Computing the Median with Uncertainty

Tomas Feder* Rajeev Motwani† Rina Panigrahy‡ Chris Olston§
Jennifer Widom¶

Abstract

We consider a new model for computing with uncertainty. It is desired to compute a function $f(X_1, \ldots, X_n)$ where $X_1, \ldots, X_n$ are unknown, but guaranteed to lie in specified intervals $I_1, \ldots, I_n$. It is possible to query the precise value of any X_j at a cost c_j. The goal is to pin down the value of f to within a precision δ at a minimum possible cost. We focus on the selection function f which returns the value of the kth smallest argument. We present optimal offline and online algorithms for this problem.

1 Introduction

Consider the following model for computing with uncertainty. We wish to compute a function $f(X_1, \ldots, X_n)$ over n real-valued arguments. The values of the variables X_1, …, X_n are not known in advance; however, we are provided with real intervals $I_1, \ldots, I_n$ along with a guarantee that for each j, $X_j \in I_j$. Furthermore, it is possible to query the true value x_j of each X_j at a cost c_j. The goal is to pin down the value of f into an interval of size $\delta \geq 0$. Thus, we are faced with the following optimization problem: *Given a function f, precision parameter δ, real intervals $I_1, \ldots, I_n$, and query costs $c_1, \ldots, c_n$, pin down the value of f to an interval of size δ using a set of queries of minimum total cost.* Note that there are two natural versions of this problem: in the *online* version, the sequence of queries is chosen *adaptively* in that each successive query is answered before the next one is chosen; and, the *offline* version, where the entire set of queries must be specified completely before the answers are provided and it must be guaranteed that f can be pinned-down as desired regardless of the results of the queries.

This model is motivated[1] by the work of Olston and Widom [2] on query processing over replicated databases, where local cached copies of databases are used to support quick processing of queries at client sites. There is a master copy of the data where all the updates to the database are maintained. The frequency of updates makes it infeasible to maintain consistency between the cached copies and the master copy, and the data values in the cache are likely to become stale and drift from the master values. However, the cached copies store for each data value an interval that is guaranteed to contain the master value. In processing an aggregation query at a client cache, it is desired to compute a function f defined over the data values to within a specified precision δ. For each data value X_j, it is possible to query the master copy for the exact value at a cost c_j, rather than using the interval I_j at the cached copy. The goal then is to compute the result of an aggregation query to within the desired precision at minimum possible cost. Systems considerations sometimes make it desirable to perform all queries to the master copy en masse, motivating the offline version of the problem.

Olston and Widom [2] considered functions f which are simple aggregation functions, including:

$$\begin{aligned} \mathrm{SUM}(f(X_1, \ldots, X_n) &= \sum_{j=1,n} X_j); \\ \mathrm{MIN}(f(X_1, \ldots, X_n) &= \min_{j=1,n} X_j); \text{ and,} \\ \mathrm{MAX}(f(X_1, \ldots, X_n) &= \max_{j=1,n} X_j). \end{aligned}$$

It was observed that the SUM problem is isomorphic to the knapsack problem [1]: Consider the set of X_j's that are *not* queried; clearly, the sum of the corresponding interval sizes must be at most δ and the objective function is equivalent to maximizing the corresponding costs. The case of the selection function, i.e., where the function f returns the value of its kth smallest argument, and particularly the median, was left open.

In this paper, we resolve the complexity of the problem of computing the median (and, in general, the kth smallest element) under the model of computing with uncertainty, in both the offline and the online settings. We begin by expressing the offline selection problem in terms of an integer

*E-mail: tomas@theory.stanford.edu.

†Department of Computer Science, Stanford University, CA 94305. E-mail: rajeev@cs.stanford.edu. Supported by ARO MURI Grant DAAH04–96–1–0007 and NSF Grant IIS–9811904.

‡Cisco Systems, San Jose, CA. E-mail: rinap@cisc o.com.

§Department of Computer Science, Stanford University, CA 94305. Supported by NSF Grant IIS–9811947 and an NSF Graduate Research Fellowship. E-mail: olston@cs.stanford.edu.

¶Department of Computer Science, Stanford University, CA 94305. E-mail: widom@cs.stanford.edu. Supported by NSF Grant IIS–9811947.

[1] There are several other applications of this model, e.g., in numerical computation, that we will discuss in the final version.

linear program (Section 3). Not only is this integer program's structure critical to the development of our offline algorithms, but it also helps provide a useful lower bound for the online selection problem. Based on these insights, we provide a polynomial-time algorithm for the offline case with unit costs and a general offline algorithm with running time exponential in k (Section 4). Then we apply this tool to the development of an optimal online algorithm and provide a tight relationship between the performance of the optimal online and offline algorithms (Section 5). We extend our results to obtaining a polynomial-time algorithm for the general offline case based on the use of linear programming (Section 6). We also present a simple approximation algorithm that does not rely on linear programming. Finally, we define a class of problems called *interval problems* and show that this is equivalent to the offline median problem, and includes weighted bipartite matching as a special case (Section 7). This suggests that there is unlikely to be a simple combinatorial algorithm for the offline selection problem. As further evidence, we demonstrate that a mild generalization thereof is NP-hard.

2 Preliminaries

An instance of the *selection problem* consists of n intervals $I_1, I_2, \ldots, I_n$, an associated real cost $c_j \geq 0$ for each interval $I_j = [l(I_j), r(I_j)]$, an integer $1 \leq k \leq \lceil \frac{n}{2} \rceil$, and a value $\delta \geq 0$. Each interval I_j has an unknown point p_j. The aim is to estimate the kth smallest p_j with precision δ. An algorithm can query an interval I_j, paying cost c_j, and obtain the point p_j. The interval I_j is then replaced with the interval $I'_j = [p_j, p_j]$. At any point in time, for the current intervals I_j, the kth smallest point can be pinned down into an interval $[l, r]$:

Lemma 1 *Given the intervals I_j, the kth smallest point must lie in the interval $[l, r]$, where l is the kth smallest $l(I_j)$, and r is the $(n-k+1)$th largest $r(I_j)$.*

Note that this is the smallest interval to which the kth smallest point can be pinned, in absence of any other information.

When the algorithm terminates, we must have $r - l \leq \delta$. We seek an algorithm that minimizes the worst-case total cost to achieve this bound. In the unit-cost case, all $c_j = 1$, and the aim is to minimize the number of intervals queried. Two special cases of interest seek the smallest element, with $k = 1$, or the median of the elements, with n odd and $k = \lceil \frac{n}{2} \rceil$. The problem as described above is an online problem, that is, we can use the points returned by previous queries to decide which interval to query next. We are also interested in the offline problem, where the algorithm must decide which intervals to query at the same time, and guarantee that regardless of the points obtained in these queries, the estimate $[l, r]$ will have $r - l \leq \delta$.

3 An Integer Programming Formulation

We shall express the offline problem as an integer linear program. The structure of this integer program is critical to the development of our algorithms; also, we obtain from the constraints of this integer program a lower bound on the worst-case online cost. The integer program has a variable $x_j \in \{0, 1\}$ that expresses whether the interval I_j is queried. The aim is then to minimize $\sum c_j x_j$. To describe the constraints, we introduce some terminology. Consider an interval $O = [l(O), r(O)]$ of size $|O| = r(O) - l(O) > \delta$. Let a be the number of I_j with $l(I_j) \leq l(O)$, and let b be the number of I_j with $r(I_j) \geq r(O)$. If $a \geq k$ and $b \geq n - k + 1$, then we say that O is an *obstruction.* We shall show that, for all obstructions O, the offline algorithm must satisfy $\sum x_j \geq a + b - n$, where the sum is over the x_j such that I_j contains O. That is, at least $a + b - n$ intervals containing O must be queried. Furthermore, satisfying these constraints guarantees the bound δ on the final estimate.

To obtain a finite number of constraints, we introduce some additional terminology. A *proper obstruction* is an obstruction O such that for some input intervals $I_j, I_{j'}$, we have $l(O) = l(I_j)$ and $r(O) = r(I_{j'})$. A *minimal proper obstruction* is a proper obstruction that does not contain any other proper obstructions. The integer program is then to minimize $\sum c_j x_j$, with $x_j \in \{0, 1\}$, subject to the constraints, for each minimal proper obstruction O_i, that $\sum x_j \geq a_i + b_i - n$. where the sum is over the x_j such that I_j contains O_i. Notice that there are at most k minimal proper obstructions, since $r(O_i)$ must be one of the k smallest $r(I_j)$.

It will sometimes be convenient, for an obstruction O, to write $a + b - n = d - e$, where d is the number of I_j containing O, and e is the number of I_j inside O, that is, with $l(I_j) > l(O)$ and $r(I_j) < r(O)$. That is, the number of intervals containing O not queried is at most the number of intervals inside O. To see why this equality holds, write it as $n = a + b - d + e$; that is, the n intervals can be counted as a intervals with $l(I_j) \leq l(O)$, b intervals with $r(I_j) \geq r(O)$, subtracting the d intervals that satisfy both conditions, and adding the e intervals that satisfy neither condition.

For the online problem, we let V be the maximum over all minimal proper obstructions O_i of the minimum of $\sum c_j x_j$ with $\sum x_j = a_i + b_i - n$, where the sums are over the x_j with I_j containing O_i. That is, for each minimal proper obstruction O_i, we determine the sum of the $(a_i + b_i - n)$ smallest costs of intervals containing O_i, and find the worst such O_i.

Proposition 1 *The integer program solves the offline selection problem. The quantity V is a lower bound on the maximum total cost for the online selection problem.*

Proof Sketch: Say that a choice of queried intervals I_j *clears* an obstruction O if, regardless of the points p_j returned for the queried intervals, the interval O will no longer be an obstruction after the queried I_j are replaced by $I'_j = [p_j, p_j]$. We show that O is cleared if and only if $\sum x_j \geq a + b - n$, where the sum is over the intervals I_j containing O.

If $\sum x_j \leq a + b - n - 1 = (a - k) + (b - (n - k + 1))$, then for each I_j queried not containing O, we can return $p_j = l(I_j)$ if $r(I_j) < r(O)$, and return $p_j = r(I_j)$ otherwise, with $l(I_j) > l(O)$. Clearly, this choice of p_j does not decrease the values a, b for O. For the I_j queried containing O, we can return $p_j = l(I_j)$ for at most $b - (n - k + 1)$ of them, and return $p_j = r(I_j)$ for at most $a - k$ of them. Then O will still be an obstruction, since there will be still at least k intervals with $l(I'_j) \leq l(O)$, and at least $n - k + 1$ intervals with $r(I'_j) \geq r(O)$. That is, the obstruction is not cleared.

For the converse, suppose $\sum x_j \geq a + b - n = (a - k) + (b - (n - k + 1)) + 1$. Then each interval I_j containing O queried, either a decreases by 1 if $p_j > l(O)$, or b decreases by 1 if $p_j < r(O)$. So in the end, we will have either $a' < k$ or $b' < n - k + 1$, so O will no longer be an obstruction. That is, the obstruction is cleared. This completes the proof of the equivalence.

A choice of intervals to be queried solves the offline problem if and only if it clears all obstructions. We show that it is sufficient to consider minimal proper obstructions. For an obstruction O, the smallest proper obstruction O' containing it has $a' = a$, $b' = b$, and the intervals I_j containing O are the same as those containing O'. Therefore the linear constraints for O and O' are the same, and it is sufficient to consider proper obstructions. If an obstruction is cleared, then every obstruction containing it is also cleared, so it is sufficient to consider minimal proper obstructions. This completes the proof of the characterization of the offline problem as an integer program.

We prove the lower bound V for the online problem. Let O be an obstruction giving the value V. In the proof that if $\sum x_j \leq a+b-n-1$, then the obstruction is not cleared, we could have chosen the points returned for the queries one by one, as the queried intervals are chosen; that is, in an online fashion. Therefore we must have $\sum x_j \geq a+b-n$, paying for the $a+b-n$ intervals of least cost containing O.

■

4 Offline Problem with Unit Costs

We now show how to solve the offline problem efficiently, in the unit-cost case, using the integer program described above. List the minimal proper obstructions in order, letting $O_1 < O_2$ if $l(O_1) < l(O_2)$ and $r(O_1) < r(O_2)$. For the leftmost minimal proper obstruction O_1, since all the intervals I_j containing it have the same cost, we can greedily select I_j with $r(I_j)$ as large as possible, so as to possibly cover as many O_j as possible with it, until we have chosen a_1+b_1-n intervals containing O_1. We then move on to O_2, with O_1 already covered, and chose additional intervals to cover O_2 as needed, again with their right endpoint as far to the right as possible, until we have chosen at least a_2+b_2-n intervals to cover O_2. We proceed in this fashion, satisfying the constraints on minimal proper obstructions in order, from left to right.

Theorem 1 *In the unit-cost case, the integer program that characterizes the offline selection problem can be solved in polynomial time by a greedy algorithm.*

This approach does not seem to work in the general offline case with arbitrary costs, since longer intervals might have a larger cost, so that it is not clearly an advantage to choose them. However, we can obtain an exponential-time algorithm as follows. Let r be the $(n-k+1)$th largest $r(I_j)$. At most $k-1$ intervals I_j have $r(I_j) < r$. Call these the special intervals, and chose which ones will be queried in all possible ways, that is, 2^{k-1} ways. All obstructions have $r(O) \leq r$, so we can again examine the minimal proper obstructions in order from left to right, covering them with intervals I_j of least possible cost which are not special, as many as needed. The right endpoint of intervals that are not special does not matter, since it is at least r.

This gives an algorithm with a time bound of $2^k poly(n)$. A tighter time bound follows from observing that there are at most k minimal proper obstructions O_i, and by writing $a_i+b_i-n = d_i-e_i$, we observe that the number e_i of intervals inside O_i is at most $k-1$, so at most $k-1$ intervals containing O_i will not be queried. We can then just identify the at most $k-1$ intervals of largest cost that are not special and have O_i as the first minimal proper obstruction they cover. That is, after $poly(n)$ preprocessing time, each of the 2^{k-1} cases involves at most k obstructions and k^2 identified intervals, so it uses $poly(k)$ time.

Theorem 2 *With arbitrary costs, the integer program that characterizes the offline selection problem can be solved in time* $poly(n) + 2^k poly(k)$.

A different approach will later enable us to obtain a polynomial-time algorithm for the general offline case.

5 The Online Problem

We now turn to the online problem, and consider an algorithm for the general case with arbitrary costs. The greedy algorithm is as follows. Determine the interval $[l, r]$, where l is the kth smallest $l(I_j)$ and r is the $(n-k+1)$th largest $r(I_j)$. If $r - l \leq \delta$ then we are done. Otherwise $[l, r]$ is an obstruction O. Query the I_j containing O of least cost. Once the point p_j is obtained, replace I_j by $I'_j = [p_j, p_j]$, and go back to the beginning of the algorithm.

Consider an execution of the greedy algorithm. Let O_1, O_2, ..., O_s be the sequence of obstructions obtained, with O_t containing O_{t+1}. For the last obstruction O_s, let H be the set of the $a_s + b_s - n$ intervals of least cost containing O_s. Clearly, the total cost of H is at most V. We show that the interval queried at stage t with $1 \leq t \leq s$ is in H, completing the proof.

The proof is by induction on t. Since the intervals in H clear the obstruction O_s, they also clear the obstruction O_t which contains O_s. In fact the intervals in H containing O_t clear the obstruction O_t, because only containing intervals matter in clearing an obstruction. By inductive hypothesis, the $t-1$ intervals previously queried are in H. However, these $t-1$ intervals have not cleared O_t. Therefore H must have some interval containing O_t other than the $t-1$ intervals previously queried, and such an interval in H of least cost c will be queried. In the case where there are also intervals of cost c not in H (that is, c is the largest cost in H), then an interval of cost c not in H may be queried, but then we can change H by switching this interval for the interval of cost c containing O_t in H, without increasing the total cost of H.

Theorem 3 *With arbitrary costs, the greedy polynomial online selection algorithm achieves the cost V of Proposition 1, and is therefore optimal.*

We will now compare the performance of the offline algorithm with the worst-case performance of the online algorithm, both in the unit-cost case and in the general case with arbitrary costs. For the unit-cost case, we will transform the problem, as follows.

Let P be an instance of the problem. Suppose P contains two intervals I_1 and I_2 with I_2 inside I_1, that is $l(I_1) < l(I_2)$ and $r(I_2) < r(I_1)$. We construct a new instance P' by replacing these two intervals by $I'_1 = [l(I_1), r(I_2)]$ and $I'_2 = [l(I_2), r(I_1)]$. We repeatedly perform this transformation until we obtain an instance $\hat{P}$ with no interval inside another interval.

We first look at the offline problem. Consider the optimal solution for P'. If both I'_1 and I'_2 are queried in this solution, obtain a candidate solution for P by querying both I_1 and I_2. If only one of I'_1 or I'_2 is queried, then query only I_1. If neither I'_1 nor I'_2 is queried, then query neither I_1 nor I_2.

Since P and P' have the same left and right endpoints of intervals, the minimal proper obstructions O are the same for P and P', and have the same $a+b-n$. If O is such an obstruction, then if both I'_1 and I'_2 contain O, then both I_1 and I_2 contain O; and if only one of I'_1 and I'_2 contain O, then I_1 contains O. Therefore the candidate solution for P is indeed a solution, with the same cost, that is, number of queried intervals, as the solution for P'.

Consider now the online problem. Here, the worst-case performance is given by the minimal proper obstruction O with the largest value $a+b-n$. This value, as we said before, is the same for P and P'.

Proposition 2 *Consider the unit-cost case. For the offline selection problem, the performance on $\hat{P}$ is at least as bad as the performance on P. For the online selection problem, the worst-case performance is the same for $\hat{P}$ as for P.*

We can now compare the performance of the optimal offline algorithm with the worst-case performance of the optimal online algorithm.

Theorem 4 *The worst-case performance ratio between offline and online selection algorithms is $\frac{2k-1}{k} < 2$ in the unit-cost case. So both algorithms have the same performance for the smallest element problem, while the ratio for the median problem is $\frac{2n}{n+1}$. In the case of arbitrary costs, the worst-case ratio equals k.*

Proof Sketch: We begin with the unit-cost case. By the preceding proposition, it is sufficient to consider an instance $\hat{P}$ with no interval inside another interval. The intervals can then be ordered from left to right, breaking ties between identical intervals arbitrarily, and letting otherwise $I_1 < I_2$ if $l(I_1) \leq l(I_2)$ and $r(I_1) < r(I_2)$, or if $l(I_1) < l(I_2)$ and $r(I_1) \leq r(I_2)$.

Let $I_1, I_2, \ldots, I_n$ be the intervals in this order. Consider the interval I_k. Clearly I_k is also the largest obstruction, and so all minimal proper obstructions are contained in it. We can assume $|I_k| > \delta$, otherwise no queries are needed. Since no interval is inside an obstruction, all intervals containing an obstruction will be queried. Let I_{k-s} be the first I_j with $j \leq k$ such that $r(I_j) - l(I_k) > \delta$. Similarly, let I_{k+t} be the last I_j with $j \geq k$ such that $r(I_k) - l(I_j) > \delta$. Clearly $0 \leq s \leq k-1$ and $0 \leq t \leq n-k$.

The intervals preceding I_{k-s} do not contain any obstruction and will therefore not be queried by either algorithm. Similarly, the intervals following I_{k+t} will not be queried by either algorithm. The intervals from I_{k-s} to I_{k+t} contain at least one obstruction and will therefore be queried by the offline algorithm. So the offline algorithm makes $s+t+1$ queries.

For the online algorithm, it is sufficient to consider the minimal proper obstruction with the largest number of intervals containing it. The first minimal proper obstruction is contained at least in the intervals from I_{k-s} to I_k. The last minimal proper obstruction is contained at least in the intervals from I_k to I_{k+t}. So the online algorithm makes at least $\max(s,t)+1$ queries in the worst case, and this quantity is the worst case when the I_j with $j < k$ do not intersect the I_j with $j > k$.

The ratio $\frac{s+t+1}{\max(s,t)+1}$ with $0 \leq s \leq k-1$ and $0 \leq t \leq n-k$ is maximized at $s = t = k-1$, and it then equals $\frac{2k-1}{k}$.

Consider next the case of arbitrary costs. Since the performance of the online algorithm is given by the worst constraint for a single minimal proper obstruction, and there are at most k such obstructions, the ratio is at most k. An example that achieves k has $n = 2k-1$, and consists of k disjoint intervals I_j of unit-cost, plus $k-1$ intervals I'_j of zero cost, with the I'_j containing all the I_j. The offline algorithm will have to query all the intervals, incurring in cost k with the intervals I_j. The online algorithm queries the I'_j first, and depending on the resulting answers, determines which single I_j to query, with total cost 1.

■

6 Offline Problem with Arbitrary Costs

The earlier algorithm for the offline selection problem with arbitrary costs has complexity exponential in k. We now provide a polynomial-time algorithm for this problem, but this algorithm is non-combinatorial and relies on linear programming.

The polytope of the offline selection problem is defined by replacing in the integer program the conditions $x_j \in \{0,1\}$ with linear constraints $0 \leq x_j \leq 1$, with the remaining linear constraints being the same.

Theorem 5 *The vertices of the polytope of the offline selection problem are all integer vertices (that is 0–1 vertices). Therefore, with arbitrary costs, the problem can be solved in polynomial time by linear programming.*

Proof Sketch: Consider a vertex x of the polytope. If some x_j is either 0 or 1 for x, then use this value in the constraints where x_j occurs. We are thus left with some h variables with $0 < x_j < 1$. Since x is a vertex of the polytope, there must be some h constraints satisfied with equality that define x uniquely. The corresponding h by h square matrix M is a 0–1 matrix M, with the property that for every column of M, the value 1 occurs in consecutive rows, since a variable x_j occurs in consecutive linear constraints. We show that the determinant of such a matrix M is either 0, 1 or -1. Therefore the solution must be integer, that is, all x_j for the vertex x are either 0 or 1, completing the proof.

So assume M has a nonzero determinant. Then some column must have a 1 in the first row. Consider the column with a 1 in the first row that has the least number r of 1s, say it is the first column. Then the 1s in the first column occur in rows $1, 2, \ldots, r$. After subtracting the first row from rows $2, \ldots, r$, we can remove the first row and the first column, and argue inductively for the resulting submatrix M'. Notice that the only columns affected by the subtraction are the columns that have a 1 in the first row. These columns, however, have a 1 in rows $1, 2, \ldots, r$ by the choice of the first column. For such a column, after the subtraction, the 1s in rows $2, \ldots, r$ become 0s, and after the first row is removed, the remaining 1s in the column will be in consecutive rows. So the matrix M' has the property of consecutive 1s in each column, and the induction goes through.

A linear programming algorithm that finds a vertex of the polytope minimizing the objective function $\sum c_j x_j$ thus solves the problem.

■

Unfortunately, a linear-programming algorithm is not very practical for the applications at hand. It therefore becomes interesting to seek a combinatorial algorithm that is polynomial-time. But, as shown in the next section, the weighted bipartite matching problem is a special case of our offline selection problem, and therefore we cannot really hope for a simple combinatorial algorithm. In practice, it might be better to use the following approximation algorithm.

Theorem 6 *The offline selection problem with arbitrary costs has a $2\log_2 k$-approximation polynomial-time algorithm.*

PROOF SKETCH: Construct a binary tree whose leaves are the minimal proper obstructions $O_1, O_2, \ldots, O_{k'}$ in left to right order, with $k' \leq k$. Place an interval at a node q if the minimal proper obstructions it contains, $O_s, O_{s+1}, \ldots, O_t$, are precisely the leaves below node q.

Unfortunately, not all intervals correspond to a single node; we argue that in general, the minimal proper obstructions covered by an interval can be decomposed into at most $2\log_2 k$ groups, with each group corresponding to a single node q. To show this, let P_i be the set of nodes on the path from the root to the leaf O_i, and consider in particular P_{s-1} and P_{t+1}, which are taken to be empty if $s = 1$ or $t = k'$, respectively. Then the nodes selected are precisely the nodes q not on P_{s-1} such that q is the right child of a node on $P_{s-1} - P_{t+1}$, plus the nodes q not on P_{t+1} such that q is the left child of node on $P_{t+1} - P_{s-1}$. If the binary tree is chosen to be balanced, the bound of $2\log_2 k$ on the number of nodes q selected follows. (By carefully choosing the binary tree, the bound can be improved by an asymptotic factor of 2.)

The algorithm then represents each interval by at most $2\log_2 k$ intervals at nodes q, each with the same cost as the original interval. This may increase the cost of an optimal solution by a factor of $2\log_2 k$. The algorithm then solves the problem with each interval at a single node q in polynomial time.

The algorithm computes at each node q, starting at the leaves O_i and moving up to the root, how to select intervals at node q so as to leave a requirement of r intervals having to be chosen at nodes higher than q, on the path from the root to q, for all possible values r, which we call the *demand* at node q. Initially, at a leaf O_i, before choosing which intervals to select at O_i, we have $r = a_i + b_i - n$. Suppose a node q has inherited demands of r' and r'' from its children, and wishes to achieve demand $r \leq \max(r', r'')$, to pass on to its parent. (At a leaf, there is a single inherited demand r' as described above.) Then we must select the $\max(r', r'') - r$ intervals of least cost at q. For a given r, we carry out the calculation of total cost with the possible choices of r', r'', and select the one that gives the least total cost for the intervals chosen at q and its descendants, for the demand r under consideration. At the root, we force $r = 0$, since no demand can be satisfied at nodes higher than the root. ■

7 Interval Problems and Weighted Bipartite Matching

We now examine the expressive power of the offline selection problem. Specifically, we define the notion of an "interval problem," show that it is equivalent to the offline median problem, and includes weighted bipartite matching as a special case. This makes it seem unlikely that we can obtain a simple combinatorial algorithm for the selection problem. In fact, we will also show that a mild generalization of the interval problem is in fact NP-hard.

Definition 1 *Define an* interval problem *to be to minimize $\sum c_j x_j$ with $x_j \in \{0,1\}$ subject to k constraints $\sum_{S_i} x_j \geq f_i$, with the property that each x_j occurs in consecutive constraints.*

We know that the offline selection problem is an interval problem. We now show the converse. The median problem will have $\delta = 1 - \frac{1}{2(k+1)}$ and k minimal proper obstructions $O_i = [i-1, i]$ for $1 \leq i \leq k$. If a variable x_j occurs in constraints $s, s+1, \ldots, t$, then we represent it by the interval $I_j = [s-1, t]$. We must also have $f_i = a_i + b_i - n = d_i - e_i$, with $a_i, b_i \geq \lceil \frac{n}{2} \rceil$. In order to have $f_i = d_i - e_i$, where d_i is the number of variables in the constraint and e_i is the number of intervals inside O_i, we put e_i single-point intervals $I_j = [p_j, p_j]$ inside O_i at $i - \frac{i}{k+1}$. To make O_i a proper obstruction, we add a zero cost interval $I_j = O_i$. To ensure that $a_i, b_i \geq \lceil \frac{n}{2} \rceil$, we add a sufficiently large number of cost zero intervals $I_j = [0, k]$. It is then clear that the O_i will be the minimal proper obstructions and will give the corresponding constraint when the zero cost intervals that were added are selected.

Proposition 3 *Every interval problem is an offline median problem.*

We may sometimes want to force a constraint $\sum_{S_i} x_j \geq f_i$ to hold with equality. To achieve this, we choose a large $L > \sum c_j$ and add $L \sum_{S_i} x_j$ to the objective function. If the constraint can be satisfied with equality, this will be forced by the minimization, making the added term equal $L f_i$. The same L can be used to force several different constraints to hold with equality, when this is feasible.

Theorem 7 *The interval problem can express the weighted bipartite matching problem.*

PROOF SKETCH: The case where all constraints must hold with equality can express the minimum cost bipartite perfect matching problem. To see this, consider a bipartite graph G with k vertices on each side and costs on the edges, that has a perfect matching; we seek a minimum cost perfect matching. Let $1, 2, \ldots, k$ be the k vertices on the left and let $1', 2', \ldots, k'$ be the k vertices on the right. The corresponding interval problem will have $2k$ constraints corresponding to these vertices, in the order $1, 2, \ldots, k, k', \ldots, 2', 1'$. An edge from i to j' will correspond to a variable occurring in the constraints from i to j' in this order. To force exactly one of the edges incident to vertex 1 to be selected, we use the bound $f_1 = 1$ for the first constraint, with equality. To force exactly one additional edge incident to vertex 2 to be selected, we use the bound $f_2 = 2$ for the second constraint, with equality. In general, we have $f_i = f_{i'} = i$, with equality. This completes the representation. ■

Consequently, the weighted bipartite matching problem is a special case of the weighted offline median problem. We use this expressiveness of the weighted offline median problem to show that a slight generalization is computationally hard.

The *median problem with multiplicities* is the median problem with the additional provision that some specified intervals, when queried, return two points instead of just one. This is like having, in the median problem, pairs of identical intervals that must be simultaneously queried or not queried.

Theorem 8 *The weighted offline median problem with multiplicities is NP-complete.*

PROOF SKETCH: The problem corresponds as before to a general interval problem, except that now, for some of the variables, the quantity $2x_j$ instead of x_j appears in the constraints. We can again force equality to hold in the constraints by using appropriate costs. The special case of bipartite matching from before now generalizes to a bipartite flow problem. In this problem, vertices on the left have a given supply amount, vertices on the right have a given demand amount, and vertices on the left are joined to vertices on the right by edges of capacity 1 or 2. The supplies and demands must be satisfied by sending flow across the edges, but in such a way that the flow across an edge is either zero or the full capacity of the edge.

We show that this bipartite flow problem is NP-complete. The reduction is from 3SAT. In fact, we assume that every variable x in the 3SAT instance occurs at most twice as a positive literal x, and at most twice as a negative literal $\overline{x}$. Every 3SAT instance can be made to satisfy this assumption, by replacing every variable with several variables forced to be equal by a cycle of implications; the implications in the cycle account for one positive and one negative occurrence, so each variable in the cycle can be used again, once as positive and once as negative.

To express such a 3SAT instance as a bipartite flow problem, we put a vertex on the right for each literal x or $\overline{x}$, with demand 2. We put a vertex on the left for each clause, with supply 1, and with three capacity 1 edges joining it to the literals occurring in the clause on the right. Thus the supply 1 must be sent to one of the three literals occurring in the clause, which we can take to be a satisfying literal for the clause. Notice that the demand bound of 2 will hold, since no literal occurs in more than two clauses. We must ensure that complementary literals are not both chosen as satisfying literals. To achieve this, we put a vertex on the left for each variable x, with supply 2, and with capacity 2 edges joining it to both literals x and $\overline{x}$. Thus only one of the two literals will have demand left to be chosen by the clauses on the left. Thus the supply on the left can be sent while satisfying the demand bounds on the right if and only if the instance has a solution. It then remains to force the demands of 2 to be met exactly, and this is done by providing additional $2v - w$ units of supply to be sent along capacity 1 edges to any vertex on the right, where v is the number of variables and w is the number of clauses. This completes the proof of NP-completeness.

■

It is natural to consider the related problem where, when an interval I_j is queried, instead of returning a single point p_j in I_j, we obtain a subinterval of I_j of length at most δ'. If the new parameter δ' satisfies $\delta' \leq \delta$, where δ is the parameter for the required precision as before, then all the results obtained above go through (unit-cost or arbitrary cost, online or offline), since an obstruction O has length greater than δ, so that an interval containing it will no longer contain it after the interval length is reduced to at most δ'. On the other hand, if $\delta' > \delta$, then the problem has no solution (unless no queries are needed), since queried intervals I_j can be answered so as to still contain a sufficiently small obstruction O.

References

[1] O.H. Ibarra and C.E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM* 22 (1975): 463–468.

[2] C. Olston and J. Widom. Bounded Aggregation: Offering a Precision-Performance Tradeoff in Replication Systems. *Submitted for publication.* Stanford Technical Report `http://www-db.stanford.edu/pub/papers/trapp-ag.ps`.