Input-Thrifty Extrema Testing

Kuan-Chieh Robert Tseng * and David Kirkpatrick

Department of Computer Science,
University of British Columbia, Canada.

Abstract. We study the complexity of one-dimensional extrema testing: given one input number, determine if it is properly contained in the interval spanned by the remaining n input numbers. We assume that each number is given as a finite stream of bits, in decreasing order of significance. Our cost measure, referred to as the *leading-input-bits-cost* (or LIB-cost for short), for an algorithm solving such a problem is the total number of bits that it needs to consume from its input streams.

An *input-thrifty algorithm* is one that performs favorably with respect to this LIB-cost measure. A fundamental goal in the design of such algorithms is to be more efficient on "easier" input instances, ideally approaching the minimum number of input bits needed to certify the solution, on all orderings of all input instances.

In this paper we present an input-thrifty algorithm for extrema-testing that is log-competitive in the following sense: if the best possible algorithm for a particular problem instance, including algorithms that are only required to be correct for presentations of this one instance, has worst-case (over all possible input presentations) LIB-cost c, then our algorithm has worst-case LIB-cost $O(c \lg \min\{c, n\})$.

In fact, our algorithm achieves something considerably stronger: if any input sequence (i.e. an arbitrary presentation of an arbitrary input set) can be tested by a monotonic algorithm (an algorithm that preferentially explores lower indexed input streams) with LIB-cost c, then our algorithm has LIB-cost $O(c \lg \min\{c, n\})$. Since, as we demonstrate, the cost profile of any algorithm can be matched by that of a monotonic algorithm, it follows that our algorithm is to within a log factor of optimality at the level of input sequences. We also argue that this log factor cannot be reduced, even for algorithms that are only required to be correct on input sequences with some fixed intrinsic monotonic LIB-cost c.

The extrema testing problem can be cast as a kind of list-searching problem, and our algorithm employs a variation of a technique called *hyperbolic dovetailing* that was introduced in that context. Viewed in this light, our results can be interpreted as another variant of the well-studied cow-path problem, with applications in the design of hybrid algorithms.

* This is an expanded version of a paper, with the same title, that will appear in the proceedings of ISAAC 2001 (LNCS 7074; T. Asano et al., Eds.). Many of the results appeared in a preliminary form in the first author's B.Sc. thesis entitled "Input Thrifty Algorithm for the Strict Containment Problem", Computer Science Department, U.B.C., April 2009.

1 Introduction

1.1 Input-thrifty algorithms

In many problem settings, precise inputs can be expensive to obtain. Typically, the more precise the input is, the higher the cost incurred. For example, suppose the input is acquired from some physical measurements. Obtaining imprecise measurements may be cheap, but more precision may require the use of progressively more expensive technology. Similarly, even precise measurements of smoothly time-varying data may degrade in proportion to the delay in transmitting the data. Another possibility arises if the inputs are produced by a numerical algorithm, such as quadrature or bisection. Again, acquiring more precise input will require additional computation. In all such cases, we are motivated to solve the underlying problem using as little input precision as possible. Algorithms that minimize the total amount of input precision used to solve the problem are referred to as *input-thrifty algorithms*.

It is clear that for some problems, full input precision is required to provide a correct solution. Finding the sum or product of input numbers are two obvious examples. In such cases, all algorithms are trivially input-thrifty and are not interesting to analyze. However, for problems such as finding the maximum of a set of numbers, some but not all problem instances require full precision of all inputs. Thus, an important property of input-thrifty algorithms is that they are *adaptive*–they perform better on easier problem instances.

Consequently, we do not measure the effectiveness of input-thrifty algorithms by the total amount of precision required in the worst case over all problem instances of some fixed size. Instead, we compare the amount of precision required relative to the *intrinsic cost* of each individual problem instance - a measure of how "hard" the instance is. Intuitively, the intrinsic cost should be the cost paid by the best algorithm that solves the problem instance since every algorithm will have to incur at least this cost. Analysis of an algorithm A relative to the intrinsic cost of individual problem instances, amounts to a kind of competitive analysis, where the A is competing against the the best possible algorithm, sometimes most naturally modeled as an "informed" algorithm that exploits knowledge of the input not available to A.

1.2 The uncertainty model

The problem that we consider takes as input a sequence $p_0, \ldots, p_n$ of real numbers in the half-open interval $[0, 1)$. An algorithm can access each such number $p = \sum_{j>0} p^{(j)} 2^{-j}$ via its binary representation $p^{(1)}, p^{(2)}, \cdots$, and this happens by examining the bits $p^{(j)}$ individually *in order of decreasing significance*, i.e. an algorithm can examine bit $p^{(j)}$ only after it has already examined bits $p^{(1)}$ through $p^{(j-1)}$.

Although the restriction of accessing input information one bit at a time in order of decreasing significance is perfectly natural, one could also adopt a more general model in which individual inputs are represented as a sequence of

nested *uncertainty intervals*. It turns out all of our results apply in this more general model. Nevertheless, we choose to first develop our results in the more restrictive, but less cumbersome, LIB-cost model.

1.3 Competitive analysis

An algorithm that knows (or guesses) the numbers in the input sequence can, at least in principle, realize the *certifying* LIB-cost, defined as the minimum number of input bits needed to certify the solution for that input. In general, however, algorithms must deal not only with uncertainty concerning the *membership* of the input set (that completely determines its certifying LIB-cost) but also the *presentation* of these numbers as a sequence. For this reason, it is natural to choose, as a basis for comparison, the behavior of algorithms aggregated over all possible presentations of a given input set.

For a given presentation S_π of an input set S, the *LIB-cost* of a deterministic algorithm A *on* S_π refers to total number of bits that A consumes from its input streams. The *maximum-LIB-cost* (respectively, *average-LIB-cost*) of A is defined to be the maximum (respectively, average), over all input sequences S_π that are presentations of S, of the LIB-cost of A on S_π. The *intrinsic* maximum-LIB-cost (respectively, average-LIB-cost) of an input set S is defined to be the minimum, over all algorithms B that are only guaranteed to solve the problem for inputs that are presentations of S, of the maximum-LIB-cost (respectively, average-LIB-cost) of B over S.

With this notion of intrinsic cost in hand, it is possible to do a *competitive analysis*[1] with respect to the family of algorithms that are constrained to answer correctly only when the input sequence is a presentation of S (i.e. relative to what we have called the intrinsic LIB-cost of S). Algorithms that are competitive on all instances are said to be *instance-efficient*; if the competitive ratio is a constant, they are *instance-optimal* (cf. [7], [1]).

While our results can be interpreted in this light, we are able to claim something that ties more closely to the certification cost of individual input presentations. Specifically, we define a restriction on algorithms that they explore input sequences in a monotonically decreasing fashion. Such algorithms are shown to be universal in the sense that the LIB-cost profile (the set of all LIB-costs over all presentations of an input set S) of an arbitrary algorithm A is exactly matched by the LIB-cost profile of some monotonic algorithm B. We then define the *monotonic-certification* LIB-cost of an input sequence S_π to be the minimum. over all monotonic algorithms, of the number of input bits needed to certify the solution for that input. We demonstrate an algorithm that is log-competitive on all presentations of all inputs with respect to the monotonic-certification LIB-cost: if input sequence S_π has monotonic-certification LIB-cost c, then our

[1] We speak of competitive analysis by analogy with the analysis (cf. [14]) of the performance of on-line algorithms (whose access to input information is restricted) in relation to the performance of the most efficient off-line algorithm (which have full access to the input).

algorithm has LIB-cost $O(c \lg \min\{c, n\})$. Furthermore, we argue that this log-factor cannot be reduced, even for algorithms that are only required to be correct on input sequences with fixed monotonic-certifification LIB-cost c.

1.4 Extrema testing

Our study originated with the following question posed by Leo Guibas at a Computational Geometry Seminar held at Schloss Dagstuhl in 1993:

> Given n points in the plane, does their convex hull contain the origin? Suppose further that the points have $\lg n$-bit coordinates. How many bits of the input do we need to examine to answer the question, in the worst case? More specifically, if there is a proof using only d bits, can we find it by examining only $O(d)$ bits?

Guibas' *origin-containment problem* is a natural example of a problem that can be addressed in our framework (where the input numbers are the coordinates of the points in question). Furthermore, the LIB-cost of algorithms, and the comparison with the certifying LIB-cost of the input, capture the essence of the associated questions.

In general, determining the LIB-complexity of certifying geometric predicates in two or more dimensions has proved to be quite difficult. We consider instead the following problem that is easily seen to be equivalent to the one-dimensional version of Guibas' origin-containment problem: suppose we are given a set S of real numbers in the interval $[0, 1)$, and another real number $a \in [0, 1)$, do there exists numbers x and y in S such that a is contained in the interval (x, y)? Clearly, solving this problem requires either (i) certifying that a is extreme in $S \cup \{a\}$, that is either $a \leq x$, for all $x \in S$ or $a \geq x$, for all $x \in S$, or (ii) certifying that a is not extreme, by demonstrating x and y in S such that $x < a < y$.

Certifying that a is extreme has basically a *fixed cost*, dependent on S but independent of the presentation of S. This follows because the construction of a certificate that $a < x$, for every $x \in S$, must entail the exploration of at least k_x bits, for each $x \in S$, where k_x denotes the index of the first bit position at which x and a differ. Thus, in both the worst and average cases, the intrinsic LIB-cost of certifying that a is extreme is just $\sum_{x \in S} k_x$. For this reason we hereafter focus on the more interesting case where a is *not* extreme.

Note that certifying that a is not extreme entails certifying that there exists a number x in S such that $a \neq x$. This simpler non-degeneracy problem has been explored in detail in [12]. Without loss of generality, we assume such an x exists and that $x < a$. So our problem reduces to that of determining whether there exists a number $y \in S$ such that $a < y$?

We begin by modeling our problem as a kind of signed list traversal problem in Section 2. As we shall see in that section, the worst case can be addressed directly by a straightforward modification of techniques for standard (unsigned) list traversal developed in [12]. Thus we focus our attention, in Section 3, on lower bounds for the intrinsic cost of individual presentations of the list traversal problem, relative to list traversal algorithms of a structurally-restricted, but

nevertheless (from a minimum cost viewpoint) universal, normal form. In Section 4, we prove that the cost incurred by a slightly modified hyperbolic dovetailing algorithm is within a logarithmic factor of this intrinsic cost on *all* input presentations, and that this logarithmic factor cannot be eliminated. It follows immediately that this modified hyperbolic dovetailing algorithm achieves an optimal logarithmic competitive-ratio, in both the worst and average cases, over all presentations of any fixed input.

2 Preliminaries

2.1 Related Work

As we have already indicated the results of this paper build directly on those of [12] where the *LIB-cost* model, input-thrifty algorithms and the hyperbolic dovetailing technique were all introduced in the context of a basic list-searching problem: given a set of lists with unknown lengths, traverse to the end of any one of the lists with as little total exploration as possible. Hyperbolic dovetailing was shown to provide a search cost within a logarithmic factor of the optimal (intrinsic) search cost for all problem instances, in both the worst and average case over all instance presentations.

A variety of other papers have dealt with the solution of fundamental computational problems in the presence of imprecise input. For example, Khanna *et al.* [11] analyzed how to compute common database aggregate functions such as sum and mean of a set of numbers, and Feder *et al.* [8] analyzed how to compute the median of a set of numbers. Other applications include finding the MST [6] and the convex hull [4]. Although the fundamental issues are clearly related, it should be noted that the framework used in these studies differs from our model in that they assume a fixed (indivisible) cost to determine specified inputs to (typically) full precision, whereas we insist that algorithms obtain extra precision of inputs in an incremental fashion.

2.2 The List Traversal Model

For each element in $x \in S$, we can certify that $x \neq a$ by obtaining some minimum number k_x of bits from the input stream of x (i.e. the k_x^{th} bit is the most significant bit on which x and a differ); if $x = a$ we take k_x as infinity. We model the input stream for x as a list with length k_x. We mark the end of the list with a "+", if the k_x^{th} bit of x is greater than the corresponding bit of a, and "−" otherwise. Thus, we can model S as a set of lists with unknown lengths, each marked with a positive or negative sign at the end. An algorithm is allowed to query any list at an integer position k, provided it has already queried that same list at all lower-indexed positions. If the list is queried at its last position, the query returns the corresponding sign. The goal is to find a + mark, at the end of any list, using the minimum number of queries.

We model a deterministic algorithm A as a ternary tree $\mathcal{T}$. At each node of $\mathcal{T}$, the index of the list to query next is specified. Depending on the outcome,

which could be either finding the end of a negative/positive list or simply discovering that the end of the list has not yet been reached, we traverse down one of the left/right/middle branches of $\mathcal{T}$ respectively. We make the assumption that the algorithm terminates if it has encountered a positive sign, and it never queries a list again if it has confirmed that the list is a negative list. Randomized algorithms can be modeled simply as probability distributions over the set of all deterministic algorithms.

The cost incurred by a list searching algorithm depends on both the input set and its presentation. The *maximum cost* of a deterministic algorithm, for a particular input, is the maximum number of queries the algorithm makes over all possible presentations of that input. The *average cost* of a deterministic algorithm is the number of queries the algorithm makes averaged over all possible presentations. For a randomized algorithm, the *maximum cost* will be the maximum number of *expected* queries the algorithm make over all possible presentations. Similarly, the *average cost* of a randomized algorithm is the expected number of queries averaged over all possible presentations. It should be clear that the maximum and average costs correspond directly to the maximum and average LIB-cost of certifying that $a < y$, for some $y \in S$.

We will denote the list traversal problem described above as the *POSITIVE_LIST_TRAVERSAL* (PLT, for short) problem. The input for the problem is described by a set $\Lambda = \Lambda^+ \cup \Lambda^-$, where Λ^+ is the set of positive lists, marked with a $+$ sign at the end, and Λ^- is the set of negative lists, marked with a $-$ sign at the end. Let $n = |\Lambda^+|$ and $m = |\Lambda^-|$ and denote by λ_i^+ (resp., λ_i^-) the length of the i^{th} shortest list in Λ^+ (resp., Λ^-). Thus, $\lambda_1^+ \leq \lambda_2^+ \leq \ldots \leq \lambda_n^+$ and $\lambda_1^- \leq \lambda_2^- \leq \ldots \leq \lambda_m^-$. Since the length of the list is the only property we are interested in, we will sometimes abuse notation, referring to the i^{th} shortest positive (resp., negative) list itself as λ_i^+ (resp., λ_i^-). When we are describing presentations of the input, it is convenient to adopt a *standard presentation* $\langle \lambda_1^+, \ldots, \lambda_n^+, \lambda_1^-, \ldots, \lambda_m^- \rangle$ and describe other presentations as permutations of this standard presentation. Specifically, if π is any permutation of $\langle 1, 2, \ldots n+m \rangle$ then we denote by Λ_π the presentation $\langle \lambda_{\pi(1)}, \lambda_{\pi(2)}, \ldots, \lambda_{\pi(m+n)} \rangle$.

Of course, if the set Λ^- is empty then the PLT problem reduces to the list traversal problem studied in [12]. Similarly, if we are interested in the maximum search cost, it is not hard to see that a simple generalization of the adversary strategy described in Theorem 5 of [12], forces $\xi(\Lambda) = \min_{1 \leq i \leq n} [i\lambda_i^+ + \sum_{1 \leq j \leq m} \min\{\lambda_j^-,\ \lambda_i^+\}]$ list probes in the worst case. Furthermore, by a modest generalization of Theorem 8 of [12], a simple hyperbolic traversal can be shown to achieve search cost $O(\xi(\Lambda) \lg \min\{|\Lambda|,\ \xi(\Lambda)\})$. Thus we focus our attention in the remainder of this paper on a finer grained cost measure, when the set Λ^- is not empty.

3 Intrinsic cost

For a competitive analysis of algorithms for the PLT problem, it is desirable to have some sort of intrinsic cost associated with individual problem instances

(that is, some way of distinguishing between "easy" and "hard" instances). Given a deterministic algorithm A and a fixed input instance $\Lambda = \Lambda^+ \cup \Lambda^-$, the cost of A on a particular presentation Λ_π is denoted by $c_A(\Lambda_\pi)$. With this we can define the *certification cost* of Λ_π to be minimum, over all deterministic algorithms A that behave correctly on Λ_π, of the cost $c_A(\Lambda_\pi)$. Using the certification cost as a notion of intrinsic cost for competitive analysis would be equivalent to comparing a given algorithm that knows nothing about the input with a competitor that knows not only the input instance Λ but also its presentation Λ_π.

As suggested above, a more realistic notion that takes into account the uncertainty associated with the given presentation can be formulated by aggregating costs over all possible presentations of Λ. We denote the average cost of A by $c_A(\Lambda)$. We can restrict our attention to deterministic algorithms since is straightforward to see that the average, over all presentations of Λ, of the expected cost of any randomized list traversal algorithm, is no less than the average cost of the best deterministic algorithm:

Lemma 1. *Let Λ be any instance of the PLT problem. For any randomized list traversal algorithm B, there exists a deterministic algorithm A such that $c_A(\Lambda) \leq c_B(\Lambda)$.*

Proof. Let X denote the set of all deterministic algorithms that solves the PLT problem for input Λ, and for each algorithm $A \in X$ denote by p_A the probability that B follows algorithm A. By definition, the average cost of B is:

$$\begin{aligned} c_B(\Lambda) &= \frac{1}{|\Lambda|!} \sum_{\pi} E[c_B(\Lambda_\pi)] \\ &= \frac{1}{|\Lambda|!} \sum_{\pi} \sum_{A \in X} p_A * c_A(\Lambda_\pi) \\ &= \sum_{A \in X} p_A \left[\frac{1}{|\Lambda|!} \sum_{\pi} c_A(\Lambda_\pi) \right] \\ &= \sum_{A \in X} p_A * c_A(\Lambda) \end{aligned}$$

We are allowed to swap the summation since there are only finitely many deterministic algorithms. For the same reason, we can choose a deterministic algorithm A_0 such that $c_{A_0}(\Lambda)$ is lowest among all deterministic algorithms A and $p_{A_0} > 0$. Then

$$c_B(\Lambda) \geq \sum_{A \in X} p_A * c_{A_0}(\Lambda) = c_{A_0}(\Lambda)$$

□

An even better approach that retains some of the sensitivity of certification cost of specific presentations, without presupposing an all-knowing competitor,

is to restrict the class of algorithms to a normal form that reflects the inherent uncertainly associated with the order of inputs. For every deterministic algorithm A and every input instance Λ, we can define the associated *cost profile*, which is just the multiset $\{c_A(\Lambda_\pi) \mid \Lambda_\pi \text{ is a presentation of } \Lambda\}$. We say that two algorithms are *cost-equivalent* on Λ if their cost profiles are identical.

A (deterministic) algorithm A, is said to be *monotonic* with respect to Λ_π if at every stage in the algorithm, for any $i < j$, the number of queries made in list $\lambda_{\pi(i)}$ is greater or equal to the number of queries made in list $\lambda_{\pi(j)}$, unless $\lambda_{\pi(i)}$ has been confirmed to be a negative list. The algorithm is monotonic with respect to Λ if it is monotonic with respect to all possible presentations of Λ. At first glance monotonicity may seem overly restrictive. However, since our analysis is with respect to all possible input presentations, it is at least intuitively clear that, when two lists are indistinguishable, nothing is lost by exploring the lower indexed list preferentially. The following lemma makes this intuition rigorous.

Lemma 2. *For any fixed input Λ and any algorithm A, there exists a monotonic algorithm $\widehat{A}$ that is cost-equivalent to A on Λ.*

Proof. Let $\mathcal{T}$ be the ternary tree that models algorithm A. At each node u of $\mathcal{T}$, we store the set of presentations that lead the algorithm from the root to u (the *plausible presentation* set or $PP(u)$) and the the number of times the list at each position has been queried (the *query profile*).

The root node's *plausible presentation* set will consists of all presentations and the *query profile* will have 0 queries for all positions. Each path from the root to a leaf of $\mathcal{T}$ represents one possible sequence of queries the algorithm will make. At each leaf node, the algorithm has found the end of a positive list. The *plausible presentation* set of the leaf node represent the presentations of Λ for which the algorithm will make this sequence of queries and the depth of the leaf node indicate the cost of the algorithm with respect to the presentations in the *plausible presentation* set.

To show that there exists a monotonic deterministic algorithm $\hat{A}$ that is cost-equivalent to A on Λ, it suffices to show that we can construct a tree that is *equivalent* to the tree of A in the sense that there exists a bijection f between the nodes of the two trees satisfying:

1. u is a parent of v in the tree of A iff $f(u)$ is a parent of $f(v)$ in the tree of $\hat{A}$.
2. for any node u in the tree of A, the *query profile* of u is equivalent to the *query profile* of $f(u)$ up to permutations.
3. for any node u in the tree of A, $|PP(u)| = |PP(f(u))|$.

It is not hard to see that this is indeed an equivalence relation. To construct the tree, we first start with the tree $\mathcal{T}$. Starting from the root node and propagating down the tree, we change the list to be queried at each node. If the list queried is at position k, which currently has been queried p times, then we will change the node to query the list at position i, where i is the smallest position in the *query profile* of the current node that has been queried p times so far and the list at position i has not been confirmed to be a negative list (i.e. in the

path from the root node to the current node, we have not queried position i and traversed down the middle edge). After changing this node, we then go through every node in the subtree rooted at the current node and propagate the change: if the node queries the list at position i, we change it to k and vice versa. We then repeat the process for every child of the node.

Since the new tree is based on $\mathcal{T}$, except that queries are changed at each step, it follows that the new tree describes an algorithm. Let $\hat{A}$ be the algorithm represented by the tree constructed in this way. It is clear at each node in the new tree, the query profile has the property that for any $i < j$, the number of queries made at position i is less than at position j, unless it has been confirmed that the list at position i is a negative list. Thus, $\hat{A}$ must be a monotonic algorithm by definition.

It remains to show that the two trees are equivalent. Note that we have not changed the tree structure of A in the new tree, so there is an obvious bijection f between the two trees and it is clear that f satisfies condition 1. To prove condition 2 and 3, we show that after updating node u in the new tree, the subtree rooted at u is equivalent to the subtree rooted at $f(u)$. Since updating the node u only affects the nodes in the subtree rooted at u, proving the claim shows that the trees are equivalent after updating a single node. By transitivity, we can conclude that the resulting tree is equivalent to the tree representation of A.

Suppose that we are updating node u and instead of querying the list at position k, we query the list at position i. Let σ be the permutation that swaps position k and i and leaves everything else unchanged. To prove that condition 2 is met, simply note that for every node in the subtree, the *query profile* has simply been permuted by σ. Let v be a node in the subtree rooted at u, we introduce the notation $PP(v)$ to be the *plausible presentation* set of v before updating and $PP'(v)$ to be the set after updating. The following claim demonstrates that condition 3 is met:

Claim: For every node v in the subtree rooted at u, $\Lambda_\pi \in PP(v) \Leftrightarrow \Lambda_{\pi\sigma} \in PP'(v)$.

Proof. For the case $v = u$, note that by our algorithm for updating, the number of queries made at position k and i is the same (denote this number by p). Thus, at this point in the algorithm, we know that the list at position k and i are both longer than p. This means that if we swap $\lambda_{\pi(i)}$ and $\lambda_{\pi(k)}$ while keeping everything else the same, the algorithm will not be able to distinguish the new presentation from Λ_π. So the algorithm will take the same path from the root to u and $\Lambda_{\pi\sigma} \in PP'(v)$.

Now suppose v is the left child of u and $\Lambda_\pi \in PP(v)$. This means that $\lambda_{\pi(k)} = p+1$ and $\lambda_{\pi(k)}$ is a positive list. However, by the observation above, we know that the presentation $\Lambda_{\pi\sigma} \in PP'(u)$ and that $\lambda_{\pi\sigma(i)} = \lambda_{\pi(k)} = p+1$ and is a positive list. Thus, $\Lambda_{\pi\sigma} \in PP'(v)$. Similarly, we can prove that if $\Lambda_{\pi\sigma} \in PP'(v)$, then $\Lambda_\pi \in PP(v)$ by noting $\sigma^{-1} = \sigma$. This proves the claim for the case v is the

left child of u. The case where v is the middle or right child of u can be shown similarly.

We will use induction on the depth of v relative to u. The base case is when the depth is 1 and v is a child of u. We have already shown the result above. Now, let v have depth l and denote the path from u to v by $u, v_1, \ldots, v_{l-1}, v$. Let $\Lambda_\pi \in PP(v)$, then $\Lambda_\pi \in PP(v_{l-1})$ and $\Lambda_{\pi\sigma} \in PP'(v_{l-1})$ (by induction hypothesis). Now, if v_{l-1} queries at position j, where $j \neq k, i$, then since $\lambda_{\pi\sigma(j)} = \lambda_{\pi(j)}$, the algorithm will receive the result of the query and will not change after updating u. So $\Lambda_{\pi\sigma} \in PP'(v)$. Otherwise, suppose v_{l-1} queries at position k. Once again, we will receive the same result from the query after updating, v_{l-1} will query position i after updating and $\lambda_{\pi\sigma(i)} = \lambda_{\pi(k)}$. The same argument applies if v_{l-1} queries at position i. So $\Lambda_\pi \in PP(V)$ implies $\Lambda_{\pi\sigma} \in PP'(v)$. A parallel argument using the fact $\sigma^{-1} = \sigma$ in the reverse direction completes the proof. □

We are now in a position to provide a more robust definition of intrinsic cost that coincides with the aggregated certification costs described earlier but is more realistic at the level of individual presentations. We define the *monotonic-certification cost* of Λ_π, denoted $\xi(\Lambda_\pi)$ to be minimum, over all deterministic monotonic algorithms $\widehat{A}$ that behave correctly on Λ_π, of the cost $c_{\widehat{A}}(\Lambda_\pi)$.

Lemma 3. $\xi(\Lambda_\pi) = \min\limits_{1 \leq i \leq n} \sum_{p=1}^{\pi^{-1}(i)} \min\{\lambda_{\pi(p)}, \lambda_i\}$.

Proof. Suppose that algorithm $\widehat{A}$ terminates after discovering the end of list λ_k^+ at position $\pi^{-1}(k)$ in the input presentation. Then, by the monotonicity of $\widehat{A}$,

$$c_{\widehat{A}}(\Lambda_\pi) \geq \sum_{p=1}^{\pi^{-1}(k)} \min\{\lambda_{\pi(p)}, \lambda_k^+\} \geq \min_{1 \leq i \leq n} \sum_{p=1}^{\pi^{-1}(i)} \min\{\lambda_{\pi(p)}, \lambda_i\}$$

But if i^* satisfies $\sum_{p=1}^{\pi^{-1}(i^*)} \min\{\lambda_{\pi(p)}, \lambda_{i^*}\} = \min\limits_{1 \leq i \leq n} \sum_{p=1}^{\pi^{-1}(i)} \min\{\lambda_{\pi(p)}, \lambda_i\}$ then the monotonic algorithm that explores lists, in order, to depth $\lambda_{i^*}^+$ realizes the minimum possible cost on Λ_π. □

4 Hyperbolic Dovetailing

For the problem where every list is marked with a positive sign, Kirkpatrick [12] proposed a hyperbolic dovetailing algorithm whose cost is only a logarithmic factor more than the aggregated intrinsic cost. In this section, we extend the hyperbolic dovetailing algorithm to solve the current problem. We first recall the hyperbolic dovetailing algorithm. A phase of the algorithm involves iterating through every list and extending the exploration up to a depth determined by the list index and the phase number. We define the *rank* of the t^{th} position on the i^{th} list as $\sum_{j=1}^{i} \min\{\lambda_{\pi(j)},\ t\}$. At the completion of phase $r \geq 1$ all list positions of rank at most r have been explored.

HYPERBOLIC-DOVETAIL(Λ_π)
  $r \leftarrow 1$;
  while (end of a positive list has not been encountered)
    for ($i = 1$; $i \leq m + n$; $++i$)
      continue exploration of list $\pi(i)$ up to the last position of rank at most r
    $r \leftarrow r + 1$;

Theorem 1. *Given a fixed input sequence Λ_π, the cost of the hyperbolic dovetailing algorithm is $O(\xi(\Lambda_\pi) \lg \min\{\xi(\Lambda_\pi), n + m\})$.*

Proof. Suppose the hyperbolic dovetailing terminates with $r = \alpha$. We consider the total number of queries by considering how many lists has been queried at least once, twice, etc. By the definition of rank, the number of lists that has been queried h times is at most $\min\{\alpha/h, n + m\}$. Thus, the total number of queries is:

$$\sum_{h=1}^{\alpha} \min\{\frac{\alpha}{h}, n + m\} = O(\alpha \ \lg \min\{\alpha, n + m\}).$$

It suffices to sum up to α since no list will be queried more than α times by the definition of rank. Finally, note that hyperbolic dovetailing will terminate successfully when $r = \xi(\Lambda_\pi)$. □

Thus hyperbolic dovetailing algorithm achieves the monotonic-certification cost of *every* presentation of *every* input instance, up to at most a logarithmic factor. It is not hard to see that this logarithmic factor cannot be avoided. In particular, if there are no negative lists the problem reduces to the standard list searching problem studied in [12]. In that setting, a family of lists was described that force a logarithmic competitive ratio. In our setting, this same family shows that any deterministic monotonic algorithm that is only required to be correct on input presentations whose monotonic-certification cost is c, has cost $c \lg \min\{c, n + m\}$ for at least one such sequence.

5 Conclusion

This paper has investigated the problem of determining if a given number is extreme with respect to a given set S of numbers. We recast the problem as a signed list-search problem. One of the main results is the specification of a notion of intrinsic cost (monotonic-certification cost) for all presentations of all instances of this problem. This is made possible by a normal-form result that shows that every algorithm that is tailored to a specific problem instance has a cost-equivalent monotonic algorithm for that instance. Our second contribution is a simple modification of the hyperbolic dovetailing algorithm of [12], that can be shown to solve the signed list-search problem within a logarithmic factor of the monotonic-certification cost of every presentation of every problem instance. Finally, we argued that this log-competitiveness cannot be improved, even by algorithms that know the monotonic-certification cost of their input sequence.

The question investigated in this paper has several natural generalizations. Foremost, it still remains to settle Guibas' original origin-containment problem, the problem that motivated the investigation in this paper, in two or higher dimensions. It should be noted that although the list traversal model was developed to model the LIB-cost of certain fundamental problems defined on integers presented as a stream of bits, it can also be used to model the more general situation where inputs are presented as a sequence of nested uncertainty intervals. For our problem it suffices to interpret a positive (resp., negative) sign in the t^{th} position of list i as an assertion that the t^{th} uncertainty interval associated with the i^{th} input x_i is disjoint from and larger (resp., smaller) than the t^{th} uncertainty interval associated with the number a. Under this interpretation, all of our results extend to this more general uncertainty model.

Finally, we remark that the signed list-search problem has applications that go beyond the realm of graph search. For example, following a similar idea presented by Kao [10], suppose we wish to solve a generally intractable problem and we have at our disposal several heuristics, demonstrated to be effective for certain classes of problem instances. On a particular problem instance, some heuristics might perform well, while others will not help at all. The problem then it to determine, without knowing the nature of the problem instance, how we should dovetail among the available heuristics. This can clearly be modeled by the list-traversal problem, where each list represents the full computation associated with one heuristic, and the depth of traversal at any point in time corresponds to the resource allocated to the associated heuristic. A positive list represent a heuristic that completes successfully and a negative list represents a heuristic that completes unsuccessfully.

Acknowledgements

The authors gratefully acknowledge helpful discussions with Raimund Seidel.

References

1. P. Afshani, J. Barbay, and T. M. Chan. Instance-optimal geometric algorithms. *50th Annual IEEE Symposium on Foundations of Computer Science* (2009), 129-138.
2. Y. Azar, A. Z. Broder and M. S. Manasse. On-line choice of on-line algorithms. *Proc. 4th Annual ACM-SIAM Symposium on Discrete Algorithms* (1993), 432-440.
3. R. A. Baeza-Yates, J. C. Culberson, and G. J. E. Rawlins. Searching in the plane. *Information and Computation* 106,2 (1993), 234-252.
4. R. Bruce, M. Hoffmann, D. Krizanc and R. Raman. Efficient update strategies for geometric computing with uncertainty. *Theory of Computing Systems* (2005), 411-423.
5. R. Dorrigiv and A. Lopez-Ortiz. A survey of performance measures for on-line algorithms. *ACM SIGACT News* 36,3(2005), 67-81.
6. T. Erlebach, M. Hoffmann, D. Krizanc, M. Mihal'ák and R. Raman. Computing minimum spanning trees with uncertainty. ArXiv e-prints (2008).

7. R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *Proc. 20th ACM Symposium on Principles of Database Systems* (2001), 102-113
8. T. Feder, R. Motwani, R. Panigrahy, C. Olston and J. Widom. Computing the median with uncertainty. *Proc. 32nd Annual ACM Symposium on Theory of Computing* (2000), 602-607.
9. M. -Y. Kao and M. L. Littman Algorithms for informed cows. *AAAI-97 Workshop on On-Line Search* (1997).
10. M. -Y. Kao, Y. Ma, M. Sipser, and Y. Yin. Optimal constructions of hybrid algorithms. *J. Algorithms* 29,1, (1998), 142-164.
11. S. Khanna and W. -C. Tan. On computing functions with uncertainty *Proc. 20th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (2001), 171-182.
12. D. Kirkpatrick. Hyperbolic dovetailing. *Proc. European Symposium on Algorithms* (2009), 516-527.
13. M. Luby, A. Sinclair and D. Zuckerman Optimal speedup of Las Vegas algorithms *Proc. Second Israel Symposium on Theory of Computing and Systems*, Jerusalem, (June 1993), 128-133.
14. D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules.. *Comm.ACM* 28 (Feb. 1985), 202-208.