On Computing Functions with Uncertainty

Sanjeev Khanna[*]
University of Pennsylvania
sanjeev@cis.upenn.edu

Wang-Chiew Tan[†]
University of Pennsylvania
wctan@saul.cis.upenn.edu

ABSTRACT

We study the problem of computing a function $f(x_1, ..., x_n)$ given that the actual values of the variables x_i's are known only with some uncertainty. For each variable x_i, an interval I_i is known such that the value of x_i is guaranteed to fall within this interval. Any such interval can be probed to obtain the actual value of the underlying variable; however, there is a cost associated with each such probe. The goal is to adaptively identify a minimum cost sequence of probes such that regardless of the actual values taken by the unprobed x_i's, the value of the function f can be computed to within a specified precision.

We design online algorithms for this problem when f is either the selection function or an aggregation function such as sum or average. We consider three natural models of precision and give algorithms for each model. We analyze our algorithms in the framework of competitive analysis and show that our algorithms are asymptotically optimal. Finally, we also study online algorithms for functions that are obtained by composing together selection and aggregation functions.

1. INTRODUCTION

Caching of data tends to improve the performance of data retrieval in a client-server environment. Besides speeding up data retrieval, caching also allows the client to reduce its total bandwidth requirements by avoiding to lookup the server everytime the same data is needed. However, since the server may have updated its data in the meantime, data obtained from the cache may be stale. The tradeoff issue between performance and staleness of data has been the subject for much research (for instance, see [2]). We consider the following variation of this problem. Suppose the cache stores for each value, an upper bound and a lower bound that specifies the range in which the current value must reside. If the computation being performed by the client can tolerate some error, then the data in the cache can be used to quickly compute the result. If the computed result based on the data in the cache is not within the error required by the client, one can update a few values in the cache and recompute the result. As an example, suppose a client wishes to find the minimum among a set of elements stored in the cache. The precise values of these elements are unknown but instead for each element, an upper and a lower bound on its underlying value is given. The client states a precision parameter which specifies that it only needs to find an element that does not deviate too far from the actual minimum. In general, it may be impossible to identify such an element from the given bounds in the cache, and it may be necessary to obtain precise values for some of the elements in the set. The precise value of an element can only be obtained from the server and thus involves an additional cost. How does one identify a minimum cost set of elements so that based on their precise values alone, one can find such an element with certainty?

In general, how can one compute the value of a function within some precision while minimizing the cost of obtaining accurate values for various elements in the underlying data? While computing the function value exactly may require collecting accurate values for much of the underlying data, if one is willing to tradeoff the precision, in many cases, it is possible to identify only a small subset of data whose values need to be pinned down accurately. Such tradeoffs between cost and precision arise in a wide variety of settings. For instance, with priced information sources in a networked environment such as the Internet [7], one can often avoid paying the cost of retrieving up-to-date information from a provider by getting a stale copy of the same information from a cached location. As in [10, 6], we may either assume that each cached item has some known divergence function which tells us by how much the initial value may change over time or assume that the server updates the cache whenever the new server value falls out of the range of values stored within the cache.

Problem Statement. We consider the following abstraction of the scenarios outlined above. We wish to compute a function $f(x_1, ..., x_n)$ given that the actual value associated with each variable x_i is unknown but an interval I_i guaranteed to contain the actual value of x_i is known. For

[*]Supported in part by an Alfred P. Sloan Research Fellowship.

[†]Supported in part by a Digital Libraries 2 grant, DL-2 IIS 98-17444.

each variable x_i, there is an associated cost c_i such that we can probe x_i to determine its exact value at a cost of c_i. The goal is to *adaptively* identify a minimum cost sequence of variables to probe so that regardless of the actual values taken by the unprobed variables, the value of the function f can be determined to within a specified accuracy. The resulting set of values or intervals associated with each x_i is a *witness* or a *proof* that the value of f lies within the required accuracy. Our goal is to design online algorithms that find such a witness at a minimum possible cost.

We study the performance of such algorithms in the framework of competitive analysis for online algorithms [1]. Given an instance I, let $A(I)$ denote the cost of an algorithm A on the instance I. Moreover, let $\mathrm{OPT}(I)$ denote the cost of the optimal algorithm. Then the *competitive ratio* of an online algorithm A is defined to be β if for any finite input I, $A(I) \leq \beta \cdot \mathrm{OPT}(I)$. Notice that an optimal algorithm is one that always finds a witness of the minimum possible cost. To see the difficulty of finding a witness with small overall cost, consider the following simple example. Suppose we wish to find the minimum of n variables each of which is given to be contained in the interval $[0, 1]$. Consider the following instance constructed in an adversarial manner: the first $n-1$ variables probed by the online algorithm are assigned a value of 1 while the last variable probed is assigned a value of 0. Clearly, any online algorithm must make all n probes since otherwise it can not ascertain the minimum value. On the other hand, an optimal algorithm simply probes the variable with value 0 and immediately ascertains the minimum to be 0. Thus the competitive ratio of any online algorithm is at least n for this problem.

Results. We study the problem for various choices of function f that arise in basic database queries. Specifically, we allow f to be an aggregation operator such as sum or average, or a selection operator that outputs an element of a specified rank from a given ordered set. We design online algorithms for all the problems stated above and establish matching upper and lower bounds on the competitive ratio. We also design competitive online algorithms for functions which are obtained by composing these operators. An interesting aspect of our results is that they highlight a structural parameter that determines the complexity of any given instance for essentially all functions considered here. This structural parameter is the maximum clique size of the interval graph associated with the set of input intervals.

For the aggregation operators, we consider two basic measures of precision for the computed function value, namely the *relative precision* model and the *absolute precision* model. In the relative precision (absolute precision) model, we say that an answer is α-*precise* if the actual value of the function is within a multiplicative (additive) factor of α of the answer. We mainly focus on the relative precision model which we believe is a more natural model for specifying precision since absolute precision model seems only effective when we have some knowledge about the magnitude of the actual answer. However, in general, it is harder to design and analyze algorithms in the relative precision model. For instance, for the sum problem where each variable has a unit cost of probing, a simple greedy algorithm can be shown to be optimal for the absolute precison model. In contrast, one can show that in the relative precision model, any online algorithm can be made to probe $\Omega(\min\{\alpha, k\})$ times as many variables as probed by an optimal algorithm where k is the maximum clique size in the associated interval graph. For selection problems, in addition to the above models, there is another natural model of precision, namely the *rank precision* model. In this model, we say that an answer is α-*precise* for the problem of identifying an element of rank p if the rank of the element output is within the range $[p-\alpha, p+\alpha]$. For the aggregation functions sum and average, we design an online algorithm that is $O(\min\{\alpha, k\})$-competitive. We show that no online algorithm can achieve an asymptotically better competitive ratio. For the selection problem, in all three precision models, we design an algorithm that is $O(k)$-competitive for any specified precision α. Moreover, we show that no online algorithm can achieve a competitive ratio better than $\Omega(k)$ in any of the precision models.

We also design and analyze online algorithms for functions obtained by composing the operators studied above. In particular, we study computations of the form $f(g(), g(), ..., g())$ where f and g range over the operators minimum, maximum, sum and average. While for many composition operations, the complexity of the problem is not too different from the complexity of each of the individual operators, when f is the operation minimum (or maximum) and g is sum (or average), the problem becomes significantly harder then either one of the underlying operators.

Related Work. Some of the problems considered here were first studied by Olsten and Widom in [10]. Their TRAPP (Tradeoff in Replication Precision and Performance) system aims to increase performance by caching an interval containing the actual value instead of the actual value. By storing a large interval, one can reduce the likelihood that any changes for a value at the source would require updating this value at the cache. As an example, suppose a source consists of 5 items, each with value 1. If the cached value is $[0, 10]$ for each item, then no update needs to be made to the cache whenever a change in a source value still falls within the range 0 to 10. However, it is more likely that the precision constraint specified by an incoming query cannot be answered by the cached values with large intervals. Continuing with the example, the cached values cannot be used to answer a sum query to within an absolute precision of, say 5. This would have been possible if each cached value is say, $[0, 1]$. The problem of what precision to give the cached values dynamically so as to achieve the best possible performance under varying workloads has been investigated in [9]. In the situation when the precision constraint specified by an incoming query cannot be satisfied, some of the source values needs to be queried for the actual values in order to answer the query to within the required precision. This problem has been studied in [10] when the incoming query is a sum or average or minimum / maximum and only for the absolute precision model. An important difference between [10] and our work is that the authors in [10] consider only *oblivious* algorithms where the set of variables to be probed is determined in a non-adaptive manner.

The general selection problem – finding the pth smallest element among n elements – in the absolute precision model was studied by Feder *et al* in [6]. They analyze the benefit of using an online (adaptive) strategy over an oblivious algorithm. They show that the ratio between the worst case performance of an oblivious algorithm and the worst case online algorithm is bounded by at most 2 when each probe costs a unit amount and at most p when the cost of each probe can be arbitrary. However, we note here that the ratio between the worst case performance of the oblivious algorithm and the optimal algorithm is bounded by n for the unit cost case and may be infinite for the arbitrary cost case. In particular, suppose we wish to find the minimum element among n identical intervals, say $[0, 10]$ to within an absolute precision of 1. It is easy to see that an oblivious algorithm will need to probe all n elements. On the other hand, an optimal algorithm requires only one probe if there were some interval with precise value 0. For the unit cost model, the ratio between the number of probes required by the oblivious algorithm to optimal algorithm is clearly n. However, if the cost of every interval that contains 10 is infinite and the cost of the interval that contains 0 is a unit amount, the ratio is clearly infinite for the arbitrary cost case. Our focus in this paper is to measure the worst-case performance of our online algorithm against the optimal algorithm over all inputs. We note that neither the relative precision model for selection and aggregation nor the rank precision model for selection have been considered by earlier works.

There has been significant works on top-k selection queries, for instance, by Fagin in [4, 5] and by Chaudhuri and Gravano in [3]. In these works, the goal is to find the top-k results of a query involving several multimedia attributes (e.g. image or audio attributes). Since a predicate on such type of attribute often does not involve an exact match, the goal is thus to find tuples that best match the given predicates. It is worthwhile to point out the differences between our selection problem with rank precision model and the problem of top-k selection queries. The input to our problem is a set of intervals (imprecise data) while the input to the top-k selection problem is the underlying data source (precise data). In the rank precision model, our goal for the "top-k" selection problem is to find *an interval* that is guaranteed to have rank between 1 and k inclusive. In other words, if we somehow know the actual values of all intervals, the interval that we return as the result is guaranteed to hold a value that is among the top k values. For the top-k selection problem, their output is a graded sequence of k elements (in the order of non-increasing grades). The element with the highest grade signifies the answer that best matches the given predicates.

Organization. Section 2 describes our notation and terminology. In Section 3 and Section 4, we design and analyze algorithms for the selection and aggregation operators respectively. Finally, in Section 5, we consider algorithms for functions considered by composing these operators.

2. PRELIMINARIES

We use $[l, r]$ to denote a range of values from l to r inclusive. For any interval I, $l(I)$ and $r(I)$ represent the left and right end-points of the interval respectively. Given a variable x with an associated interval I, we will say that we probe I to simply mean that we probe the variable x to determine its precise value. Every interval (variable) I has an associated cost of probing (or querying) and is denoted by $c(I)$. Costs and intervals are assumed to be non-negative. We use $a(I)$ to denote the actual (or precise) value underlying an interval. That is, after probing I, we get the precise interval $[a(I), a(I)]$. As a convention, we will always return the left end-point of an interval as an answer. An interval I *overlaps* with an interval J iff $r(I) \geq l(J)$ and $l(I) \leq l(J)$ or vice versa.

The *interval graph* $G(V, E)$ of a set of intervals $S = \{I_1, ..., I_n\}$ is defined as follows. The vertex set V contains a vertex v_i for each interval I_i, and the edge set E contains an edge $(v_i, v_j) \in E$ if I_i overlaps with I_j. The *width* of S is defined to be the the size of the largest clique in G and is equal to the largest number of intervals that pairwise overlap with one another. A well-known fact is that any collection S of intervals of width k can be partitioned in polynomial-time into k sets $S_1, S_2, ..., S_k$ such that no two intervals in any S_j overlap (see, for instance, [8]). We will frequently use such a partition to design our algorithms. Throughout, we will denote by α the specified precision and by k the width of the given instance.

3. SELECTION

The selection problem is to find the pth largest element among a set of elements. We will study the selection problem under all three models of precision. Throughout our discussion we will assume that we have a partition of the intervals associated with the variables into k sets of non-overlapping intervals denoted by $S_1, S_2, ..., S_k$.

3.1 Selection with Rank Precision

In rank precision model, the objective is to return an element whose rank lies within the range $[p - \alpha, p + \alpha]$ by adaptively probing a minimum cost set of intervals. In what follows, we consider the more general problem of finding an element within any given rank range $[s, t]$. Given a set of intervals $I_1, ..., I_n$, we say that $a(I_j)$, for some j, is an element whose rank lies in the range $[s, t]$ iff the rank of $a(I_j)$ amongst $a(I_1), ..., a(I_n)$ is in the range $[s, t]$.

3.1.1 The Unit Cost Model

We first show that the rank of an element is within the rank range $[s, t]$ if there exist $s - 1$ intervals to its left and $n - t$ intervals to its right. Notice however that the converse may not be true. Having such partition of intervals is a witness which establises that the element is within the rank range regardless of the outcome of the actual values taken by each interval. The proof of the following proposition is straightforward and can be found in the appendix.

PROPOSITION 3.1. *Let $L_J = \{I \mid I \neq J$ and $r(I) \leq l(J)\}$ and $R_J = \{I \mid I \neq J$ and $l(I) \geq r(J)\}$ for some interval J. Then $a(J)$ is an element whose rank lies in the range $[s, t]$ if $|L_J| \geq s - 1$ and $|R_J| \geq n - t$.*

As an example, consider the set of intervals shown in Figure 1(a). 5 is an element that lies within the rank range $[3, 4]$ since there are two intervals to the left of $[5, 11]$ and one interval to its right. Similarly, 7 is also an element within the same rank range.

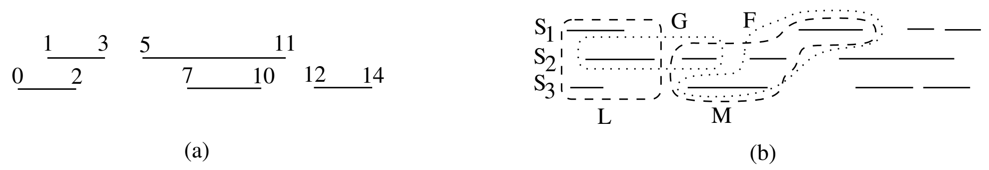


Figure 1: (a) A set of intervals with width 2. (b) A set of intervals with width 3. L, M, F and G are selected with respect to a rank of $p = 5$.

Our main algorithm Select-Rank-Unit, which will be described shortly, adaptively finds such an interval J which lies between $s-1$ intervals on its left and $n-t$ intervals on its right. Once we identify J, it suffices to return $l(J)$ as the answer. We first present an algorithm Weak-rank which will be used as a subroutine by our main algorithm Select-Rank-Unit. The subroutine Weak-rank takes as input a rank value p and returns a subset of $O(k)$ intervals so that the pth element is guaranteed to be among these intervals. It ensures this by first selecting a set of intervals (which we will denote by $L \cup M$) so that any interval that is not in this set has rank strictly greater than p. A second selection (which we will denote by $F \cup G$) is made out of intervals in $L \cup M$. This is done in such a way that any interval in $(L \cup M) - (F \cup G)$ has rank strictly less than p. A nice property of $F \cup G$ is that it has a cardinality of at most $2k$ which we will exploit in achieving a k-competitive algorithm for this problem.

Algorithm Weak-rank. We first select the leftmost interval in each partition. Let $L = \{s_1, ..., s_k\}$ where s_i is the interval in S_i with the smallest left value. Index the remaining intervals according to their left values in non-decreasing order and let $J_1, ..., J_{n-k}$ denote this sequence. Let $M = \{J_1, ..., J_{p-1}\}$ be the first $p-1$ intervals. Note that we have selected $k+p-1$ intervals in $L \cup M$ so far. Next, we would like to find the rightmost interval in each partition of $L \cup M$. Let $F = \{s'_1, ..., s'_k\}$ where s'_i is the interval in $S_i \cap (L \cup M)$ with the largest right value. Index the remaining intervals according to their right values in non-decreasing order and let $J_{p-1}, ..., J_1$ denote this sequence. We select the last $k-1$ intervals and denote this set by G. Observe that $G = \{J_{k-1}, ..., J_1\}$ and G includes all intervals if $p \le k$.

An example of L,M,F and G selected by Algorithm Weak-rank with $p = 5$ is shown in Figure 1(b).

LEMMA 3.1. *If $p \le k$ then the pth largest element in the original instance is the pth largest element in $F \cup G$. Otherwise, the pth largest element in the original instance is the kth largest element in $F \cup G$.*

PROOF SKETCH. We consider two cases depending on whether $p \le k$ or $p > k$. We first show that any interval not in $L \cup M$ must have rank greater than p. Observe that if $p \le k$, $F \cup G = L \cup M$. Therefore the pth largest element must be the pth largest element in $L \cup M$. If $p > k$, we show also that any element in $(L \cup M) - (F \cup G)$ has rank less than p. Since there are $p-k$ elements in $(L \cup M) - (F \cup G)$, the pth largest element must be the kth largest element in $F \cup G$. The full proof is shown in the appendix. □

We show next a k-competitive algorithm for finding an element within the rank range $[s, t]$.

Algorithm Select-Rank-Unit. The algorithm looks for two sets of intervals X and Y X is a set of $s-1$ intervals with the leftmost right values. Y is a set of $n-t$ intervals from the remaining intervals with the rightmost left values. Then depending on whether the boundaries of X and Y overlap, we select either k or at most $2k$ intervals to probe.

1. We first index the input intervals according to their right endpoints in non-decreasing order. If two intervals I and I' have the same right endpoint but $l(I) < l(I')$ then I comes before I' in the indexed sequence. Let X be the first $s-1$ intervals in this order and let r_X denote the largest right value among intervals in X. If $s-1 = 0$ then $r_X = -\infty$.
2. Now we index the remaining intervals according to their left endpoints in non-decreasing order. If two intervals I and I' have the same left endpoint but $r(I') > r(I)$ then I comes before I' in the sequence. Let Y denote the last $n-t$ intervals in this order and let l_Y denote the smallest left value among intervals in Y. If $n-t = 0$ then $l_Y = \infty$.
3. If there exists an interval $J \notin X \cup Y$ such that $l(J) \ge r_X$ and $r(J) \le l_Y$ then return $l(J)$. $l(J)$ is the element whose rank is guaranteed to lie within $[s, t]$. Otherwise, consider two cases:
   (a) $r_X \le l_Y$. Probe any interval $J \notin (X \cup Y)$ and consider three cases:
      i. If $r_X \le a(J) \le l_Y$ then return $a(J)$.
      ii. $a(J) < r_X$. Consider the subset of intervals $X' = X \cup \{J\}$. Observe that J is the rightmost interval in $(X' \cap S_i)$ for some i. For each S_j where $j \ne i$, we probe the rightmost interval in $(X' \cap S_j)$. Let $J_1, ..., J_m$ denote the intervals probed in this manner where $m \le k-1$. Return the largest value among $\{a(J_1), ..., a(J_m), a(J)\}$.
      iii. $a(J) > l_Y$. Similarly, consider the subset of intervals $Y' = Y \cup \{J\}$. J is the leftmost interval in $(Y' \cap S_i)$ for some i. For each S_j where $j \ne i$, we probe the leftmost interval in $(Y' \cap S_j)$. Let $J_1, ..., J_m$ denote the intervals probed in this manner where $m \le k-1$. Return the smallest value among $\{a(J_1), ..., a(J_m), a(J)\}$.
   (b) $r_X > l_Y$. Let $p = s$. We invoke the algorithm Weak-rank to find the subset of $k+p-1$ or $2k-1$ intervals depending on whether $p \le k$ or $p > k$. In the former case, we probe all intervals and return

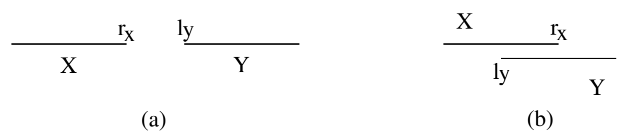


Figure 2: The two possible situations corresponding to Step 3(a) and 3(b). $|X| = s-1$, $|Y| = n-t$.

the pth value. In the latter case, we probe all intervals and return the kth value.

Correctness of Select-Rank-Unit Algorithm. Observe that in Step 3, if the algorithm finds two such collections X and Y and an interval J between X and Y then one can construct a witness according to Proposition 3.1. Accordingly, $l(J)$ is an element whose rank must lie within the required range. Conversely, it is easy to see that if there exists some interval J lying between two witnessing collection of intervals, the algorithm is guaranteed to find it without any further probes. In step 3(a)(ii), we have a total of s intervals in X'. Let $I \in \{J_1, ..., J_m, J\}$ such that $a(I)$ is the largest among $a(J_1), ..., a(J_m), a(J)$. We claim that the rank of $a(I)$ must be at least s and at most t. Suppose that on the contrary, the rank of $a(I)$ is less than s. This can only happen if $a(I) < a(I')$ for some $I' \in X'$. If I and I' belong to the same partition S_i for some $i \in [1, k]$ then I' must lie to the right of I. Therefore $I \notin \{J_1, ..., J_m, J\}$ since it is not the rightmost interval. If I and I' belong to different partitions then either I' is to the right of I or I' overlaps with I. Note that I' cannot be to the left of I since $a(I') > a(I)$. Nevertheless, whether I' is to the right of I or I' overlaps with I, either I' is among $\{J_1, ..., J_m, J\}$ or there is some interval to the right of I' which is among $\{J_1, ..., J_m, J\}$. In either case, this implies that $a(I)$ cannot be the largest among $a(J_1), ..., a(J_m), a(J)$. Also, since there are $n-t$ intervals in Y and $r_{X'} \leq l_Y$, $a(I)$ has rank at most t. A similar argument can be given for the situation in Step 3(a)(iii). For Step 3(b), the correctness follows from Lemma 3.1.

We show next that Select-Rank-Unit is k-competitive and in fact, no online algorithm can achieve a competitive ratio better than k.

THEOREM 3.1. *Select-Rank-Unit is a k-competitive algorithm.*

PROOF. It is easy to see that if the optimal algorithm can present a witness without any probes then algorithm Select-Rank-Unit will also find the witnessing collections X and Y by Step 3 without any probes. In step 3(a)(ii) or 3(a)(iii), algorithm Select-Rank-Unit makes at most k probes in total to return an answer. For Step 3(b), we first claim that in this situation, the optimal algorithm requires at least 2 probes to return an answer. Since algorithm Select-Rank-Unit makes at most $2k-1$ probes in this case, the ratio is at most k. The proof of claim is shown in the appendix. □

THEOREM 3.2. *No online algorithm can achieve a competitive ratio better than k.*

PROOF. Consider the following instance of k input intervals where each interval has the range $[0, 2]$. $k-1$ of the intervals have 1 as their precise value and one interval has 0 as its precise value. To find the element with the minimum rank, the optimal algorithm requires only one probe. On the other hand, any online algorithm can be made to probe k intervals — by returning 1 for the first $k-1$ probes and 0 for the last probe. Observe that if each of the input intervals has range $[0, 1]$, the online algorithm will only require $k-1$ probes at most. □

3.1.2 The Arbitrary Cost Model

We now briefly describe an $O(k)$-competitive algorithm when each interval has an arbitrary cost.

Algorithm Select-Rank-Arb. We first determine if there is a dividing interval between $s-1$ intervals with leftmost right endpoints and $n-t$ intervals with rightmost left endpoints (as in steps 1 and 2 of algorithm Select-Rank-Unit). If there is such an interval then we return the left endpoint of this interval. Otherwise, we first identify two sets of intervals X' and Y' of size $\Theta(k)$ each such that every interval that is not in X' but is to the left of an interval in X' has rank less than s and every interval that is not in Y' but to the right of an interval in Y' has rank greater than t. X' and Y' can be found by invoking algorithm Weak-Rank with $p = s$ and $p = t$ respectively. Let $X = X' \cup X''$ where X'' contains the rightmost interval to the left of X' in each partition S_i. Similarly, let $Y = Y' \cup Y''$ where Y'' contains the leftmost interval to the right of Y' in each S_i. Observe that we have $\Theta(k)$ intervals in X and Y. We then probe the intervals in $X \cup Y$ in the order of non-decreasing costs until we find a witness.

THEOREM 3.3. *Select-Rank-Arb is an $O(k)$-competitive algorithm. Furthermore, no online algorithm can achieve a competitive ratio better than $\Omega(k)$.*

PROOF. Observe that if there is a dividing interval between the $s-1$ and $n-t$ selected intervals, the online algorithm does not require any probes and therefore performs as well as the optimal algorithm. Otherwise, we first show that the optimal algorithm will only probe intervals in $X \cup Y$. Suppose the optimal algorithm probed some interval I to the left of X in order to reveal an interval within the required rank range. Then the leftmost interval to the right of I and in X (call it I') must have a rank greater than s. This is a contradiction since $I' \in X''$ and intervals in X'' have rank strictly less than s. A similar argument can be given to show that the optimal algorithm never probes any interval beyond Y. Hence, the optimal algorithm only probes intervals in $X \cup Y$. The online algorithm probes intervals in $X \cup Y$ with smallest costs first, hence we obtain an $O(k)$-competitive algorithm. The lower bound is immediate from Theorem 3.2. □

3.2 Selection with Relative or Absolute Precision

Recall that to find an element of rank p within a relative precision of α is to return a value v so that the actual value deviates from v by at most a multiplicative factor of α where $\alpha \geq 1$. For absolute precision, the goal is to return a value v so that the actual value differs from v by at most α where $\alpha \geq 0$. It was observed by Feder *et al* in [6] that the pth smallest value is in the range $[l, r]$ where l is the pth smallest value among $l(I_1), ..., l(I_n)$ and r is the $(n-p+1)$th largest value among $r(I_1), ..., r(I_n)$.

As an example of selection with relative (absolute) precision, consider the intervals shown in Figure 1(a). Since the third largest element lies in the interval $[5, 10]$, 5 is the third largest element within a relative precision of 2. 5 is also the third largest element within an absolute precision of 5.

Our main results are $O(k)$-competitive algorithms for this problem in relative (absolute) precision in both cost models. We also show that this is asymptotically optimal in all cases.

3.2.1 The Unit Cost Model

We describe the online algorithm introduced in [6] for absolute precision model next and show that it can be easily adapted for relative precision.

Algorithm Select-Absolute-Unit. We let l be the pth largest left value among all left endpoints and let r be the $(n-p+1)$th largest right value among all right endpoints. If $r - l \leq \alpha$ where α is the required absolute precision then return l as the answer. Otherwise, probe an interval containing $[l, r]$ and repeat the algorithm. An interval I *contains* an interval J if $l(I) \leq l(J)$ and $r(I) \geq r(J)$. **Algorithm Select-Relative-Unit** is essentially the same as algorithm Select-Absolute-Unit with the condition $r - l \leq \alpha$ replaced with $r/l \leq \alpha$ where α is the required relative precision.

Theorem 3.4. *Algorithm Select-Relative-Unit (or Select-Absolute-Unit) is k-competitive.*

Proof. Consider the sequence $[l_1, r_1]$,...,$[l_s, r_s]$ of l and r values obtained by the online algorithm over the iterations. Let the sequence of intervals that are chosen to be probed be $J_1, J_2, ..., J_s$. In other words, for all $i \in [1, s]$, J_i contains $[l_i, r_i]$. Observe that for every J_i that is probed, $a(J_i) \geq r_i$ or $a(J_i) \leq l_i$ or $l_i < a(J_i) < r_i$. If $a(J_i) \geq r_i$ then the new pth smallest left value will be at least as large as before and the $(n-p+1)$th largest right value will remain the same ($l_i \leq l_{i+1}$ and $r_i = r_{i+1}$). If $a(J_i) \leq l_i$, the opposite happens ($l_i = l_{i+1}$ and $r_i \geq r_{i+1}$). If $l_i < a(J_i) < r_i$ then the pth smallest left value will be as least as large as before and the $(n-p+1)$th largest right value will be at least as small as before ($l_i \leq l_{i+1}$ and $r_i \geq r_{i+1}$). In all cases, $l_1 \leq l_2 \leq ... \leq l_s$ and $r_1 \geq r_2 \geq ... \geq r_s$ and clearly J_i overlaps J_j for all $i, j \in [1, s]$. Since we have a set of intervals with width k, s is at most k. □

Theorem 3.5. *In either preicision model, no online algorithm can achieve a competitive ratio better than k.*

Proof. In the relative precision model, consider the following instance of input intervals where k intervals have the range $[0, 1]$. Only one of the intervals has a precise value 0 and the rest of the intervals has precise value 1. To find the minimum element to within α relative precision (for any $\alpha \geq 1$), the optimal algorithm simply probes the interval with precise value 0 and returns 0 as the answer. On the other hand, any online algorithm can be made to probe k intervals, by returning 1 for the first $k-1$ probes and 0 for the last probe. For the absolute precision model, we can consider the same input instance with an absolute precision requirement of say, 0.5. As before, only one interval has precise value 0 and the rest of the intervals have precise value 1. To find the minimum element, the optimal algorithm requires only one probe – by probing the interval with precise value 0 and returns 0 as the answer. However, any online algorithm can be made to probe k intervals – by returning 1 for the first $k-1$ probes and finally 0. □

3.2.2 The Arbitrary Cost Model

We describe an $O(k)$-competitive algorithm for the selection problem in either precision model with arbitrary costs.

Algorithm Select-Absolute/Relative-Arb. We identify a set of intervals X' of size $\Theta(k)$ each such that every interval not in X' and to the left of an interval in X' has rank less than p and every interval not in X' and to the right of X' has rank greater than p. X' can be found be invoking algorithm Weak-Rank. Let $X = X' \cup X_l \cup X_r$ where X_l contains the rightmost interval to the left of X' in each S_i and X_r contains the leftmost interval to the right of X' in each S_i. We probe the intervals in X in the order of non-decreasing costs until the pth smallest left value and the $(n-p+1)$th largest right value is within the required accuracy.

Theorem 3.6. *Algorithm Select-Relative-Arb (or Select-Absolute-Arb) is $O(k)$-competitive. Moreover, in either precision model, no online algorithm can achieve a competitive ratio better than $\Omega(k)$.*

Proof. We first claim that the optimal algorithm does not probe any interval that is not in X. Suppose the optimal algorithm probes some interval I to the left of X so as to reveal the $(n-p+1)$th largest right value, then the leftmost interval to the right of I and in X (call it I') has a rank greater than p. This is a contradiction since $I' \in X_l$ and intervals in X_l have rank less than p. Suppose the optimal algorithm probes I to reveal the pth smallest left value, call it l_p. We show that l_p cannot be the pth smallest thus arriving at a contradiction. Let the interval to the right of I and in X_l be I_r. Since there is an interval I_r in X_l to the right of I, there can be at most $k-1$ intervals in X_l lying to the left of this value l_p. In addition, there can be at most another $k-1$ intervals from X' with their left value less than or equal to l_p. For each of these intervals from X', either it is a rightmost interval in X' or their right values must be greater than or equal to $r(I)$. If there were some interval $I' \in X'$ with $r(I') < r(I)$ and I' is not a rightmost interval in X' then I would have been selected to be in X' before I'. (Recall that Weak-Rank selects intervals with larger right values first to be in the set G.) Notice that there cannot be any intervals to the right of X' whose left value is less than l_p. Suppose there were such an interval I'', then $l(I'') \leq l_p < l(I_r)$. This implies that Weak-Rank would have selected I'' to be in the set M before I_r which is a contradiction since I'' is not in M. Hence we may assume that no intervals to the right of X' have left value less than or equal to l_p. Since are $p - 2k$ intervals to the left of X and one of them is used to reveal l_p, there can be at most $p-2k-1$ intervals with whose left value is less than l_p. Also, at worst $2k - 2$ intervals from X' have left values smaller than l_p. Therefore, l_p can have rank at most $p - 2$. A similar argument can be made to show that the algorithm does not probe any interval to the right of X. The lower bound is immediate from Theorem 3.5. □

4. SUM AND AVERAGE

The sum (average) problem is to find the sum (average) of n variables to within a specified precision. Since the average of n variables differs from their sum by a fixed factor of n, any results for one problem can be directly adapted for the other. So without any loss of generality we will restrict our discussion only to the sum problem.

4.1 The Unit Cost Model

Let $\{I_1, ..., I_n\}$ be the set of intervals associated with the underlying variables $x_1, ..., x_n$. Define $L = \sum_{i=1}^{n} l(I_i)$ and $R = \sum_{i=1}^{n} r(I_i)$. If the ratio $R/L \leq \alpha$, then clearly we can simply return L as the answer. On the other hand, if it is not so, we must continue probing intervals till this condition becomes true. Consider the following greedy online algorithm. At each step, find an interval I that maximizes $r(I) - l(I)$ and probe it. We now analyze the performance of this algorithm and show that it is essentially the best possible algorithm.

Fix an optimal strategy and consider the state of the online algorithm just before it makes its last probe. We partition all the intervals into four sets: X_1, X_2, X_3 and X_4 where $X_1 \cup X_2$ are the intervals probed by the online algorithm so far, $X_2 \cup X_3$ are the intervals probed by the optimal algorithm, and X_4 is the set of intervals that are not probed by either algorithm. Thus X_2 contains the intervals probed by both the online the optimal algorithms. We shall assume X_3 to be non-empty since if X_3 is empty, the online algorithm must have already reached its stopping condition.

Let $L(X)$, $P(X)$ and $R(X)$ denote $\sum_{I \in X} l(I)$, $\sum_{I \in X} a(I)$ and $\sum_{I \in X} r(I)$ respectively. Observe that $L(X) \leq P(X) \leq R(X)$. We know that the optimal algorithm ensures the following:

$$\begin{aligned} P(X_2) + P(X_3) + R(X_1) + R(X_4) \leq \\ \alpha(P(X_2) + P(X_3) + L(X_1) + L(X_4)) \end{aligned} \tag{1}$$

On the other hand, since the online algorithm is not yet done, it must be the case that

$$\begin{aligned} P(X_1) + P(X_2) + R(X_3) + R(X_4) > \\ \alpha(P(X_1) + P(X_2) + L(X_3) + L(X_4)) \end{aligned} \tag{2}$$

From (1), we get

$$\begin{aligned} R(X_1) - L(X_1) \leq &(\alpha - 1)(P(X_2) + P(X_3) + L(X_1)) + \\ &\alpha L(X_4) - R(X_4) \end{aligned} \tag{3}$$

Let $q = |X_1|/|X_3|$. By the greedy property of the online algorithm, it follows that the largest size interval in X_3 is at most the size of the smallest interval in X_1. Therefore, we have $R(X_1) - L(X_1) \geq q(R(X_3) - L(X_3))$. Substituting (3) into this and rearranging, we get

$$\begin{aligned} (\alpha - 1)P(X_2) \geq qR(X_3) - (\alpha - 1)P(X_3) - qL(X_3) - \\ (\alpha - 1)L(X_1) + R(X_4) - \alpha L(X_4) \end{aligned}$$

And since $P(X_3) \leq R(X_3)$ and $L(X_1) \leq P(X_1)$, we get

$$\begin{aligned} P(X_2) \geq &\frac{q - \alpha + 1}{\alpha - 1} R(X_3) - \frac{q}{\alpha - 1} L(X_3) - \\ &P(X_1) + \frac{1}{\alpha - 1} R(X_4) - \frac{\alpha}{\alpha - 1} L(X_4) \end{aligned} \tag{4}$$

Rearranging (2), we have

$$\begin{aligned} (\alpha - 1)P(X_2) < &R(X_3) - \alpha L(X_3) + P(X_1) - \\ &\alpha P(X_1) + R(X_4) - \alpha L(X_4) \end{aligned}$$

which becomes

$$\begin{aligned} P(X_2) < &\frac{1}{\alpha - 1} R(X_3) - \frac{\alpha}{\alpha - 1} L(X_3) - \\ &P(X_1) + \frac{1}{\alpha - 1} R(X_4) - \frac{\alpha}{\alpha - 1} L(X_4) \end{aligned} \tag{5}$$

By comparing the coefficients of (4) and (5), we can conclude that $q < \alpha$. Thus the total cost of the online algorithm can be bounded by $\alpha|X_3| + |X_2| + 1$. Comparing with the cost incurred by the optimal strategy, we can conclude that the online algorithm incurs a cost of at most $(\alpha + 1)$ times optimal. Thus we can conclude the following lemma.

LEMMA 4.1. *There is an $(\alpha+1)$-competitive algorithm for the sum problem with relative precision and unit costs.*

We can obtain better performance guarantees when the parameter α is large compared to the width k of the given instance.

LEMMA 4.2. *For any $\alpha \geq 2$, there is a k-competitive algorithm for the sum problem with relative precision and unit costs.*

PROOF. We fix a partition of the input instance into k sets $S_1, S_2, ..., S_k$ of non-overlapping intervals. Consider any set S_i, and let $[l_1, r_1]$,$[l_2, r_2]$,...,$[l_m, r_m]$ be the set of intervals in S_i where $r_j < l_{j+1}$ for $1 \leq j < m$. We claim that we can determine the sum of the variables underlying S_i to within a relative precision of 2 by simply probing the last interval $[l_m, r_m]$. To see this, observe that $(r_1 + ... + r_{m-1} + p_m)/(l_1 + ... + l_{m-1} + p_m) \leq 2$ where p_m denotes the precise value of the interval $[l_m, r_m]$. Hence with a single probe in each S_i, we can determine the overall sum to within a relative precision of 2. □

We can combine both algorithms above to obtain the following result and we show that the bound given is tight.

THEOREM 4.1. *There is an $O(\min\{\alpha, k\})$-competitive algorithm for the sum problem with relative precision and unit costs.*

PROOF. If $\alpha \leq k$ or $\alpha < 2$, one can use the greedy algorithm which achieves $(\alpha + 1)$-competitiveness. Otherwise, one can use the k-competitive strategy outlined in Lemma 4.2 to achieve a relative precision of 2. The theorem follows. ☐

THEOREM 4.2. *No online algorithm can achieve a competitive ratio better than $\Omega(\min\{\alpha, k\})$ for the sum problem with relative precision and unit costs.*

PROOF. Consider the following input instance which consists of $\min\{\alpha, k\}$ intervals, each with the range $[0, 1]$. Only one interval has precise value 1 and the rest have precise value 0.

When $\alpha \leq k$, the optimal algorithm probes the interval with precise value 1 and can immediately achieve the required precision of α. However, any online algorithm can be made to probe all α intervals — by returning 0 for the first $\alpha - 1$ probes and finally 1. When $\alpha > k$, the optimal algorithm makes one probe to the interval with precise value 1 to achieve a relative precision of k while any online algorithm can be made to probe all k intervals. ☐

4.2 The Arbitrary Cost Model

When each interval has an arbitrary associated cost, a natural extension of the greedy algorithm considered in the previous section is to probe intervals in non-increasing order of density. The *density* of an interval I is defined to be $(r(I) - l(I))/c(I)$. The algorithm greedily probes intervals in this order until the condition $\sum_I r(I)/\sum_I l(I) \leq \alpha$ is satisfied. However, it is easy to construct examples where this greedy approach leads to solutions that have an arbitrarily large cost compared to the optimal. The problem is only in the last interval choosen by the greedy algorithm. The following lemma formalizes this.

LEMMA 4.3. *Let $I_1, I_2, ..., I_p$ be the sequence of intervals probed by the online algorithm. Then $\sum_{i=1}^{p-1} c(I_i) \leq \alpha OPT$ where OPT is the cost incurred by an optimal algorithm.*

PROOF. The proof is similar to that of Lemma 4.1. In this case every interval in X_1 has a higher density than any interval in X_3. Since the smallest density interval in X_1 is at least as large as the largest density interval in X_3, we have $(R(X_1) - L(X_1))/C_1 \geq (R(X_3) - L(X_3))/C_3$ where C_1 and C_3 are the total costs of all intervals in X_1 and X_3 respectively. Let $C_1 = qC_3$ and from an analysis similar to one shown in Section 4.1, we have $q < \alpha$. Let C denote the total cost incurred by the online algorithm in probing $X_1 \cup X_2$ (all the intervals except the last interval I_p). Surely $C = C_1 + C_2 \leq \alpha C_3 + C_2$ where C_2 is the cost of intervals in X_2. Since $\text{OPT} = C_2 + C_3$, and $C \leq \alpha(C_3 + C_2)$, we have $C \leq \alpha\text{OPT}$. ☐

The lemma above suggests that the greedy strategy would work well if we could somehow bound the cost of the last interval probed by the algorithm. Suppose the online algorithm knew the optimal cost OPT. Then the greedy algorithm could simply discard all intervals of cost greater than OPT from consideration and using the lemma above, one can immediately conclude an $(\alpha+1)$-competitive algorithm. However, since the online algorithm does not know the optimal value, it needs to guess it in a careful manner. The following modified greedy algorithm is based on these ideas — it guesses the value of OPT iteratively. We let L and R denote the current sum of left and right endpoints respectively. L and R will reflect the new sum values at each iteration (since an interval may be probed at each iteration).

```
1. Initialize a counter $i$ with value 0.
2. Set OPT $= 2^i$.
3. If $R/L \leq \alpha$, return $L$ as the answer. Otherwise, find
   an unprobed interval $I$ of largest density such that
   $c(I) \leq$ OPT. If there is no such interval, increment
   the counter $i$ and goto step 2.
   (a) If the total cost (including the cost of $I$) exceeds
       $(\alpha + 1)$OPT then increment counter $i$ and goto
       step 2.
   (b) Otherwise, probe $I$ and goto step 3.
```

Suppose the α precision is satisfied when the counter value is j for some $j \geq 0$. Since the iteration terminates at j and not earlier, we know that $\text{OPT} > 2^{j-1}$. At each iteration i, the greedy algorithm incurs a cost of at most $(\alpha + 1)2^i$. Therefore the total cost incurred by the greedy algorithm is $\leq (\alpha + 1)(\sum_{i=0}^{j} 2^i) \leq (\alpha + 1)2^{j+1}$. Hence the ratio of total cost incurred by greedy algorithm to the optimal cost is at most $((\alpha + 1)(2^{j+1}))/2^{j-1} = 4(\alpha + 1)$, which gives the following lemma.

LEMMA 4.4. *There is a $4(\alpha+1)$-competitive algorithm for the sum problem with relative precision and arbitrary costs.*

We next explore if better performances can be achieved on restricted width instances. Unfortunately, in constrast to the unit cost model, even when $k = 1$, we can show a lower bound of $\Omega(\min\{\alpha, n\})$.

Consider the following set of n intervals of width 1: $[1, 2)$,$[2, 3)$,...,$[n - 1, n)$,$[n, H)$. Let $L = n(n - 1)/2$ $R = (n - 1)(n + 2)/2$ be the sum of the left and right values of the first $n - 1$ intervals. We define H to be $(\alpha - 1)L + (\alpha - 1)n + \alpha + 1$. Let the cost of the last interval be infinite while the cost of the each of the first $n - 1$ intervals is one. Furthermore, only one of the intervals, say I in the first $n - 1$ intervals has precise value $a(I) = r(I)$ while the rest have precise value $a(I) = l(I)$. It is easy to see that the optimal algorithm requires only one probe (by probing the interval with $a(I) = r(I)$) to obtain a precision of α since $(R + H)/(L + n + 1) \leq \alpha$. We now show that the online algorithm will be required to make $\min\{\alpha, n - 1\}$ probes. Let q denote the number of probes made by the online algorithm prior to the last probe. Hence $(R + H - q)/(L + n) > \alpha$ which implies that $q < \alpha$. Therefore, any online algorithm can be made to probe $\min\{\alpha, n - 1\}$ intervals in order to achieve α precision. It follows that no online algorithm can be better than $\Omega(\min\{\alpha, n\})$-competitive.

THEOREM 4.3. *No online algorithm can achieve a competitive ratio better than $\Omega(\min\{\alpha, n\})$ for the sum problem with relative precision and arbitrary costs.*

THEOREM 4.4. *There is an $O(\min\{\alpha, n\})$-competitive algorithm for the sum problem with relative precision and arbitrary costs.*

PROOF. When $n \geq \alpha$, we use the $4(\alpha+1)$-competitive described in Lemma 4.4. When $n < \alpha$, we probe the intervals in the order of non-decreasing costs until the α precision is achieved. $\square$

We next show that one can in fact obtain a bicriteria online algorithm which is $O(k)$-competitive when $k \leq \alpha$, to achieve a precision of $2\alpha + 2$.

LEMMA 4.5. *There is an $O(k)$-competitive algorithm for the sum problem with relative precision and arbitrary costs that achieves a precision of $2\alpha + 2$.*

PROOF. As before, we assume that there is a partition of intervals into k sets of non-overlapping intervals, $S_1, ..., S_k$. Let OPT denote the cost incurred by the optimal algorithm. Let X be the set of intervals each with cost less than or equal to OPT and Y be the rest of the intervals. Observe that the set of intervals probed by the optimal algorithm comes from X. Let X_1 denote the set of rightmost intervals in $S_i \cap X$ and let $X_2 = X - X_1$. Note that $|X_1| = k$ and as observed in Lemma 4.2, $(P(X_1) + R(X_2))/(P(X_1) + L(X_2)) \leq 2$. We show next that this simple approach allows us to obtain a precision that is less than or equal to $2\alpha + 2$.

Observe that the precision obtained by our algorithm is

$$\frac{P(X_1) + R(X_2) + R(Y)}{P(X_1) + L(X_2) + L(Y)} \leq 2 + \frac{R(Y)}{P(X_1) + L(X_2) + L(Y)}$$

A lower bound on the precision that can be obtained by probing all intervals in X_2 is

$$\frac{P(X_1) + L(X_2) + R(Y)}{P(X_1) + R(X_2) + L(Y)} \geq \frac{R(Y)}{2(P(X_1) + L(X_2) + L(Y))}$$

Let α^* denote the precision obtained by the optimal algorithm. α^* is no less than the above precision. Hence, it is easy to see that the precision obtained by online algorithm is $\leq 2\alpha^* + 2 \leq 2\alpha + 2$.

The above procedure suggests that one can obtain an $O(k)$-competitive algorithm to achieve a precision to within $2\alpha+2$ if one knows the value OPT. As before, we can guess OPT iteratively, setting OPT to be increasing powers of 2. We first check if the intervals are already α-precise. If not, at each iteration, we check if $(P(X_1) + R(X_2) + R(Y))/(P(X_1) + L(X_2) + L(Y)) \leq 2\alpha + 2$. If the condition is true, we return $P(X_1) + L(X_2) + L(Y)$ as the answer. Otherwise, we increase OPT by a factor of 2 and repeat the process. The total cost incurred by our online algorithm over all iterations is to within $O(k)$ of the optimal. Assume that the algorithm stops at the iteration where OPT is guessed to be 2^j for some j. This means that OPT $> 2^{j-1}$. The cost incurred in iteration i is bounded by $k2^i$ since we probe only intervals in X_1. And since there are j iterations, the total cost incurred is at most $k2^{j+1}$. The ratio when compared with the optimal is thus $\leq 4k$. $\square$

THEOREM 4.5. *There is an $O(\min\{\alpha, k\})$-competitive algorithm for the sum problem with relative precision and arbitrary costs that achieves a relative precision of $2\alpha + 2$.*

PROOF. The theorem is immediate from the previous two lemmas. When $k > \alpha$, we use the $4(\alpha+1)$-competitive algorithm that achieves a precision of α as observed in Lemma 4.4. If $k \leq \alpha$, we use the $O(k)$-competitive algorithm described in Lemma 4.5. $\square$

5. COMPOSED FUNCTIONS

We now briefly describe some results for functions that are obtained by composing together two functions f and g where f and g can be one of the following functions: min, max, sum and average. We consider compositions $f \circ g$ of the form $f(g(x_{11}, ..., x_{1n_1}), ..., g(x_{m1}, ..., x_{mn_m}))$ where f is said to be the outer operator and g is said to be the inner operator. We study here only the *unit cost* case. In what follows, we let G_i denote the set of intervals $\{x_{i1}, ..., x_{in_i}\}$ and $n = n_1 + ... + n_m$, the total number of intervals. As before, we assume a partition of all intervals into sets $S_1, ..., S_k$.

Min∘Min/Max∘Max/Sum∘Sum/Avg∘Sum. When f as well as g are both the operator min (max), it is equivalent to a single application of min (max) and hence identical results apply. When f is either the average or sum operator and g is the sum operator, it is easy to see that this is the same as a direct application of the sum operator and hence earlier results apply as well.

Avg∘Avg/Sum∘Avg. When both f and g are the average operator, we describe an $O(\min\{\alpha, k \log n\})$-competitive algorithm for this problem. The lower bound is $\Omega(\min\{\alpha, k\})$ and it remains open whether there is an algorithm which is $O(\min\{\alpha, k\})$-competitive. We will use the observation that avg(avg($x_{11}, ..., x_{1n_1}$),...,avg($x_{m1}, ..., x_{mn_m}$)) is equivalent to sum($x_{11}/(n_1 * m)$,...,$x_{1n_1}/(n_1 * m)$,..., $x_{m1}/(n_m * m)$,...$x_{mn_m}/(n_m * m)$). If $\alpha < k \log n$ or $\alpha < 4$, we use the $(\alpha+1)$-competitive algorithm as described in Lemma 4.1 for the sum problem on the scaled inputs. Otherwise, we use the following approach. We partition the n intervals into clusters C_1,...,$C_{\log n}$. such that $C_i = \{\bigcup G_j \mid 2^i \leq |G_j| < 2^{i+1}\}$. For the set of intervals in C_i, we probe the rightmost interval in each $C_i \cap S_j$ for $1 \leq j \leq k$. This allows us to estimate sum(C_i) within a precision of 2. We next scale down our estimate of sum(C_i) by $2^{i+1} * m$. It is easy to see that this gives us an estimate of the average with precision 4. Combining together these scaled estimates for all clusters, we obtain an estimate of the overall average with precision 4. When f is the sum operator and g is the average operator, the same procedure can be used except that our estimate of of sum(C_i) is scaled by only 2^{i+1}.

Max∘Min/Min∘Max. If f is the max operator and g is the min operator (or vice versa), the problem becomes

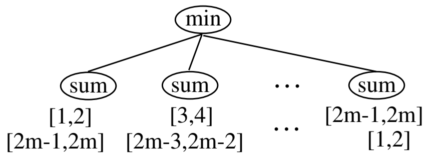


Figure 3: A width-2 input instance to min∘sum **function.**

somewhat difficult and we give an $O(k^2)$-competitive algorithm for this case. It is easy to see that the lower bound is $\Omega(k)$ and it remains open whether there exists an $O(k)$-competitive algorithm for this problem.

For each G_i, we let r_i denote the smallest right value among intervals in G_i. Let $G'_i = \{[l(I), r_i] \mid I \in G_i, l(I) \leq r_i\}$. Observe that $|G'_i| \leq k$ and the minimum value in G'_i is the same as the minimum value in G_i. Intervals in G'_i are such that the left value is less than or equal to r_i and the right value of each interval never exceeds r_i. Without loss of generality, we may assume G'_is are indexed in non-decreasing order of r_i. Let $G = \{G'_1, ..., G'_m\}$. Consider any pair of elements in G, say G'_i and G'_j where $i < j$. If the smallest left value in G'_j is strictly greater than r_i, we discard G'_i from G. We repeat the process until no more elements from G can be discarded. After this process, $|G| \leq k$. For if this is not the case, the width of the input intervals must be more than k which is a contradiction. Since for each $G'_i \in G$, $|G'_i| \leq k$ and $|G| \leq k$, we have a total of $O(k^2)$ remaining intervals. For each $G'_i \in G$, probe every interval $I \in G'_i$ and return the smallest element. Among all the smallest elements, we return the largest one.

Sum∘Min/Sum∘Max/Avg∘Min/Avg∘Max. When f is an aggregate operator and g is either min or max, we can also give an $O(k^2)$-competitive algorithm using similar ideas described earlier. Assume that f is the sum operator, g is the min operator and we have $G'_1, ..., G'_m$ as described before. Let l_i and r_i denote the leftmost and rightmost values for intervals in G'_i. Observe that we can view the input to sum function as simply the set of m intervals $[l_1, r_1]$,...,$[l_m, r_m]$. If $\alpha \geq 2$, we do the following: Consider a partition of intervals $[l_1, r_1]$,...,$[l_m, r_m]$ into at most k non-overlapping sets $S_1, ..., S_k$ (since the width of the original instance is k). For the rightmost interval in each S_i, say $[l_j, r_j]$, we probe all intervals in G'_j and return the minimum element. Since the size of G'_j is at most k, we probe at most k^2 intervals in this way. This gives an estimate of the sum with precision 2 as described in Lemma 4.2. If $\alpha < 2$, we view this as a sum problem with unit costs and apply the greedy strategy as described Lemma 4.1. However, whenever we pick an interval to probe, say $[l_j, r_j]$, we now need to probe all intervals in G'_j and return the minimum element. We get an $O(\alpha k)$-competitive algorithm and since $\alpha < 2$, this is clearly $O(k^2)$-competitive.

Min∘Sum/Min∘Avg/Max∘Sum/Max∘Avg. Finally, when f is either the operator min or max and g is an aggregate operator, then we can show that no online algorithm can be $o(n)$-competitive even when the input instance is of width 2. To see this, consider the example shown in Figure 3 where f is min and g is sum and the set of input intervals has width 2. The result of each sum node is an interval $[2m, 2m+2]$. Since all resulting intervals are the same and there are m sum nodes, the input to the min operator is an instance of width $m = \Omega(n)$. It is easy to see that an adversarial strategy can now be used to make the online algorithm make $\Omega(n)$ probes whereas the optimal can obtain the minimum in $O(1)$ probes.

6. REFERENCES

[1] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.

[2] M. J. Carey, M. J. Franklin, M. Livny, and E. J. Shekita. Data caching tradeoffs in client-server dbms architectures. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 357–366, 1991.

[3] S. Chaudhuri and L. Gravano. Evaluating top-k selection queries. In *Proc. of 25th Intl. Conf. on Very Large Data Bases (VLDB)*, pages 397–410, 1999.

[4] R. Fagin. Combining fuzzy information from multiple systems. In *Proc. of the 15th Symposium on Principles of Database Systems (PODS)*, pages 216–226, 1996.

[5] R. Fagin. Fuzzy queries in multimedia database systems. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 1–10, 1998.

[6] T. Feder, R. Motwani, R. Panigrahy, C. Olsten, and J. Widom. Computing the median with uncertainty. In *Proc. of the 32nd Symposium on Theory of Computing*, pages 602–607, 2000.

[7] H. Garcia-Molina, S. Ketchpel, and N. Shivakumar. Safeguarding and charging for information on the internet. In *Proc. of the 14th Intl. Conf. on Data Engineering*, 1998.

[8] M. C. Golumbic. *Algorithmic Graph Theory.* Academic Press, 1980.

[9] C. Olston, B. T. Loo, and J. Widom. Adaptive precision setting for cached approximate values. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, 2001.

[10] C. Olston and J. Widom. Offering a precision-performance tradeoff for aggregation queries over replicated data. In *Proc. of 26th Intl. Conf. on Very Large Data Bases*, pages 144–155, 2000.

Appendix to Section 3.

PROPOSITION 3.1. *Let $L_J = \{I \mid I \neq J$ and $r(I) \leq l(J)\}$ and $R_J = \{I \mid I \neq J$ and $l(I) \geq r(J)\}$ for some interval J. Then $a(J)$ is an element whose rank lies in the range $[s, t]$ if $|L_J| \geq s-1$ and $|R_J| \geq n-t$.*

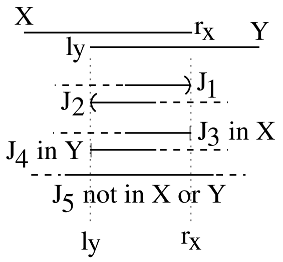
PROOF. Suppose J is an interval such that $|L_J| \geq s-1$ and $|R_J| \geq n-t$. We show that the rank of $a(J)$ must be within $[s, t]$. Let Z be the set of intervals not in $L_J \cup R_J \cup \{J\}$. For every $I \in Z$, $r(I) > l(J)$ and $l(I) < r(J)$. In one extreme setting, the precise values of all intervals in Z are greater than $a(J)$. However, the rank of $a(J)$ in this case is $|L_J|+1$ which is within the required range. Conversely, all intervals in Z could have precise values less than $a(J)$. The rank of $a(J)$ in this case is $|L_J|+|Z|+1 = |L_J|+n-|L_J|-|R_J|-1+1 = n-|R_J| \leq t$. $\square$

LEMMA 3.1. *If $p \leq k$ then the pth largest element in the original instance is the pth largest element in $F \cup G$. Otherwise, the pth largest element in the original instance is the kth largest element in $F \cup G$.*

PROOF. Let U denote the set of n intervals. Let I be an interval in $U-(L \cup M)$ and let $M_1, M_2, ..., M_{p-1}$ be the order in which the intervals in M are selected. This means that $l(M_1) \leq l(M_2) \leq .... \leq l(M_{p-1})$. Since I is not selected to be in M, $l(M_{p-1}) \leq l(I)$. Consider the interval graph of $M \cup \{I\}$ and let m be the size of the maximum clique that includes I in this interval graph. If $m=1$ (I does not overlap with any interval in M) then $l(I) > r_M$ where r_M is the largest right value of the intervals in M. Hence all intervals in M are to the left of I. Since I belongs to S_i for some $i \in [1,k]$, there is at least one more interval to the left of I in $L \cap S_i$. Hence the rank of I is more than $|M|+1=p$. If $m>1$ then let $\{I, J_1, ..., J_{m-1}\}$ be the set of intervals in $M \cup \{I\}$ that overlaps with one another. Surely $l(J_j) \leq l(I)$. And since there is an interval in $L \cap S_i$ that lies to the left of I and an interval in $L \cap S_j$ to the left of J_j for every $j \in [1, m-1]$, there must be m intervals to the left of I. Also since I does not overlap with the rest of the intervals in M ($|M|-(m-1)$ of them), there is an additional $p-m$ intervals to the left of I. Therefore there is a total of $m+p-m$ intervals to the left of I giving I a rank of more than p.

Suppose $p>k$. Let R denote the set of intervals in $L \cup M-(F \cup G)$. Observe that that $|R|=p-k$. We know that pth element must be among $L \cup M$ from the argument above. Next, we show that any element in R has rank less than p. Therefore the pth element must be among $F \cup G$. Let I be the interval in R that contains the largest precise value. We show that the rank of $a(I)$ can be $p-1$ at most. Let the partition which I belongs to be S_i for some $i \in [1,k]$. We claim that for every S_j such that $j \neq i$, there can be at most one interval $J \in S_j \cap (U-R)$ such that $a(J) \leq a(I)$. Suppose on the contrary that there are two intervals J_1 and J_2 in $S_j \cap (U-R)$ such that $a(J_1) \leq a(I)$ and $a(J_2) \leq a(I)$. Without loss of generality, we may assume J_1 is to the left of J_2. Observe that since $a(J_2) \leq a(I)$ and J_1 precedes J_2, $r(J_1) < r(I)$ and $l(J_2) \leq r(I)$. Observe also since $J_1 \in U-R$, $J_1 \in G$ or $J_1 \in F$ or $J_1 \in (U-(L \cup M))$. If $J_1 \in G$ then $I \in G$ (recall that we select intervals with the largest right value first in G) contradicting the assumption that $I \notin F \cup G$. If $J_1 \in F$ then $J_2 \in (U-(L \cup M))$. And since $I \in R$, there must be an interval I' to the right of I in $S_i \cap F$. This implies $l(J_2) \leq l(I')$ and therefore if $I' \in M$ then $J_2 \in M$ contradicting the assumption that $J_2 \in (U-(L \cup M))$. If $J_1 \in (U-(L \cup M))$ then $J_2 \in (U-(L \cup M))$. From the same argument as before, we conclude that $J_1, J_2 \in (L \cup M)$ contradicting the assumption that $J_1, J_2 \notin (L \cup M)$. Hence there are at most $k-1$ elements from $U-R$ to the left of $a(I)$. Since there are $p-k$ elements in R. The rank of $a(I)$ is at most $k-1+p-k=p-1$. Observe that since every element in R has rank at most $p-1$ and there are $p-k$ of them, the kth largest in $F \cup G$ must be the pth largest in $L \cup M$. $\square$

Claim. In the situation where $r_X > l_Y$, the optimal algorithm needs more than one probe.



PROOF. We show that the optimal algorithm uses more than one probe by showing that the optimal algorithm cannot produce a witness in just one probe. This is shown by case analysis on all types of interval that the optimal algorithm may probe. Throughout the rest of the discussion, we use J to denote the first interval probed by the optimal algorithm. In addition, we use X', Y' to denote the new collection of intervals obtained by Select-Rank-Unit in steps (1) and (2) respectively, after J has been probed. Also, let $r_{X'}$, $l_{Y'}$ denote the new rightmost and leftmost boundaries of X' and Y' respectively. For convenience, we will also let J' to denote the probed interval J. In other words, J'=$[a(J), a(J)]$. We say the boundaries of X' and Y' *crosses* if $r_{X'} > l_{Y'}$ and there is a *gap* between X' and Y' if $r_{X'} \leq l_{Y'}$.

- Suppose the optimal algorithm probes an interval J of type J_1 as shown in the figure above. Intervals of type J_1 have right values strictly less than r_X. From the way intervals are selected to be in X, we are certain that $J \in X$ since $r(J) < r_X$. Then $X' = X - \{J\} \cup \{J'\}$. Since the newly selected set of intervals X' remain the same, $Y' = Y$. Therefore the boundaries remain crossed since $r_{X'} > l_{Y'}$.
- Suppose the optimal algorithm probes an interval J of type J_2. Intervals of type J_2 have left values strictly greater than l_Y and right values greater than or equal to r_X, i.e., $l(J) > l_Y$ and $r(J) \geq r_X$. If $r(J) < r_X$, we may think of J as a type J_1 interval. If $r(J)$ is strictly greater than r_X, i.e., $r(J) > r_X$ then $J \in Y$. In addition,
  - if $a(J) \geq r_X$ then $X' = X$ and $Y' = Y - \{J\} \cup \{J'\}$. The new collect of intervals X' is the same as the old collection and Y' is essentially the same as Y except that J probed to become J'. Therefore $r_{X'} > l_{Y'}$ and hence the boundaries remain crossed.
  - if $a(J) < r_X$, X' and Y' change in the following way. For X, it discards a rightmost interval, say K, and includes some the probed interval J', i.e., $X' = X - \{K\} \cup \{J'\}$. For Y, it loses J and therefore has to acquire some new interval K', ie, $Y' = Y - \{J\} \cup \{K'\}$. As a result, the new boundary of X' may shrink, i.e., $r_{X'} \leq r_X$. However, since $l_Y < a(J) <$

r_X, $r_{X'} > l_Y$ and $l_{Y'} \leq l_Y$ whether K' is K or some interval not previously acquired. Therefore the boundaries remain crossed since $r_{X'} > l_Y$.

The case when $l(J) > l_Y$ and $r(J) = r_X$ is subsumed in the following discussion.

- Suppose the optimal algorithm probes an interval J of type J_3 where where $r(J) = r_X$ and $l(J) \leq r_X$ and $J \in X$. If a gap is produced in X' and Y' after probing J, we show that there cannot be a dividing interval $I \notin X' \cup Y'$ lying within this gap.
  - If $a(J) < r_X$ then $X' = X - \{J\} \cup \{J'\}$ and $Y' = Y$. The probed interval "remains" in the collection X' and Y' is the same as Y. Hence $l_{Y'} = l_Y$ and a gap can be produce when $r_{X'} \leq l_Y$. If there is a dividing interval $I \notin X' \cup Y'$ lying within this gap then I must be in X and therefore I is also in X' contradicting the assumption that $I \notin X' \cup Y'$.
  - If $a(J) = r_X$ then $r_{X'} = r_X$. If $X' = X - \{J\} \cup \{J'\}$, we have $Y' = Y$ and the boundaries will remain crossed. However, if $X' = X - \{J\} \cup \{K\}$ (since K is selected to be in X' in preference over J', K must be such that $r(K) = r_X$ and $l(K) < r_X$ and $K \notin X$), l_Y may shift right because Y' may include J' since it is now not included in X', i.e., $Y' = Y - \{K'\} \cup \{J'\}$ where K' is a leftmost interval in Y. In this situation, we may have $r_X = l_{Y'}$ producing a gap. If there is a dividing interval $I \notin X' \cup Y'$, with $l(I) = r(I) = r_{X'}$. It must be that $I \in Y - \{K'\}$ Therefore $I \in X' \cup Y'$ arriving at a contradiction.
- We show a similar argument when the optimal algorithm probes an interval of type J_4 where $l(J) = l_Y$ and $r(J) \geq l_Y$ and $J \in Y$. If a gap is produced after probing J, we show that there cannot be a dividing interval $I \notin X' \cup Y'$ lying within the gap.
  - Suppose $a(J) \geq r_X$ then $X' = X$ and therefore $r_{X'} = r_X$. Also, $Y' = Y - \{J\} \cup \{J'\}$. Suppose $l_{Y'} \geq r_X$ and there is an interval $I \notin X' \cup Y'$ which lies within the gap. Then $I \in Y$ implying $I \in X' \cup Y'$ which is a contradiction.
  - If $l_Y \leq a(J) < r_X$ and a gap is produced then $X' = X - \{K\} \cup \{J'\}$ and $Y' = Y - \{J\} \cup \{K'\}$ where K is a rightmost interval in X and K' is a new included interval. Notice that J is dropped by Y and included in X'. If $l_{Y'} < r_X$ then any interval I which lies within the gap must be in $X - \{K\}$ and therefore in $X' \cup Y'$ (contradiction). If $l_{Y'} = r_X$ it must be that $l(K') = r(K') = r_X$ and $K' = K$. The newly included interval in Y' is the discarded interval of X. Observe that since K is a point interval originally in X then any interval I with $r(I) = r_X$ and $l(I) < r_X$ must be in $X - \{K\}$ and therefore in $X' \cup Y'$. If $r(I) = l(I) = r_X$ then $I \in Y - \{J\}$ and therefore in $X' \cup Y'$.
- Suppose the optimal algorithm probes an interval J not in $X \cup Y$ where $r(J) \geq r_X$ and $l(J) \leq l_Y$ (type J_5 intervals).
  - If $a(J) < r_X$, we have $X' = X - \{K\} \cup \{J'\}$ where K is a rightmost interval in X. If $Y' = Y$ then a gap can only be produced if $r_{X'} \leq l_{Y'} = l_Y$. In this situation any interval in this gap must belong to $X - \{K\}$ and therefore belong to $X' \cup Y'$. Now suppose $Y' = Y - \{K'\} \cup \{K''\}$ where K' is a leftmost interval in Y and K'' is some newly included interval previously not in Y. If $l_{Y'} < r_X$ and a gap is present then any interval I which lies within the gap must be in $X - \{K\}$. If $l_{Y'} = r_X$ then it must be that $l(K) = r(K) = r_X$ and $K = K''$. The interval discarded by X is the newly included interval in Y'. Observe that since K is a point interval originally in X then any interval I with $r(I) = r_X$ and $l(I) < r_X$ must also be in X (since the algorithm would have selected I before K to be in X). Thus any interval that lies within the gap must belong to X or $Y - \{K'\}$ and therefore belong to $X' \cup Y'$. Observe that $l_{Y'} \leq r_X$ since any interval excluded from X cannot "help" increase the value of $l_{Y'}$ beyond the point r_X.
  - If $a(J) \geq r_X$ then $X' = X$ and $Y' = Y - \{K\} \cup \{J'\}$ where K is a leftmost interval in Y. Since $X = X'$ then $r_{X'} = r_X$. A gap is produced only when $l_{Y'} \geq r_X$. Let I be an interval lying within the gap. If $l(I) = r(I) = r_X$ then either $I \in X$ or $I \in Y - \{K\}$. If $l(I) = r_X$ and $r(I) > r_X$ then $I \in Y - \{K\}$. Therefore $I \in X' \cup Y'$.

□