How to Quickly Find a Witness

Daniel Kifer, Johannes Gehrke,
Cristian Bucila[*]
Cornell University
{dkifer,johannes,cristi}@cs.cornell.edu

Walker White
University of Dallas
wmwhite@udallas.edu

ABSTRACT

The subfield of itemset mining is essentially a collection of algorithms. Whenever a new type of constraint is discovered, a specialized algorithm is proposed to handle it. All of these algorithms are highly tuned to take advantage of the unique properties of their associated constraints, and so they are not very compatible with other constraints. In this paper we present a more unified view of mining constrained itemsets such that most existing algorithms can be easily extended to handle constraints for which they were not designed a-priori. We apply this technique to mining itemsets with restrictions on their variance — a problem that has been open for several years in the data mining community.

1. INTRODUCTION

Constrained Itemset Mining is a very important data mining problem [13]. It can be stated as follows. Let $\mathcal{I}$ be a set of distinct "items" (where an item is an undefined primitive). A transaction t is a set of items (a nonempty subset of $\mathcal{I}$) and a database $\mathcal{D}$ is a multiset of transactions. In constrained itemset mining, we would like to find all subsets of $\mathcal{I}$ that satisfy a *constraint*, a user-defined property designed to tailor the output of the data mining algorithm to the user's preferences. Such constraints can be the traditional "minimum support constraint", where we are only interested in sets $X \subseteq \mathcal{I}$ such that there exist at least s transactions $t \in \mathcal{D}$ with $X \subseteq t$, or more complex constraints such as "the average price of the items has to be larger than c", or "the variance of the prices of the items has to be smaller than c". Three important classes of constraints have been studied: monotone, antimonotone, and convertible constraints [13, 16], and each class has its own set of efficient mining algorithms [12, 16, 14, 4, 5, 6]. Some of these algorithms have a certain degree of flexibility – they can efficiently mine constraints from several of these classes simultaneously.

For example, several algorithms can simultaneously mine monotone and antimonotone constraints [14, 4, 5, 6], or mine convertible combined with either monotone or antimonotone constraints [16]. Unfortunately, as we will show later in the paper, the flexibility of these algorithms is very limited, especially when convertible constraints are involved.

In this paper, we present a unified framework for constrained itemset mining that applies to any type of constraint. Our framework is based on the concept of efficiently finding a *witness*, which is a single itemset X on which we can test whether the constraint holds. This test will provide information about properties of other itemsets. This information can then be used for pruning the search space. The notion of a witness has conceptual implications. For example, we now can efficiently mine all three types of constraints *simultaneously* (by finding witnesses for each constraint), and we can also mine complicated constraints that are neither monotone, antimonotone, nor convertible. As a demonstration, we will introduce an efficient algorithm for finding a witness for constraints involving the variance of a set of items.

Our paper makes the following contributions:

- We introduce the concept of a witness, which decouples the strategy for traversing the search space from the efficiency of pruning it (using constraints). This transforms the traversal strategy from a necessary restriction on an algorithm into an optimization heuristic. To illustrate the concept of a witness, we show a very efficient algorithm for finding a witness for a large class of functions which we call *stable functions*. (Section 2)
- We show how to efficiently find a witness for the constraints $\mathrm{var}(S) \leq c$ and $\mathrm{var}(S) \geq c$, and therefore show how to prune using those constraints - a problem that has been open in the literature for several years. (Section 3)
- We outline several heuristics that further improve the efficiency of finding witnesses. (Section 4)

In the remainder of this section, we take the reader on a short tour of this paper. We introduce some terminology and helpful notation in Section 1.1 and then give an overview of our results in Section 1.2. The technical part of the paper continues in Section 2.

1.1 Preliminaries

Let $\mathcal{I}$ be a set of distinct "items" (where an item is an undefined primitive). A transaction t is a set of items (a nonempty subset of $\mathcal{I}$) and a database $\mathcal{D}$ is a multiset of transactions. Given a function whose domain is $\mathcal{I}$, such as $price : \mathcal{I} \to \mathbb{R}$, we extend it to sets of items in the natural way, e.g., $\mathrm{price}(S)$ is the multiset $\{\mathrm{price}(x) : x \in S\}$. We are also given a real-valued function whose domain is $2^{\mathcal{I}}$, the powerset of $\mathcal{I}$. An example of such a function is $\mathrm{support}(S)$, which is the number of transactions in $\mathcal{D}$

[*]This work was supported by NSF Grants IIS-0084762 and IIS-0121175, and by gifts from Microsoft and Intel. Walker White was also supported by the Cornell Intelligent Systems Institute, and Dan Kifer was also supported by an NSF Fellowship.

that are supersets of S. We will use such functions to define *constraints*. For example, if we want to find all sets of items that have support greater than some constant c, we say we are mining with the constraint $\text{support}(S) > c$.

Let us now examine some classes of constraints

Definition 1 (Antimonotone). *A constraint P is antimonotone if whenever $A \subseteq B \subseteq \mathcal{I}$ then $P(B) \Rightarrow P(A)$, or equivalently, $\neg P(A) \Rightarrow \neg P(B)$.*

Definition 2 (Monotone). *A constraint Q is monotone if whenever $A \subseteq B \subseteq \mathcal{I}$ then $Q(A) \Rightarrow Q(B)$, or equivalently, $\neg Q(B) \Rightarrow \neg Q(A)$.*

Note that both antimonotonicity and monotonicity are useful properties. Once we know that itemset A does not satisfy an antimonotone constraint P we don't need to look at supersets of A, and if itemset B satisfies P then we know that all subsets of B satisfy P. Similarly, once we know that B does not satisfy a monotone constraint Q we don't need to look at B's subsets, and if A satisfies Q then so do all supersets of A.

These two classes of constraints have another useful feature: they are both closed under logical conjunction (AND). If P_1 and P_2 are antimonotone (resp., monotone) constraints then so is $P_1 \wedge P_2$ and we can use existing algorithms to prune with this compound constraint.

To define convertible constraints, we need to discuss the notion of a prefix. Fix an ordering on the elements of $\mathcal{I}$. We can therefore treat $S_1 \subseteq \mathcal{I}$ and $S_2 \subseteq \mathcal{I}$ as two sequences. Let ℓ_1 be the length of S_1, and ℓ_2 be the length of S_2. Then S_1 is a *prefix* of S_2 if $\ell_1 \leq \ell_2$ and the first ℓ_1 elements of S_2 are exactly S_1.

Definition 3 (Convertible). *A constraint R is convertible monotone if there is an ordering ω_1 such that whenever S_1 is a prefix of S_2 then $R(S_1) \Rightarrow R(S_2)$ (i.e., $\neg R(S_2) \Rightarrow \neg R(S_1)$) and R is convertible antimonotone if there is an ordering ω_2 such that if T_1 is a prefix of T_2 then $R(T_2) \Rightarrow R(T_1)$ (i.e. $\neg R(T_1) \Rightarrow \neg R(T_2)$). R is convertible if it is both convertible monotone and convertible antimonotone.*

In order to prune with convertible constraints efficiently, an algorithm must examine itemsets in a restricted order. An example of a convertible constraint is $R \equiv \text{avg}(\text{price}(S)) > c$. If the items are sorted by price in ascending order then R is convertible monotone; if the items are sorted in descending order then R is convertible antimonotone. Similarly, the constraint $\text{avg}(\text{price}(S)) < c$ is also convertible.[1] But suppose we want to mine with the following constraint that involves the functions price and weight:

$$\Big(\text{avg}(\text{price}(S)) \leq c\Big) \;\wedge\; \Big(\text{avg}(\text{weight}(S)) \leq d\Big)$$

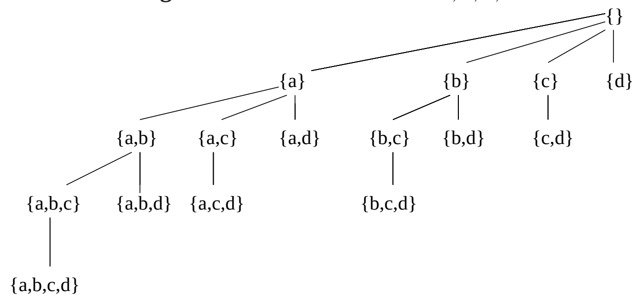
If we assume that price and weight are not correlated, then this conjunction of convertible constraints is *not* convertible. Thus existing algorithms for convertible constraints will not prune efficiently - they will use only one of these constraints and then post-process the output. This is an unfortunate situation, since many interesting predicates are conjunctions of convertible monotone or convertible antimonotone constraints.

1.2 Catching a Witness: An Overview

As an elementary example, suppose that $\mathcal{I} = \{a, b, c, d\}$ and that database D consists of the following sets: $\{a, b, c\}$, $\{b, c, d\}$, $\{b, c\}$,$\{b, d\}$. Assume we are interested in all subsets of $\mathcal{I}$ that

[1]Note that we can replace $<$ by $\leq$ and $>$ by $\geq$ without changing any of the properties stated so far.

Figure 1: An enumeration of a, b, c, d



have support ≥ 2. To find these sets, we must enumerate all possible subsets of $\mathcal{I}$ and then test this property for each of them. At one point we will consider the set $\{a\}$. It is included in only one set in D and therefore it is not interesting to us. We could add more elements to $\{a\}$, in fact we could add any subset of $\{b, c, d\}$ (call this set $\mathcal{F}(\{a\})$) to get another set in our enumeration. However, support is an antimonotone constraint and so any set containing $\{a\}$ will have support less than 2. Thus we can *prune* from consideration all sets X such that $\{a\} \subseteq X \subseteq \mathcal{F}(\{a\}) \cup \{a\}$ — 8 sets in all. Let us call this collection of sets $\mathcal{A}(\{a\})$. We say that $\{a\}$ *is a negative witness for* $\mathcal{A}(\{a\})$ because once we know that our constraint does not hold for $\{a\}$, we know it does not hold for any set in $\mathcal{A}(\{a\})$. If $\{a\}$ had been included in more than one transaction in D, then we could not conclude anything about *all* sets in $\mathcal{A}(\{a\})$ and we would have to examine them further. Thus having $\{a\}$ as a witness for $\mathcal{A}(\{a\})$ allows us to prune a large part of the search space. In general, if P holds for $\{a\}$ implies P holds for all sets in $\mathcal{A}(\{a\})$ then we call $\{a\}$ a positive witness for $\mathcal{A}(\{a\})$ with respect to P and a negative witness with respect to $\neg P$.

A property P can have both positive and negative witnesses. If X is a positive witness then $P(X) = \texttt{true}$ implies that $P(Y) = \texttt{true}$ for all $Y \in \mathcal{A}(X)$ and so we save time by not evaluating $P(Y)$. If X is a negative witness (then it is a positive witness for $\neg P$) then $P(X) = \texttt{false}$ implies that $P(Y) = \texttt{false}$ for all $Y \in \mathcal{A}(X)$ and so we save time by pruning $\mathcal{A}(X)$.

Let us investigate how witnesses actually work when mining with constraints. Each itemset mining algorithm enumerates candidate itemsets in some order, for example through a tree structure (see Figure 1 for an enumeration for a, b, c, d) that is traversed in a depth-first or breadth-first manner. When we examine a set X, we need to find a witness for the subtree rooted at X. In simple cases, such as mining with a single antimonotone constraint, X is this witness, whereas in other cases finding a witness is not so trivial.

For an example where X is not a witness for $\mathcal{A}(X)$, suppose that the prices of a, b, c, d are $1, 7, 6, 5$, respectively, and that we are interested in all sets whose average is at least 6. If we examine node $\{a\}$ in Figure 1 then clearly the average price of $\{a\}$ does not tell us much about the average price of nodes in $\mathcal{A}(\{a\})$, the subtree rooted at $\{a\}$. However, if we add to $\{a\}$ all items with price ≥ 6 we obtain the candidate witness $\{a, b, c\}$. Since we added as many items with price ≥ 6 as possible, if $\{a, b, c\}$ does not have an average ≥ 6, then no set in $\mathcal{A}(\{a\})$ can have an average ≥ 6, and thus $\{a, b, c\}$ is a negative witness. The average price of $\{a, b, c\}$ actually is less than 6 and so same is true for any set in $\mathcal{A}(\{a\})$. Thus this witness allows us to prune the complete subtree. Had the price of item a been 5 or higher, the witness would not give us enough information and we would have to traverse $\mathcal{A}(\{a\})$. Thus finding witnesses for constraints involving an average is rather straightforward: we add all items with value greater than the threshold to obtain the witness itemset, and then we test its average.

Now let us consider a more difficult case. Assume that our constraint states that the *variance* of the prices must be $\leq k$ (or $\geq k$) for some constant k. Now when we examine node $\{a\}$, good witnesses are elements in $\mathcal{A}(\{a\})$ that have maximal or minimal variance. But how do we find an element with maximal variance? Intuitively, the variance of a set is large if the elements are far away from the average. This motivates the following simple algorithm: We add to $\{a\}$ the item x that is furthest from the average of $\{a\}$, then add the item y that is furthest away from the average of $\{a, x\}$, etc. This simple algorithm overlooks the subtlety that we want to add elements that are furthest away from the average of the final witness, rather than the average of $\{a\}$ – but we do not know the average of the final witness a priori. Nevertheless, as we will show in Section 2, a variant of this algorithm actually finds the itemset with maximal variance.

Now let us consider the case where we want to find an itemset with minimal variance. Intuitively we want to include items that are close to the average of the final witness, but not items that are far from the average. Here we run into the same subtlety — the average we are talking about is the average of the witness, not the average of $\{a\}$. These subtleties present significant hurdles to the development of an efficient algorithm for finding a witness. As examples, assume that we are currently examining node X in a search tree. The following two approaches are doomed to fail:

First Algorithm

1. Start at $C = \mathcal{I}$ (all of the items).
2. Remove from C the element in $C \setminus X$ which is furthest away from the current average of C, and return true if this new set has variance $\leq k$
3. Repeat step 2.

Second Algorithm

1. Start at $C = X$.
2. Add to C the element closest to $\operatorname{avg}(C)$, and return true if this new set has variance $\leq k$
3. Repeat step 2.

We can construct an example where both algorithms fail to return the correct witness. Let X be $\{45, 55\}$, the set of two items with prices 45 and 55, respectively. Assume that the subtree rooted at X contains the following items: $1,000,000$ items with price 100; $999,999$ items with price 0; one item with price 30 and another item with price 15. From this example it is clear that there is only one set with minimal variance and we obtain it by adding to X all elements with price 100. Let k be slightly larger than the minimal variance but smaller than the variance of any other set containing X. The first algorithm will fail because it will add the item with price 30. The second algorithm will fail because the average of all prices is slightly less than 50, and thus the algorithm will remove all items with price $= 100$.

From this example we see that we can lower the variance by adding a "dense cluster" — many items with similar values. If we order the items on a line by price and slide an appropriately sized window, we may be able to find a good cluster that lowers the variance enough. In Section 3.2 we will explain the structure of such a window. However, the size of the window depends not only on the *values* of the elements in the window, but also on the *number* of elements the window contains. In fact, as we slide the window, it can shrink: as the left endpoint of the window moves to the right, the right endpoint of the window might move to the left! In Section 3.2, using subtle reasoning about the structure of the space, we describe an algorithm that finds a witness in time *linear* in the number of items.

2. WITNESSES

The execution of a typical data mining algorithm for antimonotone constraints looks like a tree. At the root is the empty set and all other nodes are non-empty sets of items. A child is a superset of its parent and contains one more item than its parent (see Figure 1 for an example). Let n be some node in the tree. Let $B(n)$ be the set of items associated with n, and let $\operatorname{Free}(n)$ be the collection of items that can be added to $B(n)$. $\operatorname{Free}(n)$ is the minimal set such that for any descendant n' of n, $B(n') = B(n) \cup J$ where $J \subseteq \operatorname{Free}(n)$. For example in Figure 1, if $B(n) = \{a, b\}$ then $\operatorname{Free}(n) = \{c, d\}$. Let $\mathcal{A}(n)$ be the collection of sets X such that $B(n) \subseteq X \subseteq B(n) \cup \operatorname{Free}(n)$. As is done in practice, we assume constraints have the following form: $f(X)\#c$ where $\#$ is either $<, \leq, >$ or $\geq$, c is a constant, X is a set and f is a real-valued function whose domain is $2^{\mathcal{I}}$, the powerset of $\mathcal{I}$.

Definition 4 (Witness). *Given a fixed constant c, node n and a function $f : 2^{\mathcal{I}} \to \mathbb{R}$, a set $\mathcal{Y}_n \in \mathcal{A}(n)$ is called a* large witness *if*

$$f(\mathcal{Y}_n) \leq c \Rightarrow \forall X \in \mathcal{A}(n) : \ f(X) \leq c$$

A set $\mathcal{Z}_n \in \mathcal{A}(n)$ is called a small witness *if*

$$f(\mathcal{Z}_n) \geq c \Rightarrow \forall X \in \mathcal{A}(n) : \ f(X) \geq c$$

For a general predicate P, $W_n \in \mathcal{A}(n)$ is a positive witness if

$$P(W_n) = \texttt{true} \Rightarrow \forall X \in \mathcal{A}(n) : \ P(X) = \texttt{true}$$

and W_n is a negative witness if

$$P(W_n) = \texttt{false} \Rightarrow \forall X \in \mathcal{A}(n) : \ P(X) = \texttt{false}$$

The intuition behind this nomenclature is that a set in $\mathcal{A}(n)$ that maximizes f (over $\mathcal{A}(n)$) is a *large* witness and a set that minimizes f is a *small* witness. When it is unambiguous, the notational dependency on n will be dropped. We will use $\mathcal{Y}$ to represent a large witness and $\mathcal{Z}$ to represent a small witness. Note that

$$f(\mathcal{Y}) < c \Rightarrow \forall X \in \mathcal{A} : \ f(X)) < c$$

and

$$f(\mathcal{Z}) > c \Rightarrow \forall X \in \mathcal{A} : \ f(X) > c$$

When we are mining for itemsets X that satisfy $f(X) \geq c$, if $f(\mathcal{Y}_n) < c$ then clearly we can prune out $\mathcal{A}(n)$ — we do not need to look at any set in that collection. If $f(\mathcal{Y}_n) \geq c$ then we do not have enough information to prune and must examine the children of n. If $f(\mathcal{Z}_n) \geq c$ then we do not need to evaluate f on the sets in $\mathcal{A}(n)$ — we know the result will be greater than or equal to c. If $f(\mathcal{Z}_n) < c$ then we do not have enough information and must examine the children of n. Analogous statements are true when we have constraints $f(X)\#c$, where $\#$ is $>, <$, or $\leq$.

2.1 Comparison to Existing Methods

When mining with antimonotone constraints, such as $\operatorname{support}(X) > c$, then for any node n, clearly $B(n)$ is a negative witness and $B(n) \cup \operatorname{Free}(n)$ is a positive witness. For monotone constraints, such as $\operatorname{support}(X) < c$, $B(n)$ is a positive witness and $B(n) \cup \operatorname{Free}(n)$ is a negative witness. For the *function* $\operatorname{support}(X)$, $B(n)$ is a large witness and $B(n) \cup \operatorname{Free}(n)$ is a small witness (clearly a small or large witness is negative or positive depending on the inequality used in the constraint). Thus we have generalized pruning with monotone and antimonotone constraints. Most algorithms that prune with monotone and/or antimonotone constraints can easily be modified to search for witnesses in order to prune efficiently.

Algorithm 1 : AVGminer

```
Require: antimonotone $P$, node $n$,Free$(n)$, $Zsum$, $Zcount$
 1: if $n =$root then
 2:     $\text{Free}(n) \leftarrow \mathcal{I}$
 3:     $Zsum \leftarrow \sum_{x \in \text{Free}, \text{price}(x) \leq c} \text{price}(x)$
 4:     $Zcount \leftarrow \sum_{x \in \text{Free}, \text{price}(x) \leq c} 1$
 5: else if $\neg P(n) \vee Zsum/Zcount > c$ then
 6:     RETURN (no set in $\mathcal{A}$ satisfies both constraints)
 7: else if $P(n) \wedge \text{avg}(\text{price}(n)) \leq c$ then
 8:     OUTPUT $B(n)$
 9: end if
10: $Temp \leftarrow \text{Free}(n)$
11: while $Temp \neq \emptyset$ do
12:     choose some $x \in Temp$; $Temp \leftarrow Temp \setminus \{x\}$
13:     if $x \leq c$ then
14:         $Zsum \leftarrow Zsum - x$; $Zcount \leftarrow Zcount - 1$
15:     end if
16:     create child $n'$ such that $B(n') = B(n) \cup \{x\}$
17:     AVGminer($P$,$n'$,$Temp$,$Zsum + x$,$Zcount + 1$)
18: end while
```

Witness-based pruning can also handle many convertible constraints. One of the most interesting convertible constraints is average (i.e., average price). Assuming all items have a price,

$$\mathcal{Y}_n = B(n) \cup \{x \in \text{Free}(n) : \text{price}(x) \geq c\}$$

is clearly a large witness, and

$$\mathcal{Z}_n = B(n) \cup \{x \in \text{Free}(n) : \text{price}(x) \leq c\}$$

is a small witness.

Naively, it may take $O(\text{Free}(n))$ time to find a witness and calculate its average. However, we just need to know the average of a witness and this can be maintained incrementally in constant time per node. Algorithm 1 shows this technique applied to a simple depth-first algorithm for antimonotone constraints. Note the $O(\mathcal{I})$ initialization step done once at the beginning of the algorithm. We can apply this technique in a straightforward manner to many other algorithms for mining monotone and antimonotone constraints, including DualMiner [6]. One advantage of this technique is that the modified algorithms can handle conjunctions of constraints. This is possible by simply searching for a witness for each constraint. Thus our technique can efficiently find sets of items X such that $R \equiv \text{avg\_price}(X) < c \wedge \text{avg\_weight}(X) < d$, whereas an algorithm designed for convertible constraints cannot prune with R — despite the fact that R is simply a conjunction of convertible constraints. Another advantage of our approach is that the presence of a conjunction of several constraints does not restrict the order in which nodes can be evaluated. This gives our technique an extra degree of freedom for optimization of traversal strategies with heuristics. Note that our technique can even be used to modify existing algorithms for convertible constraints.

Let us introduce some notation before we discuss some conceptual extensions. For convenience we will start identifying items x_i with their prices ($\text{price}(x_i)$). Since several items may have the same price we are now dealing with multisets and as a reminder of this fact, $\oplus$ will represent multiset union and $\ominus$ will represent multiset set-difference. Therefore when we talk about a set in $\mathcal{A}$ we are really talking about a multiset.

The same witnesses that work for average also work for a more general class, the class of stable functions, as introduced in the following definition.

Definition 5. *A real-valued function f is* stable *if, for any c*

$$f(A), f(\{x\}) \geq c \quad \Rightarrow \quad f(A \oplus \{x\}) \geq c, \textit{ and}$$
$$f(A), f(\{x\}) \leq c \quad \Rightarrow \quad f(A \oplus \{x\}) \leq c.$$

The predicates $f(S) \geq c$ and $f(S) \leq c$ are called stable constraints.

Examples of stable functions are average, median, and even linear combinations of moments. A linear combination of moments has the form

$$f(X) = \sum_j \frac{a_j}{n} \sum_{i=1}^{n} x_i^j = \frac{\sum_{i=1}^{n} \sum_j a_j x_i^j}{n} = \frac{\sum_{i=1}^{n} f(\{x_i\})}{n}$$

and is clearly stable. The following theorem shows how to find witnesses $\mathcal{Y}$ and $\mathcal{Z}$ for stable functions.

Theorem 2.1. *Let n be a node and f a stable function. Then*

$$f(B(n) \oplus \{x \in \text{Free}(n) : f(x) \leq c\}) \leq c$$
$$\textit{if and only if} \quad \exists X \in \mathcal{A}(n) \textit{ such that } f(X) \leq c.$$

Also

$$f(B \oplus \{x \in \text{Free}(n) : f(x) \geq c\}) \geq c$$
$$\textit{if and only if} \quad \exists X \in \mathcal{A}(n) \textit{such that} f(X) \geq c.$$

A stable function f is invertible if given $f(x_1, x_2, \ldots, x_k)$ and x_i (for $1 \leq i \leq k$) we can compute
$f(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_k)$. In this case we can use the same approach we used for average to maintain (in constant time per node) the value of f of a witness. For example, linear combinations of moments are invertible. It should be noted that the common convertible constraints are included in the class of invertible stable functions.

3. MINING VARIANCE

We now apply our framework to solve an open problem: mining variance. Due to space constraints, all proofs can be found in the appendix. We have essentially reduced the problem from finding itemsets to searching for a particular node in a lattice. The following key property will be used extensively.

Lemma 3.1. *If M is a multiset and $c, d \in \mathbb{R}$ such that*

$$|d - \text{avg}(M)| \geq |c - \text{avg}(M)|,$$

then we have

$$\text{var}(M \oplus \{d\}) \geq \text{var}(M \oplus \{c\}).$$

This is intuitively obvious; the further away an element is from the average, the larger the variance. This leads to the following simple corollary.

Corollary 3.1. *If $c \in M$ and either $d \geq c \geq \text{avg}(M)$ or $d \leq c \leq \text{avg}(M)$ then $\text{var}(M \ominus \{c\} \oplus \{d\}) \geq \text{var}(M)$.*

To find itemsets that satisfy $\text{var}(X) > c$ we need to prune sets where $\text{var}(X) \leq c$. Thus we need to find a witness $\mathcal{Y}$ such that $\text{var}(\mathcal{Y}) \leq c \Rightarrow \text{var}(X) \leq c$ for any $X \in \mathcal{A}$. The next subsection shows how this is done.

3.1 Finding Maximal Variance

An obvious choice for such a witness $\mathcal{Y}$ is a set in $\mathcal{A}(n)$ that has maximal variance. We begin by examining what such a witness looks like. For any set S let $\min_k(S)$ be the k smallest elements of S and $\max_k(S)$ be the k largest elements of S. Ties are broken according to some convention $>_\kappa$; that is if $a = b$ with $a \in X$ and $b \notin X$ then

- if $a > \text{avg}(X)$ let $a >_\kappa b$
- if $a \leq \text{avg}(X)$ then $a <_\kappa b$

We break all other ties arbitrarily.

Lemma 3.2. *Given a node n, then for any element X in $\mathcal{A}(n)$ with maximal variance, there exist two nonnegative integers L and R such that*

$$X = B(n) \oplus \min_L\{y \in \text{Free}: \ y \leq \text{avg}(X)\} \oplus \max_R\{y \in \text{Free}: \ y > \text{avg}(X)\}$$

In other words, in addition to B, X contains the L^{th} most extreme elements on the left and the R^{th} most extreme elements on the right.

A naive approach to finding a witness $\mathcal{Y}$ would look at all pairs of integers L, R and use Lemma 3.2, but that would result in an $O(|\text{Free}|^2)$ algorithm. We need to find the elements that are furthest away *from the average of* $\mathcal{Y}$ without knowing what this average is. Because of this subtlety, it is surprising not only that a linear time algorithm exists, but also that this algorithm is greedy.

However, we have one precondition. Before we begin to mine, we must sort all elements by value. The sorted order of $\text{Free}(n)$ can easily be maintained by most algorithms as they examine different nodes n. Thus we pay a one-time $O(|\mathcal{I}| \log |\mathcal{I}|)$ startup cost – which is not so bad considering how much time mining algorithms take – and a constant cost per node maintaining this order. Algorithm 2 shows the witness-search algorithm. It returns true if the witness has variance greater than c, false otherwise.

Algorithm 2 : Maximal Variance

```
Require: node $n$, $\text{Free}(n)$ is sorted
 1: $C_0 \leftarrow B(n)$, $i \leftarrow 0$
 2: if $\text{var}(C_0) > c$ then
 3:     RETURN true
 4: end if
 5: $Temp \leftarrow \text{Free}(n)$
 6: while $Temp \neq \emptyset$ do
 7:     choose $x \in Temp$ with $|x - \text{avg}(C_i)|$ largest
 8:     $C_{i+1} \leftarrow C_i \oplus \{x\}$
 9:     if $\text{var}(C_{i+1}) > c$ then
10:         RETURN true
11:     else if $\text{var}(C_{i-1}) \geq \max\big(\text{var}(C_i), \text{var}(C_{i+1})\big)$, $i \geq 1$ then
12:         RETURN false
13:     end if
14:     $i \leftarrow i + 1$
15: end while
16: RETURN false
```

In algorithm 2 we keep adding elements that are furthest away from the *current* average until we find a $Y \in \mathcal{A}$ with $\text{var}(Y) > c$ or we reach the stopping condition. The stopping condition essentially says that we get two chances to keep the variance growing.

In order to show that this algorithm is correct, we need only show that it visits an element with maximal variance and that the condition for returning "false" is correct. Hence correctness is immediate from the following two theorems.

Theorem 3.1. *Without any stopping conditions, Algorithm 2 will visit an element in $\mathcal{A}(n)$ with maximal variance.*

Theorem 3.2. *Let C_i be a multiset such that $\text{var}\, C_{i+1} \leq \text{var}\, C_i$ and $\text{var}\, C_{i+2} \leq \text{var}\, C_i$. Then for any $j \geq i$, $\text{var}(C_j) \leq \text{var}(C_i)$.*

The implication of this theorem is that if no set has variance greater than c, then we will find this out two iterations after we reach a node with maximal variance. The reason for this is that the variance does not grow monotonically, but instead zigzags. This is clear from the following example.

Example. Let $B = \{-40, -40, 40, 40\}$ and $\text{Free} = \{-42, -42, 42, 42\}$. The chain of sets produced is

$C_0 = \{-40, -40, 40, 40\}$	$\text{var}(C_0) = 1600$
$C_1 = C_0 \oplus \{42\}$	$\text{var}(C_1) = 1562.24$
$C_2 = C_1 \oplus \{-42\}$	$\text{var}(C_2) = 1654\frac{2}{3}$
$C_3 = C_2 \oplus \{42\}$	$\text{var}(C_3) = 1634\frac{2}{7}$
$C_4 = C_3 \oplus \{-42\}$	$\text{var}(C_4) = 1682$

Algorithm 2 is not just correct; it is also optimal.

Theorem 3.3. *If there exists a set X with $\text{var}(X) \geq c$, then Algorithm 2 will find the shortest path to any node whose variance $\geq c$.*

3.2 A Small Witness for Variance

Now that we know how to find a large witness $\mathcal{Y}$, we need an algorithm to find a witness $\mathcal{Z}$ such that $\text{var}(\mathcal{Z}) > c \Rightarrow \text{var}(X) > c$ for all $X \in \mathcal{A}$. This is a much more difficult problem. To see why, note that we used the following property to show that a greedy algorithm worked for finding maximal variance.

Lemma 3.3. *For any constant h, if $\text{var}(Y) \geq h$ and $\text{var}(X) \geq h$ then $\text{var}(X \oplus Y) \geq h$.*

This allowed us to add elements that had the largest effect on the variance without worrying too much about the structure of the set we were creating. The constraint $\text{var}(X) < c$ does not have a similar property and this suggests that a greedy algorithm to find a witness Z does not exist. Thus the intuitive algorithms in Section 1.2 do not work. Instead, the following lemma describes what a witness should look like.

Lemma 3.4. *For any element X in (B, T) with minimal variance, there exist two nonnegative integers L and R such that*

$$X = B(n) \oplus \max_L\{y \in \text{Free}: \ y \leq \text{avg}(X)\} \oplus \min_R\{y \in \text{Free}: \ y > \text{avg}(X)\}$$

In other words, if we order the points in Free on a line, X contains $B(n)$ and only the points in some window of size $L + R$ over this line.

It is clear from Lemma 3.4 that if there exists a set X with $\text{var}(X) < c$ then there exists a witness $\mathcal{Z}$ with $\text{var}(\mathcal{Z}) < c$ and that $\mathcal{Z}$ is the multiset union of $B(n)$ and some window over $\text{Free}(n)$. The next lemma states that this window does not have to be too big.

Lemma 3.5. *Let C be a set, $a \in C$, $n = |C|$ and let $D \geq \mathrm{var}(C)$. If $(a - \mathrm{avg}(C \ominus \{a\}))^2 > \frac{n}{n-1}D$ then $\mathrm{var}(C \ominus \{a\}) < D$*

We can derive an easy $O(|\,\mathrm{Free}\,|^2)$ search algorithm using Lemma 3.4, but it is possible to do better. The algorithm to determine if there is a set $\mathcal{Z}$ with $\mathrm{var}((Z)) \leq c$ is a two-step sliding window algorithm. In the first step, we start with a window whose right endpoint is the largest element in Free. We slide the window to the left until the right endpoint is no longer greater than or equal to $\mathrm{avg}(B(n))$. If we have not found a witness, we repeat the same thing, but on the left hand side. We can reflect all points around the y-axis (i.e. multiply them by -1) without affecting the variance of any set, and so by symmetry we only need to describe the first step of the algorithm.

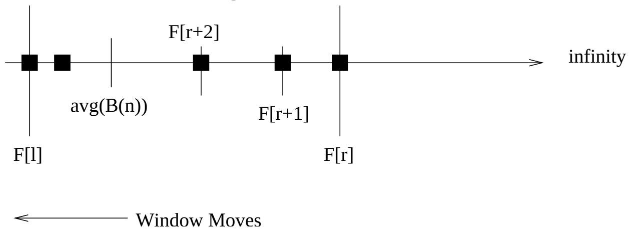
We can use Lemma 3.5 to define a suitable window. Note that the window size depends on the number of points in the window. We also have no guarantees that the window associated with the witness (by Lemma 3.4) is the same size as the algorithm's window. Therefore we must be careful about checking for witnesses to avoid a quadratic search algorithm. Once again, the algorithm assumes the elements of Free are maintained in sorted order. Let $\mathcal{F}$ be the array of elements of Free sorted in descending order. Given the index r of the right endpoint, we want the largest ℓ such that

$$T_{r,\ell} = \{\mathcal{F}[r], \mathcal{F}[r+1], \ldots, \mathcal{F}[\ell]\} \oplus B(n) \qquad (1)$$

satisfies the following properties.

1. $\mathcal{F}[r] - \mathrm{avg}(T_{r,\ell} \ominus \mathcal{F}[r]) \leq \sqrt{ck/(k-1)}$ - where $|T_{r,\ell} \ominus \mathcal{F}[r]| = k - 1$
2. $\mathcal{F}[r] - \mathcal{F}[\ell] \leq 2\sqrt{ck/(k-1)}$

Figure 2: The Window



The first condition states that we do not want the right endpoint to be further away from the average (without the endpoint) than allowable by the hypothesis of Lemma 3.5. Thus given a right endpoint, we know what the smallest allowable average is. Condition 2 states that we do not want the left endpoint to be further away from this quantity than is allowable by Lemma 3.5. The window defined by r and ℓ is W_r, our *target window for r*.

The window associated with a set M of minimal variance is a subset of W_r for some r. To see why, suppose this were not the case. Then one of the two conditions is false. This means that either the left endpoint or the right endpoint of M's window is too far from $\mathrm{avg}(M)$, so by Lemma 3.5 we can remove this endpoint and decrease the variance further.

Theorem 3.4. *Let $\mathcal{Z}$ be a witness which has a window associated to it as in Lemma 3.4. Then there exists a witness $\mathcal{Z}'$ whose window is a subset of the window of $\mathcal{Z}$, and the window of $\mathcal{Z}'$ is contained in a target window W_r for some r.*

For some choices of r, it may not be possible to get a window which brings the average close enough to $\mathcal{F}[r]$. In this case set ℓ to be the largest integer such that the set in equation (1) satisfies

1*. $\mathcal{F}[r] - \mathcal{F}[\ell] \leq \sqrt{ck/(k-1)}$ but
$\mathcal{F}[r] - \mathrm{avg}(T_{r,\ell} \ominus \mathcal{F}[r]) > \sqrt{ck/(k-1)}$

The intuition behind this idea is that we add all the elements that are greater than or equal the minimum average allowed by Lemma 3.5. If this cannot get the average (without $\mathcal{F}[r]$) high enough, then no window will. But if this does move the average close enough, we can keep adding elements that satisfy condition 2. In any case we can move the left endpoint to the left until we reach ℓ and we will recognize ℓ as soon as we see it. Note that this does not change the truth of Theorem 3.4 since this added definition only enlarges windows that would have had length 0 otherwise.

The problem with target windows is that sliding this window to the left may cause the left endpoint to move to the right. In other words, it is possible that $W_{r+1} \subseteq W_r$ and therefore sliding this window over $\mathcal{F}$ may require an $O(|\mathcal{F}|^2)$ computation. For example, suppose $\mathcal{F}[r] = \mathcal{F}[r+1]$ and equality holds in condition 1. Sliding the window over would cause the average to move further away from $\mathcal{F}[r+1]$ and thus violate condition 1. Because of this our algorithm will maintain a window larger than the target window by simply leaving the left endpoint fixed in such cases. Furthermore, if the left endpoint is defined by condition 1*, the left endpoint will never move to the right.

Algorithms 3 and 4 show how to slide the window.

Algorithm 3 : SlideWindow

```
Require: $r, \ell$
 1: $r \leftarrow r + 1$
 2: if $\mathrm{var}(T_{r,\ell}) \leq c$ then
 3:     RETURN (true, $r$, $\ell$)
 4: end if
 5: $\ell \leftarrow ExpandWindow(r, \ell)$
 6: if $\mathrm{var}(T_{r,\ell}) \leq c$ then
 7:     RETURN (true,$r$,$\ell$)
 8: else
 9:     RETURN (false,$r$,$\ell$)
10: end if
```

Algorithm 4 : ExpandWindow

```
Require: $r, \ell$
 1: while $\ell < |\mathcal{I}| - 1$ do
 2:     $k \leftarrow |T_{r,\ell+1}|$
 3:     if $\mathcal{F}[r] - \mathcal{F}[\ell+1] \leq \sqrt{ck/(k-1)}$ then
 4:         $\ell \leftarrow \ell + 1$ (Condition 1*)
 5:     else if $\mathcal{F}[r] - \mathrm{avg}(T_{r,\ell+1} \ominus \mathcal{F}[r]) \leq \sqrt{ck/(k-1)}$, $\mathcal{F}[r] - \mathcal{F}[\ell+1] \leq 2\sqrt{ck/(k-1)}$ then
 6:         $\ell \leftarrow \ell + 1$
 7:     else
 8:         BREAK
 9:     end if
10:     if $\mathrm{var}(T_{r,\ell}) \leq c$ then
11:         BREAK
12:     end if
13: end while
14: RETURN $\ell$
```

Notice that ExpandWindow (Algorithm 4) checks the variance as it moves the left boundary. Since the left and right endpoints shift in one direction only, the variance can be computed in constant time by incrementally maintaining the number of elements in the window, their sum, and the sum of their squares. If the algorithm finds a witness, then it returns immediately and SlideWindow (Algorithm 3) will know this.

The main algorithm is shown in Algorithm 5, where we assume, for simplicity, that $\mathcal{F}[-1] = \infty$. This algorithm returns true if there is an element with $\mathrm{var} \leq c$ and false otherwise.

Algorithm 5 : SmallVar

```
1: $r \leftarrow -1, \ell \leftarrow 0$
2: **while** $\mathcal{F}[r] \geq \mathrm{avg}(B(n))$ **do**
3:     $(result, r, \ell) \leftarrow SlideWindow(r, \ell)$
4:     **if** $result$ =true **then**
5:         RETURN true
6:     **end if**
7: **end while**
8: Repeat with window to the left of $\mathrm{avg}(B(n))$.
9: RETURN false
```

This algorithm runs in $O(|\mathcal{F}|)$ time because variance is computed once each time we move the right endpoint and once each time we move the left endpoint. Although SlideWindow is called $O(|\mathcal{F}|)$ times, it either does not move the left endpoint (hence doing a constant unit of work) or it moves the left endpoint to the left. Thus overall it does $O(|\mathcal{F}|)$ + $O(|\mathcal{F}|)$=$O(|\mathcal{F}|)$ units of work.

Theorem 3.5. *If there is some set in $\mathcal{A}$ with variance not greater than c then SmallVar (Algorithm 5) will find one such set.*

4. HEURISTICS

In practice, we do not always want to run a linear time (or greater) search algorithm to find a witness. Although a linear time algorithm may allow us to prune away an exponential number of sets, sometimes our negative witness satisfies the constraint. In those cases we cannot prune away $\mathcal{A}(n)$ and our time is wasted.

There are two techniques to deal with this problem. It may be possible to amortize the cost of the search by maintaining state that avoids redundant computation. For example, when we showed how to mine average, we maintained the average of the witness incrementally instead of recomputing it every time.

When amortization is not possible, we can use heuristics to tell us when to run the search algorithm. For example, if we are mining with a constraint $\mathrm{var}(S) < c$ then we want to prune sets with variance greater than or equal to c. We can use the observation that the higher the value of $\mathrm{var}(B(n))$, the less likely that $\mathrm{var}(S) < c$ for some $S \in \mathcal{A}$. Thus we can set some threshold τ on $\mathrm{var}(B(n))$, and if the variance is larger than the threshold we search for witness. A similar approach works for the constraint $var(S) > c$. A heuristic can also be based on some precomputed statistics.

When using such constraints we can also benefit from a heuristic which chooses the order in which elements are added to $B(n)$ to create children of $B(n)$. Thus we can try to arrange it so that we see many nodes n for which $B(n)$ has high variance. One such heuristic could be to order all items (in descending order) by their distance from the overall average of $\mathcal{I}$. This is very similar to the approach taken by the convertible algorithms [16].

We should note that in most cases amortization is possible by avoiding redundant computation. For example, suppose we have the constraint $\mathrm{var}(S) > c$ and that we are currently examining a node n. We run Algorithm 2 but find a set with variance $> c$. We cannot prune the subtree rooted at n but we can amortize the cost of the search. Let $a_1, a_2, \dots, a_k$ be the elements that were added to $B(n)$ by the algorithm in that order. Because we cannot prune, we will eventually have to visit the nodes represented by the sets $B(n) \oplus \{a_1\}$, $B(n) \oplus \{a_1, a_2\}$, $B(n) \oplus \{a_1, a_2, a_3\}$, etc, in order to traverse the subtrees rooted at those nodes.

If we run the algorithm at those nodes we will get the same witness as when we ran it at n. Thus at those nodes we can choose not to run the algorithm. Since we visit these nodes anyway, the amortized cost of the search is at most a constant per node plus the cost of maintaining this information. By doing a depth-first traversal of nodes, we can arrange it so that the next k nodes that the algorithm traverses are these k nodes for which we already know the result of the witness search. In this case, maintaining extra state is constant per node. Otherwise we just need to maintain two numbers - the smallest a_i that is at least $\mathrm{avg}(B(n))$ and the largest a_j that is less than $\mathrm{avg}(B(n))$. Then whenever we come to a node of the form $B(n) \oplus J$ (where $J \subseteq \{a_1, a_2, \dots, a_k\}$) we do not have to run the algorithm again since the same witness is also valid.

Similarly, if we had the constraint $\mathrm{var}(S) \leq c$, we run Algorithm 5 on a node n only to discover a set with variance $\leq c$. Again we cannot prune the subtree rooted an n. We let $a_1, \dots, a_k$ be the consecutive sequence of points that define the window of the witness we have found. Clearly this would also be a witness when we examine a node represented by $B(n) \oplus J$ (where $J \subseteq \{a_1, \dots, a_k\}$). We still have to traverse to these nodes to examine their subtrees, however we do not need to run the algorithm again. To maintain this state we need just two numbers – the left and right endpoints of the window of this witness.

5. RELATED WORK

Agrawal et al. first introduced the problem of mining frequent itemsets as a first step in mining association rules [1]. They also considered item constraints such as an item must or must not be contained in an association rule. Agrawal and Srikant introduced the Apriori algorithm and some variations of it [3, 2]. Srikant et al. generalized this mining problem to item constraints over taxonomies[18]. Other types of constraints were introduced later by Ng et al. [13, 12]. These papers introduced the concepts of antimonotone and succinct constraints and presented methods for using them to prune the search space. These classes of constraints were also studied in the case of 2-variable constraints [10] and along with monotone constraints were further generalized and studied by Pei et al. [16, 14]. Boulicant and Jeudy present algorithms for mining frequent itemsets with both antimonotone and non-antimonotone constraints [4, 5]. However they assume that the minimal itemsets satisfying the monotone constraint are easy to compute, also the minimum size of such itemsets is one and there is no gap in the sizes of itemsets that satisfy all the constraints - assumptions that frequently do not hold. This problem was also given a theoretical treatment by Gunopoulos et al. [8]. DualMiner is the first algorithm that simultaneously uses both monotone and antimonotone constraints for pruning the search space [6]. Some recent papers study the problem in the context of multi-attribute data of high dimensionality [17] or take another approach to the problem, such as not pushing the constraints deeply into the mining process, but enforcing the constraints in a final phase [9]. Other papers present specializations of previous algorithms, based on FP-trees [11] or based on projected databases [15].

6. REFERENCES

[1] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In P. Buneman and S. Jajodia, editors, *Proc. SIGMOD 1993*, pages 207–216. ACM Press, 1993.

[2] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast Discovery of Association Rules. In U. M.

Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, chapter 12, pages 307–328. AAAI/MIT Press, 1996.

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In J. B. Bocca, M. Jarke, and C. Zaniolo, editors, *Proc. VLDB 1994*, pages 487–499. Morgan Kaufmann, 1994.

[4] J. Boulicaut and B. Jeudy. Using constraints during set mining: Should we prune or not, 2000.

[5] J.-F. Boulicaut and B. Jeudy. Mining free itemsets under constraints. In *International Database Engineering and Application Symposium*, pages 322–329, 2001.

[6] C. Bucila, J. E. Gehrke, D. Kifer, and W. White. Dualminer: A dual-pruning algorithm for itemsets with constraints. In *Proc. SIGKDD 2002*, Edmonton, Alberta, Canada, July 2002.

[7] A. Delis, C. Faloutsos, and S. Ghandeharizadeh, editors. *SIGMOD 1999, Philadephia, Pennsylvania, USA*. ACM Press, 1999.

[8] D. Gunopulos, H. Mannila, R. Khardon, and H. Toivonen. Data mining, hypergraph transversals, and machine learning. In *Proc. PODS 1997*, pages 209–216, 1997.

[9] J. Hipp and U. Guntzer. Is pushing constraints deeply into the mining algorithms really what we want? *SIGKDD Explorations*, 4(1):50–55, 2002.

[10] L. V. S. Lakshmanan, R. T. Ng, J. Han, and A. Pang. Optimization of constrained frequent set queries with 2-variable constraints. In Delis et al. [7], pages 157–168.

[11] C. K.-S. Leung, L. V. Lakshmanan, and R. T. Ng. Exploiting succinct constraints using fp-trees. *SIGKDD Explorations*, 4(1):31–39, 2002.

[12] R. T. Ng, L. V. S. Lakshmanan, J. Han, and T. Mah. Exploratory mining via constrained frequent set queries. In Delis et al. [7], pages 556–558.

[13] R. T. Ng, L. V. S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained association rules. In L. M. Haas and A. Tiwary, editors, *Proc. SIGMOD 1998*, pages 13–24. ACM Press, 1998.

[14] J. Pei and J. Han. Can we push more constraints into frequent pattern mining? In *ACM SIGKDD Conference*, pages 350–354, 2000.

[15] J. Pei and J. Han. Constrained frequent pattern mining: A pattern-growth view. *SIGKDD Explorations*, 4(1):31–39, 2002.

[16] J. Pei, J. Han, and L. V. S. Lakshmanan. Mining frequent item sets with convertible constraints. In *ICDE 2001*, pages 433–442. IEEE Computer Society, 2001.

[17] C.-S. Perng, H. Wang, S. Ma, and J. L. Hellerstein. Discovery in multi-attribute data with user-defined constraints. *SIGKDD Explorations*, 4(1):56–64, 2002.

[18] R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. In *Proc. KDD 1997*, 1995.

APPENDIX

A. PROOFS

Lemma 3.2. *Given a node n, then for any element X in $\mathcal{A}(n)$ with maximal variance, there exist two nonnegative integers L and R such that*

$$X = B(n) \oplus \min_L\{y \in \text{Free}:\ y \leq \text{avg}(X)\} \oplus \max_R\{y \in \text{Free}:\ y > \text{avg}(X)\}$$

Proof. We need only show the existence of L, as the existence of R is analogous. Let $S_1 = \{y \in X \ominus B:\ y \leq \text{avg}(X)\}$ and let $F_1 = \{y \in \text{Free}:\ y \leq \max(S_1)\}$. If $S_1 = \emptyset$ then $L = 0$ and if $S_1 = F_1$ then $L = |S_1|$. Otherwise, let $m = \max(S_1)$ and choose $y = \min(F_1 \ominus S_1)$. If $y = m$ then $L = |S_1|$ and we are done by the tie-breaking convention. The only other possibility is $y < m$, in which case it is further away from $\text{avg}(X)$ than m and, since $y \notin X$, by Corollary 3.1 we can replace m with y in X to increase the variance. Clearly this case can't happen and so a suitable L always exists. □

Theorem 2.1. *Let n be a node and f a stable function. Then $f(B(n) \oplus \{x \in \text{Free}(n):\ f(x) \leq c\}) \leq c$ if and only if $\exists X \in \mathcal{A}(n)$ such that $f(X) \leq c$. Also, $f(B \oplus \{x \in \text{Free}(n):\ f(x) \geq c\}) \geq c$ if and only if $\exists X \in \mathcal{A}(n)$ such that $f(X) \geq c$.*

Proof. We only prove the first statement, as the second is similar. One direction is obvious. Assume there is some $X \in \mathcal{A}$ such that $f(X) \leq c$. Since f is stable, it follows by induction that $f(X \ominus \{x \in \text{Free}:\ f(x) > c\}) \leq c$. Then, $f(B \oplus \{x \in \text{Free}:\ f(x) \leq c\}) = f(X \ominus \{x \in \text{Free}:\ f(x) > c\} \oplus \{x \in \text{Free} \ominus X:\ f(x) \leq c\}) \leq c$. □

Lemma 3.1. *If M is a multiset and $c, d \in \mathbb{R}$ such that $|d - \text{avg}(M)| \geq |c - \text{avg}(M)|$, then we have $\text{var}(M \oplus \{d\}) \geq \text{var}(M \oplus \{c\})$.*

Proof. Assume $|d - \text{avg}(M)| \geq |c - \text{avg}(M)|$. Let $n = |M|$. Given a constant k,

$$\begin{aligned}
&\text{var}(M \oplus \{k\}) \\
&\quad = \frac{k^2 + \sum_M x_i^2}{n+1} - \frac{\left(k + \sum_M x_i\right)^2}{(n+1)^2} \\
&\quad = \frac{k^2 + \sum_M x_i^2}{n+1} - \frac{k^2 + 2k\sum_M x_i + \left(\sum_M x_i\right)^2}{(n+1)^2}
\end{aligned}$$

Thus

$$\begin{aligned}
&\text{var}(M \oplus \{d\}) - \text{var}(M \oplus \{c\}) \\
&\quad = \frac{d^2 - c^2}{n+1} - \frac{d^2 - c^2 + 2(d-c)\sum_M x_i}{(n+1)^2} \\
&\quad = \frac{n}{(n+1)^2}\left[(d^2 - c^2) - 2(d-c)\,\text{avg}(M)\right]
\end{aligned}$$

The last equation is nonnegative if and only if

$$(d-c)(d+c) \geq (d-c)2\,\text{avg}(M) \qquad (2)$$

Case 1 If $d, c \geq \text{avg}(M)$ then by hypothesis $d \geq c$ and so Equation (2) is satisfied.

Case 2 If $d, c \leq \text{avg}(M)$ then by hypothesis $d \leq c$ and clearly $d + c \leq 2\,\text{avg}(M)$. Therefore, Equation (2) is also satisfied.

Case 3 If $d \geq \operatorname{avg}(M) \geq c$ then $d - c \geq 0$ and by hypothesis $d - \operatorname{avg}(M) \geq \operatorname{avg}(M) - c$ and so $d + c \geq 2\operatorname{avg}(M)$. Thus Equation (2) is satisfied.

Case 4 Finally, if $d \leq \operatorname{avg}(M) \leq c$ then $d - c \leq 0$ and by hypothesis $\operatorname{avg}(M) - d \geq c - \operatorname{avg}(M)$ and so $2\operatorname{avg}(M) \geq d + c$. Even in this case Equation (2) is satisfied.

□

Theorem 3.1. *Without any stopping conditions, Algorithm 2 will visit an element in $\mathcal{A}(n)$ with maximal variance.*

Proof. Let T be the set $B(n) \oplus \operatorname{Free}(n)$. Without stopping conditions, the execution of this algorithm produces a chain of sets $B(n) = C_0 \subset C_1 \subset \cdots \subset C_k = T$ and for all i, $|C_{i+1} \ominus C_i| = 1$. If T has maximal variance then we are done. If not, let j be the largest index such that C_j is a subset of an element with maximal variance but C_{j+1} is not. If C_j has maximal variance then we are done. Otherwise, let M be some superset of C_j that has maximal variance. Also let $c = C_{j+1} \ominus C_j$. By definition, c is chosen by the algorithm because it is the free element furthest away from $\operatorname{avg}(C_j)$.

Because of symmetry, we can assume $c \geq \operatorname{avg}(C_j)$. By the definition of c we know that $M \ominus C_j$ can only contain elements less than c. If some element is larger, it is further away from $\operatorname{avg}(C_j)$; if some element equals c, then we can just replace it with c without affecting variance, which violates the definition of C_j. This means that $c \geq \operatorname{avg}(M)$ since we can not add $M \ominus C_j$ to C_j and increase the average beyond c.

From Corollary 3.1, we know that $M \ominus C_j$ contains no element $\geq \operatorname{avg}(M)$. Otherwise we could replace it with c and the variance will not decrease. Therefore $\max(M \ominus C_j) = m < \operatorname{avg}(M)$.

Now we claim that $\operatorname{avg}(C_j) \geq m$. If this is not the case, then $m > \operatorname{avg}(C_j)$ and adding $M \ominus C_j$ to C_j would not raise the average past m. This implies $m \geq \operatorname{avg}(M)$, which cannot happen. Thus $\operatorname{avg}(C_j) \geq m$ and adding $M \ominus C_j$ to C_j would only lower the average. Since $C_j \subset M$, it follows that $\operatorname{avg}(C_j) \geq \operatorname{avg}(M)$. If $\delta = \min(M \ominus C_j)$ (which is $< \operatorname{avg}(M)$) it also follows that $\operatorname{avg}(C_j) \geq \operatorname{avg}(M \ominus \{\delta\})$. As c is the free element furthest away from $\operatorname{avg}(C_j)$, we see that

$$\begin{aligned} c - \operatorname{avg}(M \ominus \{\delta\}) &\geq c - \operatorname{avg}(C_j) \geq \operatorname{avg}(C_j) - \delta \\ &\geq \operatorname{avg}(M \ominus \{\delta\}) - \delta \geq 0 \end{aligned}$$

By Lemma 3.1 $\operatorname{var}(M \ominus \{\delta\} \oplus \{c\}) \geq \operatorname{var}(M)$, a contradiction. Therefore the greedy algorithm visits the node with the largest variance. □

Theorem 3.2. *Let C_i be a multiset such that $\operatorname{var}(C_{i+1}) \leq \operatorname{var}(C_i)$ and $\operatorname{var}(C_{i+2}) \leq \operatorname{var}(C_i)$. Then for any $j \geq i$, $\operatorname{var}(C_j) \leq \operatorname{var}(C_i)$.*

For this theorem, we need the following results.

Lemma A.1. $\operatorname{var}(C_i) \geq \operatorname{var}(C_{i+1})$ *if and only if $d = C_{i+1} \ominus C_i$ satisfies the condition*

$$n(d - \operatorname{avg}(C_i))^2 \leq (n+1)\operatorname{var}(C_i), \text{ where } n = |C_i|$$

Alternatively, if C_i has average 0 and sum of squares Q, then $n^2 d^2 \leq (n+1)Q$.

Proof. Let $X = C_i$, $n = |X|$ and $Y = C_{i+1}$. Let $d = Y \ominus X$. First suppose that $\operatorname{avg}(X) = 0$ and without loss of generality assume that $d \geq 0$. Let Q be the sum of squares in X. Since $\operatorname{var}(X) \geq \operatorname{var}(Y)$,

$$\begin{aligned} &\operatorname{var}(Y) - \operatorname{var}(X) = \\ &\quad \frac{Q + d^2}{n+1} - \frac{d^2}{(n+1)^2} - \frac{Q}{n} = \\ &\quad \frac{nd^2 - Q}{n(n+1)} - \frac{d^2}{(n+1)^2} \leq 0 \end{aligned}$$

This is true if and only if

$$\begin{aligned} &\frac{(n^2+n)d^2 - (n+1)Q - nd^2}{n(n+1)} \leq 0 \\ \Leftrightarrow\ &\frac{n^2 d^2 - (n+1)Q}{n(n+1)} \leq 0 \\ \Leftrightarrow\ &n^2 d^2 - (n+1)Q \leq 0 \\ \Leftrightarrow\ &n^2 d^2 \leq (n+1)Q \end{aligned}$$

If X does not have average 0 then since variance is translation invariant, we can apply this result to

$$X' = \{x - \operatorname{avg}(X) : x \in X\}$$

and to $Q' = \sum_{x \in X} (x - \operatorname{avg}(X))^2$, $d' = d - \operatorname{avg}(X)$. Now Q' is just $n \operatorname{var}(X)$ and so

$$n(d - \operatorname{avg}(X))^2 \leq (n+1)\operatorname{var}(X)$$

□

When k is an integer and S is a multiset, we use $k \cdot S$ to denote k multiset unions of S with itself.

Lemma A.2. *Let C be a set, a be an element and let $D \geq \operatorname{var}(C)$ and $p \in \mathbb{Z}^+$ be constants. If*

$$\operatorname{var}(C \oplus (p \cdot \{a\})) > D$$

then

$$(a - \operatorname{avg}(C))^2 > \frac{|C|+1}{|C|} D$$

Proof. Let $Q = \operatorname{var}(C)$. Let

$$C' = \{x - \operatorname{avg}(C) \ : \ x \in C\}$$

and let $n = |C|$ and $b = a - \operatorname{avg}(C)$. Clearly

$$\operatorname{avg}(C') = 0$$

and $\operatorname{var}(C') = Q$. nQ is the sum of squares of C' and $\operatorname{var}(C' \oplus (p \cdot \{b\})) = \operatorname{var}(C \oplus (p \cdot \{a\}))$.

$$\begin{aligned} &\operatorname{var}(C \oplus (p \cdot \{a\})) \geq D \Rightarrow \operatorname{var}(C' \oplus (p \cdot \{b\})) \geq D \\ &\Rightarrow \frac{nQ + pb^2}{n+p} - \frac{p^2 b^2}{(n+p)^2} \geq D \\ &\Rightarrow (n+p)nQ + (n+p)pb^2 - p^2 b^2 \geq D(n+p)^2 \\ &\Rightarrow (n+p)nQ + npb^2 \geq D(n+p)^2 \\ &\Rightarrow (n+p)nD + npb^2 \geq D(n+p)^2 \\ &\Rightarrow npb^2 \geq npD + p^2 D \\ &\Rightarrow b^2 \geq D + pD/n \geq \frac{n+1}{n} D \end{aligned}$$

□

Proof of Theorem 3.2. Let $n = |C_i|$ and $e = C_{i+1} \ominus C_i$, $f = C_{i+2} \ominus C_{i+1}$. This theorem is obvious if $n < 2$ so we can assume that $n \geq 2$. Without loss of generality we can assume that $\text{avg}(C_i) = 0$ and that $e > 0$.

Let Q be the sum of squares of C_i. Suppose there exists a $j > i$ such that $\text{var}(C_j) > \text{var}(C_i)$. Then let $J = C_j$. Let a be the largest value in $J \ominus C_i$ and b be the smallest value. Clearly $a = e$ and $b \leq f$. a satisfies the conditions in Lemma A.1. If $a^2 \leq \frac{Q}{n}$ then this theorem is obviously true, so we can assume $a^2 > \frac{Q}{n}$. Since f is at least as far from $\text{avg}(C_{i+1})$ as b, Lemma 3.1 implies that $\text{var}(C_i \oplus \{a, b\}) \leq \text{var}(C_i)$. Thus if $b \leq -\sqrt{\frac{Q}{n}}$ then by Lemma 3.1,

$$\text{var}(C_i \oplus \{a,b\}) \geq \text{var}\left(C_i \oplus \left\{\sqrt{\frac{Q}{n}}, -\sqrt{\frac{Q}{n}}\right\}\right) = \text{var}(C_i)$$

So $b > -\sqrt{\frac{Q}{n}}$, which means that $|a| > |b|$ and $a + b > 0$.

From J we will inductively construct a multiset J^* such that $\text{var}(J^*) \geq \text{var}(J)$. Let $J^0 = J$ and given J^k, pick some element c from the set

$$H = \{x \in J^k \ominus C_i : \ a > x > b\}$$

If $c \geq \text{avg}(J^k)$ then we know $a > c$ and so let $J^{k+1} = J^k \ominus \{c\} \oplus \{a\}$. Similarly, if $c \leq \text{avg}(J^k)$ then we know $c > b$ and so let $J^{k+1} = J^k \ominus \{c\} \oplus \{b\}$. By Lemma 3.1, $\text{var}(J^{k+1}) \geq \text{var}(J^k)$. If H is empty and we cannot choose an element c, then let $J^* = J^k$. Clearly $J^* = C_i \oplus (p \cdot \{a, b\}) \oplus (q \cdot \{x\})$ for some integers p and q, where x is either a or b.

Now suppose $q \geq 1$ and x is the element b. If $p = 0$ then a is further away from $\text{avg } J^* \ominus \{b\}$ than b. If $p \geq 1$ then

$$\text{avg } J^* \ominus \{b\} \leq p(a+b)/(n+2p+q) \leq (a+b)/2$$

and so a is also further away from $\text{avg } J^* \ominus \{b\}$ than b. By Lemma 3.1, $\text{var}(J^* \ominus \{b\} \oplus \{a\}) \geq \text{var}(J)$ and

$$J^* \ominus \{b\} \oplus \{a\} = C_i \oplus ((p+1) \cdot \{a,b\}) \oplus ((q-2) \cdot \{b\})$$

Note this is also true when $q = 1$ if we interpret $B \oplus (-1) \cdot A$ as $B \ominus A$. Using this argument repeatedly, we get the set

$$C_i \oplus ((p + \lfloor q/2 \rfloor) \cdot \{a,b\}) \oplus ((q \mod 2) \cdot \{a\})$$

which has variance $\geq \text{var}(J^*)$. Thus, without loss of generality we can assume that x is the element a.

By hypothesis, $\text{var}(C_i) - \text{var}(C_{i+2}) \geq 0$, and so

$$\begin{aligned}
&\text{var}(C_i) - \text{var}(C_{i+2}) = \\
&\quad \frac{Q}{n} - \frac{Q + a^2 + b^2}{n+2} + \frac{(a+b)^2}{(n+2)^2} = \\
&\quad \frac{2Q - na^2 - nb^2}{n(n+2)} + \frac{(a+b)^2}{(n+2)^2} \quad \geq 0
\end{aligned}$$

which implies that $\frac{2Q}{n} - a^2 - b^2 + \frac{(a+b)^2}{n+2} \geq 0$.

Let $I = C_i \oplus (p \cdot \{a, b\})$. Then we have that

$$\begin{aligned}
&\text{var}(C_i) - \text{var}(I) = \\
&\quad \frac{Q}{n} - \frac{Q + pa^2 + pb^2}{n+2p} + \frac{p^2(a+b)^2}{(n+2p)^2} = \\
&\quad \frac{2pQ - npa^2 - npb^2}{n(n+2p)} + \frac{p^2(a+b)^2}{(n+2p)^2} = \\
&\quad \frac{p}{n+2p}\left(\frac{2Q}{n} - a^2 - b^2 + \frac{p(a+b)^2}{n+2p}\right) = \\
&\quad \frac{p}{n+2p}\left(\frac{2Q}{n} - a^2 - b^2 + \frac{p(n+2)}{n+2p}\frac{(a+b)^2}{n+2}\right) = \\
&\quad \frac{p}{n+2p}\left(\frac{2Q}{n} - a^2 - b^2 + \frac{pn+2p}{n+2p}\frac{(a+b)^2}{n+2}\right) \geq \\
&\quad \frac{p}{n+2p}\left(\frac{2Q}{n} - a^2 - b^2 + \frac{(a+b)^2}{n+2}\right) = \\
&\quad \frac{p}{n+2p}\left(\text{var}(C_i) - \text{var}(C_{i+2})\right) \geq 0
\end{aligned}$$

Thus $\text{var}(I) \leq \text{var}(C_i)$ and $J^* = I \oplus (q \cdot \{a\})$. By Lemma A.2, it is only possible for $\text{var}(J^*) > \text{var}(C_i)$ if $(a - \text{avg}(I))^2 > \frac{n+1}{n}\text{var}(C_i)$. Since $a + b \geq 0$, we have $a \geq \text{avg}(I) \geq 0$ and so $a \geq (a - \text{avg}(I)) \geq 0$.

From Lemma A.1

$$\text{var}(C_i)\frac{n+1}{n} \geq a^2 \geq (a - \text{avg}(I))^2$$

Therefore $\text{var}(J^*)$ cannot be larger than $\text{var}(C_i)$, and so when the variance of a set is not less than the variance of either its two successors, its variance is not less than the variance of any of its successors. □

Theorem 3.3. *If there exists a set X with $\text{var}(X) \geq c$, then Algorithm 2 will find the shortest path to any node whose variance $\geq c$.*

Proof. Let C_i be the chain of sets from the proof of Theorem 3.1. We call a node X *quick* if $\text{var}(X) \geq c$ and if $\forall Y \in (B, T)$, then $|Y| \leq |X| \Rightarrow \text{var}(Y) < c$. We need to show that for some i, C_i is quick.

Quickness is a maximality property. Therefore, to complete the proof, we simply carry out the proof of Theorem 3.1, substituting "quick" for "maximal variance". □

Lemma 3.3. *For any constant h, if $\text{var}(Y) \geq h$ and $\text{var}(X) \geq h$ then $\text{var}(X \oplus Y) \geq h$.*

Proof. For convenience, let $A = \sum_{y_i \in Y} y_i$ and $B = \sum_{y_i \in Y} y_i^2$ and $n = |Y|$. Also let $C = \sum_{x_i \in X} x_i$ and $D = \sum_{x_i \in X} x_i^2$ and $m = |X|$. Since variance is invariant under translation, we can assume that the elements of X and Y are nonnegative. We know that

$$\text{var}(Y) = \frac{B}{n} - \frac{A^2}{n^2} \geq h \Rightarrow B \geq nh + \frac{A^2}{n}$$
$$\text{var}(X) = \frac{D}{m} - \frac{C^2}{m^2} \geq h \Rightarrow D \geq mh + \frac{C^2}{m}$$

Therefore

$$\begin{aligned}
B + D &\geq nh + \frac{A^2}{n} + mh + \frac{C^2}{m} \\
&= h(n+m) + \frac{mA^2 + nC^2}{mn} \\
&= h(n+m) + \frac{(mn+m^2)A^2 + (mn+n^2)C^2}{mn(m+n)} \\
&\geq h(n+m) + \frac{mnA^2 + mnC^2 + 2mnAC}{mn(m+n)} \\
&= h(n+m) + \frac{A^2 + C^2 + 2AC}{m+n} \\
&= h(n+m) + \frac{(A+C)^2}{m+n}
\end{aligned}$$

This implies $h \leq \frac{B+D}{m+n} - \frac{(A+C)^2}{(m+n)^2} = \operatorname{var}(X \oplus Y)$ □

Lemma 3.4. *For any element X in (B, T) with minimal variance, there exist two nonnegative integers L and R such that*

$$\begin{aligned}
X = B(n) &\oplus \max_L \{y \in \text{Free} : \ y \leq \operatorname{avg}(X)\} \\
&\oplus \min_R \{y \in \text{Free} : \ y > \operatorname{avg}(X)\}
\end{aligned}$$

Proof. The proof is analogous to that of Lemma 3.2 and again assumes that ties are broken according to the convention $>_\kappa$.

□

Lemma 3.5. *Let C be a set, $a \in C$, $n = |C|$ and let $D \geq \operatorname{var}(C)$. If $(a - \operatorname{avg}(C \ominus \{a\}))^2 > \frac{n}{n-1} D$ then $\operatorname{var}(C \ominus \{a\}) < D$.*

Proof. We prove the lemma by contradiction. Assume $var(C \ominus \{a\}) \geq D$. Without loss of generality, assume $avg(C \ominus \{a\}) = 0$. Let Q be the sum of squares of $C \ominus \{a\}$. So, we have $Q \geq (n-1)D$ and $(n-1)a^2 > nD$. Then $var(C) = (Q + a^2)/n - a^2/n^2 = (nQ + (n-1)a^2)/n^2 > (nQ + nD)/n^2 \geq (n(n-1)D + nD)/n^2 = D$. Thus, $var(C) > D$, contradiction, therefore $var(C \ominus \{a\}) < D$.

□

Theorem 3.5. *If there is some set in $\mathcal{A}$ with variance not greater than c then SmallVar (Algorithm 5) will find one such set.*

To prove this theorem, we need the following result.

Lemma A.3 (The Expanding Window). *If there exists a witness $\mathcal{Z}$ with $\operatorname{var}(\mathcal{Z}) \leq c$ and and window defined by right endpoint a and left endpoint b then if $d > b$ and $\operatorname{avg}(\mathcal{Z}) - \mathcal{F}[d] \leq \sqrt{c(|\mathcal{Z}|+1)/|\mathcal{Z}|}$ then there is a witness $\mathcal{Z}'$ with $\operatorname{var}(\mathcal{Z}') \leq c$ and window defined by endpoints a and d.*

Proof. If $d = b+1$ then this is obvious by Lemma A.2. If $d > b+1$ then this is true by induction. □

Proof of Theorem 3.5. The algorithm's window can be in 3 states.

State 1: The window satisfies condition 1*.

State 2: The window satisfies conditions 1 and 2.

State 3: The left endpoint didn't move and the previous window satisfied conditions 1 and 2 (otherwise the current window satisfies condition 1* and is in state 1).

It is clear that if the algorithm is in state 1, sliding the window will only move it to states 1 or 2. If it is in state 2 then it will only go to states 2 or 3. Finally, if it is in state 3, it can go to any other state.

We already know that there exists a witness $\mathcal{Z}$ that is characterized by Lemma 3.4 – it has an associated window. By Lemma 3.5 we can restrict ourselves to only look for witnesses whose right endpoints are close enough to the average of the rest of the witness (i.e. satisfy condition 1). Without loss of generality we can assume that the window is minimal in the sense that no witness has an associated window that is a proper subset of this. By symmetry, we can also assume that the right endpoint of this window is not less than $\operatorname{avg}(B(n))$. Thus by sliding the algorithm's window over to the left, at some point the right endpoint r of the algorithm window will also be the right endpoint of the minimal witness.

When this happens, by Theorem 3.4, the target window, W_r, contains the window of the minimal witness. When the algorithm window is in states 2 or 3 and has right endpoint r, it will contain the target window and therefore the window of the minimal witness. Since r is also the right endpoint of the window of the minimal witness, the algorithm window with right endpoint r can not satisfy condition 1* and so it will not be in state 1. Thus it is sufficient to prove two things.

(i) If the algorithm is in states 2 or 3 and a witness's window is contained within the algorithm's current window *and* both windows have the same right endpoint, then the algorithm will find this out.

(ii) If the window of a witness is inside the algorithm's window and the algorithm's window is in state 1, then when we slide the window it will still contain the window of a witness.

We prove property (ii) first. If the window satisfies condition 1* and a witness has its window inside the algorithm's window, then we test if $\operatorname{var}(T_{r,\ell}) \leq c$. If the variance is greater than c then we claim there is a witness $\mathcal{Z}$ whose window is inside of this window, but that the window of the witness has a right endpoint different from r. If there was a witness whose right endpoint was r then by condition 1*, $\mathcal{F}[r]$ is far away from $\operatorname{avg} \mathcal{Z} \ominus \{\mathcal{F}[r]\}$, and by Lemma 3.5 $\mathcal{Z} \ominus \{\mathcal{F}[r]\}$ is also a witness. Furthermore, the right endpoint of this witness's window will still be larger than $\operatorname{avg}(B(n))$ by the minimality assumption. At this point the algorithm would slide the window over and the window of $\mathcal{Z}$ would still be contained in the algorithm's window.

Property (i) follows by induction. The inductive hypothesis will also maintain the fact that if the algorithm window is in state 3 then if the window of a witness is inside the algorithm's window, then we can extend the left boundary of the witness's window to the left endpoint of the algorithm's window and so create a set with variance not greater than c.

The base case of the induction is the beginning of the algorithm. The first window must be either in state 1 or 2. If it is in state 1 then property (i) is vacuously true for the first algorithm window. If it is in state 2, let r be the right endpoint. ExpandWindow will initially start with the with right and left endpoints r and r and will check if we have a witness whose window has right endpoint r every time it moves the left endpoint. Thus we will know if a witness's window is inside the algorithm window and shares a right endpoint with it.

Let $g(r)$ be the largest integer such that

$$\mathcal{F}[r] - \mathcal{F}[g(r)] \leq \sqrt{ck/(k-1)}$$

where $k = |T_{r,g(r)}|$. Note that $g(r) = \ell$ if condition 1* holds and $g(r) \leq \ell$ for states 2 and 3. We will left r_1, ℓ_1 be the boundaries of the previous algorithm window and r_2, ℓ_2 be the boundaries of the current window. As we can treat any occurrence of state 1 as the initial state, the inductive step has five cases.

Case (I) The current window is in state 2 and the previous window is in state 1. Since the algorithm explicitly checks for the variance of $T_{r_2,\ell_1}, \ldots T_{r_2,\ell_2}$, we must show that we do not miss anything by not checking the variance of $T_{r_2,r_2}, \ldots, T_{r_2,\ell_1-1}$. Since $\ell_1 = g(r_1) \leq g(r_2)$ it is sufficient to show that we do not need to check the variance of $T_{r_2,r_2}, \ldots, T_{r_2,g(r_2)-1}$. Suppose there is a witness $\mathcal{Z}$ whose window has right endpoint r and left endpoint L between r_2 and $g(r_2) - 1$. Because the witnesses must satisfy condition 1, the distance between $\operatorname{avg} Z$ and $\mathcal{F}[g(r_2)]$ is not greater than the distance between $\mathcal{F}[r]$ and $\mathcal{F}[g(r_2)]$. Since $|\mathcal{Z}| \leq |T_{r_2,g(r_2)}|$ and

$$(|\mathcal{Z}| + 1)/|\mathcal{Z}| \geq (|T_{r_2,g(r_2)}| + 1)/|T_{r_2,g(r_2)}|$$

the $g(r_2)$ and $\mathcal{Z}$ satisfy the conditions of the Expanding Window Lemma. So $T_{r_2,g(r_2)}$ would also be a witness and would be explicitly checked by the algorithm.

Case (II) Both the current window and the previous window are in state 2. Once again we must show that we do not miss anything by not checking the variance of the sets $T_{r_2,r_2}, \ldots, T_{r_2,\ell_1-1}$. Suppose there is a witness $\mathcal{Z}$ whose window has right endpoint r and left endpoint L between r_2 and $\ell_1 - 1$. If $L \leq g(r_2)$ then we use the same arguments as in Case (I). Otherwise $L > g(r_2) \geq g(r_1)$ and so $T_{r_1,L}$ must satisfy Condition 1. But if $T_{r_2,L}$ were a witness then by Lemma A.2 so is $T_{r_1,L}$ and this would have been discovered by the inductive hypothesis.

Case (III) The current window is in state 2 and the previous window is in state 3. Again we show that we do not miss anything by not checking the variance of $T_{r_2,r_2}, \ldots, T_{r_2,\ell_1-1}$. Suppose there is a witness $\mathcal{Z}$ whose window has right endpoint r and left endpoint L between r_2 and $\ell_1 - 1$. Then the window of the witness is bounded by r_1, ℓ_1 and so by the inductive hypothesis we can extend its left endpoint and so $T_{r_2,L}$ will have $\operatorname{var} \leq c$. This set will then be checked by the algorithm.

In the last two cases, the current state is 3. We must show that any witness whose window fits inside the algorithm's current window can have its left endpoint extended to the algorithm's window's left endpoint. From this it follows that if a witness has right endpoint equal to r_2, then the variance of T_{r_2,ℓ_2} is $\leq c$ and this is checked by the algorithm. If $\ell_2 \leq g(r_2)$ then we should actually be in state 3. Therefore $\ell_2 > g(r_2)$.

Case (IV) The current window is in state 3 and the previous window is in state 2. In this case $\ell_1 = \ell_2$ and T_{r_1,ℓ_2} satisfies both conditions 1 and 2 but T_{r_2,ℓ_2} does not. Since the fewer the elements in the window, the larger $2\sqrt{ck/(k-1)}$ is, condition 2 is automatically satisfied. Hence condition 1 must be false. Thus if there is a witness whose window has endpoints R, L where $r_2 \leq R \leq L < \ell_2$ and if $\operatorname{avg}(T_{R,L}) \geq \mathcal{F}[\ell_2]$ then

$$|\operatorname{avg}(T_{R,L}) - \mathcal{F}[\ell_2]| \leq \sqrt{ck/(k-1)}$$

where $k - 1 = |T_{r_1,\ell_2}|$. Since $k^* = |T_{R,L}| < |T_{r_1,\ell_2}|$, we have $|\operatorname{avg}(T_{R,L}) - \mathcal{F}[\ell_2]| \leq \sqrt{c(1 + |T_{R,L}|)/|T_{R,L}|}$ and we can use the Expanding Window Lemma. If, on the other hand $\operatorname{avg}(T_{R,L}) < \mathcal{F}[\ell_2]$ then $\mathcal{F}[\ell_2]$ is closer to $\operatorname{avg}(T_{R,L})$ than $\mathcal{F}[R]$ is and because of our restriction on witnesses we can again use the Expanding Window Lemma to expand the left endpoint from L to ℓ_2.

Case (V) Both the current window and the previous window are in state 3. This follows trivially from the inductive hypothesis since $r_1 \leq r_2$ and $\ell_1 = \ell_2$ and so the current algorithm window is contained in the previous algorithm window. $\square$