

Some Quick Ideas

April 15, 2009

1 Introduction

Sashka's Technique applied to pFBP The key step in Sashka and Russel's technique to prove lower bounds for pFBT was to prove that for each path p of length ℓ , $\Pr_I[S(I) \vdash PS_p \mid I \vdash PI_p] \leq \frac{1}{f}$. We extend this to a technique for pFBP. The key step is to prove that for each node v at depth ℓ , $\Pr_I[S(I) \vdash PS_v \mid I \vdash PI_v] \leq \frac{1}{f}$. My intuition was that $\Pr_I[S(I) \vdash PS_v \mid I \vdash PI_v] \leq \max_{p \in v} \Pr_I[S(I) \vdash PS_p \mid I \vdash PI_p]$. However, in general this is not true. See Section 2

Tree Width vs Parallel Width: I have often explained the pBT model by saying that it can have w different solutions going at a time. But this is not really correct. Allan was wondering about the power of *tree width* verses that of *parallel width*. The first is the pFBT concept defined before, while the later imposes a restriction that all of the branches start at the beginning of the computation and survive until the end. Essentially, the computation consists of W independent parallel priority computations, resulting in the best of these. We prove that there is a problem that can be done with tree width $W = 2$, but that requires parallel width $W = 2^{\Omega(n)}$. It is analogous to many instances of a problem solved in serial being able to reusing its bounded memory, while the memory requirements blow up when the instances must be solved in parallel. See Section 3. The upper bound given below also shows the power of the tree model to reuse width.

Large Data Items: A complaint about the priority model has been that the results depend too much on the input representation. To address this, we define a model that allows the algorithm to address as much local information about an object as it wants before it is forced into an irrevocable decision about it. With in this model, we are able to prove two tight lower bounds, one on 3-SAT (see Section 4) and one on a version of the Steiner tree problem (or *st*-connectivity) (see Section ??).

Extra Info: Allan and I discussed giving the pFBT algorithm bits of information about the input before it began. For example, it may be helpful to give away the number of nodes in the instance. We noted, however, that giving away $k(n)$ bits can be implemented by having $2^{k(n)}$ times the width in the tree. Hence, a $2^{\Omega(n)}$ lower bound on the width automatically gives a lower bound with $k(n) = \mathcal{O}(n)$.

2 Sashka's Technique applied to pFBP

We will now attempt to apply Sashka and Russell's lower bound technique to the pFBP model. Assume that the q value for each node is binary and monotone increasing moving back up the DAG.

A path p of length ℓ in the DAG_A specifies $PI_p = \langle PI_p^{in}, PI_p^{out} \rangle$ and PS_p where PI_p^{in} is the set of ℓ data items that are known to be in the instance, PI_p^{out} is the set of data items that are known not to be in the instance, and PS_p specifies the decisions made about each data item in PI_p^{in} . Define $I \vdash PI_p$ to mean that instance I travels down path p , i.e. this path is in $DAG_A(I)$, because I is consistent with PI_p , i.e. contains

the data items in PI_p^{in} and not those in PI_p^{out} . Define $S(I) \vdash PS_p$ to mean that the decisions made about each data item in PI_p^{in} is consistent with the/a solution for I . There technique for pFBT was to prove that $\forall p, \Pr_I[S(I) \vdash PS_p \mid I \vdash PI_p] \leq \frac{1}{f}$.

Let v be a node at level ℓ in the DAG. It is labeled with $\langle PI_v, PS_v \rangle$, where $PI_v = \langle PI_p \mid p \text{ is a path to } v \rangle$ and $PS_v = \langle PS_p \mid p \text{ is a path to } v \rangle$. Define $I \vdash PI_v$ to mean that instance I travels through node v , i.e. node v is in $DAG_A(I)$, because I is consistent with at least one of the paths p to it, i.e. $OR_{p \in v} [I \vdash PI_p]$. Consider any such instance I , if there is a path in $DAG_A(I)$ leading from node v to an accepting node, i.e. to a sink node labeled one, then all the nodes in the DAG from this accepting node to v and back up through every path in $DAG_A(I)$ to the root will all be labeled one. The adversary might choose any of these paths p from the root back down to node v and on to the accepting node. Hence, PS_p must be a valid partial solution for I for each such paths. Note that $PS_{p'}$ does not need to be a valid partial solution for I if this path is in DAG_A but not in $DAG_A(I)$. Define $S(I) \vdash PS_v$ if this is the case, i.e. $AND_{p \in v} [I \vdash PI_p \Rightarrow S(I) \vdash PS_p]$.

Theorem 1.

1. Give a probability distribution on instances I .
2. Prove that whp every set PI_p^{out} in $DAG_A(I)$ is “small”. (Assume from here on in that this is the case.)
3. Prove that $\forall v, \Pr_I[S(I) \vdash PS_v \mid I \vdash PI_v] \leq \frac{1}{f}$.

The theorem then assures that any valid pFBP has width $w \geq f$.

Proof. of Theorem 1 The challenge is that for each instances I , there is a possibly different $DAG_A(I)$.

Define: $Tried = \{\langle I, v \rangle \mid I \vdash PI_v\}$.

Lemma 2. $|Tried| \leq w \cdot |\{I\}|$.

Proof. of Lemma 2 Every instance I has at most w nodes at level ℓ . □

Define: $Succeed = \{\langle I, v \rangle \mid I \vdash PI_v \text{ and } S(I) \vdash PS_v\}$.

Lemma 3. $|Succeed| \geq |\{I\}|$

Proof. of Lemma 3 Every instance I needs at least one path on which it succeeds and the node v at level ℓ along this path will be such that $\langle I, v \rangle \in Succeed$. □

Lemma 4. $|Succeed| \leq \frac{1}{f} \cdot |Tried|$

Proof. of Lemma 4 By step 3,

$$\begin{aligned}
|Succeed| &= \sum_v |\{\langle I, v \rangle \mid I \vdash PI_v \text{ and } S(I) \vdash PS_v\}| \\
&= \sum_v \Pr_I [I \vdash PI_v \text{ and } S(I) \vdash PS_v] \cdot |\{I\}| \\
&= \sum_v \Pr_I [S(I) \vdash PS_v \mid I \vdash PI_v] \cdot \Pr_I [I \vdash PI_v] \cdot |\{I\}| \\
&\leq \sum_v \frac{1}{f} \cdot \Pr_I [I \vdash PI_v] \cdot |\{I\}| \\
&\leq \frac{1}{f} \cdot \sum_v |\{\langle I, v \rangle \mid I \vdash PI_v\}| \\
&= \frac{1}{f} \cdot |Tried|
\end{aligned}$$

□

By Lemma 3, Lemma 4, and Lemma 2, we have that $|\{I\}| \leq |Succeed| \leq \frac{1}{f} \cdot |Tried| \leq \frac{1}{f} \cdot w \cdot |\{I\}|$. Hence, $w \geq f$ as needed. □

Shortest Path can be done in poly-time in pFBP but not in pFBT. This (I believe) is because $\Pr_I[S(I) \vdash PS_p \mid I \vdash PI_p] \leq \frac{1}{f}$ is not true for specially cooked sets $\langle PI_p, PS_p \rangle$, even though it is true for the sets that arise in practice. For example, suppose that PI_p^{in} said that there is a partial path of surprisingly low (or high) weight in the graph and PS_p said that that path is (is not) in the solution. Then they would probably be right.

If pFBP is more powerful than pFBT, then it is because it pFBP is able to find such situations, where pFBT is not.

The next conjecture lets us reuse for pFBP many pFBT lowerbounds proofs without modification.

Conjecture 1. $\Pr_I[S(I) \vdash PS_v \mid I \vdash PI_v] \leq (\ell + 1) \max_{p \in v} \Pr_I[S(I) \vdash PS_p \mid I \vdash PI_p]$.

I am quite sure it is true. $S(I) \vdash PS_v$ consists of a big AND, requiring many conditions of the form $S(I) \vdash PS_p$. The probability all of them are true is clearly smaller than the probability that only one is true. For intuition lets change this condition to simply that the algorithm needs to know what decision to make for l of the data items (but it does not matter which). Suppose the algorithm has been told $I \vdash PI_v$. This is a big OR. Telling it $I \vdash PI_p$ is more information. Surely telling it more information before needing to guess the decisions for the l data items can only help. Especially, since, the $p \in v$ is chosen to be the most useful.

I thought it was true for sure but here is why it might not be. Let $S_p = \{I \mid I \vdash PI_p\}$. Think of $\{S_p \mid p \in v\}$ as being a collection of sets of red and blue balls. The ball I is red if $S(I) \vdash PS_v$. For each i , $q_p = \frac{|S_p \cap red|}{|S_p|}$ is the density of red balls in the i^{th} set and $q = \frac{|(\cup_p S_p) \cap red|}{|(\cup_p S_p)|}$ is the density red balls over all. My intuition was that $q \leq \max_p q_p$. However, this is not true. Let there be m sets. Each S_p consist of $q'n$ red balls that are not contained in any other set and of $(1 - q')n$ blue balls that are common to every set. Then the density of each S_p is $q_p = q'$. The total number of red balls is $m \cdot q'n$ and the total number of blue balls is $1 \cdot (1 - q')n$ giving an over all density of $\frac{mq'n}{mq'n + (1 - q')n} \approx 1$. This is counter example to Conjecture 1. It, however, requires lots of intersection between the sets S_p on the blue balls. The next lemma, however, proves that this cant happen because the intersections are small while the number of blue balls huge.

To make things slightly easier, lets assume that the data items in each instance are indexed from one to n , i.e. $I \in \Pi_{i \in [n]} D_i$. Recall that PI_p^{in} fixes the values in I for a subset of these indexes and that PI_p^{out} throws out possibilities for the remaining indexes, i.e. $S_p = \{I \mid I \vdash PI_p\} = PI_p^{in} \times \Pi_{i \in [n] - PI_p^{in}} D_{\langle p, i \rangle}$, where $D_{\langle p, i \rangle} = D_i - PI_{\langle p, i \rangle}^{out}$ is the set of values still possible for this index. Because the path p is not *bad*, for each $i \in [n] - PI_p^{in}$, $|D_{\langle p, i \rangle}| \geq \frac{|D_i|}{e^{2\epsilon^2 n}}$. Let us denote this size as d .

Lemma 5. (1) If PI_p^{in} and $PI_{p'}^{in}$ do not fix the same set of indexes (or fix the same set of indexes but in different ways), then $|S_p \cap S_{p'}| \leq \frac{1}{d} \max(|S_p|, |S_{p'}|)$. (2) If PI_p^{in} and $PI_{p'}^{in}$ fix the same set of indexes in the same way, the difference between S_p and $S_{p'}$ depends only on the PI_p^{out} and **** we will ignore this case from here on in. ****

Proof. of Lemma 5 If there is some index i , where the i^{th} value in I is fixed in S_p or in $S_{p'}$, but not in the other. Hence, $|S_p \cap S_{p'}| \leq \frac{1}{d} \max(|S_p|, |S_{p'}|)$. □

Conjecture 2. I believe the following is the worst case.

We start by colouring a small number of the balls “red”. Let $S(I) \vdash PS_p$ be true for I if $I_i = 0$ for at least one $i \leq \ell + 1$. We want the sets S_p to not overlap on these red balls but overlap a lot everywhere else. For each $\hat{i} \leq \ell + 1$, and for each $\alpha \in \Pi_{i \in [\ell+1] - \hat{i}}[D_i - 0]$, let there be a path p such that PI_p^{in} fix the indexes $i \in [\ell+1] - \hat{i}$ of I to be the nonzero values α . This gives $S_p = \{I \mid I \vdash PI_p\} = \alpha \times \Pi_{i \in [n] - [\ell+1] + \hat{i}} D_i$. Note that the only way for $I \in S_p$ to be red is to have a zero at index \hat{i} . This gives that $\Pr_I[S(I) \vdash PS_p \mid I \vdash PI_p] = \frac{1}{d}$. On the other hand, $\cup_{p \in v} S_p$ is almost the full domain of possible instances. The only restriction is that the first $\ell + 1$ indexes have at most one zero. There are still $\ell + 1$ indexes such that if this index is zero then I is red. This gives that $\Pr_I[S(I) \vdash PS_v \mid I \vdash PI_v] = \Pr_I[\text{one zero}] / \Pr_I[\text{at most one zero}] = \frac{\ell+1}{\ell+d} = \frac{\ell+1}{d} = (\ell + 1) \max_{p \in v} \Pr_I[S(I) \vdash PS_p \mid I \vdash PI_p]$. This is in line with Conjecture 1.

Coverings: Let S_p be an ℓ -rectangle if $S_p = \alpha \times \Pi_{i \in [n] - i_p} D_i$, where $i_p \subset [n]$ specifies ℓ indexes and $\alpha \in \Pi_{i \in i_p} D_i$. Say that a set $\{S_p\}$ of ℓ -rectangles covers maximally if each S_p covers some I that the others do not. Let $U(r) = \max_{\{S_p \mid p \in [r]\}} |\{I \mid I \notin \cup_{p \in [r]} S_p\}|$ be the maximum number of uncovered I .

Conjecture 3. $U(r) \leq (1 - d^{-\ell})^r d^n$, giving $U(r) \leq qd^{n-\ell}$, when $r = (\ell \ln d + \ln(1/q))d^\ell$.

“Proof” of Conjecture 3: We want the S_p to be different but overlap as much as possible. This is done by having i_p and $i_{p'}$ differ by at most one index. This is what is done in the conjectured worse case.

Another option is to choose the S_p randomly. First choose the i_p using your favorite distribution and then choose α uniformly at random. $(1 - d^{-\ell})^r d^n$ is the expected number of uncovered I . ■

“Proof” of Conjecture 1: We will now use Conjecture 3 to prove a weaker version Conjecture 1. Let v be a node. Let $r = (\ell \ln d + \ln(1/q))d^\ell$. Let $\{S_p \mid p \in [r] \subset v\}$ be the first r rectangles that cover maximally. Let $q = \max_{p \in v} \Pr_I[S(I) \vdash PS_p \mid I \vdash PI_p]$. The number of red balls in any one S_p is at most $q|S_p| = qd^{n-\ell}$. Hence, the number in $\{S_p \mid p \in [r] \subset v\}$ is at most $rqd^{n-\ell}$. Lets assume that all I uncovered by $\{S_p \mid p \in [r] \subset v\}$ are red. There are at most $U(r)$ of these. Hence, the total number of red balls over all of D^n is $rqd^{n-\ell} + U(r) \leq (r + 1)qd^{n-\ell}$. The number of I covered by all $\{S_p \mid p \in v\}$ is at least $d^n - U(n) \approx d^n$. This gives that $\Pr_I[S(I) \vdash PS_v \mid I \vdash PI_v] \leq \frac{\text{the number of red balls}}{d^n - U(n)} \leq \frac{rqd^{n-\ell} + U(r)}{d^n - U(n)} \leq \frac{(r+1)qd^{n-\ell}}{d^n} = (\ell \ln d + \ln(1/q))q$. ■

3 Tree Width vs Parallel Width

Allan was wondering about the power of *tree width* verses that of *parallel width*. The first is the pFBT concept defined before, while the later imposes a restriction that all of the branches start at the beginning of the computation and survive until the end. Essentially, the computation consists of W independent parallel priority computations, resulting in the best of these. We prove that there is a problem that can be done with tree width $W = 2$, but that requires parallel width $W = 2^{\Omega(n)}$. It is analogous to many instances of a problem solved in serial being able to reusing its bounded memory, while the memory requirements blow up when the instances must be solved in parallel.

Theorem 6. *There is a problem that can be done with tree width $W = 2$, but that requires parallel width $W = 2^{\Omega(n)}$.*

Proof. of Theorem 6 Let P denote any computation problem that can be solved with tree width $W = 2$ but cannot be solved with width $W = 1$, even with lots of free data items. For example, the Bor et al paper proves that width $W = 2$ is needed to solve knapsack with some competitive ratio. We could also scale down the $2^{\Omega(\sqrt{n})}$ SAT result. Let the constant a be the smallest number of data items needed in the input to obtain this result. Let the constant $D = d^a$ be the total number of different input instances to this problem. Let $P_a^{\frac{n}{a}}$ denote the problem consisting of $\frac{n}{a}$ independent instances of P . The input will consist of n data items, a for

each of the $\frac{n}{a}$ instances. Each data item will clearly say which of the $\frac{n}{a}$ instances it is associated with. The task is to solve each of these $\frac{n}{a}$ instances to the required level of accuracy.

The problem can be solved with tree width $W = 2$. The algorithm simply solves the instances one at a time sequentially. After seeing the a data items for the current instance, one of the two branches will contain an valid solution for this instance. The other branch dies. The computation continues on the living branch. In the end, the final branch will contain an valid solution for each of the $\frac{n}{a}$ instances.

On the other hand, simple counting shows that any algorithm parallel width W that solves this problem requires $W \geq e^{\frac{n}{aD}} = 2^{\Omega(n)}$. Consider any branch of any such algorithm. Consider how this branch solves the i^{th} instance of P for $i \in [\frac{n}{a}]$. This branch cannot solve this instance of P for all D input instance. This would contradict the fact that there is no $W = 1$ with algorithm for P . (Note the lower bound for P needed to allow for the algorithm to also receive dummy data items. See Lemma 7.) To be nice, assume that it solves P on $D-1$ of the instances. Similarly, this same branch solves $D-1$ of each of the $\frac{n}{a}$ instances. In order for this branch to solve the instance of $P^{\frac{n}{a}}$, it must solve each of the $\frac{n}{a}$ instances of P . Hence, this branch solves at most $(D-1)^{\frac{n}{a}}$ instances of $P^{\frac{n}{a}}$. However, there are $D^{\frac{n}{a}}$ instances of $P^{\frac{n}{a}}$ to be solved. Hence, the number of branches must be at least $W \geq [D^{\frac{n}{a}}]/[(D-1)^{\frac{n}{a}}] \approx e^{\frac{n}{aD}}$.

It is interesting that this reduction is almost tight. Suppose that for each of the D possible instances to P , there is a $W = 1$ width algorithm that fails to solve this instance but solves the $D-1$ other instances. Choose an algorithm for $P^{\frac{n}{a}}$ by choosing independently for each of its W branches and for each its $\frac{n}{a}$ instances of P which of these D algorithms for P to use. Consider one of the $D^{\frac{n}{a}}$ instances of $P^{\frac{n}{a}}$, one branch, and one $i \in [\frac{n}{a}]$. The probability that this branch solves the i^{th} instances is $\frac{D-1}{D}$. The probability that it solves each of the $\frac{n}{a}$ instances is $[\frac{D-1}{D}]^{\frac{n}{a}} \approx e^{-\frac{n}{aD}}$. The probability that each of the W branches fails to solve this instance of $P^{\frac{n}{a}}$ is $[1 - e^{-\frac{n}{aD}}]^W \approx e^{-(e^{-\frac{n}{aD}})W}$. That this occurs for one of the $D^{\frac{n}{a}}$ instances is at most $e^{-(e^{-\frac{n}{aD}})W} \cdot D^{\frac{n}{a}}$. Setting $W > e^{\frac{n}{aD}} \cdot \frac{n \ln D}{a}$ gives that this probability is strictly less than one. Hence, an algorithm exists that solves each of these instances.

I have not been able yet, however, able to prove the lemma I first wanted. (Though it might now be of limited interest.) It is basically the same question solved above, except the P requires w parallel width and we want to know the parallel width requirement of P^2 , consisting of two independent copies of P . I conjecture that width $W = \Omega(w^2)$ is needed. Consider the following problem of covering a matrix with rectangle. The rows of the matrix are indexed by instances of P and the columns by instances of P' . Each entry of the matrix is an instance of $P \times P'$. A single branch solves the instances in a rectangle $S \times S'$ where S are the instances of P solved and S' the instances of P' . An algorithm for $P \times P'$ with W parallel width covers all the entries of the matrix with W such rectangles. Assuming that no w of these rectangles can cover all of the rows and no w' of them can cover all of the columns, prove that $\Omega(w w')$ rectangles are needed.

===

As seen in Section ??, free data items can provide extra power. In addition to the n data items one also receives m free data items chosen from some large domain. These have no bearing on the solution of the problem and the algorithm need not make any decision about them. We show here that when the width is $W = 1$, the power of free data items is limited.

Lemma 7. *When the width is only $W = 1$, the free data items seems to be limited to increasing IP^{out} at only three times the speed that it is increased with out them.*

This is not a proof.

The extra power comes as follows. The algorithm orders the data items that remain possible. This ordering includes the possible free items. The algorithm learns the first data item in this ordering that is in the actual input instance. If this data item is a free item, the algorithm still learns that all the possible data items that were before it in the ordering are not in the input instance, i.e. are added to PI^{out} . The scary

thing is that this extra information comes at no extra cost. But it is not so bad. If the possible actual data items and the free data items are distributed uniformly, then $\frac{m+n}{n}$ is the expected number of free data items seen before a real data item is seen. For each of these free data items, the expected number of possible real data items added to PI^{out} is $q_i = \frac{d_i \ln W}{m}$. This number is given in the upper bound Theorem ?? and will be needed in the lower bound. In total, the expected number of possible real data items added to PI^{out} is before a real data item is seen is then $\frac{m+n}{n} \times q_i \leq \frac{2d_i \ln W}{n}$. This is only a factor of two away from the number added to PI^{out} when a real data item is seen. Hence, all is well.

4 Large Data Items

Previous Model:

Contents of a Data Item: Suppose at first that the computational problem's input is a graph G and that decision are made about its edges e (or its nodes v). The most natural way to represent the input would be to have one data item for each edge containing the names of the two adjacent nodes and the weight of the edge. Sometimes, however, to have an efficient algorithm, the degrees of these nodes is needed and maybe even their adjacency lists. Even though decisions are being made about the edges, another option is to have a data item for each node containing its adjacency list. One concern about the priority model is that the complexity of a problem can depend a great deal on such representation details. What is clear, however, is that the more information included, the better off the algorithm is.

The Decisions Made: If each data item contains a node and its adjacency list and decisions need to be made about the edges, then the standard thing to do is to require the algorithm to make an irrevocable decision about each of the edges seen. It would be more generous to the algorithm to only require it to make a decision at this time about one specific edge.

The New Model:

The Broad Local Priority and BT Model: Our idea is to include in each data item as much local information about an object as the algorithm wants before it is forced into an irrevocable decision about it and then to require it to make a decision about only this one object at this time.

The parameter q will specify the amount of information accessible to the algorithm. For each object e , let G_e be the sub-graph containing all edges of G that are reachable from e by paths of length ℓ_e , where ℓ_e is chosen dynamically so that G_e contains approximately q edges. The data item for the edge e will specify all of G_e . When a priority or BT algorithm sorts the edges it can do so using all of this information. When it receives the next edge e , it receives all of this information. Then using this information, it must make an irrevocable decision about e . It can delay making decisions about the remaining edges in G_e until it receives the data items associated with these edges.

3-SAT: The same ideas can be used for any input type as long as there is a sense of the locality the objects. 3-SAT, for example, has a set of clauses as inputs. The standard BT model has a data item for each variable and to include in this data item a full description of all clauses that the variable is in. Our model, on input Φ , defines a graph on the variables with an edge between two variables if they share a clause. It stores Φ_x in the data item associated with variable x , where Φ_x contains the full description of every clause containing a variable that is reachable in this graph from x by a path of length ℓ_x , where ℓ_x is chosen dynamically so that Φ_x contains approximately q variables.

BT Width: Borodin et al defines the width w of a BT computation to be the width of the tree of decisions or equivalently the number of decision vectors that can be simultaneously kept.

Theorem 8. “Any” computation problem with data item size q can be solved optimally with decision tree width $w = 2^{\mathcal{O}(n/q)}$.

Theorem 9. 3-SAT with data item size q requires tree width $w = 2^{\Omega(n/q)}$.

Proof. of Theorem 8 We prove that any computation problem with data item size q can be solved optimally with decision tree width $w = 2^{\mathcal{O}(n/q)}$.

Each data items reveals q of the n objects in the input instance. The algorithm greedily ask for a data item that gives as much as possible of the input not seen before. (**Not Proved**) After $\mathcal{O}(\frac{n}{q})$ data items, the algorithm has seen the entire input. Hence, with its unbounded computation power, it knows the optimal solution. As it sees the $\mathcal{O}(\frac{n}{q})$ data items, the algorithm builds a tree of width $w = 2^{\mathcal{O}(n/q)}$ considering every possible answer for these. The optimal solution will be an extension of one of these partial solutions. \square

Proof. of Theorem 9 By way of contradiction, suppose there is an algorithm with data item size q that can solve 3-SAT with decision tree width $w = 2^{\Omega(n'/q)}$, where n' is the number of variables.

The proof is a reduction to 3-SAT in the model where each data item contains a variable and a full description of all the clauses that the variable is in. Consider an input instance Φ with n variables and m 3-clauses. We map it to an input instance Φ' with $n' = 6qm$ variables as follows. Start by building a graph with a node u_x for each variable x in Φ . Node u_x will be the center of a star of paths fanning out. For each clause c containing x , there will be a path $u_x = v_{\langle x,c,1 \rangle}, \dots, v_{\langle x,c,2q \rangle}$ of length $2q$. The instance Φ' will have a variable for each node in this graph. For each edge $\{v_{\langle x,c,i \rangle}, v_{\langle x,c,i+1 \rangle}\}$, there will be two clauses $\neg v_{\langle x,c,i \rangle} \wedge v_{\langle x,c,i+1 \rangle}$ and $v_{\langle x,c,i \rangle} \wedge \neg v_{\langle x,c,i+1 \rangle}$. Note that for both of these clause to be satisfied, these two variable must be given the same values. Following this logic along the paths, all the variables in the star associated with a variable x must all have the same value. To conclude the construction, the instance Φ' will have the original clauses c , but instead of using the original variable x , it will use the new variables $v_{\langle x,c,2q \rangle}$. In the graph, this puts a clique between the nodes $v_{\langle x,c,2q \rangle}$, $v_{\langle y,c,2q \rangle}$, and $v_{\langle z,c,2q \rangle}$, when c contains the three variables x , y , and z .

The rest of the reduction is identical to Nassim wrote up. The each of our data items, being associated with a variable x , is mapped to the entire star of variables $v_{\langle x,c,i \rangle}$. When the mirrored algorithm receives all clauses within distance q from $v_{\langle x,c,i \rangle}$, it learns no more than we learn, namely a full description of the clauses that x is in. We set x the same that it sets $v_{\langle x,c,i \rangle}$. We do not need to worry about “free branches” because when their algorithm later sets $v_{\langle x,c,i' \rangle}$, we only need to follow one of their branches. \square