}})}$ . When  $m = \mathcal{O}(n \log d)$ , it decreases all the way to  $\mathcal{O}(n \log d)^3$  even without free branching, i.e. in pBT.*

*Proof.* We first consider the case without free data items. We nonconstructively construct the deterministic algorithm by proving that the probability of our randomized algorithm finding a correct solution for a fixed input instance is less than one over the number  $d^n$  of instances. Hence, by the union bound we know that the probability that the correct solution is not found for some input instance is less than one. Hence, there must exist an algorithm that solves every input instance correctly.

In its first stage, the algorithm learns what the input instance is, not by learning each of the data items in the instance but by learning the complete set of  $|\mathcal{D}| - n$  data items that are not in the input instance. It does this without having to commit to values for more than  $\ell = \mathcal{O}(\sqrt{n \log d})$  variables. The algorithm branches when making a decision about each of these variables. Hence, the width is  $\approx |\Sigma|^\ell = 2^{\mathcal{O}(\sqrt{n \log d})}$ . In its second stage, the algorithm uses its unbounded computational power to instantly compute the solution for the now known instance. This solution will be consistent with one of its  $|\Sigma|^\ell$  branches.

We now focus on the how the algorithm can use free branches to learn lots of data items that are not in the instance. The algorithm randomly orders the potential data items and learns the first one that is in the input instance. The items earlier in the ordering are learned to be not in the instance and are added to  $PI^{out}$ . This is expected to consist of a  $\frac{1}{n}$  fraction of the remaining potential items. However, if instead, the algorithm free branches to repeat this experiment  $F = 2^\ell = 2^{\mathcal{O}(\sqrt{n \log d})}$  times, then one of these branches will result in  $\Theta(\frac{\ln(F)}{n}) = \Theta(\frac{\sqrt{\log d}}{\sqrt{n}})$  fraction of the remaining potential data items being added to  $PI^{out}$ . The algorithm might want to keep the one branch that performs the best, but this would require “cross talk” between the branches. Instead, it keeps any branch that performs sufficiently well. The threshold  $q_i$  is carefully set so that the expected number of branches alive stays the constant  $r$  from one iteration to the next.

**algorithm**  $pFBP(n, d)$

**$\langle pre-cond \rangle$ :**  $n$  is the number of data items in the input instance and  $d$  is the number of potential data items

**$\langle post-cond \rangle$ :** Forms a pFBP tree such that whp one of the leaves knows the solution

begin

$$\ell = \mathcal{O}(\sqrt{n \log d})$$

$$F = 2^\ell$$

$$r_0 = r = \mathcal{O}(\ell^2 n \log d) = \mathcal{O}(n \log d)^2$$

$$\text{Width} \leq (2r) \times F \times 2^\ell = 2^{\mathcal{O}(\sqrt{n \log d})}$$

Free branch to form  $r_0$  branches

Loop:  $i = 1, 2, \dots, \ell$

**(loop-invariant)**: The current states in the pFBT tree form a  $r_{i-1} \times |\Sigma|^{i-1}$  rectangle. The  $r_{i-1}$  rows were formed using free branches. For each such row, the states know a set  $PI^{in}$  of  $i-1$  of the data items known to be in the input instance. The column of  $|\Sigma|^{i-1}$  states within this row were formed from decision branches in order to make all  $|\Sigma|^{i-1}$  possible decisions on these  $i-1$  variables. Except for making different decision, these states are identical. Each of the  $r_{i-1}$  rows of states also knows a set  $PI^{out}$  of the data items known to be not in the instance.  $PI^{out}$  has grown large enough so that there are only  $d_{i-1} = d - |PI^{in}| - |PI^{out}|$  remaining potential data items.

Each state free branches with degree  $F$  forming a total of  $r_{i-1}F$  rows

For each of these  $r_{i-1}F$  rows of states

Randomly order the  $d_{i-1}$  remaining potential data items

$D_i$  = the first data item that is in the input

$q$  = # of data items before  $D_i$  in the order

$q_i = \Theta\left(\frac{d_i \ln(F)}{n}\right)$ , set so that  $\Pr(q \geq q_i) = \frac{1}{F}$

if( $q \geq q_i$ ) then

add  $D_i$  to  $PI^{in}$  and these  $q$  earlier data items to  $PI^{out}$

$d_i = d_{i-1} - q - 1$  remaining potential data items

Make a decision branch to decide each possibility in  $\Sigma$  about  $D_i$

else

Abandon this branch

end if

end for

$r_i$  = the total # of these  $r_{i-1}F$  rows that survive

if( $r_i \notin [1, 2r]$ ) abort

end loop

We know what the input instance is by knowing all the data items that are not in it.

We use our unbounded power to compute the solution.

This solution will be consistent with one of our  $2^\ell$  states.

end algorithm

The first thing to ensure is that  $d_\ell = d - |PI^{in}| - |PI^{out}|$  becomes  $n - \ell$  so the remaining possible data items must all be in the instance. At the beginning of the  $i^{th}$  iteration, there are  $d_{i-1}$  remaining potential data items. The algorithm randomly chooses one of these to be first in the ordering. It is in our fixed input instance with probability at most  $\frac{n}{d_{i-1}}$ . Conditional on this one not being in the instance, the next has probability at most  $\frac{n}{d_{i-1}-1}$  of not being in the instance, while the last of concern has probability at most  $\frac{n}{d_i}$ . Let  $q$  denote the number of these chosen before the first appears that is in the instance.  $\text{Exp}[q] \geq \frac{d_i}{n}$ .  $\Pr[q \geq q_i] \geq (1 - \frac{n}{d_i})^{q_i} \approx e^{-nq_i/d_i}$ . Because we have  $F$  chances and we want the expected number of these to survive to be one, we set  $q_i = \frac{d_i \ln(F)}{n}$  so that this probability is  $\frac{1}{F}$ . For the states that achieve  $q \geq q_i$ , the number of remaining potential data items is  $d_i = d_{i-1} - q - 1 \leq d_{i-1} - \frac{d_i \ln(F)}{n} = d_{i-1} - \frac{d_i \ell}{n} = \frac{d_{i-1}}{1 - \frac{\ell}{n}}$ . Hence, in the end  $d_\ell = \frac{d}{(1 - \frac{\ell}{n})^\ell} \approx \frac{d}{e^{\frac{\ell^2}{n}}} = \frac{d}{e^{\frac{\sqrt{n \ln d^2}}{n}}} < 1$ . Before this occurs, the input instance is known.

*Free data items* provide even more power than free branching does. Just as done above the set  $PI^{out}$  of data items known not to be in the instance grows every time one of these data items is received, however, because they have no bearing on the solution of the problem, the algorithm need not branch making all possible decisions  $\Sigma^m$  about them. The algorithm is identical to that above except that  $m' = \min(m, n)$

of the free data items are randomly ordered into the remaining real data items. The next data item received will either be one of the  $n$  real data items or one of these  $m'$  free data items. As before, let  $q$  denote the number of data items before this received data item in the priority ordering, i.e. the number added to  $PI^{out}$ .  $\Pr[q \geq q_i] \geq (1 - \frac{n+m'}{d_i})^{q_i} \geq e^{-2nq_i/d_i}$ . During this first stage, the algorithm wants to receive all  $m$  of the free data items and only receive  $\ell$  of the real data items. If a real data item is read after already having received  $\ell$  of them, then the branch dies.  $\Pr[q \geq q_i \text{ and the data item received is a free one}] \geq e^{-2nq_i/d_i} \times \frac{m'}{n+m'}$ . Still having  $F = 2^\ell$  chances and wanting the expected number of these to survive to be one, we set  $q_i = \frac{d_i(\ln(F) - \ln(\frac{m'}{n+m'}))}{2n} \geq \frac{d_i \ell}{3n}$  so that this probability is  $\frac{1}{F}$ . ( $\ln(F) = \ell$  will be set to  $\mathcal{O}(\frac{n \log d}{m}) \gg \ln(\frac{m'}{n+m'})$ .) Hence,  $d_i = d_{i-1} - q_i - 1 \leq \frac{d_{i-1}}{1 - \frac{\ell}{3n}}$ , giving in the end  $d_{m+\ell} = \frac{d}{(1 - \frac{\ell}{3n})^{m+\ell}} \approx \frac{d}{e^{\frac{\ell(m+\ell)}{3n}}} < 1$ , when  $\ell = \Omega(\frac{n \log d}{m + \sqrt{n \log d}})$ . This gives a total width of  $W_{\mathcal{A}}(n) \leq (2r) \times F \times |\Sigma|^\ell = 2^{\mathcal{O}(\frac{n \log d}{m + \sqrt{n \log d}})}$  as required.

We must also bound the probability that the algorithm aborts to be less than one over the number  $d^n$  of instances. It does so if  $r_\ell \notin [1, 2r]$  because it needs at least one branch to survive and we don't want the computation tree to grow too wide. Lemma 2 proves that this failure probability is at most  $2^{-\Theta(r/(m+\ell)^2)}$ , which is  $d^{-n}$  as long as  $r = \Theta((m+\ell)^2 n \log d)$ . This is  $r = \Theta(n \log d)^2$  when  $m = 0$  and grows to  $\Theta(n \log d)^3$  when  $m = \mathcal{O}(n \log d)$ .

The remaining point is that this can be achieved without free branching, i.e. in pBT, when  $m = \mathcal{O}(n \log d)$  and  $W_{\mathcal{A}}(n) \leq 2r = \mathcal{O}(n \log d)^3$ . In this case, a branch terminates every time a real data item is received, i.e.  $\ell = 0$ . Because only  $F = 2$  branches are needed each time a free data item is received, this can be achieved by having the algorithm branch on the  $\Sigma$  different possible decisions for this received free data item.  $\square$

We now bound the probability that the number of rows starting at  $r$  goes to zero or expands past  $2r$ .

**Lemma 2.** *Start with  $r$  rabbits. Each iteration  $i \in [\ell]$ , all the rabbits currently alive have  $F$  babies (and dies). Each baby lives independently with probability  $\frac{1}{F}$ . Hence, the expected number of rabbits remains at  $r$ . The game succeeds if the population does not die out and there are never more than  $2r$  rabbits. The probability of failure is at most  $2^{-\Theta(r/\ell^2)}$ .*

*Proof.*  $\text{Exp}[r_i] = r_{i-1}$  and the probability that it deviates by more than  $h\sqrt{r_{i-1}}$  is at most  $2^{-\Theta(h^2)}$ . If this does not occur in  $\ell$  rounds, then  $r_i$  deviates from  $r$  by at most  $\ell h\sqrt{r} < r$ . Requiring  $h = \frac{\sqrt{r}}{\ell}$ . The probability that this does not happen is at most  $\ell 2^{-\Theta(h^2)} = 2^{-\Theta(r/\ell^2)}$ .  $\square$

### 3.2 Upper Bounds for $k$ -SAT

The width  $2^{\mathcal{O}(\sqrt{n \log |\mathcal{D}|})}$  of the algorithm described in the last section is  $2^{\mathcal{O}(\sqrt{n \log n})}$  for  $k$ -SAT when restricted to having only  $\mathcal{O}(1)$  clauses per variable, because then the domain  $\mathcal{D}$  of data items is no bigger than  $n^{\mathcal{O}(1)}$ . However, for general  $k$ -SAT problem, this result does not directly improve the width because there are  $|\mathcal{D}| = n \cdot 2^{2(2n)^{k-1}}$  potential data items. Despite this, this algorithm can surprisingly be extended to  $2^{\mathcal{O}(n^{\frac{1}{3}} m^{\frac{1}{3}} \log n)}$  when the instance is restricted to having only  $m$  clauses.

**Theorem 3** ( $k$ -SAT  $m$ -clauses). *If there are only  $m$  clauses, then  $k$ -SAT can be done with  $2^{\mathcal{O}(n^{\frac{1}{3}} m^{\frac{1}{3}} \log n)}$  width.*

This is  $2^{\mathcal{O}(n^{\frac{2}{3}} \log n)}$  when there are only  $m = \Theta(n)$  clauses and  $2^{\mathcal{O}(n)}$  when there are only  $m = o(n^2)$  clauses. The problem is that there may be  $m = \Theta(n^k)$  clauses.

*Proof.* The first step is to ask for all variables that appear in at least  $q = n^{-\frac{1}{3}}m^{\frac{2}{3}}$  clauses. There are at most  $\frac{m}{q} = n^{\frac{1}{3}}m^{\frac{1}{3}}$  of these. Branching on each of these requires width  $2^{\mathcal{O}(n^{\frac{1}{3}}m^{\frac{1}{3}})}$  width. For each of the remaining variables, only  $q \log((2n)^k)$  bits are needed to specify the  $q$  clauses that they are in. Hence, the number of possible data items for each of these is at most  $d = (2n)^{qk}$ . Theorem 1 then gives a pFBT algorithm for this problem with width  $w = 2^{\mathcal{O}(\sqrt{n \log d})} = 2^{\mathcal{O}(\sqrt{nq \log n})} = 2^{\mathcal{O}(n^{\frac{1}{3}}m^{\frac{1}{3}} \log n)}$ .  $\square$

Disappointingly, when the number of clauses exceeds  $m = n^2$ , this algorithm fails to do any better than have a width of  $2^{\mathcal{O}(n)}$  and we have not been able to determine whether or not a better width can be obtained. This is particularly surprising given the fact that our intuition is that  $k$ -SAT is easier to solve when there are lots of clauses. Capturing this idea, the *sparsification lemma* from [?] proves that *back-tracking* can be made to work better than initially expected as long the formula has many more clauses than variables. We are easily able to implement this algorithm in pFBT. The problem is that the tree produced by the sparsification lemma, though it is proved to have very few leaves, is very unbalanced with one path being as long as  $\Omega(n)$ . For each of the nodes in this long path, the pFBT must branch making a decision about the value of the variable read. As far as we can see, this has the danger of blowing the width up to  $2^{\Omega(n)}$ .

## 4 Reductions from Hard to Easy Problems

This section does reductions to show that if pBT or pFBT can't do well on the simple matrix problems then they can't do well on the hard problems either.

**Theorem 4** (Lower Bounds for Hard Problems). *7-SAT requires width  $2^{\Omega(n)}$  in pBT and  $2^{\Omega(\sqrt{n})}$  in pFBT. Subset sum required width  $2^{\Omega(n/\log n)}$  in pFBT and  $2^{\Omega(n)}$  in pFBT<sup>-</sup>. Finally, subset sum without carries requires width  $2^{\Omega(n)}$  in pFBT.*

*Proof.* Lemma 5 gives a reduction from 7-SAT to the matrix inversion problem in both the pBT and the pFBT models. Lemma 6 gives a reduction in pFBT from subset sum (with and without carries) to the matrix parity problem with free data items. Lemma 7 gives a reduction from subset sum in the pFBT<sup>-</sup> model to the matrix parity problem with no free data items in the pFBT model. (Lemma ?? gives the same reduction in a slightly stronger model.) The results then follow from the lower bounds for the matrix problems given in Theorem 8.  $\square$

**Lemma 5.** *[Reduction to 7-SAT] If 7-SAT can be solved with width  $W$  in the pBT (pFBT) model, then the matrix inversion problem can be solved with the same width.*

*Proof.* Given a pBT (pFBT) algorithm for 7-SAT, we design a pBT (pFBT) algorithm for the matrix inversion problem as follows. Consider a matrix inversion data item. Let  $x_j$  be the variable specified and  $b_i$  be one of the at most  $K$  bits from  $b$  involving this variable, i.e  $M_{\langle i,j \rangle} = 1$ . Because the  $i^{\text{th}}$  row of  $M$  has at most seven ones, its contribution to  $Mx = b$  is that the parity of these corresponding seven  $x$  variables must be equal to  $b_i$ . This parity can be expressed as the AND of 64 clauses involving these seven variables. These clauses for each of the  $K$  bits  $b_i$  are included in the 7-SAT data item corresponding to this matrix inversion data item. Note that as required for the 7-SAT problem, this constructed data item contains a variable  $x_j$  and the list of clauses containing this variable. Each clause contains at most seven variables. As with a matrix inversion data item, the only new information that an algorithm who is aware of the reduction learns from a new 7-SAT data item is the value of the at most  $K$  bits  $b_i$  from  $b$  involving this variable. Finally, note that the output of both the matrix inversion problem and of 7-SAT is  $x \in \{0, 1\}^n$  such that  $Mx = b$ .  $\square$

**Lemma 6.** *[Reduction to Subset Sum with free data items] If Subset Sum can be solved with width  $W$  in pFBT, then the matrix parity problem with the introduction of  $m = \mathcal{O}(n \log n)$  data items can be solved*

with the same width. If the version without carries can be solved then only  $m = \mathcal{O}(n)$  data items need to be added to the matrix parity problem.

*Proof.* Given a pFBT algorithm for SS (subset sum), we design a pFBT algorithm for the MP (matrix parity problem) as follows. Like in the standard reduction from SS, we will have  $2n$  special bit indexes into the binary integers:  $c_j$  for each column  $j$  and  $r_i$  for each row  $i$  of the matrix  $M$ . These special bit indexes will be separated by at least  $2 \log_2 n$  indexes so that the sum of the integers for one special bit index does not carry into another. Let  $D_j$  denote the MP data item containing the variable name  $x_j$  and the  $j^{\text{th}}$  column of the matrix  $M$ . This data item will get mapped to two SS data items  $D_{\langle j,+ \rangle}$  and  $D_{\langle j,- \rangle}$ . In order to contain the same information as  $D_j$ , both  $D_{\langle j,+ \rangle}$  and  $D_{\langle j,- \rangle}$  will be one in their  $c_j^{\text{th}}$  bit and store the  $j^{\text{th}}$  column of the matrix, namely for  $i \in [n]$ ,  $i \neq j$ , the  $r_i^{\text{th}}$  bit is  $M_{\langle i,j \rangle}$ . The only exception is that while the  $r_j^{\text{th}}$  bit of  $D_{\langle j,- \rangle}$  stores the diagonal entry  $M_{\langle j,j \rangle}$  as it does with the other entries, the  $r_j^{\text{th}}$  bit of  $D_{\langle j,+ \rangle}$  instead stores the complement of this value. Both  $D_{\langle j,+ \rangle}$  and  $D_{\langle j,- \rangle}$  will be zero in every other bit. For each bit index  $k \in [1, \lceil \log n \rceil]$  and each row  $i \in [n]$ , SS will have a dummy data item  $E_{\langle i,k \rangle}$ . The integer in  $E_{\langle i,k \rangle}$  will be one only in the  $r_i + k^{\text{th}}$  bit. The target  $T$  will be one in the  $c_j^{\text{th}}$  bit and the  $r_i + \lceil \log n \rceil^{\text{th}}$  bit for each  $i, j \in [n]$ .

Having mapped an instance of the MP problem to an instance of the SS problem, we must now prove that the solutions to these two instances correspond to each other. Let  $S_{sum}$  be a solution to the SS instance described above. We first note that for each  $j$ ,  $S_{SS}$  must accept exactly one of  $D_{\langle j,+ \rangle}$  and  $D_{\langle j,- \rangle}$  because these are the only integers with a one in the  $c_j^{\text{th}}$  bit and the target  $T$  requires a one in this bit. Let  $S_{MP}$  be the solution to the MP instance in which  $x_j = 1$  iff  $D_{\langle j,+ \rangle}$  is accepted by  $S_{SS}$ . We now prove that this is a valid solution, i.e.  $\forall i \in [n]$ ,  $x_i = \bigoplus_{j \in [n]} M_{\langle i,j \rangle}$ . Consider some index  $i$ . Because  $S_{SS}$  is a valid solution, we know that its integers sum up to  $T$  and hence the  $r_i^{\text{th}}$  bit of the sum is zero. Because there is no carry to the  $r_i^{\text{th}}$  bit, we know that the parity of the  $r_i^{\text{th}}$  bit of the integers in  $S_{SS}$  is zero. The dummy integers  $E_{\langle i,k \rangle}$  are zero in this bit. For  $j \neq i$ , both  $D_{\langle j,+ \rangle}$  and  $D_{\langle j,- \rangle}$  have  $M_{\langle i,j \rangle}$  in this bit and as seen exactly one of them is in  $S_{SS}$  and hence these integers contribute  $M_{\langle i,j \rangle}$  to the parity. For  $j = i$ , it is slightly trickier.  $D_{\langle i,- \rangle}$ , if it is in  $S_{SS}$ , also contributes  $M_{\langle i,i \rangle}$  to the parity, while  $D_{\langle i,+ \rangle}$  contributes the compliment  $M_{\langle i,i \rangle} \oplus 1$ . In this first case,  $x_i = 0$  and in the second  $x_i = 1$ . Hence, we can simplify this by saying that together  $D_{\langle i,- \rangle}$  and  $D_{\langle i,+ \rangle}$  contribute  $M_{\langle i,i \rangle} \oplus x_i$ . Combining these gives that the parity of the  $r_i^{\text{th}}$  bit of the integers in  $S_{SS}$  is  $0 = \left[ \bigoplus_{j \neq i} M_{\langle i,j \rangle} \right] \oplus [M_{\langle i,i \rangle} \oplus x_i]$ . This gives as required that  $x_i = \bigoplus_{j \in [n]} M_{\langle i,j \rangle}$ . And hence,  $S_{MP}$  is a valid solution.

Now we must go in the opposite direction. Let  $S_{MP}$  be a solution for our MP problem. Put  $D_{\langle j,+ \rangle}$  in  $S_{SS}$  if  $x_j = 1$ , otherwise put  $D_{\langle j,- \rangle}$  in it. For each row  $i$ , let  $N = 2^{\lceil \log n \rceil}$  and  $N_i = N - (x_i + \sum_{j \in [n]} M_{\langle i,j \rangle})$  and let the binary expansion of  $N_i$  be  $[N_i]_2 = \langle N_{\langle i, \lceil \log n \rceil \rangle}, \dots, N_{\langle i, 0 \rangle} \rangle$ , so that  $N_i = \sum_{k \in [0, \lceil \log n \rceil]} N_{\langle i,k \rangle} 2^k$ . For  $k \neq 0$ , include the integer  $E_{\langle i,k \rangle}$  in  $S_{SS}$  iff  $N_{\langle i,k \rangle} = 1$ . We must now prove that the integers in  $S_{SS}$  add up to  $T$ . The same argument as above prove that the  $c_j^{\text{th}}$  and the  $r_i^{\text{th}}$  bits of the sum are as needed for  $T$ . What remains is to deal with the carries. The  $c_j^{\text{th}}$  bits will not carry. The  $r_i^{\text{th}}$  bits in the  $D_{\langle j,+ \rangle}$  or  $D_{\langle j,- \rangle}$  integers of  $S_{SS}$  add up to  $x_i + \sum_{j \in [n]} M_{\langle i,j \rangle}$ . For  $k \in [1, \lceil \log n \rceil]$ , the integer in  $E_{\langle i,k \rangle}$  is one only in the  $r_i + k^{\text{th}}$  bit and hence can be thought of contributing  $2^k$  to the  $r_i^{\text{th}}$  bit. Together, they contribute  $\sum_{k \in [1, \lceil \log n \rceil]} N_{\langle i,k \rangle} 2^k$ , which by construction is equal to  $N_i - N_{\langle i,0 \rangle}$ . Because  $S_{MP}$  is a solution,  $N_i = N - (x_i + \sum_{j \in [n]} M_{\langle i,j \rangle})$  is even, making  $N_{\langle i,0 \rangle} = 0$ . The total contribution to the  $r_i^{\text{th}}$  bit is then  $(x_i + \sum_{j \in [n]} M_{\langle i,j \rangle}) + N_i = N = 2^{\lceil \log n \rceil}$ , which carries to be zeros everywhere except in the  $r_i + \lceil \log n \rceil^{\text{th}}$  bit. This agrees with the target  $T$ .

What remains to prove is that the MP computation tree can state by state mirror the SS computation tree. It is the  $m = \mathcal{O}(n \log n)$  free data items that the MP problem has that allows this reduction to be straight forward. For each  $i \in [n]$ , the MP problem will have a free data item labeled  $D_{\langle i, 2^{nd} \rangle}$  and for

each  $i \in [n]$  and  $k \in [1, \lceil \log n \rceil]$ , it will have one labeled  $E_{\langle i, k \rangle}$ . In each state of the computation tree, the SS algorithm specifies an priority ordering of its data items  $D_{\langle i, + \rangle}$ ,  $D_{\langle i, - \rangle}$ , and  $E_{\langle i, k \rangle}$ . The simulating MP algorithm constructs as follows its ordering of its data items  $D_i$ ,  $D_{\langle i, 2^{nd} \rangle}$ , and  $E_{\langle i, k \rangle}$ . If neither  $D_{\langle i, + \rangle}$  nor  $D_{\langle i, - \rangle}$  have been seen yet, then the first occurrence of them in this ordering is replaced by  $D_i$ . The second occurrence of them can be ignored because it will never come into play. If at least one of  $D_{\langle i, + \rangle}$  or  $D_{\langle i, - \rangle}$  has been seen already, then the occurrence of the other one in this SS ordering is replaced by the free data item  $D_{\langle i, 2^{nd} \rangle}$ . Any dummy data item  $E_{\langle i, k \rangle}$  in the SS ordering are replaced by the corresponding free data item  $E_{\langle i, k \rangle}$  in the MP ordering. If according to this SS priority ordering, SS receives the first of  $D_{\langle i, \pm \rangle}$ , then MP according to its mirrored ordering will receive  $D_i$ . Note that MP learns the same information about its instance that SS does. If on the other hand, SS receives the second of  $D_{\langle i, \pm \rangle}$  or receives a dummy data item  $E_{\langle i, k \rangle}$ , then MP will receive the corresponding free data item. The MP algorithm (and we can assume the SS algorithm) knows that its instance is coming from this reduction and hence they knew before receiving it that this dummy/free data item is in the instance and hence neither gains any new information from this fact when it is received. The information of interest that they both inadvertently learn is that all data items  $D'$  in the ordering before this received data item are learned to not be in the instance and as such are added to  $PI^{out}$ . Because the MP algorithm always learns the same information that SS does, MP is able to continue simulating SS's algorithm.

If we are reducing from the carry free version of subset sum then the dummy data items  $E_{\langle i, k \rangle}$  are not needed.  $\square$

**Lemma 7.** [Reduction to Subset Sum in pFBT<sup>-</sup>] *If Subset Sum can be solved with width  $W$  in the pFBT<sup>-</sup> model, then the matrix parity problem without free data items can be solved in pFBT with the same width.*

*Proof.* When SS receives the first of  $D_{\langle i, \pm \rangle}$ , the adversary worries that SS may use the second of the pair as a *free data item*. Hence, at this point in the current computation, the adversary designates that this second data item is a *known data item* by revealing to SS that it is in the instance. The  $E_{\langle i, k \rangle}$  data items are considered to be known from the beginning. The pFBT<sup>-</sup> model then does not allow SS to have *mixed* reading states that intertwine the unknown and known data item together in its priority orderings. Consider an *unknown* reading state of SS, i.e. one that puts all of the possible unknown data items before the known ones. The simulating MP algorithm constructs its ordering of its data items  $D_i$  by replacing the first occurrence of  $D_{\langle i, \pm \rangle}$  with  $D_i$ . If SS receives one of  $D_{\langle i, \pm \rangle}$ , then MM receives  $D_i$ . If SS receives a known data items, then this means that the current computation path is done because the input instance is completely known. Now consider a *known* reading state of SS, i.e. one that puts all of the known ones before the possible unknown ones. Both SS and MP know that these known data items are in SS's instance and hence know that SS will receive the first one in this order. In fact, there was no point in this read state at all. The negative thing about such a read is that it unnecessarily forces SS to make an irrevocable decision about this known data item. If SS forks in order to to make more than one such decisions, then MP makes this same branches using a free branch. This free branch was one of the initial motivators of having free branches.  $\square$

## 5 Lower Bounds for the Matrix Problems

This section will prove the following lowerbounds for the matrix problems.

**Theorem 8** (Lower Bounds for the Matrix Problems). *The matrix inversion problem requires width  $2^{\Omega(n)}$  in the pBT model and width  $2^{\Omega(\sqrt{n})}$  in the pFBT model. The matrix parity problem requires width  $2^{\Omega(n)}$  in the pFBT model. If included in these problems are  $m$  free data items, then widths  $2^{\Omega\left(\frac{n}{m+\sqrt{n}}\right)}$  and  $2^{\Omega\left(\frac{n^2}{m+n}\right)}$  are still needed.*

## 5.1 The Lower Bound Technique

We will begin developing a general technique for proving lower bounds on the width of the computation tree of any algorithm solving a given problem in either the pBT or the pFBT models. We will then show how it is applied to get all of the results in Theorem 8.

For each input instance  $I$ , the computation tree  $\mathcal{T}_A(I)$  has height  $n = |I|$  measured in terms of the number of data items received. Let  $l$  be a parameter,  $\Theta(n)$  or  $\Theta(\sqrt{n})$  depending on the result. We will focus our attention on *partial paths*  $p$  from the root to a state at level  $l$  in  $\mathcal{T}_A(I)$ . Recall that such states are uniquely identified with  $\langle PI_p^{in}, PI_p^{out}, PS_p, f_p \rangle$ .  $PI_p = \langle PI_p^{in}, PI_p^{out} \rangle$ , consisting of the  $l$  data items known to be in the instance and those inadvertently learned to be not in the instance, constitutes the sum knowledge that algorithm  $\mathcal{A}$  knows about the instance  $I$  in this state. With only this limited knowledge, it is unlikely that the irrevocable the decisions  $PS_p$  that the algorithm has made about the data items in  $PI_p^{in}$  are correct. Formally, we prove that  $\Pr_I[S(I) \vdash PS_p \mid I \vdash PI_p]$  is small, where  $I \vdash PI_p$  is defined to mean that instance  $I$  is consistent with  $p$ , i.e. contains the data items in  $PI_p^{in}$  and not those in  $PI_p^{out}$  and  $S(I) \vdash PS_p$  is defined to mean that the decisions  $PS_S$  are consistent with the/a solution for  $I$ .

With free branching, in depth only  $l_{upper} = \mathcal{O}(\sqrt{n \log d})$ , the unreasonable upper bound in Theorem 1 manages to learn the entire input  $I$  not by having  $PI_p^{in}$  contain all of its data items but by having  $PI_p^{out}$  contain all of the data items not in it. This demonstrates how having  $PI_p^{out}$  extra ordinarily large, can cause  $I \vdash PI_p$  to give the algorithm sufficient uniformly about the instance  $I$  that it can deduce a correct partial solution  $PS_p$  causing  $\Pr_I[S(I) \vdash PS_p \mid I \vdash PI_p]$  to be far too big. Hence, we define a path  $p$  to be *bad* if this is the case.

A pFBT algorithm  $\mathcal{A}$  is allowed to fork both in a *free branching state* in order to try different priority orderings and in an *input reading states* in order to try different irrevocable decisions. This is what produces the tree  $\mathcal{T}_A(I)$  of parallel computation paths. It is sometimes easier to focus on the *algorithmic strategy*  $\mathcal{A}_f$  along one such computation path. Here  $f = \langle \sigma_1, \dots, \sigma_l, f_1, \dots, f_l \rangle$  denotes a *tuple of forking indexes*, where  $\sigma_i \in \Sigma$  indicates the decision made in the  $i^{th}$  input reading state and  $f_i \in \mathbb{N}$  indicates which branch is followed in the  $i^{th}$  free branching state. Note that if the width of the computation tree  $\mathcal{T}_A(I)$  is bounded by  $W_A(n)$ , then each  $f_i$  is at most  $W_A(n)$ . This allows us to define  $Forks_{pFBT} = [\Sigma \times [W_A(n)]]^l$  be the set of possible tuples of forking indexes. In contrast, a pBT algorithm is not allowed free branching, and hence  $Forks_{pBT} = \Sigma^l$ . The only difference between these two models that we will use is the sizes of these sets.

This next theorem outlines the steps sufficient to prove that the width of the algorithm must be high.

**Theorem 9** (The Lower Bound Technique).

1. Define a probability distribution  $\mathcal{P}$  on a finite family  $\mathcal{F}_n$  of valid and hard instances.

- For the matrix parity problem, the matrix  $M$  is chosen uniformly at random from  $\{0, 1\}^{n \times n}$  and for the matrix inversion problem, the vector  $b$  chosen uniformly from  $\{0, 1\}^n$ .

2. Prove that the information about the instance  $I$  gained from a good  $PI = \langle PI^{in}, PI^{out} \rangle$  is not sufficient to effectively make irrevocable decisions about the data items in  $PI^{in}$ . To do this, fix parameters  $l \in [n]$  and  $pr_{good} \in [1]$  and make slight adjustments to your definition of  $PI^{out}$  being good. Let  $PI^{in}$  and  $PI^{out}$  be arbitrary sets of data items such that  $|PI^{in}| = l$  and  $PI^{out}$  is sufficiently small so that it is considered good. Let  $PS$  be arbitrary decisions about the data items in  $PI^{in}$ . Prove that  $\Pr_{I \in \mathcal{P}, \mathcal{F}_n} [S(I) \vdash PS \mid I \vdash PI] \leq pr_{good}$ .

- If the algorithm simply guesses the decision  $PS$  for each of the  $l$  data items in  $PI^{in}$ , then it is correct with probability  $|\Sigma|^{-l} = 2^{-l}$ . For both the matrix and the matrix inversion problems, we

are able to bound  $pr_{good} \leq 2^{-\Omega(l)}$ . Recall as well, that  $l$  is  $\Theta(n)$  or  $\Theta(\sqrt{n})$  depending on the result. See Lemmas 14 and 16.

3. Prove that with probability at most  $\frac{1}{2}$ , every set  $PI_p^{out}$  in  $\mathcal{T}_A(I)$  is sufficiently small so that it is considered good. Consider a tuple of forking indexes  $f \in Forks$ . Consider some algorithmic strategy  $A_f$  along these forking indexes. Prove that  $\Pr_{I \in \mathcal{P}\mathcal{F}_n} [A_f \text{ produces a } PI^{out} \text{ that is bad}] \leq pr_{bad} \leq \frac{1}{2|Forks|}$ . Note that  $\frac{1}{2|Forks|}$  gives us the probability  $\frac{1}{2}$  after doing the union bound.

- For the matrix inversion problem, we are able to bound  $pr_{bad} \leq 2^{-\Omega(n)}$ . For the matrix parity problem, we are able to do much better bounding it by  $2^{-\Omega(n^2)}$ . See Lemmas 12 and 13.

These three steps give that any pFBT (pBT) algorithm  $A$  requires width  $W_A(n) \geq \frac{1}{2pr_{good}}$ .

From this technique and the itemized points, our lower bound results follow.

*Proof.* of Theorem 8: In all cases, the technique gives that  $W_A(n) \geq \frac{1}{2pr_{good}} \geq 2^{\Omega(l)}$ . What remains is to ensure that the probability  $pr_{bad}$  of a bad path is at most  $\frac{1}{2|Forks|}$ . For the matrix inversion problem,  $pr_{bad} \leq 2^{-\Omega(n)}$ . In the model pBT,  $|Forks_{pBT}| = 2^l$ , giving  $|Forks_{pBT}| \cdot pr_{bad} \leq \frac{1}{2}$  even when  $l \in \Theta(n)$ . In the pFBT model, however,  $|Forks_{pFBT}| = [2W_A(n)]^l = [2^{\Theta(l)}]^l = 2^{\Theta(l^2)}$ , giving  $|Forks_{pFBT}| \cdot pr_{bad} \leq \frac{1}{2}$  only when  $l \in \Theta(\sqrt{n})$ . For the matrix parity problem  $pr_{bad} \leq 2^{-\Omega(n^2)}$ , so we have no problem setting  $l \in \Theta(n)$  either way. If  $m$  free data items are included in this problem, then what changes is that instead of just considering  $l$  levels of the computation, we consider the computation up to the point at which  $l$  real data items have been received and any number of free data items have been received. This involves up to  $l+m$  levels in total. This increases  $|Forks_{pFBT}|$  from  $[2W_A(n)]^l = 2^{\Theta(l^2)}$  to  $[2W_A(n)]^{l+m} = 2^{\Theta(l(l+m))}$ . Hence, when  $pr_{bad} \leq 2^{-\Omega(n)}$ ,  $l \in \Theta\left(\frac{n}{m+\sqrt{n}}\right)$  is needed and when  $pr_{bad} \leq 2^{-\Omega(n^2)}$ ,  $l \in \Theta\left(\frac{n^2}{m+n}\right)$  is needed.  $\square$

We now prove that the technique works.

*Proof.* of Theorem 9: The following lines are annotated below.

$$\frac{1}{2} \leq \Pr_{I \in \mathcal{P}\mathcal{F}_n} [\exists p \in \mathcal{T}_A(I)_l, \text{ such that } p \text{ is good and } S(I) \vdash PS_p] \quad (1)$$

$$\leq \Pr_{I \in \mathcal{P}\mathcal{F}_n} [\exists p \in Paths_g, \text{ such that } I \vdash PI_p \text{ and } S(I) \vdash PS_p] \quad (2)$$

$$\leq \sum_{p \in Paths_g} \Pr_{I \in \mathcal{P}\mathcal{F}_n} [I \vdash PI_p \text{ and } S(I) \vdash PS_p] \quad (3)$$

$$= \sum_{p \in Paths_g} \Pr_{I \in \mathcal{P}\mathcal{F}_n} [S(I) \vdash PS_p \mid I \vdash PI_p] \cdot \Pr_{I \in \mathcal{P}\mathcal{F}_n} [I \vdash PI_p] \quad (4)$$

$$\leq pr_{good} \cdot \sum_{p \in Paths_g} \Pr_{I \in \mathcal{P}\mathcal{F}_n} [I \vdash PI_p] \quad (5)$$

$$\leq pr_{good} \cdot \sum_{p \in Paths} \Pr_{I \in \mathcal{P}\mathcal{F}_n} [p \in \mathcal{T}_A(I)] \quad (6)$$

$$= pr_{good} \cdot \text{Exp}_{I \in \mathcal{P}\mathcal{F}_n} [\text{width of } \mathcal{T}_A(I) \text{ at level } l] \leq pr_{good} \cdot W_A(n) \quad (7)$$

1. Step 3 in the technique proves that for a random instance  $I \in \mathcal{P}\mathcal{F}_n$ , the probability that  $\mathcal{T}_A(I)$  contains a bad partial path is at most  $\frac{1}{2}$ . Hence, any working algorithm  $A$  must solve the problem with probability at

least  $\frac{1}{2}$  using a full path whose first  $l$  levels is a good partial path. This requires that there exists a partial path  $p \in \mathcal{T}_A(I)$  such that the irrevocable decisions  $PS_p$  made along it are consistent with the/a solution  $S(I)$  of the instance. This is stated formally in the first line.

2. The probability in line 1 deals only with paths in  $\mathcal{T}_A(I)$ , which in turn depends on the randomly selected instance  $I$ . In contrast, the step 2 probability talks about fixed sets  $PI$  and  $PS$  that can depend on the algorithm  $\mathcal{A}$  as a whole but not on our current choice of  $I$ . To understand the algorithm  $\mathcal{A}$  independent of a particular instance  $I$ , define  $Paths = \{p \mid \exists I \text{ such that } p \text{ is a partial path of length } l \text{ in } \mathcal{T}_A(I)\}$  and  $Paths_g$  those that are good. Line 2 follows from  $p \in \mathcal{T}_A(I) \Rightarrow [p \in Paths \text{ and } I \vdash PI_p]$ .

3-5. The union bound, conditional probabilities, and plugging in the probability from step 2.

6. Translating from the algorithm  $\mathcal{A}$  as a whole back to just those paths in  $\mathcal{T}_A(I)$  requires understanding how the computation tree  $\mathcal{T}_A(I)$  changes for different instances  $I$ . Another argument for the necessity of doing this is noting that if these  $\mathcal{T}_A(I)$  were allowed to be completely different for different instances  $I$ , then for each  $I$ ,  $\mathcal{T}_A(I)$  could simply know the answer for  $I$ . The key fact that we need from the model is expressed in the following lemma.

**Lemma 10.** *If  $p \in Paths$  and  $I \vdash PI_p$ , then  $p \in \mathcal{T}_A(I)$ .*

It follows that when  $\mathcal{A}$  is in the state following  $p$ , then what it knows about the instance is that  $I \vdash PI_p$ , but from amongst these possible instances, it does not know which is the actual instance. The key for us is that this Lemma gives us line 6.

7. The width of  $\mathcal{T}_A(I)$  at level  $l$  is equal to the number of paths of length  $l$  in  $\mathcal{T}_A(I)$ . This width is bounded by  $W_A(n)$ .

Together this gives us  $W_A(n) \geq \frac{1}{2pr_{good}}$  as required.  $\square$

*Proof.* of Lemma 10: This can be proved by looking at the inductive structure of the computation tree  $\mathcal{T}_A(I)$ . Another option is to view the algorithm  $\mathcal{A}$  as a *super decision tree* with states  $\langle PI_p, PS_p, f_p \rangle$ . As with the computation tree  $\mathcal{T}_A(I)$ , the super decision tree makes free branches in order to try different priority orderings and branches when the algorithm tries different irrevocable decisions  $PS_p$ . In addition, the super decision tree also branches depending on which next data item  $D$  the algorithm receives. The set of paths in this super decision tree is then the union of the paths from the different computation trees  $\mathcal{T}_A(I)$ , i.e.  $Paths_g = \{p \mid \exists I \text{ such that } p \text{ is a path of length } l \text{ in } \mathcal{T}_A(I)\}$ .  $\square$

## 5.2 The Probability of a Bad Path

This section proves that with probability at most  $\frac{1}{2}$ , every computation path  $p$  in  $\mathcal{T}_A(I)$  is considered *good*. Recall that when the computation path  $p$  receives the data item  $D$ , in addition to learning that  $D$  is in the instance, it also learns that every data item  $D'$  before  $D$  in the priority ordering is not in the input instance. We consider the path  $p$  only while it receives its first  $l$  data items. Denoted by  $PI^{out}$  the set of data items learned in this way to be not in the instance. Informally, we defined  $PI^{out}$  to be *good* if it is sufficiently small. The proofs for the matrix inversion and for the matrix parity problems each are general enough to apply for any computation problem, but differ for two reasons. The first is that the distributions  $\mathcal{P}$  on the instances  $\mathcal{F}_n$  are different. The second is because the definition of *good* needs to be changed slightly for the matrix parity problem.

### 5.2.1 Bad Paths for the Matrix Inversion Problem

**Definition 1.** (*Q-good*) A computation path  $p$  and the set  $PI^{out}$  arising from it are considered to be *Q-good* iff  $|PI^{out}| \leq Q$ .

For the matrix inversion problem,  $Q$  is set to be  $\Theta(n)$ , which is reasonable given that the number of possible data items is  $|\mathcal{D}| = 2^K n = \Theta(n)$ .

We cannot prove anything about probabilities without stating something about the probability distribution  $\mathcal{P}$  on the instances  $\mathcal{F}_n$ .

**Definition 2.** ( $\mathcal{P}_q$ ) We say that  $\mathcal{P}$  is a  $q$  distribution on instances iff  $\Pr_{I \in \mathcal{P}\mathcal{F}_n} [D \in I \mid I \vdash PI] \geq q$ , where  $PI = \langle PI^{in}, PI^{out} \rangle$  denotes the current state of a computation, and  $D$  is a data item that is still possibly in the instance  $I$ .

**Lemma 11.** For the matrix inversion problem, the distribution  $\mathcal{P}$  is  $q = \frac{1}{2^K}$  distribution.

*Proof.* Lemma 15 will prove that  $I \vdash PI$  reveals at most  $r$  of bits of the vector  $b \in \{0, 1\}^n$ , for some  $r$  irrelevant to this proof. The conditional distribution  $\mathcal{P}$  chooses uniformly at random the remaining  $n-r$  bits. Each data item  $D$  specifies a variable  $x_j$  and up to  $K$  bits of the vector  $b$ . If this  $D$  disagrees with some of the  $r$  revealed bits of  $b$ , then  $D$  is not still possible. Otherwise, the probability that  $D$  is in the instance is  $\frac{1}{2^{K'}} \geq \frac{1}{2^K}$ , where  $K' \leq K$  is the number of the specified bits that have not been fixed.  $\square$

The next lemma then bounds the probability that a computational path is *bad*. Consider a tuple of forking indexes  $f \in \text{Forks}$ . Consider some algorithmic strategy  $\mathcal{A}_f$  along these forking indexes halting after  $l$  levels.

**Lemma 12.**  $pr_{bad} = \Pr_{I \in \mathcal{P}\mathcal{F}_n} [\mathcal{A}_f \text{ is a } Q\text{-bad path}] \leq e^{-qQ(1-\frac{l}{qQ})^2/2}$ .

For the matrix inversion problem, this gives  $pr_{bad} \leq e^{-\frac{Q}{2^{K+3}}} \leq 2^{-\Omega(n)}$  as claimed, as long as the computation depth considered is  $l \leq \frac{1}{2}qQ = \frac{Q}{2^{K+1}} \leq \mathcal{O}(n)$ .

*Proof.* The algorithmic strategy  $\mathcal{A}_f$ 's only goal is to make its computation path bad. This can be viewed as the following game. Following only one path down the computation tree, each step of  $\mathcal{A}_f$  defines an priority ordering on the data items and receives the data item  $D$  first in this ordering that is in the actual instance. The proof of Lemma 10 defined a *super decision tree*. Here we use basically the same tree except instead of having one node branching on which next data item  $D$  the algorithm receives, each such step is broken into substeps. In each of these substeps, the algorithm specifies one data item  $D$  from those that are still possible and is told whether  $D$  is or is not in the instance. If the answer is yes, then  $D$  is added to  $PI^{in}$  and if it is no, it is added to  $PI^{out}$ . Generally, we consider  $\mathcal{A}_f$ 's computation until it receives  $l$  data items. However, instead, let us consider it for  $Q$  of these substeps. For each  $t \in [Q]$ , let  $X_t$  be the random variable indicating whether the  $t^{\text{th}}$  such data item is added to  $PI^{in}$ . By definition,  $\Pr[X_t = 1] \geq q$  for distribution  $\mathcal{P}_q$ . (This is actually a martingale because the exact distribution on each  $X_t$  depends on the results of the previous ones.) Note that  $X = \sum_{t \in [Q]} X_t$  denotes the resulting size of  $PI^{in}$ . If  $X \geq l$ , then by this point in the computation  $|PI^{in}| \geq l$ , so the part of the computation we normally consider has already stopped, but  $|PI^{out}| = Q - |PI^{in}| \leq Q$  and hence this computation is considered to be good. It follows that  $\Pr[\text{path is bad}] \leq \Pr[X < l]$ . Define  $\mu = \text{Exp}[X] = qQ$  and  $\delta = 1 - \frac{l}{qQ}$  so that  $(1 - \delta)\mu = l$ . Chernoff bounds gives that  $\Pr[X < (1 - \delta)\mu] < e^{-\mu\delta^2/2} = e^{-qQ(1-\frac{l}{qQ})^2/2}$ .  $\square$

## 5.2.2 Bad Paths for the Matrix Parity Problem

As said, the definition of a path being *good* needs to be changed slightly for the matrix parity problem. For most computational problems, the input instance  $I = \langle D_1, D_2, \dots, D_n \rangle \in \mathcal{D}^n$  has each of its data items labeled by the variable  $x_j$  about which it must make an irrevocable decision. For each  $j \in [n]$ , let  $\mathcal{D}_j \subset \mathcal{D}$  denote the set of possible data items labeled by  $x_j$ . We assume in this section that the probability distribution  $\mathcal{P}$  on the instances  $\mathcal{F}_n$  selects independently for each  $j$ , one data item  $D_j \in \mathcal{D}_j$  uniformly. (This is true for the

matrix parity problem but not for the matrix inversion problem.) Similarly,  $PI^{out}$  can be partitioned into  $PI_j^{out} \subseteq \mathcal{D}_j$ . Define  $PI_j^? \subseteq \mathcal{D}_j$  to be the set of data items for which it remain unknown whether or not it is in the instance  $I$ . If  $j \notin PI^{in}$  (i.e.  $PI^{in}$  does not contain a data item from  $\mathcal{D}_j$ ), then  $PI_j^? = \mathcal{D}_j - PI_j^{out}$ . If  $j \in PI^{in}$ , then  $PI_j^? = \emptyset$ , because having received one data item from  $\mathcal{D}_j$ , the algorithm knows that no more from this domain are possible.

**Definition 3.** (*q-good*) A computation path  $p$  and the set  $PI^{out}$  arising from it are consider to be *q-good* iff  $\exists j \notin PI^{in}, |PI_j^?| \geq q|\mathcal{D}_j|$ .

For the matrix parity problem,  $|\mathcal{D}_j| = 2^n$  and  $q$  is set to be  $2^{-(1-\varepsilon)l} = 2^{-(1-\varepsilon)\varepsilon n}$ , which effectively means that at most  $(1-\varepsilon)\varepsilon n$  of the  $n$  bits have be revealed about the data item  $D_j \in \mathcal{D}_j$ .

**Lemma 13.**  $pr_{bad} = \Pr_{I \in \mathcal{P}\mathcal{F}_n} [\mathcal{A}_f \text{ is a } q\text{-bad path}] \leq (3q)^{n-l}$ .

This gives  $pr_{bad} \leq 2^{-\frac{1}{2}\varepsilon n^2}$  for the matrix parity problem as claimed.

*Proof.* The algorithmic strategy  $\mathcal{A}_f$  can be viewed as the algorithm playing  $n$  games in parallel. He loses the  $j^{th}$  game if he accidental adds  $j$  to  $PI^{in}$  before he can manage to shrink  $PI_j^?$  to be smaller than  $q|\mathcal{D}_j|$ . It is ok for him to lose  $l$  of these games, because, during the part of his computation we consider, we allow  $PI^{in}$  to grow to be of size  $l$ . However, if he loses  $l + 1$  of the games, then we claim that this computation path is *good*. At least one of these lost games had its  $j$  added to  $PI^{in}$  after we stopped considering his computation and at the point in time  $j \notin PI^{in}$  and  $|PI_j^?| \geq q|\mathcal{D}_j|$ .

In each substep of the *super decision tree* description of the computation  $\mathcal{A}_f$ , the algorithm specifies one data item  $D$  from one of the sets  $PI_j^?$  for which  $j \notin PI^{in}$  and is told whether  $D$  is or is not in the instance. If the answer is yes,  $j$  is added to  $PI^{in}$  and if it is no, then this data item is removed from its set  $PI_j^?$ . Though the algorithm plays the  $n$  games in parallel, they really are independent. Individually, for each  $j \in [n]$ , the algorithm chooses the one ordering in which it will ask about the data items in  $\mathcal{D}_j$ . The algorithm wins this game iff the data item  $D_j$  from  $\mathcal{D}_j$  that is in the instance lays in the last  $q$  fraction of this ordering. Because  $D_j$  is selected uniformly at random, the probability of the algorithm winning this game is  $q$ .

For  $j \in [n]$ , let  $X_j$  be the random variable indicating whether  $\mathcal{A}_f$  wins the  $j^{th}$  game and let  $X = \sum_{j \in [n]} X_j$  denote the number of games won. We proved that  $\Pr[X_j = 1] = q$ . We also proved that  $\Pr[\text{path is bad}] \leq \Pr[X \geq n - l]$ . Define  $\mu = \text{Exp}[X] = qn$  and  $1 + \delta = \frac{1}{q}(1 - \frac{l}{n})$  so that  $(1 + \delta)\mu = n - l$ . Chernoff bounds gives that  $\Pr[X \geq (1 + \delta)\mu] < \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^\mu \leq (3q)^{n-l}$ . □

### 5.3 Good Paths for the Matrix Parity Problem

We now consider the Matrix Parity Problem. The input consists of randomly chosen matrix  $M \in \{0, 1\}^{n \times n}$ . Each data item contains the name  $x_i$  of a variable and the  $i^{th}$  column of  $M$ . The required output  $x_i$  is the parity of the  $i^{th}$  row.

This section will prove that the information about the instance  $I$  gained from a good  $PI = \langle PI^{in}, PI^{out} \rangle$  is not sufficient to effectively make irrevocable decisions about the data items in  $PI^{in}$ . This is the only result in the lower bounds section that depends on anything specific about the computation problem and it does not depend on the specifics of the model of computation.

**Lemma 14.** Let  $PI^{in}$  be an arbitrary set of  $l$  data items,  $PS$  be arbitrary decisions about them, and  $PI^{out}$  any *q-good* set.  $pr_{good} = \Pr_{I \in \mathcal{P}\mathcal{F}_n} [S(I) \vdash PS \mid I \vdash PI] \leq \frac{1}{q}2^{-l}$ .

Here  $q = 2^{-(1-\varepsilon)l}$ , giving  $pr_{good} \leq 2^{-\varepsilon l}$ .

*Proof.* Assume the data items in  $PI^{in}$  have told the algorithm the first  $l$  columns of  $M$ , but required him to make irrevocable guesses  $PS$  of the parities  $x_1, \dots, x_l$  of the first  $l$  rows of  $M$ .  $PI^{out}$  being  $q$ -good means that  $\exists j \notin PI^{in}, |PI_j^?| \geq q|\mathcal{D}_j|$ , where  $D_j \in PI_j^? \subseteq \mathcal{D}_j = \{0, 1\}^n$  is a vector that remains possible for the  $j^{th}$  column of  $M$ . In order to be nice to the algorithm, we will give the algorithm all of  $M$  except for this column  $j$ . Then for each row  $i \in [1, l]$ , we now know the parity  $\bigoplus_{j' \neq j} M_{\langle i, j' \rangle}$  of these columns and know the parity  $x_i = \bigoplus_{j \in [n]} M_{\langle i, j \rangle}$  that the algorithm has promised for the entire row. Hence, we can compute the required value for  $M_{\langle i, j \rangle}$  in order for the algorithm to be correct. Let  $A$  be the set of vectors  $\alpha$  for the  $j^{th}$  column such that the algorithm is correct, namely  $A = \{\alpha \in \{0, 1\}^n \mid \forall i \in [1, l], \alpha_i = \text{the required } M_{\langle i, j \rangle}\}$ . Note that  $|A| = 2^{n-l}$  because there is such an  $\alpha$  for each way of setting the remain  $n-l$  bits of this column. The adversary now randomly selects  $\alpha \in PI_j^?$  that will be the  $j^{th}$  column in the instance  $M$ . If the selected  $\alpha$  is in  $A$ , then the algorithm wins. The probability of this, however, is  $|A|/|PI_j^?| \leq [2^{n-l}] / [q2^n] = \frac{1}{q}2^{-l}$ .  $\square$

## 5.4 Good Paths for the Matrix Inversion Problem

We now turn our attention to the Matrix Inversion Problem. A matrix  $M \in \{0, 1\}^{n \times n}$  is fixed and known to the algorithm. It is non-singular, has exactly seven ones in each row, at most  $K$  ones in each column, and is a  $(r, 7, c)$ -boundary expander, with  $K \in \Theta(1)$ ,  $c = 3$  and  $r \in \Theta(n)$ . A vector  $b \in \{0, 1\}^n$  is chosen randomly. The algorithm must output  $x \in \{0, 1\}^n$  such that  $Mx = b$ . To begin let us consider what the algorithm learns from  $I \vdash PI$ .

**Lemma 15.** *Let  $PI = \langle PI^{in}, PI^{out} \rangle$  be arbitrary sets of data items.  $I \vdash PI$  reveals at most  $K \cdot |PI^{in}| + |PI^{out}|$  bits of  $b$ .*

A computation path  $p$  is consider to be  $Q$ -good iff  $|PI^{out}| \leq Q$ . We set  $Q = r - Kl = \Theta(n)$ , where  $r = \Theta(n)$  is the parameter in strong expanding property that the matrix  $M$  has. Note that when  $PI^{out}$  is good, the algorithm learns at most  $K \cdot |PI^{in}| + |PI^{out}| \leq K \cdot l + (r - Kl) = r$  bits of  $b$ .

*Proof.* For every data item  $D_j$  in  $PI^{in}$ , the algorithm learns the value of at most  $K$  bits of  $b \in \{0, 1\}^n$  corresponding to corresponding  $j^{th}$  column of  $M$ . For every data item in  $PI^{out}$ , the algorithm learns a setting in  $\{0, 1\}^k$  that  $k \leq K$  of bits in  $b$  do not have. If  $k$  is large, then the immediate usefulness of this information is a little less clear. To simplify, what the algorithm does and does not know about  $b$ , the adversary chooses one of the  $k$  bits of  $b$  that the data item is about and that the adversary does not yet know the value of and reveals that this bit of  $b$  has the opposite value from that given in the data item. The algorithm, seeing this as the reason the data item is not in the instance, learns nothing more about the instance. In the end, what the algorithm learns from  $I \vdash PI$  is  $K \cdot |PI^{in}| + |PI^{out}|$  bits of  $b$ .  $\square$

The key point of this section will be to prove that learning at most  $r$  bits of  $b$  is not sufficient to effectively make irrevocable decisions about the data items in  $PI^{in}$ .

**Lemma 16.** *Let  $PI$  be from a good path so that  $I \vdash PI$  reveals at most  $r$  bits of  $b$ . Let  $PS$  be arbitrary decisions about the data items in  $PI^{in}$ .  $\Pr_{I \in \mathcal{P}\mathcal{F}_n} [S(I) \vdash PS \mid I \vdash PI] \leq 2^{-\Omega(l)}$ .*

This is when the definition a boundary expander is needed.

**Definition 4.** *(boundary expander)  $M$  is a  $(r, 7, c)$ -boundary expander if for every set  $R \subseteq [n]$  of at most  $r$  rows, the boundary  $\partial_M(R)$  has size at least  $c|R|$ . The boundary is defined to contain the column  $j$  if there is exactly one one in the rows  $R$  of this column. Section 5.5 proves that such a matrix exists.*

*Proof.* Let  $R \subseteq [n]$  denote the indexes  $i$  of the at most  $r$  bits of  $b$  that have been fixed and let  $\bar{b}_R \in \{0, 1\}^{|R|}$  be the values that these  $b_R$  bits have been fixed to. Let  $M_R$  denote the corresponding rows of the matrix  $M$ . These rows impose the requirement  $M_R \cdot x = \bar{b}_R$ . In addition,  $PS$  commits the algorithm to the values of exactly  $l$  variables. Let  $Fixed \subseteq [n]$  denote the indexes  $j$  of these variables  $x_j$  and let  $\bar{x}_{Fixed}$  denote the values committed to them.

Our probability distribution  $\mathcal{P}$  on instances  $I \in \mathcal{F}_n$  is uniform on the settings of  $b$ . Because there is a 1-1 mapping  $x = M^{-1}b$  between the settings of  $b$  and the settings of  $x$ , it is equivalent to instead consider the uniform distribution on setting of  $x$ . Our requirement  $I \vdash PI$  on our instance when considering the distribution on  $b$  was that  $b_R = \bar{b}_R$ . But when considering the distribution on  $x$ , this translates over to the requirement  $M_R \cdot x = \bar{b}_R$ . The requirement  $S(I) \vdash PS$  in both settings is that  $x_{Fixed} = \bar{x}_{Fixed}$ . An interesting effect of doing this translation is that the equations  $M_{([n]-R)} \cdot x = b_{([n]-R)}$  no longer matter. To conclude, this translation gives that

$$\begin{aligned} succ. prob. &= \Pr_{I \in \mathcal{P}\mathcal{F}_n} [S(I) \vdash PS \mid I \vdash PI] = \Pr_{x \in_u \{0,1\}^n} [x_{Fixed} = \bar{x}_{Fixed} \mid M_R \cdot x = \bar{b}_R] \\ &= \frac{|\{x \mid x_{Fixed} = \bar{x}_{Fixed} \text{ and } M_R \cdot x = \bar{b}_R\}|}{|\{x \mid M_R \cdot x = \bar{b}_R\}|} \end{aligned}$$

What we now need to compute is both the number of solutions  $x$  to the subsystem of equations  $M_R \cdot x = \bar{b}_R$  and the number of these consistent with  $x_{Fixed} = \bar{x}_{Fixed}$ . Towards this goal, we will do a Gaussian-like elimination to satisfy a subset  $Sat \subseteq R$  of the equations  $M_R \cdot x = \bar{b}_R$ . This is done by selecting a subset  $Set$  of the variables  $x_j$  that is disjoint from  $Fixed$  and setting their values as a linear combination of the variables in  $[n] - Set$  in a way that ensures the equations  $M_{Sat} \cdot x = \bar{b}_{Sat}$  are satisfied. Let  $R_0 = R - Sat$  index the equations  $M_{R_0} \cdot x = \bar{b}_{R_0}$  not yet satisfied. We will select and eliminate these equations  $i \in Sat$  one at a time. For the first one, let  $x_j$  be a boundary variable in the boundary  $\partial(R)$  that is not in  $Fixed$ . Recall that this is a column of  $M_R$  containing exactly one one. Let  $i \in R$  be the row in which  $x_j$  appears. The equations  $M_{(R-i)} \cdot x = \bar{b}_{(R-i)}$  do not use this variable  $x_j$  and hence are not effected by how it is set. The equation  $M_i \cdot x = \bar{b}_i$  can be solved for  $x_j$ . We set the value of  $x_j$  to be this linear combination of the variables in  $[n] - Set$  in order to satisfy this equation  $M_i \cdot x = \bar{b}_i$ . Being satisfied, the equation  $i$  is removed from  $R_0 = R - Sat$  and being set, the variable  $x_j$  is added to  $Set$ . We show as follows that this process can be repeated as long as  $c|R_0| > l$ . Because  $M$  is a  $(r, 7, c)$  boundary expander and because  $|R_0| \leq |R| \leq r$ , we have that  $|\partial(R_0)| \geq c|R_0| > l = |Fixed|$ . Hence, there is a variable  $x_j \in \partial(R_0) - Fixed$  to set next.

## GAUSSIAN-LIKE ELIMINATION OF ROWS

$R_0 \leftarrow R$

while ( $c|R_0| > l$ )

1. Pick  $x_j \in \partial(R_0) - Fixed$
2. Let  $i \in R$  be the row in which  $x_j$  appears.
3.  $x_j \leftarrow$  value that satisfies  $M_i \cdot x = \bar{b}_i$
4.  $R_0 \leftarrow R_0 - i$
5.  $Set \leftarrow Set + j$

This Gaussian-like elimination of rows terminates when  $|R_0| = \frac{l}{c}$ . This gives that the size of the boundary of the matrix  $M_{R_0}$  is  $|\partial(R_0)| \geq c|R_0| = l$ . This means that the matrix  $M_{R_0}$  has at least  $l$  columns with exactly one one. Another property of the matrix  $M$  is that each row has at most 7 ones. Hence these  $l$  boundary variables must be spread over at least  $\frac{l}{7}$  rows, making these  $\frac{l}{7}$  rows linearly independent. Hence, the rank  $r_0$  of the reduced system  $M_{R_0} \cdot x = \bar{b}_{R_0}$  is at least  $\frac{l}{7}$  and at most  $\frac{l}{c} = \frac{l}{3}$ . Because  $x$  consists of  $n$

boolean variables, this gives that  $|\{x \mid M_{R_0} \cdot x = \bar{b}_{R_0}\}| = 2^{n-r_0}$ . I would like to say that the same is true for  $|\{x \mid M_R \cdot x = \bar{b}_R\}|$ , however, the setting  $x_j$  for  $j \in Set$  are still part of system of equation. Hence, the number of these solutions may be  $2^{n-r_0-|Set|}$ .

When computing  $|\{x \mid x_{fixed} = \bar{x}_{fixed} \text{ and } M_R \cdot x = \bar{b}_R\}|$ , we also need to include the equations  $x_{fixed} = \bar{x}_{fixed}$ . None of the  $x_j \in Fixed$  have been eliminated. Hence, the requirements that  $x_{fixed} = \bar{x}_{fixed}$  adds  $l$  more linearly independent equations to the reduced system  $M_{R_0} \cdot x = \bar{b}_{R_0}$ . I don't know what it does to the system  $M_R \cdot x = \bar{b}_R$ .

\*\*\*\* alert \*\*\*\* alert \*\*\*\* alert \*\*\*\* alert \*\*\*\* alert \*\*\*\*

I conclude that I am at a loss to bound

$$succ. prob. = \frac{|\{x \mid x_{fixed} = \bar{x}_{fixed} \text{ and } M_R \cdot x = \bar{b}_R\}|}{|\{x \mid M_R \cdot x = \bar{b}_R\}|}$$

□

## 5.5 Boundary Expander Matrix

\*\*\*\* alert \*\*\*\* alert \*\*\*\* alert \*\*\*\* alert \*\*\*\* alert \*\*\*\*

I have not read this.

I am not sure whether it needs to be included, i.e. it may be standard and just included in the Thesis for completion.

**Definition 5.** A matrix  $A$  is  $(r, k, c)$  boundary expander if each row has at most  $k$  ones and for any set  $I$ , such that  $|I| \leq r$ , then the number of unique neighbors  $|\partial_A(I)|$  is at least  $c|I|$ :  $|\partial_A(I)| \geq c|I|$ .

Here we need to prove that a square matrix  $A \in \{0, 1\}^{n \times n}$  exists such that

1. Each row sums to exactly 7.
2. Each column sums up to  $7\Delta$  - a constant.
3.  $A$  is of full rand and hence is non-singular.
4.  $A$  is a  $(r, 7, c)$  boundary expander for  $c > 1$  and  $r \in \theta(n)$ .

Let  $\Delta$  be a constant, whose exact value will be determined later. First we define a distribution on  $\Delta n \times n$  binary matrices, such that each row has exactly 7 ones and each column sums up  $7\Delta$ . Then we show that a random matrix from that distribution is a good boundary expander. Next we argue that with high probability a matrix from that distribution has a full column rank. Then we are done because any  $n$  linearly independent rows of that matrix will inherit the expansion of the big matrix and is non-singular. Furthermore, each column will sum up to  $7\Delta$  which is a constant and does not depend on  $n$ , and each row sums to exactly 7.

The columns of the matrix are the variables. Each variable has  $7\Delta$  distinct equation in which it should appear. The rows are the equations with exactly 7 variables each. The total number of 1s in the matrix is  $7\Delta n$ . How do we match variables to equation and preserve the constraint? Model the matrix as a bipartite graph. Left side are the equations there are  $\Delta n$  of them each has 7 slots (for *distinct* variables). On the other side we have  $n$  variables each has  $7\Delta$  slots for the  $7\Delta$  equations in which a variable participates it. The distribution on matrices is all random perfect matchings of this bipartite graph.

Pick a row at random. From the available unmatched variables pick one at random, remove it (we sample from the columns without replacement). Decrease the number of remaining slots the selected variable has for future selection. Repeat until all 7 variables are chosen. We claim that any set  $I$  of at most  $r$  rows according to this distribution has boundary greater than  $c|I|$ ,  $c > 1$ .

**Lemma 17.** Let  $\Delta$  be a constant whose value will be set later. For any sufficiently large  $n$  there exists a matrix  $A \in \{0, 1\}^{\Delta n \times n}$ , which is also a  $(r, 7, 3)$  boundary expander.

*Proof.* We claim that the probability over the random choices without replacement of the variables in each row, that  $A$  is a  $(r, 7, c)$ ,  $c = 3$  boundary expander is greater than 0.

The set of rows  $I$ ,  $|I| = t \leq r$ , define  $t$  equations and  $7t$  variables of which  $B$  are boundary and the remaining  $D$  are duplicates. Initially, before the sampling begins each variable has all its  $7\Delta$  slots available. Once a variable is chosen it becomes a boundary variable and has only  $7\Delta - 1$  slots free. Consider the walk of length  $7t$  from Figure 2 beginning at state  $B=0/D=0$  and assume that  $r \in \Theta(n)$ . At time  $i$ ,  $i$  variables are picked and  $i$  of the  $7\Delta n$  slots are used. Suppose at time  $i$  the number of boundary variables is  $B$  and the duplicates is  $D$ , respectively. The total slots available to from the boundary variables is  $B(7\Delta - 1)$ . If a boundary variable is picked again then it becomes a duplicate. The number of boundary variables decreases by 1, the set of duplicates increases by one, and slots of duplicates increases by  $7\Delta - 2$ . Each duplicate has at most  $7\Delta - 2$  slots left and each time a duplicate is chosen again to participate in an equation, the number of available slots decreases. The remaining variables  $n - B - D$  have all their slots empty and available to be chosen, that is  $7\Delta(n - B - D)$  slots. We need to estimate the probability that after  $7t$  steps the chain

I DO NOT HAVE THIS FIGURE

Figure 2: Transition function of state  $B/D$ .

will end in a state where  $B < 3t$ . Assume that after  $i \leq 7t$  steps the current state is  $B/D$ . Only  $i$  slots are occupied and  $B + 2D \leq i \leq r$ ,  $r \in \Theta(n)$ . With probability

$$p_{++} = \frac{7\Delta(n - B - D)}{7\Delta n - i} > 1 - \frac{B + D}{n}$$

the next step will increase the boundary and we go to a state  $B + 1$ . With probability

$$p_{--} = \frac{(7\Delta - 1)B}{7\Delta n - i} < \frac{B}{n}$$

the next step will decrease the boundary and we go to a state  $B - 1$ . With probability

$$p_{=} \leq \frac{(7\Delta - 2)D}{7\Delta n - i} < \frac{D}{n}$$

the next step will not change the boundary and the duplicate sets.

From here on when we say *forward* for the direction of the walk, we mean increase the boundary, and as an overestimate when we say *backwards* we mean decrease or do not change the boundary<sup>3</sup>. Although those probabilities depend on the current state we will simplify the analysis by modeling the experiment as a Binomial random variable  $B(7t; \hat{p})$ . We define the success probability  $\hat{p}$  to be the largest probability that a walk of length  $7t$  can finish in a state with  $B \leq 3t$ . Note that any walk of length  $7t$  where we go to the right, that is we increase the boundary for  $5t$  steps or more will not be able go back to  $B < 3t$  state. Then  $\hat{p} < 1 - (1 - \frac{B+D}{n}) = \frac{B+D}{n} \leq \frac{5t}{n}$ . Now the event that after  $7t$  steps we end up in a state with less than  $3t$  boundary elements happens with probability less than the probability that in  $7t$  independent trials we have  $5t$  forward steps and  $2t$  backward steps (or  $2t$  successes and rest failures), which is:

$$\binom{7t}{2t} \hat{p}^{2t} (1 - \hat{p})^{5t} < \binom{7t}{2t} \hat{p}^{2t}.$$

<sup>3</sup>Note that this assumption leads to not tight analysis because some walks “backwards” will not finish in a state  $B < 3t$  but will be counted as such. We trade precise estimate for simplicity.

To find the probability  $p_t$  that there exists a set  $I$  of size at most  $t$  whose boundary is at most  $3t$  we use the union bound:

$$\begin{aligned} p_t &< \binom{\Delta n}{t} \binom{7t}{2t} \hat{p}^{2t} \leq \left(\frac{e\Delta n}{t}\right)^t \left(\frac{7e}{2}\right)^{2t} \left(\frac{5t}{n}\right)^{2t} \\ &\leq \Delta^t \cdot e^{3t} \cdot 20^{2t} \cdot \left(\frac{t}{n}\right)^t \leq \left[e^3 \cdot 20^2 \left(\frac{\Delta t}{n}\right)\right]^t \\ &< \left[(20e)^3 \left(\frac{\Delta r}{n}\right)\right]^t \end{aligned}$$

Then the probability that there exists a set of size  $t \leq r$  whose boundary is at most  $3t$  is

$$p = \sum_{t=1}^r p_t.$$

If we choose  $r = \frac{n}{3\Delta \cdot (20e)^2} \in \Theta(n)$ , then  $p_1 < \frac{1}{2}$  and  $p < 1$ , hence such an expander exists.  $\square$

Next we need to show that with high probability a random matrix chosen from the distribution has a full column rank.

**Lemma 18.** *Let  $A$  be a random  $\{0, 1\}^{\Delta n \times n}$  matrix with exactly 7 ones in each row and exactly  $7\Delta$  ones in each column. Then with high probability  $\text{rank}(A) = n$ .*

*Proof.* Suppose not. Suppose the rows of the matrix are contained in a proper subspace (a subspace of dimension  $k < n$ ). Let

$$x_{i_1} + \cdots + x_{i_k} = B \tag{8}$$

be the subspace equation, where  $B = 1$  or  $B = 0$ . To be a valid subspace the 0 vector must satisfy Eq. (8), hence it cannot be the case that  $B = 1$ , and any valid subspace equation for the matrix  $A$  (whose rows define  $\pmod{2}$  equations) is of the form:

$$\sum_{i \in S \subset [n]} x_i = 0. \tag{9}$$

Furthermore, each row equation of  $A$  must satisfy Eq. (9) therefore each row of  $A$  must contain exactly two, four, or six variables which also participate in the subspace equation. Any equation from the matrix which has odd number of variables from the subspace equation will contradict Eq. (9). Hence we need to bound the probability that a fixed subspace equation is satisfied by each matrix equation of  $A$ , given the constraints. We consider two cases when the number of variables in the subspace equation is small and when it is large.

1. Say the number of variables in the subspace equation is  $t \leq \frac{n}{12}$  (it must be the case that  $t \in \Theta(n)$  otherwise the probability all  $\Delta n$  matrix equations satisfy the subspace equation, given the distribution of the matrix  $A$  is zero). Intuitively, no such equation exists because when  $t$  is small the probability that the subspace equation will be satisfied by all rows is small.

Fix the variables in the subspace equation. Let those be  $F = \{x_{i_1}, \dots, x_{i_t}\}$ . Then the subspace equation is:

$$x_{i_1} + x_{i_2} + \cdots + x_{i_t} = 0. \tag{10}$$

So need to express the probability that a fixed equation is a valid subspace equation for  $A$ . Now we evaluate the probability that Eq. (10) is satisfied by all rows from the matrix  $A$ . Meaning each row of  $A$  must have at least two but even (four or six) number of variables from  $F$ , given the distribution that no variable in  $F$  can participate in more than  $7\Delta$  equations and that each equation has exactly 7 variables.

Now consider the matrix equations one by one. Each matrix equation must have two, four, or six variables from  $F$ . But also each variable in  $F$  can participate in exactly  $7\Delta$  equations (no more!).

**Definition 6.** *Call a matrix equation surviving if it has an even number of variables sampled from  $F$  without replacement.*

Each variable in each equation is chosen at random without replacement from the remaining available variables. Initially the  $t$  variables in  $F$  have all their  $7\Delta t$  slots available and the probability that the first matrix equation survives is at most the probability that at least one of the remaining six variables is sampled from  $F$ . Then by union bound the probability the first equation survives is

$$6 \times \frac{7\Delta t}{7\Delta n} = \frac{6t}{n}.$$

Say at time  $i$  when the first  $i$  equations have survived the variables from  $F$  have at most  $7\Delta t - 2i$  slots available and in total  $7\Delta n - 7i$  slots are used. The the probability that the  $i + 1$ -st equation survives by union bound is bounded by

$$\frac{6(7\Delta t - 2i)}{7\Delta n - 7i} \leq \frac{6t}{n}.$$

Now the probability that Eq. (10), is satisfied by all  $\Delta n$  equations of the matrix  $A$  is at most the probability that the first  $\Delta t$  equations have survived and the probability of that is at most  $\left(\frac{6t}{n}\right)^{\Delta t}$ .

Hence by a union bound the probability that there exists an equation of  $t$  variables which is satisfied by all equations of  $A$  is bounded above by:

$$\binom{n}{t} \cdot \left(\frac{6t}{n}\right)^{\Delta t} < \left(\frac{en}{t}\right)^t \cdot \left(\frac{6t}{n}\right)^{\Delta t} = (6e)^t \cdot \left(\frac{6t}{n}\right)^{(\Delta-1)t}$$

For  $t \leq \frac{n}{12}$  and  $\Delta > 5$  the probability above is  $2^{-\Omega(n)}$ .

2. Consider the case when the subspace equation has greater than  $\frac{n}{12}$  variables. We argue that a reasonable fraction of all matrix equations are likely to contradict the subspace equation because we show that it is likely that all variables from the matrix equation are also in the subspace equation. Hence the probability that many equations equations survive is exponentially small.

Initially the variables in the subspace equation have all their slots available, in total  $7\Delta t$  slots. Refer to the slots for the variables in the subspace equation as  $T$ ,  $|T| = t > \frac{n}{12}$ .

**Definition 7.** *We say that an equation is in violation if all its variables are from  $T$ , and thus will contradict the subspace equation. If an equation contradicts the subspace equation then we say that it cannot survive.*

We seek to bound from above the probability that at least the first  $\frac{\Delta n}{12}$  equations are not in violation by bounding from below the probability they contradict the subspace equation.

Consider the first matrix equation. The probability that the first variable is from  $T$  is  $\frac{7\Delta t}{7\Delta n} \geq \frac{1}{13}$  because  $t > \frac{n}{12}$ . The probability that the second variable is also from  $T$  is at least  $\frac{7\Delta t - 1}{7\Delta n - 1} > \frac{1}{13}$ . Then the probability that the first equation is in violation and all seven variables are from  $T$  is at least  $(\frac{1}{13})^7$ . And the probability that the first equation survives is at most  $1 - (\frac{1}{13})^7$ .

Given that the  $i$ -th matrix equation has survived, then the probability that the  $i + 1$ -st matrix equation is in violation is at least

$$\prod_{j=1}^7 \left( \frac{7\Delta t - 7i - j}{7\Delta n - i - j} \right) \geq \left( \frac{1}{13} \right)^7,$$

And for  $i \leq \frac{\Delta n}{13^2}$  the probability that the  $i$ -th equation survives is at most

$$\left( 1 - \left( \frac{1}{13} \right)^7 \right).$$

Fix a subspace equation with more than  $n/12$  variables. The probability that all matrix equation survive the subspace equation is at most the probability that the first  $i = \frac{\Delta n}{13^2}$  equations survive, which is at most

$$\left( 1 - \left( \frac{1}{13} \right)^7 \right)^{\frac{\Delta n}{13^2}} \leq e^{-\frac{\Delta n}{13^9}}$$

Then by union bound the probability that there exists a matrix equation with  $t > \frac{n}{12}$  variables is:

$$\binom{n}{t} \cdot e^{-\frac{\Delta n}{13^7}} < 2^n \cdot e^{-\frac{\Delta n}{13^9}}$$

As long as  $\Delta > 13^9$  the above probability is exponentially small.

□

By Lemmas 17 and 18 for reasonably large  $n$  there exists a non-singular binary matrix which is  $(r, 7, 3)$  boundary expander in which each column sums up to a constant  $K = 7\Delta$ . Choose  $\Delta = 2 \cdot 13^9$ , then  $K = 14 \cdot 13^9$ .

## References

- [AB04] Spyros Angelopoulos and Allan Borodin. On the power of priority algorithms for facility location and set cover. *Algorithmica*, 40(4):pp.271–291, 2004.
- [ABBO<sup>+</sup>05] Michael Alekhnovich, Allan Borodin, Joshua Buresh-Oppenheimer, Russell Impagliazzo, Avner Magen, and Toniann Pitassi. Toward a model for backtracking and dynamic programming. In *IEEE Conference on Computational Complexity*, pages 308–322, 2005.
- [AHI04] Michael Alekhnovich, Edward A. Hirsch, and Dmitry Itsykson. Exponential lower bounds for the running time of DPLL algorithms on satisfiable formulas. In *ICALP 2004, Turku, Finland.*, pages 84–96, July 2004.
- [AS92] Noga Alon and Joel Spencer. *The Probabilistic Method*. John Wiley, 1992.
- [BBL04] Allan Borodin, Joan Boyar, and Kim S. Larsen. Priority algorithms for graph optimization problems. In *WAOA*, pages 126–139, 2004.

- [BCM05] Allan Borodin, David Cashman, and Avner Magen. How well can primal-dual and local-ratio algorithms perform?. In *ICALP*, pages 943–955, 2005.
- [BEY98] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
- [BNR03] Allan Borodin, Morten N. Nielsen, and Charles Rackoff. (Incremental) Priority Algorithms. *Algorithmica*, 37(4):pp.295–326, 2003.
- [Bol01] Bela Bollobas. *Random Graphs*. Cambridge University Press, 2001.
- [CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition*. The MIT Press and McGraw-Hill Book Company, 2001.
- [DI07] Sashka Davis and Russell Impagliazzo. Models of greedy algorithms for graph problems. *Algorithmica*, November 2007.
- [FW98] Amos Fiat and Gerhard Woeginger. *Online Algorithms: The State of the Art*. Springer, September 1998.
- [Hoe63] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *J. of Amer. Statist. Assoc.*, 58(301):13–30, March 1963.
- [IKR85] Oscar H. Ibarra, Sam M. Kim, and Louis E. Rosier. Some characterizations of multihead finite automata. *Information and Control*, 67(1-3):114–125, 1985.
- [KT05] Jon Kleinberg and Eva Tardos. *Algorithm Design*. Addison Wesley, 2005.
- [MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
- [MU05] Michael Mitzenmacher and Eli Upfal. *Probability and Computing*. Cambridge University Press, 2005.
- [MV04] Jiri Matousek and Jan Vondrak. *The Probabilistic Method, Lecture Notes*. Manuscript, August 2004.
- [PPZ99] Ramamohan Paturi, Pavel Pudlák, and Francis Zane. Satisfiability coding lemma. *Chicago J. Theor. Comput. Sci.*, 1999.
- [RSZ02] Alexander Russell, Michael E. Saks, and David Zuckerman. Lower bounds for leader election and collective coin-flipping in the perfect information model. *SIAM J. Comput.*, 31(6):1645–1662, 2002.
- [Woe99] Gerhard J. Woeginger. When does a dynamic programming formulation guarantee the existence of fptas? In *SODA*, pages 820–829, 1999.
- [Woe01] Gerhard J. Woeginger. Exact algorithms for np-hard problems: A survey. In *Combinatorial Optimization*, pages 185–208, 2001.