

# Estimating the longest increasing sequence in polylogarithmic time

Michael Saks\*

C. Seshadhri

saks@math.rutgers.edu

csesha@us.ibm.com

Dept. of Mathematics

IBM Almaden

Rutgers University

## Abstract

Finding the length of the longest increasing subsequence (LIS) is a classic algorithmic problem. Let  $n$  denote the size of the array. Simple  $O(n \log n)$  algorithms are known for this problem. What can a sublinear time algorithm achieve? We develop a polylogarithmic time randomized algorithm that for any constant  $\delta > 0$ , estimates the length of the LIS of an array upto an additive error of  $\delta n$ . More precisely, the running time of the algorithm is  $(\log n)^c (1/\delta)^{O(1/\delta)}$  where the exponent  $c$  is independent of  $\delta$ . Previously, the best known polylogarithmic time algorithms could only achieve an additive  $n/2$  approximation.

The *distance to monotonicity*, which is the fractional size of the complement of the LIS, has been studied in depth by the streaming and property testing communities. Our polylogarithmic algorithm actually has a stronger guarantee, and gives (for any constant  $\delta > 0$ ) a multiplicative  $(1 + \delta)$ -approximation to the distance to monotonicity. This is a significant improvement over the previously known 2-factor approximations.

---

\*This work was supported in part by NSF under CCF 0832787.

# 1 Introduction

Finding the length of longest increasing subsequence (LIS) of an array is a classic algorithmic problem in computer science. We are given an array, represented by a function  $f : [n] \rightarrow \mathbb{R}$ . We will use the terms *array* and *function* interchangeably. An *increasing subsequence* of this array is a sequence of indices  $i_1 < i_2 < \dots < i_k$  such that  $f(i_1) \leq f(i_2) \leq \dots \leq f(i_k)$ . An LIS is an increasing subsequence of  $f$  having maximum size. Schensted [Sch61] first dealt with finding the LIS of arrays. The LIS problem is one of the basic examples of dynamic programming in many basic algorithms textbooks, and often ends up as an exercise problem [CLRS00]. Indeed, the LIS is exactly the longest common subsequence between  $f$  and its sorted version. This yields an  $O(n^2)$  algorithm. Fredman [Fre75] gave an  $O(n \log n)$  algorithm, which can be seen as a more clever way of maintaining the dynamic program. Aldous and Diaconis [AD99] use the elegant algorithm of *patience sorting* to find the LIS. Fredman [Fre75] gave a matching lower bound for comparison based algorithms that find the LIS. Ramanan [Ram97] subsequently strengthened this bound for algebraic decision trees and showed the  $\Omega(n \log n)$  lower bound holds even for determining the length of the LIS.

It is also interesting to consider the size of the *complement* of the LIS. This is referred to as the (edit) *distance to monotonicity*, since it is the minimum number of values that need to be changed to make  $f$  monotonically increasing. Conventionally, this number is divided by the total size  $n$ , so it is represented as a fraction. For function  $f$ , it is denoted by  $\varepsilon_f$ . Letting  $\lambda_f$  be the size of the LIS as a fraction of  $n$ , we have  $\varepsilon_f = 1 - \lambda_f$ . For exact algorithms, of course, finding  $\lambda_f$  is equivalent to finding  $\varepsilon_f$ . Approximating these quantities can be very different problems.

In recent years, motivated by the increasing ubiquity of massive sets of data, there has been considerable attention given to the study of approximate solutions of computational problems on huge data sets by judicious sampling of the input. In the context of property testing it was shown in [EKK<sup>+</sup>00, DGL<sup>+</sup>99, Fis01, ACCL07] that for any  $\varepsilon > 0$ ,  $O(\varepsilon^{-1} \log n)$  random samples are necessary and sufficient to distinguish the case that  $f$  is increasing ( $\varepsilon_f = 0$ ) from the case that  $\varepsilon_f \geq \varepsilon$ . Subsequently, results of [PRR06, ACCL07] gave *multiplicative approximations* to the distance to monotonicity in (essentially)  $O(\varepsilon^{-1} \log n)$ . For any constant  $\tau > 0$ , these algorithms output a value that is in the range  $[\varepsilon_f, (2 + \tau)\varepsilon_f]$ , which essentially gives a 2-approximation to  $\varepsilon_f$ .

While these algorithms give good approximations to the distance to monotonicity, they provide little information about the LIS if  $\lambda_f$  is between 0 and 1/2. Note that in this case  $\varepsilon_f \geq 1/2$  and so a 2-approximation to  $\varepsilon_f$  may output 1 as the estimate of  $\varepsilon_f$ , which only implies that the LIS has size 0. Indeed, there are simple examples where  $\varepsilon_f = 1/2$  and the algorithms of [PRR06, ACCL07] return an estimate of 1. The difficulty lies in the fact that a small value of  $\varepsilon_f$  means that  $f$  is nearly increasing, and this structure can be used to strongly guide the algorithm. However when  $\lambda_f$  is small (say 1/10), the structure of  $f$  is much less apparent. None of the previously known algorithms are able to detect this structure. Indeed, getting multiplicative approximations to the distance is much easier than getting approximations (even additive) to  $\lambda_f$ .

In this paper, we focus on getting additive approximations to  $\lambda_f$ , i.e. outputting a value that is between  $\lambda_f - \delta$  and  $\lambda_f$  for some small fixed  $\delta > 0$ . Notice that this is equivalent to getting an additive  $\delta$ -approximation for  $\varepsilon_f$ . The existing multiplicative 2-approximation algorithm for  $\varepsilon_f$  gives the rather weak consequence of an additive 1/2-approximation for  $\lambda_f$ . Rather surprisingly, nothing better was known.

We show that poly-logarithmic time is sufficient to get *arbitrarily* good additive approximations to  $\lambda_f$ . For any constant  $\delta$ , we get additive  $\delta$ -approximations in poly-logarithmic time. We will use *with high probability* to indicate probability at least  $1 - n^{-\Omega(\log n)}$ , where the  $\Omega$ -notation hides some fixed constant independent of any parameters.

**Theorem 1.1** *Let  $f$  be an array of size  $n$  and  $\lambda_f$  the size of the LIS divided by  $n$ . There is a randomized algorithm  $A(f, \delta)$  which takes as input an array  $f$  and parameter  $\delta > 0$ , and outputs, with high probability, a real number  $\kappa$  such that  $\lambda_f \in [\kappa, \kappa + \delta]$ . The running time is bounded by  $(1/\delta)^{O(1/\delta)}(\log n)^c$ , for some absolute constant  $c$  independent of  $n$  and  $\delta$ .*

Using our techniques, we also have a somewhat stronger algorithm, that provides a multiplicative  $(1 + \tau)$ -approximation to  $\varepsilon_f$  for any  $\tau > 0$ . We note that this is the first improvement over the  $(2 + \tau)$ -approximations of [PRR06, ACCL07]. As before,  $c$  denotes some absolute constant.

**Theorem 1.2** *Let  $\tau > 0$  and  $\varepsilon_f > 0$  be the distance to monotonicity for input array  $f$ . There exists an algorithm with running time  $(1/\varepsilon_f)^{O((1/\tau)\log(1/\tau))}(\log n)^c$  that computes a real number  $\varepsilon$  such that (with high probability)  $\varepsilon_f \in [\varepsilon, (1 + \tau)\varepsilon]$ .*

In this preliminary version, we give a complete proof of a somewhat weaker algorithm. The running times will be of the forms  $(\log n/\delta)^{O(1/\delta)}$  and  $(\log n/\varepsilon_f)^{O(1/\tau)}$  respectively. The authors have found some major simplifications for the stronger algorithm and are currently working on writing a detailed proof for that. Observe that even this weaker algorithm runs in polylogarithmic time for constant  $\tau$ .

## 1.1 Related work and relation to other models

The field of *property testing* [RS96, GGR98] deals with finding sublinear, or even constant, time algorithms for distinguishing whether an input has a property, or is far from the property (see surveys [Fis01, Ron01, Gol98]). The property of monotonicity has been studied over a variety of domains, of which the boolean hypercube and the set  $[n]$  have usually been of special interest [GGL<sup>+</sup>00, DGL<sup>+</sup>99, FLN<sup>+</sup>02, HK03, ACCL07, PRR06, BGJ<sup>+</sup>09]. Our result can be seen as a *tolerant tester* [PRR06], which can closely approximate the distance to monotonicity.

The LIS has been studied in detail in the streaming model [GJJK07, SW07, GG07, EJ08]. Here, we are allowed a small number (usually, just a single) of passes over the array and we wish to estimate either the LIS or the distance to monotonicity. The distance approximations in the streaming model are based on the sublinear time approximations. The technique of counting inversions used in the property testers and sublinear time distance approximators is a major component of these algorithms. This problem has also been studied in the communication models where various parties may hold different portions of the array, and the aim is to compute the LIS with minimum communication. This is usually studied with the purpose of proving streaming lower bounds [GJJK07, GG07, EJ08].

There has been a body of work on studying the Ulam distance between strings [AK07, AK08, AIK09, AN10]. For permutations, the Ulam distance is exactly the size of the complement of the longest common subsequence. Note that Ulam distance between a permutation and the identity permutation is the distance to monotonicity. Recently, a sublinear time algorithm for approximating the Ulam distance between two permutations was discovered [AN10]. We again note that the previous techniques for distance approximation play a role in these results. Our results may be helpful in getting better approximations for these problems.

## 1.2 Obstacles to additive estimations of the LIS

A first natural approach to estimating the length of the LIS is to look at the LIS of a small random sample  $s$ . We could try outputting  $\lambda_s$ , the fractional LIS length of the sample, as our estimate of  $\lambda_f$ . A little consideration shows that there are inputs where this algorithm will answer 1 with high probability even though  $\lambda_f$  is arbitrarily close to 0. Let  $K$  be a large constant and  $n = Kt$ . For

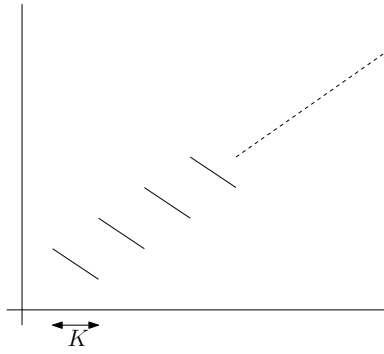


Figure 1: Small random samples will almost always be totally increasing

$0 \leq i \leq t-1$  and  $0 \leq j < K$ , set  $f(iK+j+1) = iK-j$ . Refer to Figure 1. The LIS of this function is size  $t = n/K$ , but a small random sample will almost certainly be completely increasing. This is because the scale of violations (pairs  $i < j$  with  $f(i) > f(j)$ ) is too small for the sample to detect. Violations can lie at any scale, and the algorithm needs to be able to find them wherever they are. We refer to elements in the domain  $[n]$  as indices. A natural approach to algorithms for estimating the LIS is to give an algorithm which, given an index  $i$ , classifies  $i$  as *good* or *bad* in such a way that:

- The good indices form an increasing sequence.
- The number of good indices is close to the size of the LIS, so the number of bad indices is small.

This was the approach used in [ACCL07] and [PRR06]. To classify  $i$ , the algorithm considers random samples in all intervals of the form  $[i-2^k, i+2^k]$ , for all  $k$ . This checks all scales efficiently. The idea is that if at any one of these scales,  $i$  is involved in many violations then it should be bad. This heuristic does well if  $\varepsilon_f$  is small, but does poorly otherwise. For the function shown in Figure 1, even when  $K$  is 2 (so  $\lambda_f = 1/2$ ), these algorithms will denote all indices as bad. The problem is that all indices are involved in many violations in an interval of size  $K$ . These algorithms are unable to observe that in the decreasing intervals of size  $K$ , a single index *can* be called good. A far more complicated problem is that when the LIS is not most of the points, the decision about whether to label a point as good *may involve small scale properties of the sequence far from  $i$* . Consider the following example: suppose  $n = 6k$  and divide the indices into three contiguous blocks, where the first has size  $k$ , the second has size  $2k$  and the third has size  $3k$ . Consider the sequence  $f$  whose first block is  $100k+1, \dots, 101k$ , whose second block is  $1, 101k+1, 2, 101k+2, \dots, k, 101k$  and whose third block is some increasing subsequence of  $k+1, \dots, 99k$ . Let  $f'$  be a sequence that agrees with  $f$  on the first two blocks. The final  $3k$  positions is some sequence with values in the range  $k+1, \dots, 99k$  but looks like the function in Figure 1. Refer to Figure 2 for a pictorial representation of these sequences.

Notice that the LIS of  $f$  has size  $4k$  (and excludes the first block of elements) while the LIS for  $f'$  has size  $2k$  (and includes the first block of elements). Furthermore, in  $f$ , an increasing sequence that uses any element from the first block has size at most  $2k$ . Any good approximation algorithm for the LIS must realize that elements from the first block are part of the LIS for  $f'$ , but not for  $f$ . But the only difference between  $f$  and  $f'$  is in very local properties in the third block. This is because violations in the third block (for  $f'$ ) are only present at a very small scale. So, small scale

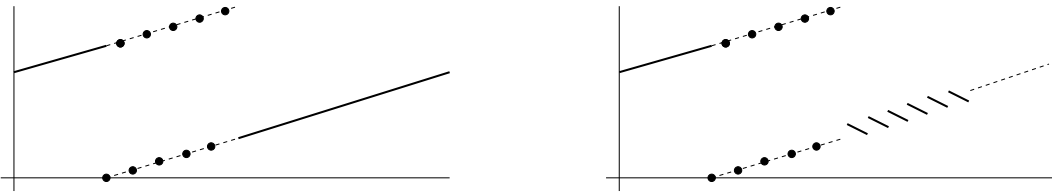


Figure 2: The sequences/functions  $f$  and  $f'$  where small scale properties in one block affect points of a distant block

violations in the third block are critical to decisions made in the first block. Since one can build many variants of this example, where the size and location of the critical block is different, and the important scale within the critical block may also vary, it seems that very global information at all scales may be required to make a satisfactory decision about any particular index.

Another perspective is to consider the dynamic program that computes the LIS. The dynamic program starts by building and storing small increasing sequences. Eventually it tries to join them to build larger and larger sequences. Any one of the currently stored increasing sequences may extend to the LIS, while the others may turn out to be incompatible with any increasing sequence close in size to the LIS. Deciding among these alternatives requires accurate knowledge of how partial sequences all over the sequence fit together. Any sublinear time algorithm that attempts to approximate the LIS arbitrarily well has to be able to (at some level) mimic this. It seems almost hopeless to do this in poly-logarithmic time. Indeed, the authors suffered for a long time in trying to deal with these issues.

## 2 The algorithmic idea and intuition

In this section we give an overview of the algorithm. It is convenient to identify the array/function  $f$  with the set of points  $\{(i, f(i))\}$  in  $\mathbb{R}^2$ . We take the natural partial order where  $(a_1, a_2) \preceq (b_1, b_2)$  iff  $\forall i, a_i \preceq b_i$ . The LIS is now the size of the longest chain in this partial order. Any chain can be seen as a piecewise linear curve with positive slopes (where we just interpolate between consecutive points of the chain). The axes of the plane will be denoted, as usual, by  $x$  and  $y$ . We use *interval* to denote an interval of indices, which geometrically is an interval on the  $x$ -axis. We use *value* to denote  $y$ -coordinates.

The first idea, which takes its inspiration from complexity theory, is to consider an *interactive protocol* for approximating  $\lambda_f$ . (Note that we will not make any mention of these protocols in the actual algorithm or in any proof but they are a nice picture to have in mind.) Suppose that we have a sequence  $f$  and two players, a prover and verifier. The prover knows  $f$  completely and the verifier has query access to  $f$ . The prover makes a claim of the form  $\lambda_f \geq b$  for some  $b \in (0, 1)$ . The verifier wishes to check this claim by asking the prover questions and querying  $f$  on a small number of indices. At the end of the interaction, the verifier either accepts or rejects. The proof must have the following (usual) properties. If  $\lambda_f \geq b$ , then there is a strategy of the prover that makes the verifier accept with high probability. If the prover is lying and  $\lambda_f < b - \delta$ , then for any strategy of the prover it is unlikely that the verifier will accept.

The protocol works as follows. Imagine that the prover has some fixed LIS in mind. The verifier maintains an interval  $I \subseteq [n]$  and a pair of values  $a_L, a_R$ . Initially  $I$  is equal to  $[n]$  and  $a_L = 0$  and  $a_R = \infty$ . The verifier asks the prover for a point in  $\mathbb{R}^2$  called a *splitter*. The  $x$ -coordinate of the

splitter must be in  $I$ , the  $y$ -coordinate must be in  $[a_L, a_R]$ , and the splitter must not be a violation with any point of the LIS. We can think of the splitter as generating a partition of  $I$  into two intervals  $I_L < I_R$  and having a value  $h \in [a_L, a_R]$ . We will alternatively refer to the triple  $(I_L, I_R, h)$  as a splitter. All values of the LIS that belong to  $I_L$  are  $\leq h$  and all LIS values that belong to  $I_R$  are  $\geq h$ . The verifier selects a uniformly random index<sup>1</sup>  $i \in I$  and then sets  $I$  to be the one of  $I_L$  and  $I_R$  that contains  $i$ . If  $I_L$  is chosen, then  $a_R$  is set to  $h$  and if  $I_R$  is chosen, then  $a_L$  is set to  $h$ . The verifier proceeds until  $I$  is a singleton point  $i$ . The verifier then queries  $f(i)$  and calls the result a *success* if  $a_L \leq f(i) \leq a_R$  and a *failure otherwise*. The verifier repeats this process some number (say  $\log n$ ) of times. If the fraction of successes is at least  $b - \delta/2$ , he accepts the proof. Otherwise, he rejects the proof.

It is easy to check that if the prover is honest, then the verifier will accept with high probability (since one run of the process is a success if and only if the final  $i$  lies on the LIS). We leave it as an exercise to the reader to show that if the LIS is at most  $(b - \delta)n$ , then the verifier will reject with high probability.

The number of rounds of the algorithm depends on how *balanced* the splitters selected by the prover are. As long as the splits are all  $\alpha$ -balanced (both  $I_L$  and  $I_R$  have size at least  $\alpha|I|$ ) for constant  $\alpha > 0$ , then the number of rounds of the basic procedure will be at most  $O(\log n/\alpha)$ .

This protocol leads to the following algorithmic idea: try to simulate the prover algorithmically. Thus, we search for a reasonably balanced splitter. Consider a potential splitter  $(I_L, I_R, h)$  for  $I$ . We say that an index  $i$  is a *violation* with this splitter if:  $i \in I_L$  and  $f(i) > h$  or  $i \in I_R$  and  $f(i) < h$ . Naturally, we cannot hope to find the exact splitter in sublinear time. To give ourselves room, we allow ourselves to find an “approximate” splitter. This is a splitter such that the number of violations of the LIS (restricted to  $I$ ) with this point is small. How small is small enough? If we can guarantee that the fraction of violations of the LIS with the splitter is at most  $\mu$ , then the total error will be  $\mu$  times the number of rounds which is  $O(\log n/\alpha)$ . Hence, it would suffice to have  $\mu$  be a small fraction of  $\alpha/\log n$ .

But since we do not know what the LIS is, how do we ensure that the number of violations of the splitter with the LIS is small? One way is be conservative and to simply count violations of the potential splitter with any index in  $I$ . Suppose we can find a splitter in  $I$  with less than  $\alpha|I|/\log n$  violations. Then, we can safely declare this as the approximate splitter and proceed with the simulation. However, there is no guarantee that we can actually find such a splitter.

To deal with this situation, we need a new idea, that of boosting the quality of the approximation. Suppose we have an additive  $\delta$ -approximation to  $\lambda_f$ . Can we use it to get an additive  $\delta'$ -approximation for some smaller  $\delta'$ ? If we could, then by using the known additive  $1/2$ -approximation algorithm as a starting point, we might be able to recursively combine these algorithms into one that achieves any desired error. Suppose it were the case that we could find a partition  $I_L, I_R$  of  $I$  with the property that for any  $h$ ,  $(I_L, I_R, h)$  has at least  $\mu|I|$  violations. Then we could proceed as follows. Choose a polylog sample of possible  $h$  values (by sampling random  $y$ -coordinates of points in  $I$ ). With high probability, one of them is close to being the “real” splitter that comes from the LIS. For each  $h$ , let  $I_L(h)$  and  $I_R(h)$  be the set of points in each half that are consistent with splitter  $(I_L, I_R, h)$ . Compute an additive  $\delta$ -approximation to the LIS length in each of  $I_L(h)$  and  $I_R(h)$  and take their sum. For the correct candidate splitter  $h$ , this sum is within  $\delta(|I_L(h)| + |I_R(h)|)$  of the true LIS. The key observation is that  $|I_L(h)| + |I_R(h)| \leq (1 - \mu)|I|$ , since  $(I_L, I_R, h)$  has at least  $\mu|I|$  violations. Hence, this error is at most  $\delta(1 - \mu)|I|$ . We are able to boost an additive  $\delta$ -approximation algorithm to get a  $\delta(1 - \mu)$  approximation. Essentially, because of strong property of the partition  $I_L, I_R$ , we can conclude that the LIS length in  $I$  is at most  $(1 - \mu)|I|$ . Note the dynamic program-

---

<sup>1</sup>We stress that the source of randomness is hidden from the prover.

ming nature of this approach. Each choice of  $h$  generates two subproblems, one to the left ( $I_L(h)$ ) and one to the right ( $I_R(h)$ ). We solve all these subproblems (with the  $\delta$ -approximation) for each choice of  $h$ . For each  $h$ , we associate the sum of estimates given by the  $\delta$ -approximation for  $I_L(h)$  and  $I_R(h)$ . The  $h$  with the maximum such value should be a good approximate splitter.

Thus it seems we have a useful dichotomy: if we have a good splitter value  $h$  for the partition  $I_L, I_R$  (one having at most  $\mu|I|$  violations with  $I$ ), then we can proceed with the simulation of the interactive proof. If, on the other hand, there is no such  $h$ , then we can boost an existing approximation algorithm.

It is not clear how to put these together to get an algorithm, but there is a more significant difficulty. For the simulation of the interactive protocol to work, we argued that  $\mu$  should be  $O(1/\log n)$ . For the recursive boosting to work efficiently, we will need  $\mu$  to be at least  $\Omega(1/\log \log n)$ . Why? For each level of recursion, we can improve the additive approximation from  $\delta$  to  $\delta(1-\mu)$ . So we will need  $1/\mu$  levels of recursion to improve from  $\delta$  to  $\delta/2$ . At each level of the recursion, we make at least 2 recursive calls, for the left and right subproblems generated by any choice of  $h$ . So the total number of iterated recursive calls is exponential in  $1/\mu$ , forcing  $\mu$  to be  $\Omega(1/\log \log n)$ . Thus, although the dichotomy is quite pleasing, we have a huge gap from an algorithmic perspective.

We seek ways to close this gap. Further consideration (and using past work in the area as a guide) it seems fruitful to look for a splitter with a much weaker property. The present procedure uses an excessively stringent condition to find a splitter. One could imagine an input where (say) the first  $99n/100$  points form an increasing sequence, and the last  $n/100$  points are violations with all the previous points. Our current procedure would not find an appropriate splitter, but it is quite believable that sublinear time procedures could find a good splitter. We redefine our notion of an approximate splitter  $(I_L, I_R, h)$ . We now allow large subintervals of  $I_L$  and  $I_R$  to contain many violations, as long as the number of these violations is a *small* fraction of the subinterval. More formally, a subinterval of  $I_L$  is relevant if it contains the right endpoint of  $I_L$  (we define analogously for  $I_R$ ). Let  $\mu$  and  $\gamma$  be some parameters. Any relevant subinterval of  $I_L$  or  $I_R$  of size at least  $\gamma|I|$  contains at most a  $\mu$ -fraction of violations with  $h$ . The advantage of this condition is that we can set  $\mu$  to be much larger than  $O(1/\log n)$ . It can be shown that a splitter which satisfies the above condition with a small constant  $\mu$  and  $\gamma = O(1/\log n)$  is a sufficiently good splitter.

So this weaker property works with a constant value of  $\mu$ , so we can ask: what if we have a partition  $I_L, I_R$  for which there is no  $h$  that satisfies the above condition for  $\mu$  and  $\gamma$ ? Is this enough to use the boosting algorithm? The answer, unfortunately, is no. But, by introducing a more sophisticated version of the boosting algorithm, we are able to get the dichotomy we need. For a given interval  $I$  either of the following holds: we can find a splitter  $(I_L, I_R, h)$  that satisfies the above conditions for  $\mu$  and  $\gamma$  (and so we can proceed with the simulation). Otherwise, we can concrete evidence that the LIS in  $I$  has size at most  $(1-\mu)|I|$ . Then, the more sophisticated boosting algorithm can improve a given  $\delta$ -approximation algorithm to a  $\delta(1-\mu)$ -approximation algorithm.

We finish this section by sketching the idea of the better boosting algorithm. Divide the interval  $I$  into  $s$  equal-sized intervals  $I_1, \dots, I_s$  with  $s$  polylogarithmic in  $n$ . Suppose we can find values  $h_0, h_1, h_2, \dots, h_s$  that are *consistent* with the LIS. This means that the values of LIS points in  $I_r$  are at least  $h_{r-1}$  and at most  $h_r$ . Refer to the left part of Figure 3. Using these values, we can estimate the LIS length in  $I$ . For each subinterval  $I_r$ , we look at all input points in  $I_r$  with values between  $h_r$  and  $h_{r+1}$ . For this set of points, we use an additive  $\delta$ -approximation algorithm to approximate the LIS length. We then add up these estimates over all  $h$ , to get our estimate for the LIS length in  $I$ . Under the hypothesis that there is no good splitter for some balanced partition  $I_L, I_R$  of  $I$ , we can show that this sum is a  $\delta(1-\mu)$ -approximation to the LIS.

Since we do not know the values  $h_1, \dots, h_s$ , we need to search for approximate versions of them. This will be done by finding the longest path in a DAG, which can be solved by a dynamic program.

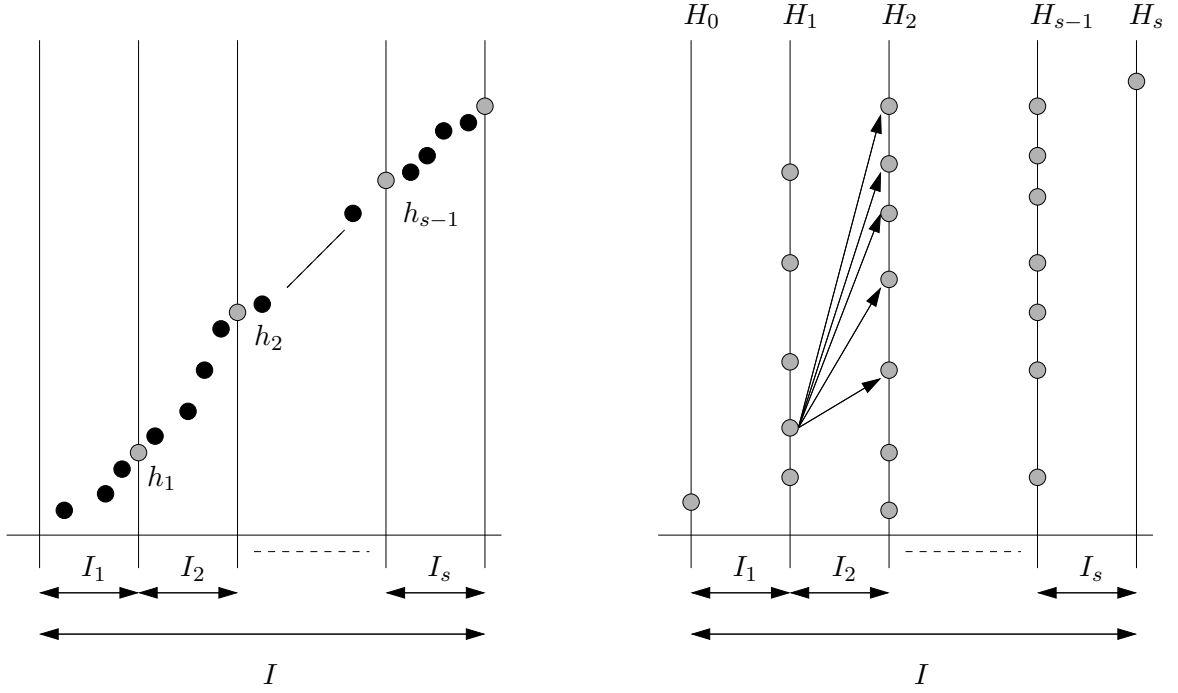


Figure 3: The figure to the left shows the placement of possible values  $h_1, h_2, \dots$ . The black points for the LIS and the grey points correspond to the  $h$  values. The figure to the right shows the construction of the graph  $G$ , and the layers of vertices  $H_i$ . The arrows give the directed edges from one vertex in layer  $H_1$ .

We choose a random sample of  $\log^2 n$  points from each  $I_i$  and let  $H_i$  denote the set of  $y$ -coordinates of these points. The set  $H_i$  is the set of candidate  $h_i$  values (for the boundary, we set  $H_0$  (resp.  $H_s$ ) to have the minimum (resp. maximum) value in  $I$ ). We need to find the best  $h_i$  values among these candidates. We construct the following directed layered graph with  $(t+1)$  layers. Refer to the right part of Figure 3. Layer  $i$  has vertices for each value in  $H_i$ . We put a directed edge between  $h_i \in H_i$  and  $h_{i+1} \in H_{i+1}$  if  $h_i \leq h_{i+1}$ . Consider the set of all points in  $I_{i+1}$  with values in the range  $[h_i, h_{i+1}]$ . We estimate the LIS length of this set using the  $\delta$ -approximation, and set this to be the weight of the edge  $(h_i, h_{i+1})$ . Observe that this graph has polylogarithmic size, and all weights can be computed with polylogarithmically many calls to the additive  $\delta$ -approximation algorithm. For a path  $h_0, h_1, \dots, h_r$ , the length is an estimate of the LIS consistent with these values. So it is natural to use the longest path as our estimate for the total LIS in  $I$ . Since this graph is a DAG, the longest path can be found in time polynomial in the graph size, which is polylogarithmic in  $n$ .

These two ingredients - the simulation of the interactive protocol and the improved boosting algorithm - are combined together to give our algorithm. There are various parameters such as  $\alpha, \mu, \gamma$  involved in the algorithm, and we must choose their values carefully. We also need to determine how many levels of boosting are required to get a desired approximation. A direct choice of parameters leads to a  $(\log n)^{1/\delta}$  approximation algorithm. The better algorithm claimed in Theorem 1.1 is obtained by a more involved version of the algorithm, which involves modifying the various parameters as the algorithm proceeds. This enables us to reduce the number of recursive calls needed for the basic boosting algorithm from polylogarithmic to a constant depending on  $\delta$ .

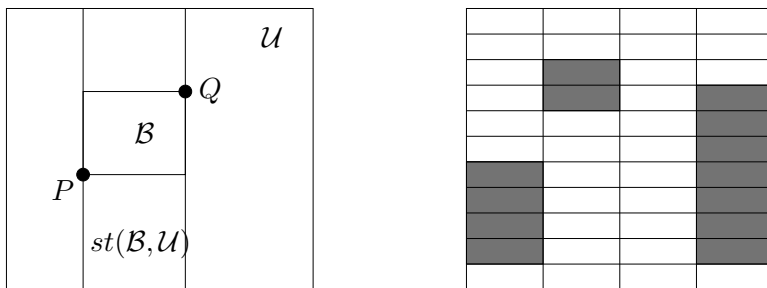


Figure 4: The left figure shows a strip. The box  $\mathcal{B}$  is contained in  $\mathcal{U}$ , and is formed by points  $P$  and  $Q$ . The strip  $st(\mathcal{B}, \mathcal{U})$  is shown. The points  $P$  and  $Q$  are endpoints for the strip. The right figure shows a grid. The grey boxes are all grid boxes. Note how they can span many horizontal lines but are always between two vertical lines.

### 3 Preliminaries

We introduce our notation and define many key concepts. As mentioned earlier, it will be convenient to think of the input as a set  $\mathcal{F}$  of (geometric) points  $\{(i, f(i))\}$  in  $\mathbb{R}^2$ , with  $x$  and  $y$  denoting the axes. We will assume, wlog, that all coordinates are positive and distinct<sup>2</sup>. We take the natural partial order where  $(a_1, a_2) \preceq (b_1, b_2)$  iff  $\forall i, a_i \preceq b_i$ . An increasing sequence is just a chain in this partial order. A point  $P$  is to the *left* (resp. *right*) of  $Q$ , if the  $x$ -coordinate of  $P$  is less (resp. more) than that of  $Q$ . We use capital letters to denote points, calligraphic letters for sets of points, and bold letter for collections of sets of points. Two points  $P, Q$  are *consistent* if  $P \prec Q$  or  $Q \prec P$ . Otherwise, they form a *violation*.

We now list out some basic definitions that will be used throughout the paper.

**Box:** A very important definition is that of a *box*. A box  $\mathcal{B}$  is the set of input points *internal* to an axis-parallel rectangle in the plane. (We do not include points on the boundary in  $\mathcal{B}$ .) Although intuitively a box is a geometric object, formally, it is just a set of points in  $\mathcal{F}$ . Abusing notation, we will refer to corners of  $\mathcal{B}$  as if it is a axis-parallel rectangle in the plane. Our algorithm will represent and store a box  $\mathcal{B}$  by just its corners (a constant size representation), and will not explicitly store the set  $\mathcal{B}$ . So when a box  $\mathcal{B}$  is passed as an argument, we just pass this representation.

The *size* of box  $\mathcal{B}$  is the number of points, also denoted by  $|\mathcal{B}|$ . We let  $\varepsilon_{\mathcal{B}}$  denote the distance to monotonicity for the input points in  $\mathcal{B}$ . Given a pair of points  $(P, Q)$ ,  $P \preceq Q$ , the box  $Box(P, Q)$  is formed by taking the axis-parallel rectangle formed by points  $P$  and  $Q$ . We will say  $Box(P, Q) \preceq Box(P', Q')$  if  $Q \preceq P'$  (so all points in the first box are dominated by all points in the latter). We also use  $\mathcal{F}$  to also denote the box containing all points.

**Chain:** A *chain of boxes* is a sequence of boxes  $\mathcal{B}_1 \preceq \mathcal{B}_2 \preceq \mathcal{B}_3 \cdots$ . A *maximal chain of boxes* is a chain where the upper right corner of  $\mathcal{B}_i$  has the same  $y$ -coordinate as the lower right corner of  $\mathcal{B}_{i+1}$ . The size of a chain is the total number of points in the chain.

**Strip:** For a box  $\mathcal{U}$ , the  $\mathcal{U}$ -strip is a box which has the same vertical extent as  $\mathcal{U}$ . We use  $x(\mathcal{B})$  to denote the  $x$ -interval corresponding by the  $\mathcal{B}$ . Let  $\mathcal{B}, \mathcal{U}$  be boxes such that  $\mathcal{B} \subseteq \mathcal{U}$ . The  $\mathcal{U}$ -strip of a box  $\mathcal{B}$ , denoted by  $st(\mathcal{B}, \mathcal{U})$ , is the box of all points in  $\mathcal{U}$  with  $x$ -coordinates in  $x(\mathcal{B})$ . In other words,  $st(\mathcal{B}, \mathcal{U})$  is a box formed by the  $x$ -extent of  $\mathcal{B}$  and the  $y$ -extent of  $\mathcal{U}$ . Refer to Figure 4. The

<sup>2</sup>We use a simple tie-breaking rule for two  $y$ -coordinates that are the same. So for  $i < j$  if  $f(i) = f(j)$ , the algorithm will assume that  $f(i) < f(j)$ .

points (if any) contained in the left and right edges of the rectangle corresponding to  $st(\mathcal{B}, \mathcal{U})$  are the *endpoints* of the strip. (By the uniqueness of coordinates, the left and right endpoints, if they exist, are unique.) The *width* of box  $\mathcal{B}$  is the size of the strip  $st(\mathcal{B}, \mathcal{F})$  and is denoted by  $w(\mathcal{B})$ .

**Grid:** Consider a box  $\mathcal{B}$ . Let  $r_y < r_x < 1$  be positive parameters. The *grid*,  $\text{Grid}_{\mathcal{B}}(r_x, r_y)$ , is formed by a set of vertical and horizontal lines inside this box.  $\text{Grid}_{\mathcal{B}}(r_x, r_y)$  has a crucial property: the number of  $\mathcal{B}$ -points between two adjacent vertical lines is at most  $r_x|\mathcal{B}|$ . The total number of vertical lines is at most  $2/r_x$ . The number of  $\mathcal{B}$ -points between adjacent horizontal lines is at most  $r_y|\mathcal{B}|$ . There are at most  $2/r_y$  horizontal lines. A *grid point* is simply a point that is an intersection point for some vertical and horizontal line of that grid. A *grid box* for  $\text{Grid}_{\mathcal{B}}(r_x, r_y)$  is a box formed by taking two consistent grid points on *adjacent* vertical lines. Note that grid boxes can overlap, but their  $x$ -intervals never have a non-trivial intersection. Refer to Figure 4.

The only points that we will ever deal with are either input points (in  $\mathcal{F}$ ) or grid points. Henceforth, we will just use *point* for an input point. For clarity, we will always use *grid point* for the corresponding concept.

A crucial definition is that of *safeness* of points. Since it involves many parameters and the reader may have to come back to it often, we put it in a separate environment. This definition is inspired from [ACCL07] (which is in turn based on ideas from [EKK<sup>+</sup>00]). The parameter  $\alpha$  should just be thought of as a small constant.

**Definition 3.1** *Let  $\mathcal{B} \subseteq \mathcal{U}$ . A point  $P \in \mathcal{B}$  is  $(\mathcal{B}, \mathcal{U}, \gamma, \mu)$ -safe if the following is true: for any  $\mathcal{U}$ -strip  $\mathcal{S}$  having  $P$  as an endpoint such that  $\mathcal{S} \subseteq st(\mathcal{B}, \mathcal{U})$  and  $|\mathcal{S}| \geq \gamma|st(\mathcal{B}, \mathcal{U})|$ , the number of violations with  $P$  in this strip is at most  $\mu|\mathcal{S}| + (\alpha^2/\log n)w(\mathcal{B})$ .*

*A point  $P$  is  $(\mathcal{B}, \mathcal{U}, \gamma, \mu)$ -unsafe if the above does not hold: some strip with endpoint  $P$  and size  $\geq \gamma|st(\mathcal{B}, \mathcal{U})|$  has at least  $\mu|\mathcal{S}| + (\alpha^2/\log n)w(\mathcal{B})$  violations with  $P$ .*

Finally, we list out some auxiliary procedures used by our LIS estimation algorithm. These can be derived from some fairly routine sampling calculations. Hence, we will not describe these completely here, but merely state the relevant claims. The proofs of these claims are placed towards the end in Section 6.1, so that they do not impede the flow of the paper.

We just state one of the slight technical issues with these procedures. We often need to get a random sample from a box  $\mathcal{B}$ . Unfortunately, since the input  $\mathcal{F}$  is represented as an array, this is not possible. However, we *can* sample from  $st(\mathcal{B}, \mathcal{F})$ , since this is just an interval of the array. As long as  $|\mathcal{B}|$  is a large enough fraction of  $w(\mathcal{B})$ , we can sample randomly from  $\mathcal{B}$ . If not, then we must analyze these boxes in a different way. This leads to the following definition.

**Definition 3.2** *A box  $\mathcal{B}$  is disposable if  $|\mathcal{B}| \leq \alpha^2 w(\mathcal{B})/\log n$ .*

We now state the claims about various auxiliary procedures used. Our procedures sometime output *labels*, usually indicating some kind of corner case. These will be represented in typewriter typeface. Greek letters will always represent parameters taking values in  $(0, 1)$ . We express the running times using the big-Oh notation, and stress that this only hides absolute constants. We do *not* consider any of our parameters to be constants.

**Claim 3.3 (Size)** *There is a procedure  $\text{Size}(\mathcal{B})$  that outputs either outputs an estimate of  $|\mathcal{B}|$  or labels  $\mathcal{B}$  as disposable. The running time is  $(\log n/\alpha)^{O(1)}$  and the following hold with high probability. If an estimate is output, then it is correct upto an additive  $\alpha^3 w(\mathcal{B})/\log n$ . If the label is output, then  $\mathcal{B}$  is disposable.*

**Claim 3.4 (Sample)** *There is a procedure  $\text{Sample}(\mathcal{B}, k)$  that outputs either a set of  $k$  points in  $\mathcal{B}$  or labels  $\mathcal{B}$  as disposable. The running time is  $(k \log n / \alpha)^{O(1)}$  and the following hold with high probability. If a set is output, then each point in the set is an independent random sample from  $\mathcal{B}$ . If the label is output, then  $\mathcal{B}$  is disposable.*

**Claim 3.5 (Grid)** *There is a randomized procedure  $\text{Grid}(\mathcal{B}, r_x, r_y)$  that with high probability, either outputs  $\text{Grid}_{\mathcal{B}}(r_x, r_y)$  or labels  $\mathcal{B}$  correctly as disposable. The running time is  $(\log n / r_y)^{O(1)}$  time.*

**Claim 3.6 (Find)** *There is a procedure  $\text{Find}(\mathcal{B}, \mathcal{U}, \gamma, \mu, \rho)$  that outputs either a point  $s$ , the label **sparse**, or the label **disposable**. The running time is at most  $(\log n / (\alpha \gamma \mu \rho))^{O(1)}$  and the following hold with high probability. If the output is  $s$ , then  $s$  is  $(\mathcal{B}, \mathcal{U}, 2\gamma, \mu + \alpha)$ -safe and there are at least  $\rho|\mathcal{B}|/5$  points in  $\mathcal{B}$  both to the left and right of  $s$ . If the output is **sparse**, the number of  $(\mathcal{B}, \mathcal{U}, \gamma, \mu - \alpha)$ -safe points in  $\mathcal{B}$  is at most  $\rho|\mathcal{B}|/2$ . If the output is **disposable**, then  $\mathcal{B}$  is disposable.*

## 4 The basic algorithm

We begin by describing the weaker algorithm. The output guarantee in its most general form is both a multiplicative and additive approximation to the distance to monotonicity. We first state the main theorem of this section that gives this guarantee. We can derive the desired purely additive approximation for the LIS and purely multiplicative approximation for the distance by a simple choice of parameters.

**Theorem 4.1** *Let  $f : [n] \rightarrow \mathbb{R}$  be an array and  $\varepsilon_f$  be the distance to monotonicity. Let positive parameters  $\delta, \tau$  be at most some small constant, and  $\delta \leq \tau$ .*

*There exists a procedure that, given oracle access to  $f$ , outputs a real number  $\varepsilon$  such that with high probability  $\varepsilon_f \in [\varepsilon, (1 + \tau)\varepsilon + \delta]$ . The running time of this procedure is  $(\log n / \delta)^{O(1/\tau)}$ .*

**Corollary 4.2** *Let  $f : [n] \rightarrow \mathbb{R}$  be an array, and  $\delta, \tau$  be positive parameters bounded above by a small constant.*

- *There exists a procedure that estimates the LIS of  $f$  upto an additive error of  $\delta n$ . The running time of this procedure is  $(\log n / \delta)^{O(1/\delta)}$ .*
- *There exists a procedure that outputs a  $(1 + \tau)$ -multiplicative approximation to  $\varepsilon_f$ . This procedure has running time  $(\log n / \varepsilon_f)^{O(1/\tau)}$ .*

Let us see the choice the parameters in Theorem 4.1 that lead to this corollary. For convenience, let the parameters in Theorem 4.1 be  $\tau'$  and  $\delta'$ , so we get an output  $\varepsilon$  such that  $\varepsilon_f \in [\varepsilon, (1 + \tau')\varepsilon + \delta']$ . If we set  $\tau' = \delta' = \delta/2$ , then we get an estimate  $\varepsilon$  such that  $\varepsilon_f \leq (1 + \tau')\varepsilon + \delta' \leq \varepsilon + \delta$ . So  $n(1 - \varepsilon)$  is an additive  $\delta n$ -approximation for the LIS.

For the second part of Corollary 4.2, we need to set  $\delta'$  to be  $O(\tau\varepsilon_f)$ . Using [ACCL07], we can get  $\varepsilon$ , such that  $\varepsilon_f \in [\varepsilon, 2.1\varepsilon]$ . We set  $\tau' = \tau/2$  and  $\delta' = \tau\varepsilon/7 \leq \tau\varepsilon_f/3$ . This shows the second part. The running times follow immediately.

### 4.1 The procedures *ApproxLIS* and *Classify*

We are now ready to describe the procedure *ApproxLIS*. Since we have an iterative boosting technique, we actually have a suite of procedures *ApproxLIS<sub>t</sub>*, for positive integer  $t > 0$ . These form a sort of hierarchy of procedures. The aim of this procedure *ApproxLIS<sub>t</sub>( $\mathcal{U}$ )* is to output an estimate

for the LIS length in the box  $\mathcal{U}$ . For convenience, it also outputs an estimate to  $\varepsilon_{\mathcal{U}}$ .  $ApproxLIS_t$  indirectly uses  $ApproxLIS_{t-1}$  as a subroutine, and provides an improved approximation.

The heart of the algorithm is actually the procedure  $Classify_t(P, \mathcal{U})$ . This procedure decides whether point  $P$  should be present on the LIS in  $\mathcal{U}$  or not. Concretely, it labels the points in  $\mathcal{U}$  a point as  $good_t$  or  $bad_t$ . The  $good_t$  points form an increasing sequence and the number of  $bad_t$  points is an approximation of the distance  $\varepsilon_{\mathcal{U}}$ .

The procedure  $Classify_t(P, \mathcal{U})$  begins by calling a procedure  $Unsplittable$ . This procedure tries to run the interactive protocol described earlier. It tries to find a splitter in  $\mathcal{U}$ . Suppose it is successful and  $P$  is consistent with this splitter. Then it goes to a smaller box containing  $P$  and tries to find a splitter again. This process is repeated until either  $P$  turns out to be inconsistent with a splitter, or  $P$  is located in an *unsplittable* box. This is a box where  $Find$  fails to locate a splitter. So  $Unsplittable$  either returns a label `reject`, indicating that  $P$  is inconsistent with some splitter, or a box  $\mathcal{B} \ni P$  that contains very few (too few to sample) potential splitters.

This box  $\mathcal{B}$  is returned to  $Classify_t$ . A logarithmic-sized grid is built on  $\mathcal{B}$  and then  $ApproxLIS_{t-1}$  is invoked on all the (logarithmically many) grid boxes. The outputs of these calls are used to determine the structure of an approximate LIS. Finally, if  $P$  is potentially on the approximate LIS, a call to  $Classify_{t-1}(P, \mathcal{C})$  (for an appropriate grid box  $\mathcal{C}$ ) is made to determine whether to label  $P$  as  $good_t$ .

For the base case for  $Classify$  and  $ApproxLIS$ , we use trivial algorithms. The procedure  $ApproxLIS_0$  always outputs 0 as the estimated length of the LIS. The procedure  $Classify_0$  always labels a point as  $bad_0$ .

We observe that  $ApproxLIS_r$  (for various values  $r$ ) is invoked on many different boxes. So we estimate the LIS length on various boxes and piece the information together to estimate the overall LIS. This leads to a fairly complex recursive scheme. Hence, there will be an array of different parameters that are used by these procedures. These are intimately related to the approximation guarantees that will be finally provided. The parameters  $\zeta_t, \xi_t, \psi_t$  quantify the approximation guarantee of the various  $ApproxLIS_t$ 's. The parameter  $\alpha$  is just some sort of error parameter and is chosen depending on the final desired approximation. The reader should just think of this as something extremely small. If we invoke some  $ApproxLIS_t$ , we will always choose  $\alpha$  so that it is at most  $\zeta_t^2/30$ . The parameters  $\mu_t$  and  $\gamma$  are used by  $Find$ . The following values are the only choice of arguments that will be used by the various procedures.

- $\beta = \alpha^2 / \log n, \gamma = \alpha^2 / \log n$
- $\zeta_1 = 3.1, \zeta_t = \zeta_{t-1} - \zeta_{t-1}^2/40$
- $\xi_1 = 0, \xi_t = (1 + \zeta_t)\xi_{t-1} + 300\alpha$
- $\psi_1 = 0, \psi_t = (1 + \zeta_t)\psi_{t-1} + 20\beta$ .
- $\mu_t = \zeta_t/5$

$ApproxLIS_t(\mathcal{U})$  ( $t > 0$ )

1. Run  $Sample(\mathcal{U}, c \log^2 n / \alpha^2)$  and call  $Size(\mathcal{U})$  to get estimate  $u$ . If  $Sample$  or  $Size$  outputs `disposable`, output 1 as estimated distance and 0 for LIS estimate. Terminate procedure. Otherwise, we have a random sample  $\mathcal{R}$ .
2. Call  $Classify_t(P, \mathcal{U})$  for each  $P \in \mathcal{R}$  and let the fraction of  $bad_t$  points be  $\varepsilon$ .
3. Output  $\varepsilon$  as the estimate of  $\varepsilon_{\mathcal{U}}$  and  $(1 - \varepsilon)u$  times as the estimate of the LIS in  $\mathcal{U}$ .

$Classify_t(P, \mathcal{U})$  ( $t > 0$ )

1. Run  $Unsplittable_t(P, \mathcal{U}, \mathcal{U})$ . If this outputs **failure**, output  $bad_t$  and terminate. Otherwise, we get box  $\mathcal{B}$ .
2. If  $w(\mathcal{B}) \leq \log n / \alpha$ , find the LIS of  $\mathcal{B}$ . If  $P$  is on the LIS output  $good_t$ , otherwise output  $bad_t$ . Terminate procedure.
3. Call  $Grid(\mathcal{B}, \alpha^3 / \log n, \alpha^4 / \log n)$  to generate a grid for  $\mathcal{B}$ . For each grid box  $\mathcal{C}$ , call  $ApproxLIS_{t-1}(\mathcal{C})$  to get an estimate on the LIS length in  $\mathcal{C}$ . Let this estimate be the *length* of the  $\mathcal{C}$ .
4. Determine the longest length grid chain (by dynamic programming)  $\mathcal{C}'_1, \mathcal{C}'_2, \dots$ . If  $P$  does not belong to the longest grid chain, output  $bad_t$ . Otherwise, let  $\mathcal{C}'_j$  be the grid box containing  $P$ . Run  $Classify_{t-1}(P, \mathcal{B})$ . If  $p$  is  $good_{t-1}$ , output  $good_t$ . Otherwise, output  $bad_t$ .

$Unsplittable_t(P, \mathcal{B}, \mathcal{U})$

1. Run  $Find(\mathcal{B}, \mathcal{U}, \gamma, \mu_t, \alpha)$ . If the output is **disposable**, output **reject** and terminate procedure.
2. If  $Find(\mathcal{B}, \mathcal{U}, \gamma, \mu_t, \alpha)$  outputs point  $S$ : Call this the *splitter* for  $\mathcal{B}$ .
  - (a) If  $P$  is a violation with  $S$ , output **reject** and terminate procedure.
  - (b) Let  $\mathcal{B}$  be  $Box(B_l, B_r)$ . Set  $\mathcal{B}_l = Box(B_l, S)$  and  $\mathcal{B}_r = Box(S, B_r)$ . Let  $\mathcal{B}'$  be the box among these that contains  $P$ . (If  $P = S$ , then set  $\mathcal{B}'$  arbitrarily.)
  - (c) Recursively call  $Unsplittable_t(P, \mathcal{B}', \mathcal{U})$ .
3. If  $Find(\mathcal{B}, \mathcal{U}, \gamma, \mu_t, \alpha)$  outputs **sparse**, output  $\mathcal{B}$ .

We observe that  $Classify_t(P, \mathcal{B})$  labels a point  $P$  with respect to a box  $\mathcal{B}$ . We will use a shorthand for saying that the output of  $Classify_t(P, \mathcal{B})$  is  $good_t$ . We will just say that  $P$  is labeled  $good_t(\mathcal{B})$  (or  $bad_t(\mathcal{B})$ ). We refer to  $t$  as the *level number*, since it denotes the number of levels of recursion used.

Before we state our main claims, it is important to look at the use of randomness by these procedures. This is a somewhat subtle point and we discuss it in the next subsection. Then, we shall be ready to state our main approximation boosting lemma.

#### 4.1.1 The use of randomness

Randomness is used through the invocations of *Sample*, *Find*, and *Grid*. It will be important for us that these procedures give the same output when given the same arguments. This will ensure consistency over various calls to the procedures. Since these are randomized algorithms, independent calls to them could give different outputs. To avoid this, we associate a fixed random seed for a given procedure with a given set of arguments.

All calls to a procedure made during a fixed level  $t$  with the same arguments will use the same random seed. Calls made in *different* levels always use different random seeds. For example, there is a fixed random seed associated with  $Find(\mathcal{B}, \mathcal{U}, \gamma, \mu_t, \alpha)$ , for a fixed choice of these arguments. Every call to this procedure will use the same random seed, thereby ensuring the output is always the same. We will always assume that  $t \leq \log n$ .

**Claim 4.3** *With high probability over the choice of random seeds, no invocation to  $Find$ ,  $Grid$ ,  $Size$ , or  $Sample$  made by any call to  $ApproxLIS_t$ ,  $Classify_t$ , or  $Unsplittable_t$  will fail.*

**Proof:** This holds by a simple union bound. The total number of boxes  $\mathcal{B}$  is  $O(n^2)$ . The total number of possible parameters is  $t \leq \log nn$ . For a given setting of arguments (for one of these procedures), by Claims 3.4, 3.5, 3.6, the probability of failure is low. There are totally only a polynomial number of possible arguments. So a union bound shows that the probability that any of these procedures fails for any choice of arguments in any level is low.  $\square$

Suppose we focus on a fixed  $t$ . For every possible of call to  $Find$ ,  $Grid$ ,  $Size$ , or  $Sample$ , every level number  $\leq t$ , and every possible choice of arguments, we can write down a random seed. The total list of these random seeds is the seed for  $ApproxLIS_t$ . Note that *any* call to  $ApproxLIS_t(\mathcal{U})$  (for any set  $\mathcal{U}$ ) behaves deterministically, now that the random seed is fixed. Note that we do not really have to know the whole seed in advance, but can generate the relevant parts of it as  $ApproxLIS_t$  proceeds. The key is that the random seed is reused, whenever an auxiliary procedure is repeatedly called with the same argument. Note a major benefit of this approach. For a level  $r$  and box  $\mathcal{B}$ , the labels  $good_r(\mathcal{B})$  and  $bad_r(\mathcal{B})$  are now fixed. Since this labeling is now static, it is convenient to make arguments about these points.

**Definition 4.4** *A random seed of  $ApproxLIS_t$  is sound if the following hold. No call to any auxiliary procedure with any choice of arguments fails. For box  $\mathcal{U}$  and level  $r$ , let  $g_r(\mathcal{U})$  (resp.  $b_r(\mathcal{U})$ ) be the number of  $good_r(\mathcal{U})$  (resp.  $bad_r(\mathcal{U})$ ) points. Let the LIS estimate of  $ApproxLIS_r(\mathcal{U})$  be  $\ell(\mathcal{U})$  and the distance estimate be  $\varepsilon(\mathcal{U})$ . For all boxes  $\mathcal{U}$  and  $r \leq t$ , we have*

$$|\varepsilon(\mathcal{U})|\mathcal{U}| - b_r(\mathcal{U})| \leq \alpha|\mathcal{U}| + \beta w(\mathcal{U}) \quad |\ell(\mathcal{U}) - g_r(\mathcal{U})| \leq \alpha|\mathcal{U}| + 2\beta w(\mathcal{U})$$

**Claim 4.5** *Let  $t \leq \log n$ . With probability at most  $n^{-\Omega(\log n)}$ , a seed for  $ApproxLIS_t$  chosen uniformly at random is unsound.*

**Proof:** By Claim 4.3, we can assume that with high probability no auxiliary procedure fails. Fix a box  $\mathcal{U}$  and level  $r$ . The procedure  $Sample(\mathcal{U}, \cdot)$  invoked by  $ApproxLIS_r(\mathcal{U})$  does not fail. If  $Sample$  or  $Size$  outputs `disposable`, then  $g_r(\mathcal{U}) \leq |\mathcal{U}| \leq \alpha^2 w(\mathcal{U}) / \log n = \beta w(\mathcal{U})$ . Similarly,  $||\mathcal{U}| - b_r(\mathcal{U})| \leq \beta w(\mathcal{U})$ . So  $ApproxLIS_r$  outputs good estimates. Otherwise  $Sample$  outputs a random sample of  $M = c \log^2 n / \alpha^2$  points from  $\mathcal{U}$ . Note that the randomness used for  $Sample(\mathcal{U}, \cdot)$  is *independent* of any randomness used for the classification (by  $Classify_r$ ) of points in  $\mathcal{U}$ . Let the number of  $good_r(\mathcal{U})$  points in the sample be  $X$ . An additive Chernoff bound gives us:  $\Pr[|(X/M)|\mathcal{U}| - g_r(\mathcal{U})| > \alpha|\mathcal{U}|] < \exp(-\alpha^2 M) \leq n^{-\Omega(\log n)}$ . So, the estimated *fraction* of good points is accurate. An identical argument works for the estimated fraction of bad points. Hence, with high probability  $|\varepsilon(\mathcal{U})|\mathcal{U}| - b_r(\mathcal{U})| \leq \alpha|\mathcal{U}| + \beta w(\mathcal{U})$ .

$ApproxLIS_r(\mathcal{U})$  outputs an estimate for the LIS *length* (not the fraction), and has to estimate  $|\mathcal{U}|$ . By Claim 3.3, we get an additive  $\alpha^3 w(\mathcal{U}) / \log n$  estimate  $u$  for  $|\mathcal{U}|$  with high probability. The estimate output by  $ApproxLIS_r(\mathcal{U})$  is  $(X/M)u$ . We have  $|(X/M)u - (X/M)|\mathcal{U}|| \leq \alpha^3 w(\mathcal{U}) / \log n \leq \beta w(\mathcal{U})$ . Combining this with the previous bound, we get the desired bound on the LIS length estimate of  $ApproxLIS_r(\mathcal{U})$ . Since all of this holds with high probability, a union bound over all  $\mathcal{U}$  and  $r$  completes the proof.  $\square$

Henceforth, we will assume that a *sound random seed* is one that is sound for a sufficiently large value of  $t$  (say  $\log n$ ). This means that, using this random seed, all calls to any of our procedures will succeed. We will assume that such a sound random seed is fixed, so all procedures are deterministic.

## 4.2 Terminal and critical boxes

We now introduce the notion of *terminal* and *critical* boxes. These are very important for the analysis, and will also give us a greater understanding of how  $Classify_t$  works. Through the use of  $Unsplittable_t$ ,  $Classify_t(\cdot, \mathcal{U})$  essentially breaks  $\mathcal{U}$  into a chain of boxes and searches for an approximate LIS inside this chain. It is very important for these definitions that we have a fixed random seed. So, we reiterate, all procedures are now deterministic. In the following, we fix a box  $\mathcal{U}$  and look at calls to  $Classify(P, \mathcal{U})$ , for all  $P \in \mathcal{U}$ .

**$\mathcal{U}$ -splitters:** Consider a call to  $Unsplittable_t(P, \mathcal{U}, \mathcal{U})$ . In the course of its running, many splitters in  $\mathcal{U}$  are found. These are  $\mathcal{U}$ -splitters for level  $t$ . Each splitter is associated with the box  $\mathcal{B}$  in which it is found (so  $Unsplittable_t(P, \mathcal{B}, \mathcal{U})$  “finds” splitter  $S$ ). The complete set of splitters found over all calls (over all  $P \in \mathcal{U}$ ) is the set of  $\mathcal{U}$ -splitters for level  $t$ . For convenience, we always consider the lower left and upper right corner of  $\mathcal{U}$  to be splitters.

**Invoked and Terminal boxes:** Consider a call to  $Unsplittable_t(P, \mathcal{U}, \mathcal{U})$  made by  $Classify_t(P, \mathcal{U})$ . The recursive nature of  $Unsplittable_t$  leads to a sequence of (recursive) calls  $Unsplittable_t(P, \mathcal{B}_0, \mathcal{U})$ ,  $Unsplittable_t(P, \mathcal{B}_1, \mathcal{U})$ ,  $\dots$ ,  $Unsplittable_t(P, \mathcal{B}_k, \mathcal{U})$ , where  $\mathcal{B}_0 = \mathcal{U}$ . All of the boxes  $\mathcal{B}_i$  are called  $\mathcal{U}$ -invoked boxes for level  $t$ . Note that we have the containment  $\mathcal{B}_0 \supset \mathcal{B}_1 \supset \mathcal{B}_2 \dots$ . Every invoked box (except for the last one  $\mathcal{B}_k$ ) contains a splitter, which is found in Step 2.

Suppose the last procedure  $Unsplittable_t(P, \mathcal{B}_k, \mathcal{U})$  returns in Steps 1 or 3. This means that  $P \in \mathcal{B}_k$ , and  $\mathcal{B}_k$  cannot be “split” further. The box  $\mathcal{B}_k$  is a  $\mathcal{U}$ -terminal box for level  $t$ . The set of  $\mathcal{U}$ -terminal boxes can be determined by considering all calls to  $Unsplittable_t(P, \mathcal{U}, \mathcal{U})$ , for all  $P \in \mathcal{U}$ .

Terminal boxes are of three types: disposable, tiny, and proper. Disposable terminal boxes are those labeled so in Step 1 of  $Unsplittable_t$ . A tiny box  $\mathcal{B}$  is one for which  $w(\mathcal{B}) \leq \log n/\alpha$ . For such a box,  $Classify_t$  finds the LIS in  $\mathcal{B}$  (in Step 2). All other terminal boxes are proper. For these boxes, a grid is constructed in them and the longest length grid chain is determined (Steps 3 and 4 of  $Classify_t$ ).

A  $\mathcal{U}$ -terminal strip for level  $t$  is of the form  $st(\mathcal{B}, \mathcal{U})$ , where  $\mathcal{B}$  is a corresponding terminal box.

**Critical boxes:** Consider a call to  $Classify_t(P, \mathcal{U})$  and suppose it proceeds to Step 4. The longest grid chain  $\mathcal{C}'_1, \mathcal{C}'_2, \dots$  is then found. All these boxes are  $\mathcal{U}$ -critical for level  $t$ . Also, if a box  $\mathcal{B}$  is a tiny terminal box, then it is also  $\mathcal{U}$ -critical. Analogous to terminal strips, we can also define critical strips.

The following claims are fairly easy to prove, given the algorithm description. It is helpful to have these formally stated, so we can refer to them later. The proofs for these are given in Section 6.2.

**Claim 4.6** *For any two  $\mathcal{U}$ -invoked boxes for level  $t$   $\mathcal{B}$  and  $\mathcal{B}'$ , either  $\mathcal{B} \subseteq \mathcal{B}'$  or  $\mathcal{B}$  and  $\mathcal{B}'$  are comparable. Hence, the set of  $\mathcal{U}$ -invoked boxes for level  $t$  form a tree (called the  $\mathcal{U}$ -invoked tree for level  $t$ ) by containment, and the leaves are exactly the set of corresponding terminal boxes. The depth of this tree is at most  $10 \log n/\alpha$ .*

**Claim 4.7** *Let the  $\mathcal{U}$ -splitters for level  $t$  be  $S_0, S_1, \dots$ , in increasing order of  $x$ -coordinate. The  $\mathcal{U}$ -terminal boxes for level  $t$  are exactly boxes of the form  $Box(S_i, S_{i+1})$ . Furthermore, all these splitters are consistent, and hence the terminal boxes form a maximal chain.*

**Claim 4.8** *The  $\mathcal{U}$ -critical boxes for level  $t$  form a chain of boxes contained in the chain of  $\mathcal{U}$ -terminal boxes for level  $t$ . All  $good_t(\mathcal{U})$  points for  $\mathcal{U}$  lie inside this chain, and form an increasing sequence.*

A basic induction argument shows that the running time of  $ApproxLIS_t(\mathcal{U})$  is polylogarithmic in  $n$ .

**Claim 4.9** *For any box  $\mathcal{U}$ , the running time of  $ApproxLIS_t(\mathcal{U})$  is  $(\log n/\alpha)^{O(t)}$ .*

**Proof:** We first observe that  $ApproxLIS_t$  invokes  $Sample$  once and  $Classify_t$   $\text{poly}(\log n/\alpha)$  times. So the running time of  $ApproxLIS_t$  is at most  $(\log n/\alpha)^a$  times the running time of  $Classify_t$ , for some constant  $a$ . Let  $c$  be a constant much larger than  $a$ .

We will show that the running time of  $Classify_t$  is  $(\log n/\alpha)^{ct}$ , by induction on  $t$ . The running time bound is proven by induction over  $t$ . The running time of  $Classify_0$  is just constant. Assume the bound holds for  $Classify_t$ . Now consider  $Classify_{t+1}$ . First, there is a call to  $Unsplittable_{t+1}$ . A call to  $Unsplittable_{t+1}$  calls  $Find$  once and then potentially makes a recursive call. The sequence of recursive calls follows some path of boxes in the tree of invoked boxes. Since the depth of this tree is  $10 \log n/\alpha$  (by Claim 4.6), the running time of  $Unsplittable_{t+1}$  is at most  $10 \log n/\alpha$  times the running time of  $Find$ . By Claim 3.6, the running time for  $Unsplittable_{t+1}$  is at most  $(\log n/\alpha)^c$ . The running times for Steps 2 and 3 of  $Classify_{t+1}$  is at most  $(\log n/\alpha)^c$ .

This leaves the running time for Step 4 of  $Classify_{t+1}$ . Since  $c$  is a sufficiently large constant, this involves at most  $(\log n/\alpha)^{c/3}$  calls to  $ApproxLIS_t$  (one call for each grid box). By the induction hypothesis, each call takes time  $(\log n/\alpha)^{ct+a}$ . The total time is at most  $(\log n/\alpha)^{ct+c/2}$ .

The procedure now has to find the longest length grid chain. This is done by dynamic programming. Construct the following layered directed graph  $G$ . The vertices form layers  $L_1, L_2, \dots$ . The vertices of  $L_i$  are the grid points in the  $i$ th (leftmost) vertical line of the grid. Connect a vertex of  $L_i$  to a vertex in  $L_{i+1}$  by a directed edge if the corresponding grid points form a grid box. Note that  $G$  is a DAG, and a chain of boxes corresponds to a directed path in  $G$ . The longest length grid chain is just the longest path in  $G$  and can be found by dynamic programming. The time required is polynomial in the size of  $G$ , which is at most  $(\log n/\alpha)^c$ . Finally, there could be a call to  $Classify_t$ , which by the induction hypothesis requires at most  $(\log n/\alpha)^{ct}$  time. Summing up all these running times, the running time for  $Classify_t$  is at most  $(\log n/\alpha)^{ct+c}$ . This completes the proof.  $\square$

### 4.3 Iterative boosting

The proof of Theorem 4.1 is done essentially through an induction argument. Assuming that  $Classify_{t-1}$  has some good guarantee, we want to show that  $Classify_t$  has a better guarantee. By quantifying this improvement, we can determine what the value of  $t$  should be for some desired approximation. We will state the boosting lemma. This will first have a “base case”, expressing the guarantee of  $Classify_1$ . Then, we have a claim that is the “induction step”, showing how approximations are improved. It will actually be convenient for the analysis to express the approximation guarantee of  $Classify_{t-1}$  using  $\zeta_t$  and  $\xi_t$  (instead of the  $(t-1)$ -subscript versions).

**Lemma 4.10** *Fix a sound seed.*

- *The number of  $bad_1(\mathcal{U})$  points is at most  $(1 + \zeta_2)\varepsilon_{\mathcal{U}}|\mathcal{U}| + \xi_2|\mathcal{U}| + \psi_2 w(\mathcal{U})$ .*
- *Suppose, for any box  $\mathcal{B}$ , the number of  $bad_{t-1}(\mathcal{B})$  points is at most  $(1 + \zeta_t)\varepsilon_{\mathcal{B}}|\mathcal{B}| + \xi_t|\mathcal{B}| + \psi_t w(\mathcal{B})$  in number. Then, for any box  $\mathcal{U}$ , the number of  $bad_t(\mathcal{U})$  points is at most  $(1 + \zeta_{t+1})\varepsilon_{\mathcal{U}}|\mathcal{U}| + \xi_{t+1}|\mathcal{U}| + \psi_{t+1} w(\mathcal{U})$  in number.*

Theorem 4.1 follows fairly directly from this lemma. But we will first need the following technical claim.

**Claim 4.11** *The parameters satisfy the following bounds:  $\zeta_t \leq O(1/t)$ ,  $\xi_t \leq \alpha/t^{O(1)}$ , and  $\psi_t \leq \beta/t^{O(1)}$ .*

**Proof:** We prove, by induction on  $t$ , that  $\zeta_t \leq c/t$ , for suitably large constant  $c$ . For  $\zeta_2 = 3.1$ , this is trivially true. Suppose  $\zeta_t \leq c/t$ . We have  $\zeta_{t+1} = \zeta_t - \zeta_t^2/40$ . The function  $z - z^2/40$  is increasing for all  $0 \leq z \leq 1$ . So we have  $\zeta_{t+1} \leq (c/t)(1 - c/40t) = c(40t - c)/40t^2$ . Since  $c$  is sufficiently large,  $(40t - c)/(40t^2) \leq 1/(t+1)$ . Hence,  $\zeta_{t+1} \leq c/(t+1)$ .

We now bound  $\xi_t$ . (An identical argument works for  $\psi_t$ .) We have  $\xi_t \leq 200\alpha t \prod_{r \leq t} (1 + \zeta_r)$ . The product is bounded above by

$$\prod_{r \leq t} (1 + \zeta_r) \leq \prod_{r \leq t} (1 + c/r) \leq e^{c \log t} = t^c$$

□

**Proof:** (of Theorem 4.1) By Claim 4.5, the random seed is sound with high probability. By a simple induction argument using Lemma 4.10, for any box  $\mathcal{U}$ , the number of  $bad_t(\mathcal{U})$  points is at most  $[(1 + \zeta_{t+1})\varepsilon_{\mathcal{U}} + \xi_{t+1}]|\mathcal{U}| + c(t+1)\beta w(\mathcal{U})$ . By Claim 4.8, all  $good_t(\mathcal{U})$  points form an increasing sequence in  $\mathcal{U}$ , so the number of  $bad_t(\mathcal{U})$  points is at least  $\varepsilon_{\mathcal{U}}|\mathcal{U}|$ .

We set  $\alpha = \delta\tau^{c'}$  (for large enough constant  $c'$ ). By Claim 4.11, there is a  $t = O(1/\tau)$  such that  $\zeta_{t+1} \leq \tau$ ,  $\xi_{t+1} \leq \delta/6$ , and  $\psi_{t+1} \leq \delta^2/\log n$ . We will claim that  $ApproxLIS_t$  (almost) outputs the desired approximation. Although the fraction of  $bad_t$  points is a good (one-sided) approximation for  $\varepsilon_f$ , we remind the reader that  $ApproxLIS_t$  has to itself approximate this fraction. So, we have the account for this additional error.

Consider  $ApproxLIS_t(\mathcal{F})$ , where  $\mathcal{F}$  is the box containing all points of the function  $f$ . Note that  $w(\mathcal{F}) = |\mathcal{F}| = n$ . Let  $\varepsilon'$  be the output distance estimate and  $b$  be the number of  $bad_t(\mathcal{F})$  points. By the choice of  $i$ ,  $b \leq (1 + \tau)\varepsilon_f n + \delta n/3$ . Since the seed is sound, by Definition 4.4,  $|\varepsilon' - b/n| \leq \alpha + \beta < \delta/6$ . We have  $b/n \in [\varepsilon_f, (1 + \tau)\varepsilon_f + \delta/3]$ . Therefore,  $\varepsilon_f \leq b/n \leq \varepsilon' + \delta/6$ . More importantly,  $\varepsilon_f \geq (1 + \tau)^{-1}(b/n - \delta/6) \geq (1 + \tau)^{-1}(\varepsilon' - \delta/2)$ . Set  $\varepsilon = (1 + \tau)^{-1}(\varepsilon' - \delta/2)$ , so we have  $\varepsilon_f \in [\varepsilon, (1 + \tau)\varepsilon + \delta]$ . □

## 5 Analysis

Our ultimate aim is to prove Lemma 4.10. This will require a significant amount for preparation. We will fix a box  $\mathcal{U}$  and a level  $t$ .

### 5.1 Grid chains and the main dichotomy

In this subsection, we prove a key tool that shows the dichotomy between finding a splitter and approximation boosting. As we mentioned earlier in Section 2, we want to show that if a splitter cannot be found, then we can deduce that LIS must be small. Step 4 in  $Classify_t$  is where this will be exploited. Consider the situation when this step is executed. We have found a box  $\mathcal{B}$  where the fraction of potential splitters is extremely small. We break  $\mathcal{B}$  into grid boxes, and use  $ApproxLIS_{t-1}$  to estimate the LIS length in each grid box. Why should the longest chain give a *better* approximation than  $ApproxLIS_{t-1}(\mathcal{B})$ ? Indeed, if we knew nothing about the grid, this would not be true.

The key observation is that when a box  $\mathcal{B}$  contains almost no splitters, *no* grid chain in the  $\mathcal{B}$  can contain too many points. This immediately implies that the LIS cannot contain too many points, since the LIS must be contained in some grid chain. Hence, this grid “certifies” that the LIS

is small. The longest path chain uses  $ApproxLIS_{t-1}$  to estimate the LIS in that chain, but because we run  $ApproxLIS_{t-1}$  on a provably smaller instance, we can boost the approximation.

We state and prove the main lemma that shows that the existence of few splitters in  $\mathcal{B}$  implies that grid chains cannot be large.

**Definition 5.1** *A grid chain in  $\mathcal{B}$  is a chain of grid boxes in  $Grid_{\mathcal{B}}$ . Let  $\mathcal{G}$  be a maximal grid chain in box  $\mathcal{B}$ . Consider  $p \in \overline{\mathcal{G}} := st(\mathcal{B}, \mathcal{U}) \setminus \mathcal{G}$ . If  $p$  is a violation with  $q \in \mathcal{P}$  such that  $p_x < q_x$ , then  $p$  is above  $\mathcal{G}$ . If  $p_x > q_x$ , then  $p$  is below  $\mathcal{G}$ .*

The following lemma shows that any grid chain with many points must contain many safe points. By Claim 3.6, if many safe points exists,  $Find$  will find one of them that can be used as a splitter. The proof of this lemma is identical to the charging argument used in Lemma 2.3 of [ACCL07].

**Lemma 5.2** *In  $Grid_{\mathcal{B}}$ , let  $\mathcal{G}$  be a maximal chain of boxes with size at least  $(1 - \mu + 3\alpha)|st(\mathcal{B}, \mathcal{U})|$ . Set  $\gamma = \alpha^2 / \log n$ . Then there exists at least  $\alpha|st(\mathcal{B}, \mathcal{U})|$   $(\mathcal{B}, \gamma, 1 - \mu + \alpha)$ -safe points in  $\mathcal{B}$ .*

*Therefore, if  $Find(\mathcal{B}, \mathcal{U}, \gamma, \mu_t, \alpha)$  outputs **sparse**, then no chain in  $\mathcal{B}$  has size more than  $(1 - \mu + 3\alpha)|st(\mathcal{B}, \mathcal{U})|$ ,*

**Proof:** We partition the points in  $st(\mathcal{B}, \mathcal{U}) \setminus \mathcal{G}$  into two sets,  $\overline{\mathcal{G}}_a$  and  $\overline{\mathcal{G}}_b$ . The former (resp. latter) contains the points in  $st(\mathcal{B}, \mathcal{U})$  above (resp. below)  $\mathcal{G}$ . Abusing notation, we will let  $[P, Q]$  denote the  $\mathcal{U}$ -strip formed with  $P$  and  $Q$  as endpoints.

We bound the total number of right-unsafe points in  $\mathcal{B}$  by  $[(1 - (\mu - 2\alpha))/(\mu - 2\alpha)]|\overline{\mathcal{G}}_b|$ . The proof is analogous for left-unsafe points. We charge points in  $\overline{\mathcal{G}}_b$  with a credit scheme. First assign one unit of credit for each right-unsafe point. We will process all the right-unsafe points in reverse order (rightmost to leftmost). Processing a point will involve moving its credit to some points in  $\overline{\mathcal{G}}_b$ . Finally, at the end of the processing, we will show an upper bound on the credit that each point in  $\overline{\mathcal{G}}_b$  possesses. This will give us our desired bound.

Here is how we process  $P$ . Consider the left end  $P$  and let  $Q_P$  be the point such that the strip  $[P, Q_P]$  contains more than a  $(\mu - \alpha)$ -fraction of violations (with  $P$ ). This strip is larger than  $\gamma|st(\mathcal{B}, \mathcal{U})| = \alpha^2|st(\mathcal{B}, \mathcal{U})| / \log n$ . The only points of  $\mathcal{G}$  that  $P$  can be in violation with are in the chain box containing  $P$ . By the spacing between the vertical lines in  $Grid_{\mathcal{B}}$ , these are at most  $\alpha^3|st(\mathcal{B}, \mathcal{U})| / \log n$ . That is at most an  $\alpha$ -fraction of the strip  $[P, Q_P]$ . This means that the strip  $[P, Q_P]$  contains at least a  $(\mu - 2\alpha)$ -fraction of violations, all of which are in  $\overline{\mathcal{G}}_b$ . We will call these  $\overline{\mathcal{G}}_b$ -violations. Spread one credit among all these violations (with  $P$ ) in  $[P, Q_P]$ . We use the word “spread” because we do not simply drop one unit of credit into one account. Let the set of  $\overline{\mathcal{G}}_b$ -violations (with  $P$ ) in this strip be  $\mathcal{V}$ . We keep adding credit (infinitesimally) to the points in  $\mathcal{V}$  that have the minimum amount of charge. So, if the charge on some point in  $\mathcal{V}$  crosses  $x$ , then *all* points in  $\mathcal{V}$  have charge at least  $x$ .

We now show that no point in  $\overline{\mathcal{G}}_b$  ever receives more than  $u = [1 - (\mu - 2\alpha)]/(\mu - 2\alpha)$  units of credit. Suppose by contradiction that this were the case. Let this happen just after processing right-unsafe  $P$ . Some point  $Q$  ends up with more than  $x$  units of credit. This point  $Q$  must be a  $\overline{\mathcal{G}}_b$ -violation with  $P$ . By the charging process, all  $\overline{\mathcal{G}}_b$ -violations with  $P$  in  $[P, Q_P]$  must have more than  $u$  units of credit. There are at least  $(\mu - 2\alpha)[P, Q_P]$   $\overline{\mathcal{G}}_b$ -violations. The total amount of credit on these violations is at least  $u(\mu - 2\alpha)[P, Q_P]$ . On the other hand, the *only* credit contribution to these points comes from chain boxes in  $[P, Q_P]$ . The total number of them is at most  $[1 - (\mu - 2\alpha)][P, Q_P]$ . So

$$[1 - (\mu - 2\alpha)][P, Q_P] > u(\mu - 2\alpha)[P, Q_P] \implies u < [1 - (\mu - 2\alpha)]/(\mu - 2\alpha)$$

That contradicts the choice of  $u$ . Applying an analogous argument for left-unsafe points, the total number of unsafe points is at most  $\{[1 - (\mu - 2\alpha)]/(\mu - 2\alpha)\}|\overline{\mathcal{G}}_a|$ .

Since  $|\mathcal{G}| \geq (1 - \mu + 3\alpha)|st(\mathcal{B}, \mathcal{U})|$ ,  $|\overline{\mathcal{G}}| < (\mu - 3\alpha)|st(\mathcal{B}, \mathcal{U})|$ . The number of safe points in  $\mathcal{B}$  (actually in  $\mathcal{G}$ ), as a fraction of  $|st(\mathcal{B}, \mathcal{U})|$  is at least

$$(1 - \mu + 3\alpha) - \frac{(\mu - 3\alpha)(1 - (\mu - 2\alpha))}{\mu - 2\alpha} > \alpha$$

□

## 5.2 Bounding the number of bad points: the proof of Lemma 4.10

At long last, we come to main technical portion, the proof of Lemma 4.10. We need to bound the number of points labeled  $bad_t$  by  $Classify_t(P, \mathcal{U})$ . For this purpose, it is convenient to talk in terms of *losses*. Fix some LIS  $\mathcal{L}$  (in case there are many) in  $\mathcal{U}$ . The number of  $bad_t$  points is denoted by  $Loss_A$ , and the number of points *not* on  $\mathcal{L}$  is  $Loss_L$ . Of course,  $Loss_L$  is exactly  $\varepsilon_{\mathcal{U}}|\mathcal{U}|$ . The number of bad points in the  $\mathcal{U}$ -strip  $\mathcal{S}$  is denoted by  $Loss_A(\mathcal{S})$ . Similarly,  $Loss_L(\mathcal{S})$  is the number of points in  $\mathcal{S}$  not on  $\mathcal{L}$ .

Lemma 4.10 has two cases,  $t = 1$  and  $t > 1$ . We will prove both these cases here. The initial set up of the proof is the same for both cases. The final charging argument has different calculations. We will take care of these cases in separate subsections. Even though it is technically possible to provide a single unified proof, this would be at the cost of lot of extra tedious calculations. The authors believe that  $t = 1$  case is a nice warm-up to understand how  $Loss_A$  can be related to  $Loss_L$ .

For  $t > 1$ , we will assume the basic premise of Lemma 4.10. So, for any box  $\mathcal{B}$ , the number of  $bad_{t-1}(\mathcal{B})$  points is at most  $[(1 + \zeta_t)\varepsilon_{\mathcal{B}} + \xi_t]|\mathcal{B}| + \psi_t w(\mathcal{B})$ . Throughout this subsection, we focus on the behavior of  $Classify_t$  on  $\mathcal{U}$ . For convenience, when we refer to just *good* or *bad*, we mean  $good_t(\mathcal{U})$  or  $bad_t(\mathcal{U})$ . Any other labels (such as  $good_{t-1}(\mathcal{B})$ ) will be explicitly stated. When we use the term *strip*, we always means a  $\mathcal{U}$ -strip. For the various parameters  $\zeta_t, \xi_t, \psi_t, \mu_t, \gamma$ , we will drop the subscript  $t$ .

The proof of Lemma 4.10 will require a careful partition of the points in  $\mathcal{L}$  into various sets. By getting an upper bound on these sets, we can lower bound  $Loss_L$ . Let the ordered (left to right) list of splitters generated by  $Classify_t$  be  $S_0, S_1, \dots$ . Note that the  $\mathcal{U}$ -strip generated by  $S_i$  and  $S_{i+1}$  form a terminal strip, called  $\mathcal{T}_i$ . The terminal box corresponding to these points is called  $\mathcal{B}_i$ . The box in which  $S_i$  is found is  $\mathcal{I}_i$ . We set  $\mathcal{L}_i = \mathcal{L} \cap \mathcal{T}_i$ , the portion of the LIS in  $\mathcal{T}_i$ .

We have a bound that does a strip-by-strip comparison, which is then summed up to prove the above. In each strip  $\mathcal{T}_i$ ,  $\mathcal{L}_i$  can be partitioned into three sets. First, we take all points in  $\mathcal{L}_i$  that are *below*  $S_i$ . Note that these are all in violation with  $S_i$ . This is the set  $\mathcal{L}_i^-$ . Then, we have points  $\mathcal{L}_i \cap \mathcal{B}_i$ , called  $\mathcal{L}_i^0$ . These points are all consistent with both  $S_i$  and  $S_{i+1}$ . Finally, we have the set of points  $\mathcal{L}_i^+$  above  $S_{i+1}$ . (Refer to Figure 5.)

To deal with  $\mathcal{L}_i^-$  and  $\mathcal{L}_i^+$ , we have to introduce to concept of the *destroyed strip*. For a splitter  $S$ , consider the points of  $\mathcal{L}$  that violate it and are in a terminal strip ending at  $S$ . Note that all these points are either to left or to the right of  $S$ . Among these violations, let  $F$  be the point that is the farthest (by  $x$ -coordinate). Let the  $\mathcal{U}$ -strip formed by  $F$  and  $S$  as endpoints be called the *destroyed strip* and denoted by  $\mathcal{D}(S)$ . This will be called a *left* or *right* strip depending on whether it is of the form  $F$  is to the left or right of  $S$ . So  $\mathcal{D}(S_i)$  is a left strip iff  $\mathcal{D}(S_i) \subseteq \mathcal{T}_{i-1}$ , and  $\mathcal{D}(S_i)$  is a right strip iff  $\mathcal{D}(S_i) \subseteq \mathcal{T}_i$ . We also set  $\mathcal{D}(\mathcal{T}_i) = (\mathcal{D}(S_i) \cup \mathcal{D}(S_{i+1})) \cap \mathcal{T}_i$ . Observe that since destroyed strips are contained in terminal strips, two left (or right) destroyed strips cannot intersect. But we can also say a little more, giving us an important property.

**Observation 5.3** *If  $\mathcal{D}(S_i)$  is a left (resp. right) strip, then  $\mathcal{L}_i^-$  (resp.  $\mathcal{L}_{i-1}^+$ ) is empty.*

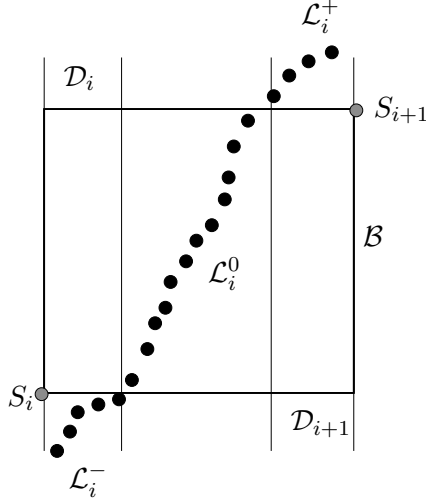


Figure 5: This shows the partition of  $\mathcal{L}_i$  into the three sets  $\mathcal{L}_i^-$ ,  $\mathcal{L}_i^0$ , and  $\mathcal{L}_i^+$ . The destroyed strips are also indicated.

*The strips  $\mathcal{D}(S_i)$  and  $\mathcal{D}(S_{i+1})$  are disjoint. Hence, all destroyed strips are disjoint.*

**Proof:** If  $\mathcal{D}(S_i)$  is a left (resp. right) strip, all violations to  $S_i$  are in  $\mathcal{T}_{i-1}$  (resp.  $\mathcal{T}_i$ ). So  $\mathcal{L}_i^-$  (resp.  $\mathcal{L}_{i-1}^+$ ) must be empty.

If  $\mathcal{D}(S_i)$  and  $\mathcal{D}(S_{i+1})$  intersect, then the former must be a right strip, and the latter must be a left strip. Some point must be between  $S_i$  and  $S_{i+1}$  and a violation with both of them. But then  $S_i$  and  $S_{i+1}$  would form a violation. Contradiction.  $\square$

We are now ready to bound the lengths  $\mathcal{L}_i^-$  and  $\mathcal{L}_i^+$  in terms of the destroyed strips.

**Claim 5.4**  $|\mathcal{L}_i^- \cup \mathcal{L}_i^+| \leq (\mu + \alpha)|\mathcal{D}(\mathcal{T}_i)| + 2\gamma|\text{st}(\mathcal{T}_i)| + 2\gamma|\text{st}(\mathcal{T}_{i+1})|$

**Proof:** We first show that  $|\mathcal{L}_i^-| \leq (\mu + \alpha)|\mathcal{D}(S_i) \cap \mathcal{T}_i| + \gamma|\text{st}(\mathcal{T}_i)|$ . If  $\mathcal{D}(S_i) \cap \mathcal{T}_i = \emptyset$ , then  $\mathcal{L}_i^- = \emptyset$ , so this trivially holds. If not, then  $\mathcal{L}_i^-$  is completely contained in  $\mathcal{D}(S_i)$ . Since  $S_i$  is a splitter, it must be  $(\mathcal{T}_i, \mathcal{U}, 2\gamma, \mu + \alpha)$ -safe. By definition, all points of  $\mathcal{L}_i^-$  are violations with  $S_i$ . If  $|\mathcal{D}(S_i)| \geq 2\gamma|\text{st}(\mathcal{T}_i)|$ , then the number of violations with in  $S$  in  $\mathcal{D}(S_i)$  is at most  $(\mu + \alpha)|\mathcal{D}(S_i)|$ . Hence, we get an upper bound of  $(\mu + \alpha)|\mathcal{D}(S_i) \cap \mathcal{T}_i| + \gamma|\text{st}(\mathcal{T}_i)|$ .

Similarly, we can prove that  $|\mathcal{L}_i^+| \leq (\mu + \alpha)|\mathcal{D}(S_{i+1}) \cap \mathcal{T}_i| + \gamma|\text{st}(\mathcal{T}_{i+1})|$ . Adding these bounds, we complete the proof.  $\square$

We now deal with  $\mathcal{L}_i^0$ , which is really the interesting part. The next claim shows a fairly simple calculation showing for  $\mathcal{L}_i^0$  can be “rounded” in the grid on  $\mathcal{B}_i$ . We get a grid chain that almost contains  $\mathcal{L}_i^0$  completely. Now, the dichotomy of Section 5.1 enters the picture. Since we are dealing with (proper) terminal boxes, we can argue that this grid chain is small compared to  $\mathcal{B}_i$ . So this shows that  $\mathcal{L}_i^0$  cannot be very large, and must exclude quite a few points of  $\mathcal{B}_i$ . This sets the stage for the approximation boosting.

**Claim 5.5** *Let  $\mathcal{B}_i$  be a proper terminal box and consider the grid on  $\mathcal{B}_i$ . There exists a maximal grid chain  $\mathcal{G}_i$  disjoint from  $\mathcal{D}(\mathcal{T}_i)$  such that the number of points of  $\mathcal{L}_i^0$  outside  $\mathcal{G}_i$  is at most  $(2r_x + 2r_y/r_x)|\mathcal{T}_i|$ . Furthermore,  $|\mathcal{G}_i| \leq (1 - \mu + 3\alpha)|\mathcal{T}_i|$ .*

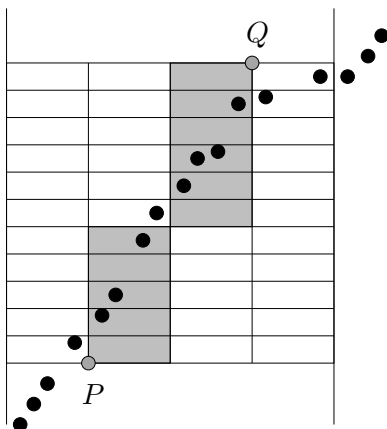


Figure 6: This shows the rounding procedure of Claim 5.5. The LIS points are colored in black. The points  $P$  and  $Q$  are shown, and the grid chain is the collection of grey boxes.

**Proof:** We will now perform a *rounding* of  $\mathcal{L}_i^0$  in the grid of  $\mathcal{B}_i$ . Refer to Figure 6 for a pictorial representation of what follows. The points of  $\mathcal{L}_i^0$  form some increasing sequence in  $\mathcal{B}_i$ . Consider the leftmost vertical grid line that has some point of  $\mathcal{L}_i^0$  to its left. Let the lowest grid point on this line be  $P$ . Similarly, consider the rightmost vertical grid line that has some point of  $\mathcal{L}_i^0$  to its right. The highest grid point on this line is  $Q$ . By the vertical distance between grid lines, at most  $2r_x|\mathcal{T}_i|$  points of  $\mathcal{L}_i^0$  are not in  $\text{Box}(P, Q)$ . All points of  $\mathcal{L}$  to the right of  $P$  are consistent with  $P$ . So  $\mathcal{D}(S_i)$  cannot contain  $P$ . Similarly,  $\mathcal{D}(S_{i+1})$  does not contain  $Q$ . The box  $\text{Box}(P, Q)$  is disjoint to  $\mathcal{D}(\mathcal{T}_i)$ .

The grid chain  $\mathcal{G}_i$  will be contained completely in  $\text{Box}(P, Q)$ . Consider the vertical grid lines  $v_1, v_2, \dots$  that contained in  $\text{Box}(P, Q)$ . If there are grid points on  $v_j$  that is consistent with  $\mathcal{L}_i^0$ , denote one of them to be  $P_j$ . Otherwise, let the highest grid point on  $v_j$  lower than  $\mathcal{L}_i^0$  be  $P_j$ . The maximal grid chain created by the sequence of boxes  $\text{Box}(P, P_1), \text{Box}(P_1, P_2), \dots, \text{Box}(P_k, Q)$  is the desired  $\mathcal{G}$ .

The points of  $\mathcal{L}_i^0$  violated by  $P_j$  are exactly those that are above and to the left of  $P_j$ . The grid point just above  $P_j$  (on  $v_j$ ) is above  $\mathcal{L}_j^0$ . So, all these points of  $\mathcal{L}_i^0$  lie between two horizontal grid lines. The total number of these is the horizontal space  $r_y|\mathcal{B}_i|$ . The total number of points of  $\mathcal{L}_i^0$  not in  $\mathcal{G}_i$  is at most  $r_y|\mathcal{T}_i|$  multiplied by the total number of vertical lines,  $2/r_x$ . This completes the proof of the first part.

The second part is very easy to prove, but is an extremely important piece in the whole analysis. So, we recommend the reader to pause to digest the import of this bound. Since  $\mathcal{B}_i$  is a proper terminal box, we know that  $\text{Find}(\mathcal{B}_i, \mathcal{U}, \gamma, \mu, \alpha)$  output **sparse**. By Lemma 5.2, no chain in the grid of  $\mathcal{B}_i$  can have size more than  $(1 - \mu + 3\alpha)|\mathcal{T}_i|$ .  $\square$

To give cleaner formulae, we define the *error term* for a terminal strip.

**Definition 5.6**  $\Delta_i := 2\gamma|\text{st}(\mathcal{I}_i)| + 2\gamma|\text{st}(\mathcal{I}_{i+1})| + (\xi + 5\alpha)|\mathcal{T}_i| + (\psi + 4\beta)w(\mathcal{T}_i)$

We will prove that in each strip,  $\text{Loss}_A(\mathcal{T}_i)$  is comparable to  $\text{Loss}_L(\mathcal{T}_i)$ , upto an additive  $\Delta_i$ . Hence, this is the “error term” associated with a strip. Eventually, we will have to add up all these errors. This is a very small term, as shown in the next claim.

**Claim 5.7**  $\sum_i \Delta_i \leq (\xi + 45\alpha)|\mathcal{U}| + (\psi + 4\beta)w(\mathcal{U})$ .

**Proof:** Since the strips  $\mathcal{T}_i$  are disjoint,  $\sum_i |\mathcal{T}_i| \leq |\mathcal{U}|$  and  $\sum_i w(\mathcal{T}_i) \leq w(\mathcal{U})$ . Dealing with  $\sum_i \gamma |st(\mathcal{T}_i)|$  is the main non-trivial part. Let us focus on the number of strips  $st(\mathcal{T}_i)$  that contain a fixed point. Note that  $\mathcal{T}_i$  is an invoked box. By Claim 4.6, the maximum number of invoked boxes that contain a fixed point (which is the depth of the tree) is at most  $10 \log n / \alpha$ .

$$\sum_i \gamma |st(\mathcal{T}_i)| \leq \gamma (10 \log n / \alpha) |\mathcal{U}| \leq (\alpha^2 / \log n) (10 \log n / \alpha) |\mathcal{U}| \leq 10\alpha |\mathcal{U}|$$

Summing all these terms, we get the desired bound.  $\square$

By doing a strip-by-strip analysis, we can show that the  $Loss_A(\mathcal{T}_i)$  is comparable to  $Loss_L(\mathcal{T}_i)$ , upto the error term  $\Delta_i$ . This bound is summed over all terminal strips  $\mathcal{T}_i$  to yield the final bound. We first state this lemma, the proof of which is really the crux of the approximation boosting. Using this, Lemma 4.10 will follow quite easily.

**Lemma 5.8** *For  $t = 1$ ,  $Loss_A(\mathcal{T}_i) \leq 4.1(Loss_L(\mathcal{T}_i) + \Delta_i)$ . For  $t > 1$ ,  $Loss_A(\mathcal{T}_i) \leq (1 + \zeta - \zeta^2/40)(Loss_L(\mathcal{T}_i) + \Delta_i)$ .*

This yields the following claim.

**Claim 5.9** *Let  $\Gamma < 5$ . If for all  $\mathcal{T}_i$ ,  $Loss_A(\mathcal{T}_i) \leq \Gamma(Loss_L(\mathcal{T}_i) + \Delta_i)$ , then*

$$Loss_A \leq \Gamma Loss_L + (\Gamma \xi + 300\alpha) |\mathcal{U}| + (\Gamma \psi + 20\beta) w(\mathcal{U})$$

**Proof:** We sum the bound in Lemma 5.8 over all terminal strips  $\mathcal{T}_i$ . We note that  $Loss_A = \sum_i Loss_A(\mathcal{T}_i)$ . This is because the strips  $\mathcal{T}_i$  are disjoint and the union  $\bigcup_i \mathcal{T}_i$  only excludes the splitters (which do not contribute to  $Loss_A$ ). We trivially have  $Loss_L \geq \sum_i Loss_L(\mathcal{T}_i)$ . So we get

$$Loss_A \leq \Gamma(Loss_L + \xi |\mathcal{U}| + \psi |w(\mathcal{U})|) + 120\alpha |\mathcal{U}| + 20\beta w(\mathcal{U})$$

$\square$

Combining Lemma 5.8 with Claim 5.9, Lemma 4.10 follows immediately. For  $t = 1$ , the value of  $\Gamma$  for Claim 5.9 is 4.1. For  $t > 1$ ,  $\Gamma = 1 + \zeta_t - \zeta_t^2/40$ .

It now remains to prove Lemma 5.8. The first step in this is to deal with dead and tiny terminal boxes separately. We can show a much stronger bound for these boxes.

**Claim 5.10** *For a dead or tiny terminal box  $\mathcal{B}_i$ ,  $Loss_A(\mathcal{T}_i) \leq (1 - \mu - \alpha)^{-1}(Loss_L(\mathcal{T}_i) + \Delta_i)$ .*

**Proof:** Suppose  $\mathcal{B}_i$  is a dead box. Then  $|\mathcal{L}_i^0| \leq |\mathcal{B}_i| \leq \beta w(\mathcal{T}_i)$ . Since  $|\mathcal{L}_i| = |\mathcal{L}_i^0| + |\mathcal{L}_i^- \cup \mathcal{L}_i^+|$ , we can bound  $|\mathcal{L}_i|$  by adding  $\beta w(\mathcal{T}_i)$  to the upper bound of Claim 5.4. So  $|\mathcal{L}_i| \leq (\mu + \alpha) |\mathcal{D}(\mathcal{T}_i)| + \Delta_i$ . We trivially bound  $Loss_A(\mathcal{T}_i) \leq |\mathcal{T}_i|$  and  $|\mathcal{D}(\mathcal{T}_i)| \leq |\mathcal{T}_i|$ . We have  $Loss_L(\mathcal{T}_i) = |\mathcal{T}_i| - |\mathcal{L}_i| \geq (1 - \mu - \alpha) |\mathcal{T}_i| - \Delta_i$ . This implies  $Loss_A(\mathcal{T}_i) \leq (Loss_L(\mathcal{T}_i) + \Delta_i) / (1 - \mu - \alpha)$ .

Now let  $\mathcal{B}_i$  be a tiny box. We have  $|\mathcal{L}_i^- \cup \mathcal{L}_i^+| \leq (\mu + \alpha) |\mathcal{D}(\mathcal{T}_i)| + \Delta_i$ . We bound  $Loss_L(\mathcal{T}_i) \geq (|\mathcal{T}_i| - |\mathcal{L}_i^0|) - (\mu + \alpha) |\mathcal{D}(\mathcal{T}_i)| - \Delta_i$ . Since  $\mathcal{L}_i^0$  is disjoint to  $\mathcal{D}(\mathcal{T}_i)$ , we have  $|\mathcal{D}(\mathcal{T}_i)| \leq (|\mathcal{T}_i| - |\mathcal{L}_i^0|)$ . Hence,  $Loss_L(\mathcal{T}_i) + \Delta_i \geq (1 - \mu - \alpha) (|\mathcal{T}_i| - |\mathcal{L}_i^0|)$ . The number of  $good_t$  points is exactly the size of the LIS in  $\mathcal{B}_i$ . Obviously,  $|\mathcal{L}_i^0|$  can be no larger than this. So  $Loss_A(\mathcal{T}_i) \leq (|\mathcal{T}_i| - |\mathcal{L}_i^0|)$ . That completes the proof.  $\square$

Now, we deal with proper boxes, where all the interesting action occurs. We split into two cases, depending on the value of  $t$ . We break this part up into separate subsections.

### 5.2.1 The case $t = 1$

The procedure *Classify*<sub>1</sub> performs a rather trivial labeling in a proper terminal box  $\mathcal{B}_i$ . The procedure *ApproxLIS*<sub>0</sub> always returns 0 as the estimated LIS length. In any grid box  $\mathcal{C}$ , *Classify*<sub>0</sub> simply labels *all* points as *bad*<sub>0</sub>. This means that *Classify*<sub>1</sub> is simply going to label all points in  $\mathcal{B}_i$  as *bad*. So why should we get any approximation? The key is Claim 5.5. We show that the LIS  $\mathcal{L}$  must exclude a constant of  $\mathcal{B}_i$ . So our trivial labeling of all points as *bad* is actually a constant factor approximation to the distance. We now prove this formally.

**Lemma 5.11** *Consider a proper terminal strip  $\mathcal{T}_i$ .  $Loss_A(\mathcal{T}_i) \leq 4.1(Loss_L(\mathcal{T}_i) + \Delta_i)$ .*

**Proof:** We trivially bound  $Loss_A(\mathcal{T}_i) \leq |\mathcal{T}_i|$ . We now give an upper bound for  $|\mathcal{L}_i|$ . We break up  $|\mathcal{L}_i|$  into  $|\mathcal{L}_i^0| + |\mathcal{L}_i^- \cup \mathcal{L}_i^+|$ . The second term is bounded by Claim 5.4. For the first term, we use Claim 5.5. We can trivially bound  $|\mathcal{L}_i^0 \cap \mathcal{G}_i|$  by  $|\mathcal{G}_i|$ . The number of remaining points of  $\mathcal{L}_i^0$  is at most  $(2r_x + 2r_y/r_x)|\mathcal{T}_i| \leq 4\alpha|\mathcal{T}_i|$ . Adding all the bounds, we get

$$\begin{aligned} |\mathcal{L}_i| &\leq |\mathcal{G}_i| + 4\alpha|\mathcal{T}_i| + (\mu + \alpha)|\mathcal{D}(\mathcal{T}_i)| + \gamma|st(\mathcal{T}_i)| + \gamma|st(\mathcal{T}_{i+1})| \\ &\leq |\mathcal{G}_i| + (\mu + \alpha)|\mathcal{D}(\mathcal{T}_i)| + \Delta_i \end{aligned}$$

Hence,  $Loss_L(\mathcal{T}_i) + \Delta_i \geq |\mathcal{T}_i| - |\mathcal{G}_i| - (\mu + \alpha)|\mathcal{D}(\mathcal{T}_i)|$ . Note that  $\mathcal{G}_i$  and  $\mathcal{D}(\mathcal{T}_i)$  are disjoint, so  $|\mathcal{G}_i| \leq |\mathcal{T}_i| - |\mathcal{D}(\mathcal{T}_i)|$ . By Claim 5.5,  $|\mathcal{G}_i| \leq (1 - \mu + 3\alpha)|\mathcal{T}_i|$ . Set  $\mu' = \mu + \alpha$  and  $\bar{\mu} = \mu - 3\alpha$ . We get  $|\mathcal{G}_i| \geq |\mathcal{T}_i| - \max(\bar{\mu}|\mathcal{T}_i|, |\mathcal{D}(\mathcal{T}_i)|)$ .

$$\begin{aligned} Loss_L(\mathcal{T}_i) + \Delta_i &\geq \max(\bar{\mu}|\mathcal{T}_i|, |\mathcal{D}(\mathcal{T}_i)|) - \mu'|\mathcal{D}(\mathcal{T}_i)| \\ &\geq (1 - \mu') \max(\bar{\mu}|\mathcal{T}_i|, |\mathcal{D}(\mathcal{T}_i)|) \geq \bar{\mu}(1 - \mu')|\mathcal{T}_i| \end{aligned}$$

Since  $\mu = 1/2$  (for  $t = 1$ ) and  $\alpha$  is much smaller than  $1/100$ , so  $Loss_L(\mathcal{T}_i) + \Delta_i \geq 0.245|\mathcal{T}_i| \geq 0.245Loss_A(\mathcal{T}_i)$ .  $\square$

Observing that  $(1 - \mu - \alpha)^{-1} \leq 4.1$  (for  $\mu = 1/2$  and sufficiently small  $\alpha$ ), we can combine Claim 5.10 and Lemma 5.11 into the following corollary. This proves the first part of Lemma 5.8.

**Corollary 5.12** *For any terminal strip  $\mathcal{T}_i$ ,  $Loss_A(\mathcal{T}_i) \leq 4.1(Loss_L(\mathcal{T}_i) + \Delta_i)$*

### 5.2.2 The general case

The general case (naturally) needs a more involved argument than the  $t = 1$  setting. We now use our induction hypothesis that says the number of *bad* <sub>$t-1$</sub> ( $\mathcal{C}$ ) points is at most  $(1 + \zeta)\varepsilon_{\mathcal{C}}|\mathcal{C}| + \xi|\mathcal{C}| + \psi w(\mathcal{C})$ . By the careful choice of parameters, we can ensure that  $Loss_A(\mathcal{T}_i)$  is “significantly” better than a  $(1 + \zeta)$ -approximation of  $Loss_L(\mathcal{T}_i)$ . This allows for the overall improved approximation of *ApproxLIS*.

We need to set some notation that will aid us in our analysis. We fix some proper terminal strip  $\mathcal{T}_i$ .

- $\mathcal{G}_i$ : This is chain of boxes obtained from Claim 5.5.
- $g$ : Any point in  $\mathcal{G}_i$  is inside some box  $\mathcal{C} \in \mathcal{G}_i$ . The total number of all such points that are labeled *good* <sub>$t-1$</sub> ( $\mathcal{C}$ ) (for any grid box  $\mathcal{C}$  in  $\mathcal{G}_i$ ) is  $g$ .
- $b$ . This is just  $|\mathcal{G}_i| - g$ , the total number of *bad* <sub>$t-1$</sub>  points in the grid chain  $\mathcal{G}_i$ .
- $d = |\mathcal{D}(\mathcal{T}_i)|$  and  $s = |\mathcal{T}_i|$

- $\zeta' = \zeta/(1 + \zeta)$ ,  $\mu' = \mu + \alpha$  and  $\bar{\mu} = \mu - 3\alpha$
- $\Delta'_i = (\mu + \alpha)|\mathcal{D}(\mathcal{T}_i)| + 2\gamma|st(\mathcal{T}_i)| + 2\gamma|st(\mathcal{T}_{i+1})| + (3\alpha + \xi)|\mathcal{T}_i| + \psi w(\mathcal{T}_i)$

**Lemma 5.13** Consider a proper terminal strip  $\mathcal{T}_i$ . Let the total number of point labeled  $good_t(\mathcal{U})$  in  $\mathcal{T}_i$  be  $g$ , and the number of  $bad_t(\mathcal{U})$  points be  $b$ . Then

$$|\mathcal{L}_i| \leq g + \zeta b/(1 + \zeta) + (\mu + \alpha)d + \Delta'_i$$

**Proof:** To bound  $|\mathcal{L}_i|$ , we break it up into  $|\mathcal{L}_i^0 \cap \mathcal{G}_i| + |\mathcal{L}_i^0 \cap \overline{\mathcal{G}}_i| + |\mathcal{L}_i^- \cup \mathcal{L}_i^+|$ . The second term is bounded by Claim 5.5 and last term by Claim 5.4. We now deal with the first term.

Let the boxes of the chain  $\mathcal{G}_i$  be  $\mathcal{C}_1, \mathcal{C}_2, \dots$ . For each grid box  $\mathcal{C}_j$ , let  $g_j$  (resp.  $b_j$ ) be the number of  $good_{t-1}$  (resp.  $bad_{t-1}$ ) points. The properties of  $Classify_{t-1}$  imply that  $b_j \leq (1 + \zeta)\varepsilon_{\mathcal{C}_j}|\mathcal{C}_j| + \xi|\mathcal{C}_j| + \psi w(\mathcal{C}_j)$ . Therefore,

$$\varepsilon_{\mathcal{C}_j}|\mathcal{C}_j| \geq \frac{b_j - \xi|\mathcal{C}_j| - \psi w(\mathcal{C}_j)}{1 + \zeta} \geq \frac{b_j}{1 + \zeta} - \xi|\mathcal{C}_j| - \psi w(\mathcal{C}_j)$$

The number of points of  $\mathcal{L}_i^0$  in  $\mathcal{C}_j$  is at most the LIS length inside  $\mathcal{C}_j$ , which is  $|\mathcal{C}_j| - \varepsilon_{\mathcal{C}_j}|\mathcal{C}_j| = g_j + b_j - \varepsilon_{\mathcal{C}_j}|\mathcal{C}_j|$ . Combining these together, we get that the  $\mathcal{L}_i^0 \cap \mathcal{C}_j$  is at most  $g_j + \zeta' b_j + \xi|\mathcal{C}_j| + \psi w(\mathcal{C}_j)$  (where  $\zeta' = \zeta/(1 + \zeta)$ ). By summing the above,  $|\mathcal{L}_i^0 \cap \mathcal{G}_i|$  is at most  $g + \zeta' b + \xi|\mathcal{T}_i| + \psi w(\mathcal{T}_i)$ .

We can now bound  $|\mathcal{L}_i|$ . Observing that  $r_x \leq \alpha^2/\log n$  and  $r_y/r_x \leq \alpha$ ,

$$\begin{aligned} |\mathcal{L}_i| &= |\mathcal{L}_i^0 \cap \mathcal{G}_i| + |\mathcal{L}_i^0 \cap \overline{\mathcal{G}}_i| + |\mathcal{L}_i^- \cup \mathcal{L}_i^+| \\ &\leq g + \zeta' b + \xi|\mathcal{T}_i| + \psi w(\mathcal{T}_i) + (2r_x + 2r_y/r_x)|\mathcal{T}_i| + (\mu + \alpha)|\mathcal{D}(\mathcal{T}_i)| + 2\gamma|st(\mathcal{T}_i)| + 2\gamma|st(\mathcal{T}_{i+1})| \\ &\leq g + \zeta' b + (\mu + \alpha)|\mathcal{D}(\mathcal{T}_i)| + 2\gamma|st(\mathcal{T}_i)| + 2\gamma|st(\mathcal{T}_{i+1})| + (3\alpha + \xi)|\mathcal{T}_i| + \psi w(\mathcal{T}_i) \end{aligned}$$

□

**Claim 5.14** The number of points in  $\mathcal{T}_i$  labeled  $good_t(\mathcal{U})$  is at least  $g - 2\alpha|\mathcal{T}_i| - 4\beta w(\mathcal{T}_i)$ .

**Proof:** Focus on calls to  $Classify_t$  that terminate in  $\mathcal{B}_i$ . In Step 3 of  $Classify_t$ , the length of each grid box  $\mathcal{C}$  is set as  $ApproxLIS_{t-1}(\mathcal{C})$ . Denote this by  $\ell(\mathcal{C})$  and let  $g_{t-1}(\mathcal{C})$  be the number of  $good_{t-1}(\mathcal{C})$  points. By the soundness of the random seed and Definition 4.4,  $|\ell(\mathcal{C}) - g_{t-1}(\mathcal{C})| \leq \alpha|\mathcal{C}| + 2\beta w(\mathcal{C})$ . Consider any grid chain  $\mathcal{G}$ . Let  $\ell(\mathcal{G})$  be the length of this chain, and  $g_{t-1}(\mathcal{G}) = \sum_{\mathcal{C} \in \mathcal{G}} g_{t-1}(\mathcal{C})$ . Since  $\mathcal{G} \subseteq \mathcal{B}_i$ ,  $|\ell(\mathcal{G}) - g_{t-1}(\mathcal{G})| \leq \alpha|\mathcal{T}_i| + 2\beta w(\mathcal{T}_i)$ . Let  $\mathcal{G}^L$  denote the longest grid chain found in Step 4. Note that  $g_{t-1}(\mathcal{G}^L)$  is exactly the number of points labeled  $good_t(\mathcal{U})$  in  $\mathcal{B}_i$ , and  $g_{t-1}(\mathcal{G}_i)$  is  $g$ . We have  $\ell(\mathcal{G}^L) \geq \ell(\mathcal{G}_i)$ . Hence  $g_{t-1}(\mathcal{G}^L) \geq g - 2\alpha|\mathcal{T}_i| - 4\beta w(\mathcal{T}_i)$ . □

**Lemma 5.15** For a proper terminal strip  $\mathcal{B}_i$ ,  $Loss_A(\mathcal{T}_i) \leq (1 + \zeta - \zeta^2/40)(Loss_L(\mathcal{T}_i) + \Delta_i)$

**Proof:** By Lemma 5.13 and Claim 5.14, we can bound the losses.

$$\begin{aligned} |\mathcal{L}_i| &\leq g + \zeta b/(1 + \zeta) + (\mu + \alpha)d + \Delta'_i \\ \implies Loss_L(\mathcal{T}_i) + \Delta'_i &\geq (s - g) - \mu'd - \zeta'b \\ Loss_A(\mathcal{T}_i) &\leq (s - g) + 2\alpha s + 4\beta w(\mathcal{T}_i) \end{aligned}$$

By Claim 5.5, the size of chain  $\mathcal{G}_i$  is at most  $(1 - \mu + 3\alpha)s$ . The size of the chain  $\mathcal{G}_i$  (given by Claim 5.5) is exactly  $g + b$ . Trivially, we can also bound  $g + b$  by  $s - d$  (the size of the complement

of the destroyed strip in  $\mathcal{T}_i$ ). This is because  $\mathcal{G}_i$  is contained in  $\mathcal{B}_i$ , which is disjoint to the destroyed strips. Hence

$$\begin{aligned} g + b &\leq \min(s - d, (1 - \mu + 3\alpha)s) = s - \max(d, \bar{\mu}s) \\ &\implies b \leq (s - g) - \max(d, \bar{\mu}s) \end{aligned}$$

We now use this to bound  $Loss_A(\mathcal{T}_i)$ .

$$\begin{aligned} Loss_L(\mathcal{T}_i) + \Delta_i &\geq (s - g) - \mu'd - \zeta'b \\ &\geq (s - g) - \mu'd - \zeta'[(s - g) - \max(d, \bar{\mu}s)] \\ &= (1 - \zeta')(s - g) - \mu'd + \zeta' \max(d, \bar{\mu}s) \end{aligned}$$

A little thought reveals that the worst case is when  $d = \bar{\mu}s$ . We also use the fact that  $\zeta' - \mu' \geq 0$ . This is because  $\zeta \leq 3.1$ ,  $\zeta' = \zeta/(1 + \zeta)$  and  $\mu = \zeta/5$ . More formally,

$$\begin{aligned} Loss_L(\mathcal{T}_i) + \Delta'_i &\geq (1 - \zeta')(s - g) - \mu'd + \zeta' \max(d, \bar{\mu}s) \\ &\geq (1 - \zeta')(s - g) - \mu' \max(d, \bar{\mu}s) + \zeta' \max(d, \bar{\mu}s) \\ &= (1 - \zeta')(s - g) + (\zeta' - \mu') \max(d, \bar{\mu}s) \\ &\geq (1 - \zeta')(s - g) + (\zeta' - \mu')\bar{\mu}s \end{aligned}$$

Combining this with the simple upper bound of  $Loss_A(\mathcal{T}_i) - 2\alpha s - 4\beta w(\mathcal{T}_i) \leq s - g$ ,

$$\frac{Loss_A(\mathcal{T}_i) - 2\alpha s - 4\beta w(\mathcal{T}_i)}{Loss_L(\mathcal{T}_i) + \Delta'_i} \leq \frac{s - g}{(1 - \zeta')(s - g) + s(\zeta'\bar{\mu} - \mu'\bar{\mu})} \leq \frac{1}{1 - \zeta' + \zeta'\bar{\mu} - \mu'\bar{\mu}} = \frac{1 + \zeta}{1 + \zeta\bar{\mu} - (1 + \zeta)\mu'\bar{\mu}}$$

We have  $\mu = \zeta/5$  and  $\alpha \leq \zeta^2/30$ . So  $\bar{\mu} = \mu - 3\alpha > \zeta/6$  and  $\mu' = \mu + \alpha < \zeta/3$  (for  $\zeta < 3.1$ ). (For large values of  $\zeta$ , we can set much better parameters, so these are mainly chosen for small  $\zeta$ .) We get

$$1 + \zeta\bar{\mu} - (1 + \zeta)\mu'\bar{\mu} = 1 + \bar{\mu}(\zeta - (1 + \zeta)\mu') \geq 1 + (\zeta/6)(\zeta/3) = 1 + \zeta^2/20$$

So  $Loss_A(\mathcal{T}_i) \leq (1 + \zeta - \zeta^2/40)(Loss_L(\mathcal{T}_i) + \Delta'_i) + 2\alpha s + 4\beta w(\mathcal{T}_i)$ . Since  $\Delta'_i + 2\alpha s + 4\beta w(\mathcal{T}_i) \leq \Delta_i$ , we get the desired bound.  $\square$

We just combine the two above claims into the following corollary, which completes the proof of Lemma 5.8.

**Corollary 5.16** *For any terminal strip  $\mathcal{T}_i$ ,  $Loss_A(\mathcal{T}_i) \leq (1 + \zeta - \zeta^2/40)(Loss_L(\mathcal{T}_i) + \Delta_i)$*

## 6 Extra proofs

### 6.1 Proofs for auxiliary procedures

For convenience, we first state the Chernoff bounds that we use.

**Theorem 6.1** (*Hoeffding-Chernoff*) *Let  $X = X_1 + X_2 + \dots + X_r$  be the sum of independent Bernoulli random variables. Then  $\Pr[|X - \mathbf{E}[X]| > \Delta] \leq 2 \exp(-2\Delta^2/r)$ .*

**Theorem 6.2** (*Multiplicative Chernoff*) Let  $X = X_1 + X_2 + \dots + X_r$  be the sum of i.i.d Bernoulli random variables. Then for  $\sigma \in (0, 1)$ ,  $\Pr[X < (1 - \sigma)\mathbf{E}[X]] \leq \exp(-\sigma^2\mathbf{E}[X]/2)$  and  $\Pr[X > (1 + \sigma)\mathbf{E}[X]] \leq \exp(-\sigma^2\mathbf{E}[X]/3)$

We will need a slightly more general version of Claim 3.4.

**Claim 6.3** *There is a procedure  $\text{Sample}(\mathcal{B}, k, \sigma)$  that outputs either a set of  $k$  points in  $\mathcal{B}$  with a size estimate for  $\mathcal{B}$  or labels  $\mathcal{B}$  as  $\sigma$ -disposable. The running time is  $2k \log^2 n / \sigma^2$  and the following hold with high probability. If a set is output, then each point in the set is an independent random sample from  $\mathcal{B}$ . The size estimate differs from  $|\mathcal{B}|$  by at most  $\sigma |st(\mathcal{B}, \mathcal{U})|$ . If the label is output, then  $|\mathcal{B}| < \sigma |st(\mathcal{B}, \mathcal{U})|$ .*

**Proof:** We choose  $m = 2k \log^2 n / \sigma^2$  uniform random samples from  $st(\mathcal{B}, \mathcal{F})$ . For each of these, we check if it is contained in  $\mathcal{B}$ . Note that because  $\mathcal{F}$  is represented as an array, and  $\mathcal{B}$  is stored by its corner points, this can be done easily. Let  $X$  be the number of points that fall in  $\mathcal{B}$ . If  $X \geq k$ , we output the first  $k$  of these points as the sample set. We set the size estimate to be  $(X/m)|st(\mathcal{B}, \mathcal{F})|$ . If  $X < k$ , we output **disposable**. Note that when a set of points is output, it is certainly an independent uniform sample of  $k$  points. It remains to show that with high probability, when  $\mathcal{B} > \sigma |st(\mathcal{B}, \mathcal{U})|$ , this procedure outputs a set and a correct size estimate.

Suppose  $\mathcal{B} > \sigma |st(\mathcal{B}, \mathcal{U})|$ . We have  $\mathbf{E}[X] = (|\mathcal{B}|/|st(\mathcal{B}, \mathcal{U})|)m \geq 2k \log^2 n / \sigma$ . The additive Chernoff bound of Theorem 6.1 gives  $\Pr[|X - \mathbf{E}[X]| < \sigma m / 2] \leq 2 \exp(-\sigma^2 m^2 / (4m)) \leq n^{-\Omega(\log n)}$ . A simple calculation yields

$$|X - \mathbf{E}[X]| < \sigma m \iff |X - |\mathcal{B}|m/|st(\mathcal{B}, \mathcal{U})|| < \sigma m \iff |(X/m)|st(\mathcal{B}, \mathcal{F})| - |\mathcal{B}| < \sigma |st(\mathcal{B}, \mathcal{U})|$$

This shows the size estimate is accurate. We have  $\sigma m / 2 = k \log^2 n / \sigma \leq \mathbf{E}[X] / 2$ . Therefore, we get at least  $k \log^2 n / \sigma$  samples in  $\mathcal{B}$ .  $\square$

Claims 3.4 and 3.3 follow directly from Claim 6.3, by setting  $\sigma = \alpha^2 / \log n$  and  $\sigma = \alpha^3 / \log n$ , respectively. We now prove that grids can be constructed quickly.

**Proof:** (of Claim 3.5) Suppose  $\mathcal{B}$  is not disposable. By Claim 3.4, we can get  $c \log^2 n / r_x^2$  independent uniform random samples in  $\mathcal{B}$  in  $2k \log^6 n / r_x^2$  time. Let their sorted list of  $x$ -coordinates be  $x_1, x_2, \dots$ . Take  $2/r_x$  equally spaced out coordinates. (These are the  $x$ 's corresponding to indices  $ic \log^2 n / (2r_x)$ , for all  $i$ .) The vertical lines of the grid will correspond to these  $x$ -coordinates. For a fixed pair of points in the sample, consider the  $\mathcal{B}$ -strip generated by them. Let the number of points in this strip be  $L$ . Let  $M = c \log^2 n / r_x^2 - 2$  be the number of sample points other than this pair. Let  $X$  be the random variable denoting the number of these sample points that fall in this strip (note that  $\mathbf{E}[X] = ML/|\mathcal{B}|$ ). We have  $\Pr[|X - \mathbf{E}[X]| > c \log^2 n / (4r_x)] < 2 \exp(-2c^2 \log^4 n / (16r_x^2 M)) \leq n^{-\Omega(\log n)}$ . The total number of such pairs is  $(\log n)^{O(1)}$ . Taking a union bound over this error probability all such pairs, we get that for *every* strip, the number of points in the strip deviates from the expectation by at most  $c \log^2 n / (4r_x)$ . Now, consider such an adjacent pair of vertical lines. For this strip, we have  $X = c \log n / (2r_x)$  by construction. Hence,  $ML/|\mathcal{B}| = \mathbf{E}[X] \leq 3c \log^2 n / (4r_x)$ , implying  $L \leq r_x |\mathcal{B}|$ . This gives the desired properties of the vertical lines.

Through Claim 3.4, we can generate a random subset of  $c \log^2 n / (r_y^2)$  samples in  $\mathcal{B}$ . Take the sorted list of these  $y$ -coordinates, we can choose  $2/r_y$  horizontal lines such that the number of  $\mathcal{B}$ -points between adjacent horizontal lines is at most  $r_y |\mathcal{B}|$ . The proof is exactly the same as above. A union bound over all the error probabilities completes the proof.  $\square$

Before proving Claim 3.6, we need a procedure that decides the safeness of points. The following is an adaptation of procedures used in [ACCL08].

**Claim 6.4** *Let  $\mathcal{B}$  be a box that is not disposable. There is a procedure  $\text{Safe}(P, \mathcal{B}, \mathcal{U}, \gamma, \mu)$  that takes as input  $P \in \mathcal{B}$  and outputs safe or unsafe. The running time of this procedure is  $c \log^2 n / (\alpha^3 \mu)$  and the following hold with high probability. If the output is safe, then the point  $P$  is  $(\mathcal{B}, \mathcal{U}, 2\gamma, \mu + \alpha)$ -safe. If the output is unsafe, then  $P$  is  $(\mathcal{B}, \mathcal{U}, \gamma, \mu - \alpha)$ -unsafe.*

**Proof:** We first use Claim 3.4 to get a random sample of size  $k = c \log^2 n / (\alpha \gamma)^2$  in  $\mathcal{B}$ . We perform the following for every point  $Q$  in the sample such that: the  $\mathcal{U}$ -strip  $\mathcal{S}$  with  $P$  and  $Q$  as endpoints contains at least  $3\gamma k / 2$  points. Invoke  $\text{Sample}(\mathcal{S}, c \log^2 n / \alpha^2, \gamma \alpha^2 / \log n)$ . If no sample is found, terminate the procedure and output arbitrarily. If a sample is found, we check the fraction of violations with  $P$ . If, for any  $\mathcal{S}$ , this fraction is more than  $\mu$ , we label the point unsafe. Otherwise, we label the point safe. We will refer to the strips  $\mathcal{S}$  as those *found* by our procedure.

Let us first prove some preliminary facts. Consider some  $\mathcal{B}$ -strip  $\mathcal{T}$  that has size at least  $\alpha \gamma |\mathcal{B}| / 10$ . Let the random variable denoting the number of points of the sample in  $\mathcal{B}$  falling in  $\mathcal{S}$  be  $X_{\mathcal{T}}$ . By the additive Chernoff bound,  $\Pr[|X_{\mathcal{T}} - \mathbf{E}[X_{\mathcal{T}}]| > k / (20\alpha\gamma)] \leq n^{-\Omega(\log n)}$ . Taking a union bound over all  $\mathcal{B}$ -strips (at most polynomially many), for every strip  $\mathcal{T}$ ,  $X_{\mathcal{T}}$  deviates from its expectation by at most  $c \log^2 n / (20\alpha\gamma)$ . For any strip  $\mathcal{S}$  found by our procedure, this implies that  $|\mathcal{S}| > \gamma |\mathcal{B}|$ . Henceforth, we assume this to hold.

Suppose point  $P$  is  $(\mathcal{B}, \mathcal{U}, 2\gamma, \mu + \alpha)$ -unsafe. Then, there is some  $\mathcal{B}$ -strip  $\mathcal{T}$  ending at  $P$  such that  $|\mathcal{T}| \geq 2\gamma |st(\mathcal{B}, \mathcal{U})|$  and  $\mathcal{T}$  has at least a  $(\mu + \alpha)$ -fraction of violations. By the above, there is some point  $Q \in \mathcal{T}$  in the sample of  $\mathcal{B}$  such that the  $\mathcal{U}$ -strip  $\mathcal{T}'$  formed by  $P, Q$  has size at least  $|\mathcal{T}|(1 - \alpha/10)$ . Since  $\mathcal{T}$  contains at least  $(\mu + \alpha)|\mathcal{T}|$  violations (with  $P$ ),  $\mathcal{T}'$  contains at least a  $(\mu + \alpha/2)$ -fraction of violations. We have the size bound,

$$|\mathcal{T}'| \geq |\mathcal{T}|(1 - \alpha/10) \geq \gamma |st(\mathcal{B}, \mathcal{U})| \geq \gamma \alpha^2 w(\mathcal{B}) / \log n \geq \gamma \alpha^2 w(\mathcal{T}') / \log n$$

By Claim 6.3, the invocation to  $\text{Sample}(\mathcal{T}', c \log^2 n / \alpha^2, \gamma \alpha^2 / \log n)$  will return a random sample of size  $k' = c \log^2 n / \alpha^2$ . Let  $X$  be the random variable denoting the number of violations with  $P$  in this sample. We have  $\mathbf{E}[X] > (\mu + \alpha/2)k'$ . By the multiplicative Chernoff bound,  $\Pr[X < (1 - \alpha/4)\mathbf{E}[X]] \leq \exp(-\alpha^2 \mathbf{E}[X] / 32)$ . Since  $\alpha < \mu$ , the probability that  $X \leq \mu k'$  is at most  $n^{-\Omega(\log n)}$ . Hence, we will find at least a  $\mu$ -fraction of violations in this strip, and declare the point unsafe with high probability.

Suppose the point  $P$  is  $(\mathcal{B}, \gamma, \mu - \alpha)$ -safe. Consider a strip  $\mathcal{S}$  found by our procedure. Since the size of this strip is at least  $\gamma |\mathcal{B}|$ , the fraction of violations is at most  $(\mu - \alpha)$ . A random sample of size  $k'$  is chosen in this strip. As before, let  $X$  be the random variable denoting the number of violations. Note that  $\mathbf{E}[X] < (\mu - \alpha)k'$  and we have  $\Pr[X > (1 + \alpha/4)\mathbf{E}[X]] \leq \exp(-\alpha^2 \mathbf{E}[X] / 64)$ . Hence,  $X \geq \mu k'$  with low probability. A union bound over all strips that are found shows that the point is declared safe with high probability. We take a union bound over all the error probabilities to complete the proof.  $\square$

**Proof:** (of Claim 3.6) Suppose  $\mathcal{B}$  is not disposable. We will show that with high probability  $\text{Find}$  will output either a point or the label **sparse**.

By Claim 3.4, we can get a random sample of  $k = c \log^2 n / \rho^2$  points from  $\mathcal{B}$ . For each point  $P$  in the sample, we run  $\text{Safe}(P, \mathcal{B}, \mathcal{U}, \gamma, \mu)$ . If at least a  $9\rho/20$ -fraction of points are deemed safe, output the middle (by  $x$ -coordinate) safe point. Otherwise, output **sparse**. By a union bound, the calls to  $\text{Sample}$  and  $\text{Safe}$  succeed with high probability. So, let us assume that no failure occurred.

For any point  $P \in \mathcal{B}$ , let  $X_P$  denote the random variable that is the number of sample points to fall to the left of  $P$  (including  $P$ ). An additive Chernoff bound shows that  $\Pr[|X_P - \mathbf{E}[X_P]| >$

$9\rho k/80] < 2 \exp(-2(9\rho/80)^2 k)$ . Taking a union bound over all points in  $\mathcal{B}$ , we conclude: with high probability, if, for any  $P \in \mathcal{B}$ ,  $X_P > 9\rho k/40$ , then  $\mathbf{E}[X_P] > \rho k/5$ . Hence the fraction of points to the left of  $P$  is at least  $\rho/5$ . A similar claim holds for points to the right of  $P$ .

Suppose a point  $P$  is output. By Claim 6.4, this point is  $(\mathcal{B}, \mathcal{U}, 2\gamma, \mu + \alpha)$ -safe. There are at least a  $9\rho/40$ -fraction of sample points (from  $\mathcal{B}$ ) both to the left and right of  $p$ . By the argument above, there must be at least  $\rho|\mathcal{B}|/5$  points in  $\mathcal{B}$  both to the left and right of  $P$ .

Suppose there are more than a  $(\rho/2)$ -fraction of  $(\mathcal{B}, \mathcal{U}, \gamma, \mu - \alpha)$ -safe points. Let  $Y$  be the random variable denoting the number of  $(\mathcal{B}, \mathcal{U}, \gamma, \mu - \alpha)$ -safe points in the sample. We have  $\mathbf{E}[Y] \geq \rho k/2$  and  $\Pr[|Y - \mathbf{E}[Y]| > \rho k/20] < 2 \exp(-(\rho/20)^2 k)$ . With high probability, at least a  $9\rho/20$ -fraction of the random sample is  $(\mathcal{B}, \mathcal{U}, \gamma, \mu - \alpha)$ -safe. By Claim 6.4, all of these points are deemed safe. Hence, **sparse** will not be output.  $\square$

## 6.2 Proofs about terminal boxes

**Proof:** (of Claim 4.6) Consider two invoked boxes  $\mathcal{B}$  and  $\mathcal{B}'$  such that neither is a subset of the other. There exists point  $P$  such that  $Unsplittable(P, \mathcal{U}, \mathcal{U})$  eventually calls  $Unsplittable(P, \mathcal{B}, \mathcal{U})$  (similarly, we have point  $P'$ ). Let us look at the recursion paths for both  $Unsplittable(P, \mathcal{U}, \mathcal{U})$  and  $Unsplittable(P', \mathcal{U}, \mathcal{U})$ . Let us set  $\mathcal{B}_0 = \mathcal{B}'_0 = \mathcal{F}$ . We have a the sequence of calls  $Unsplittable(P, \mathcal{B}_0, \mathcal{U})$ ,  $Unsplittable(P, \mathcal{B}_1, \mathcal{U})$ ,  $Unsplittable(P, \mathcal{B}_2, \mathcal{U})$ ,  $\dots$ ,  $Unsplittable(P, \mathcal{B}, \mathcal{U})$  made from  $Unsplittable(P, \mathcal{U}, \mathcal{U})$ . Similarly, we have the call sequence  $Unsplittable(P', \mathcal{B}'_0, \mathcal{U})$ ,  $Unsplittable(P', \mathcal{B}'_1, \mathcal{U})$ ,  $Unsplittable(P', \mathcal{B}'_2, \mathcal{U})$ ,  $\dots$ ,  $Unsplittable(P', \mathcal{B}', \mathcal{U})$ . Since neither  $\mathcal{B}$  or  $\mathcal{B}'$  is a subset of the other, we can find the smallest index  $j$  where  $\mathcal{B}_j \neq \mathcal{B}'_j$ . Therefore,  $\mathcal{B}_{j-1} = \mathcal{B}'_{j-1}$ . Both calls  $Unsplittable(P, \mathcal{B}_{j-1}, \mathcal{U})$  and  $Unsplittable(P', \mathcal{B}'_{j-1}, \mathcal{U})$  invoke procedure *Find* on the same arguments. Since we have a fixed random seed associated with this, for both calls to *Unsplittable* find same splitter  $S$ . The boxes  $\mathcal{B}_j$  and  $\mathcal{B}'_j$  are the two different boxes created in Step 2b and are therefore comparable. Since  $\mathcal{B} \subseteq \mathcal{B}_j$  and  $\mathcal{B}' \subseteq \mathcal{B}'_j$ ,  $\mathcal{B}$  and  $\mathcal{B}'$  are comparable.

Consider a leaf box  $\mathcal{B}$ . This means there is recursion path of calls  $Unsplittable(P, \mathcal{B}_0, \mathcal{U})$ ,  $Unsplittable(P, \mathcal{B}_1, \mathcal{U})$ ,  $\dots$ ,  $Unsplittable(P, \mathcal{B}_k, \mathcal{U})$ ,  $Unsplittable(P, \mathcal{B}, \mathcal{U})$  made from  $Unsplittable(P, \mathcal{U}, \mathcal{U})$ . Note that  $P \in \mathcal{B}_i$ , for all  $i$ . Suppose a splitter  $S$  was found in  $\mathcal{B}$ . This splits  $\mathcal{B}$  into two boxes  $\mathcal{B}_l$  and  $\mathcal{B}_r$ . At least one of the these boxes must be non-empty (otherwise, the splitter  $S$  would not be consistent with *any* point in  $\mathcal{B}$ ). Let  $P'$  (in, say,  $\mathcal{B}_l$ ) be a point in one of these boxes. The call to  $Unsplittable(P', \mathcal{U}, \mathcal{U})$  will follow the same sequence of calls as above and *then* proceed to  $Unsplittable(P', \mathcal{B}_l, \mathcal{U})$ . But this contradicts that fact that  $\mathcal{B}$  is a leaf box. Therefore, a splitter is not found in  $\mathcal{B}$ . The call  $Unsplittable(P, \mathcal{B}, \mathcal{U})$  terminates in either Step 1 or 3. So this box is terminal.

Consider a box  $\mathcal{B}' \subset \mathcal{B}$  that is a child of  $\mathcal{B}$  in this tree. A splitter is found in  $\mathcal{B}$ , and  $\mathcal{B}'$  is either  $\mathcal{B}_l$  or  $\mathcal{B}_r$  in Step 2b. By the properties of *Find* (Claim 3.6), there are at least  $\alpha|\mathcal{B}|/5$  points in  $\mathcal{B}$  to the left (and right) of  $S$ . Hence,  $|\mathcal{B}_l| \leq (1 - \alpha/5)|\mathcal{B}|$  (similarly for  $\mathcal{B}_r$ ). This means that as we go down a path in the tree, the sizes decrease by a multiplicative factor of at least  $(1 - \alpha/5)$ . So the depth can be at most  $10 \log n/\alpha$ .  $\square$

**Proof:** (of Claim 4.7) Consider two splitters  $S_i$  and  $S_j$  which we found in boxes  $\mathcal{B}_i$  and  $\mathcal{B}_j$ . By Claim 4.6, either (wlog)  $\mathcal{B}_i \subseteq \mathcal{B}_j$ , or  $\mathcal{B}_i$  and  $\mathcal{B}_j$  are comparable. In the former case,  $\mathcal{B}_i$  is comparable to the splitter  $S_j$ , so  $S_i$  is comparable to  $S_j$ . In the latter case, the splitters are trivially comparable (since they are contained in the respective boxes).

It is quite easy to see that all invoked (and hence terminal) boxes are formed by splitters at their opposite corners. Consider the box  $Box(S_i, S_{i+1})$ . Suppose that  $S_i$  is found in  $\mathcal{B}_i$  and  $S_{i+1}$

is found in  $\mathcal{B}_{i+1}$ . Note that  $\mathcal{B}_i$  cannot be comparable to  $\mathcal{B}_{i+1}$  (that would imply the existence of a splitter between  $S_i$  and  $S_{i+1}$ ). So let us assume that  $\mathcal{B}_i \subset \mathcal{B}_{i+1}$ . So in the tree of boxes,  $\mathcal{B}_i$  is a descendant of  $\mathcal{B}_{i+1}$ . Hence, in a call to  $Unsplittable_t(P, \mathcal{U}, \mathcal{U})$  that finds  $S_i$ , the splitter  $S_{i+1}$  is found first. Now, the box  $\mathcal{B}_i$  contains splitter  $S_i$  and so the top right corner of  $\mathcal{B}_i$  must be greater than  $S_i$ . So it must be  $S_{i+1}$ . The box  $\mathcal{B}_i$  has  $S_{i+1}$  at its top right corner, and  $S_i$  is found in this box. By the properties of the splitter (Claim 3.6), there are at least  $\alpha|\mathcal{B}_i|/5$  points to the right of  $S_i$  and in  $\mathcal{B}_i$ . At least one of these points (actually at least a  $(1 - \mu)$ -fraction) is consistent with  $S_i$ . Take such a point  $P'$  and consider the call to  $Unsplittable_t(P', \mathcal{U}, \mathcal{U})$ . This call must eventually end up at  $Unsplittable_t(P', \text{Box}(S_i, S_{i+1}), \mathcal{U})$ . Since a splitter cannot be found in this box, this is a terminal box.

A box of the form  $\text{Box}(S_i, S_j)$ , for  $j > i + 1$  cannot be terminal, since it contains a splitter. This completes the proof.  $\square$

**Proof:** (of Claim 4.8) The first part is easy to see. Critical boxes that are not tiny terminal boxes are found in Step 4 of *Classify<sub>t</sub>*. A chain of critical boxes, which is the longest grid chain, is found in the relevant terminal box. So the  $\mathcal{U}$ -critical boxes form a chain contained in the chain of  $\mathcal{U}$ -terminal boxes.

All  $good_t$  points are present in critical boxes, since points labeled  $good_t$  are either in tiny boxes or in the longest grid chain found in Step 4. We prove that these form an increasing sequence by induction on  $t$ . There are no  $good_0$  points, so this is vacuously true for  $t = 0$ . Suppose this is true for  $t = r$ . Let us look at  $good_{r+1}(\mathcal{U})$  within a given terminal box  $\mathcal{B}$ . If  $\mathcal{B}$  is tiny, then the  $good_{r+1}(\mathcal{U})$  points here form the LIS. If not, then the  $good_{r+1}(\mathcal{U})$  points are in the longest grid chain of boxes  $\mathcal{C}'_1, \mathcal{C}'_2, \dots$  for  $\mathcal{B}$ . The set of  $good_{r+1}$  points is exactly the union of  $good_r(\mathcal{C}'_i)$  points (for each  $i$ ). By the induction hypothesis, in each  $\mathcal{C}'_i$ , the set of  $good_r(\mathcal{C}'_i)$  points form an increasing sequence. Since the boxes  $\mathcal{C}'_1, \mathcal{C}'_2, \dots$  form a chain, all the  $good_{r+1}(\mathcal{U})$  points inside  $\mathcal{B}$  form an increasing sequence. Finally, since all the terminal boxes form a chain, all  $good_{r+1}(\mathcal{U})$  points form an increasing sequence.  $\square$

## 7 Acknowledgements

The second author would like to thank Robi Krauthgamer and David Woodruff for useful discussions.

## References

- [ACCL07] N. Ailon, B. Chazelle, S. Comandur, and D. Liu. Estimating the distance to a monotone function. *Random Structures and Algorithms*, 31(3):371–383, 2007.
- [ACCL08] N. Ailon, B. Chazelle, S. Comandur, and D. Liu. Property-preserving data reconstruction. *Algorithmica*, 51(2):160–182, 2008.
- [AD99] D. Aldous and P. Diaconis. Longest increasing subsequences: from patience sorting to the Baik-Deift-Johannson theorem. *Bulletin of the American Mathematical Society*, 36:413–432, 1999.
- [AIK09] A. Andoni, P. Indyk, and R. Krauthgamer. Overcoming the  $\ell_1$  non-embeddability barrier: algorithms for product matrices. In *Proceedings of the 20th Symposium on Discrete Algorithms (SODA)*, pages 865–874, 2009.

- [AK07] A. Andoni and R. Krauthgamer. The computational hardness of estimating edit distance. In *Proceedings of the 48th Symposium on Foundations of Computer Science (FOCS)*, pages 724–734, 2007.
- [AK08] A. Andoni and R. Krauthgamer. The smoothed complexity of edit distance. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP)*, pages 357–369, 2008.
- [AN10] A. Andoni and H. L. Nguyen. Near-optimal sublinear time algorithms for ulam distance. In *Proceedings of the 21st Symposium on Discrete Algorithms (SODA)*, 2010.
- [BGJ<sup>+</sup>09] A. Bhattacharyya, E. Grigorescu, K. Jung, S. Raskhodnikova, and D. Woodruff. Transitive-closure spanners. In *Proceedings of the 18th Annual Symposium on Discrete Algorithms (SODA)*, pages 531–540, 2009.
- [CLRS00] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2000.
- [DGL<sup>+</sup>99] Y. Dodis, O. Goldreich, E. Lehman, S. Raskhodnikova, D. Ron, and A. Samorodnitsky. Improved testing algorithms for monotonicity. *Proceedings of the 3rd International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*, pages 97–108, 1999.
- [EJ08] F. Ergun and H. Jowhari. On distance to monotonicity and longest increasing subsequence of a data stream. In *Proceedings of the 19th Symposium on Discrete Algorithms (SODA)*, pages 730–736, 2008.
- [EKK<sup>+</sup>00] F. Ergun, S. Kannan, R. Kumar, R. Rubinfeld, and M. Viswanathan. Spot-checkers. *Journal of Computer Systems and Sciences (JCSS)*, 60(3):717–751, 2000.
- [Fis01] E. Fischer. The art of uninformed decisions: A primer to property testing. *Bulletin of EATCS*, 75:97–126, 2001.
- [FLN<sup>+</sup>02] E. Fischer, E. Lehman, I. Newman, S. Raskhodnikova, R. Rubinfeld, and A. Samorodnitsky. Monotonicity testing over general poset domains. In *Proceedings of the 34th Annual Symposium on Theory of Computing (STOC)*, pages 474–483, 2002.
- [Fre75] M. Fredman. On computing the length of the longest increasing subsequences. *Discrete Mathematics*, 11:29–35, 1975.
- [GG07] A. Gal and P. Gopalan. Lower bounds on streaming algorithms for approximating the length of the longest increasing subsequence. In *Proceedings of the 48th Symposium on Foundations of Computer Science (FOCS)*, pages 294–304, 2007.
- [GGL<sup>+</sup>00] O. Goldreich, S. Goldwasser, E. Lehman, D. Ron, and A. Samordinsky. Testing monotonicity. *Combinatorica*, 20:301–337, 2000.
- [GGR98] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45(4):653–750, 1998.
- [GJKK07] P. Gopalan, T. S. Jayram, R. Krauthgamer, and R. Kumar. Estimating the sortedness of a data stream. In *Proceedings of the 18th Symposium on Discrete Algorithms (SODA)*, pages 318–327, 2007.

- [Gol98] O. Goldreich. Combinatorial property testing - a survey. *Randomization Methods in Algorithm Design*, pages 45–60, 1998.
- [HK03] S. Halevy and E. Kushilevitz. Distribution-free property testing. *Proceedings of the 7th International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*, pages 302–317, 2003.
- [PRR06] M. Parnas, D. Ron, and R. Rubinfeld. Tolerant property testing and distance approximation. *Journal of Computer and System Sciences*, 6(72):1012–1042, 2006.
- [Ram97] P. Ramanan. Tight  $\Omega(n \lg n)$  lower bound for finding a longest increasing subsequence. *International Journal of Computer Mathematics*, 65(3 & 4):161–164, 1997.
- [Ron01] D. Ron. Property testing. *Handbook on Randomization*, II:597–649, 2001.
- [RS96] R. Rubinfeld and M. Sudan. Robust characterization of polynomials with applications to program testing. *SIAM Journal of Computing*, 25:647–668, 1996.
- [Sch61] C. Schensted. Longest increasing and decreasing subsequences. *Canadian Journal of Mathematics*, 13:179–191, 1961.
- [SW07] X. Sun and D. Woodruff. The communication and streaming complexity of computing the longest common and increasing subsequences. In *Proceedings of the 18th Symposium on Discrete Algorithms (SODA)*, pages 336–345, 2007.