Discussion Paper No. 614

A LINEAR TIME RANDOMIZING ALGORITHM FOR LOCAL ROOTS AND OPTIMA OF RANKED FUNCTIONS*

by

Eitan Zemel**

September 1983
Revised July 1984

---

*Supported, in part, by NSF Grant ECS-812141, and by the Mia Fischer David Foundation through the Israel Institute of Business Research, Tel Aviv University. IIBR working papers are intended for preliminary circulation of tentative research results. Comments are welcome and should be addressed directly to the author.

**J. L. Kellogg Graduate School of Management, Northwestern University, Evanston, Illinois 60201. Part of this work was done when the author was visiting Tel Aviv University.

Abstract

Consider a family F of n functions defined on a common interval U. For each $x \in U$, let $\ell_k(x)$ be the k-th largest element in the set of function values at x. Similarly, let $L_k(x)$ be the sum of the k-largest elements in this set. We give a linear time randomizing algorithm for finding local roots, optima, deflection points, etc., of ℓ or L. The algorithm is generalized to the cost effective resource allocation problem and to various variants of the parametric knapsack problem.

Keywords: selection, rank, randomizing algorithm, parametric methods.

A LINEAR TIME RANDOMIZING ALGORITHM FOR LOCAL ROOTS AND OPTIMA OF RANKED FUNCTIONS

by
Eitan Zemel

Let $F = \{f_1,\ldots,f_n\}$ be a given set of continuous functions defined on a common interval $U = [a,b]$ whose end points may be at infinity. We are interested in functions derived from F by order information. Specifically, we deal with the two functions $\ell_k(x)$, whose value for each $x \in U$ is the k-th largest element in the set $F(x) \equiv \{f_1(x),\ldots f_n(x)\}$, and $L_k(x)$ whose value is the sum of the k largest elements in that set. In particular, we give fast (linear time) randomizing algorithms for finding local optima, roots, deflection points, etc., of ℓ and L. Our methods are generalized to yield efficient randomizing algorithm for various ratio knapsack problems and for the Cost Effective Effort Allocation Problem [1]. Similar methods yield an $O(n \log n)$ randomizing algorithm for the Weighted Euclidean One Center Problem [4], which can be solved deterministically in $O(n \log^2 n)$, Megiddo [3].

The deterministic foundation on which our algorithm is based is the parametric approach of Megiddo [2]. Indeed, the methods of [2] can yield an $O(n \log^2 n)$ deterministic algorithm for our problem. The algorithm uses internally induced randomizations in the spirit of Rabin [5]. Thus, all the probability statements provided are worst case statements and do not rely on any assumption concerning the distribution of problem instances. A key element in achieving the linear time behavior is the concept of an ε-partition, i.e., binary search which is not carried to conclusion. Without this element, the running time of the algorithm is $O(n \log n)$.

For the sake of briefness, we treat in this note the case of finding a root of ℓ. The reader should have no difficulty extending the results to the case of L or to the problems of finding a local optimum, deflection point, etc.

An earlier version of this paper was circulated as [7].

II. The Computational Model

We consider here the problem of finding a local root of ℓ over U. By finding a local root of a function, g, over an interval [x,y], we mean the following:

(i) If $f(x) \cdot g(y) = 0$, return x or y as the case may be;

(ii) If $g(x) \cdot g(y) < 0$, return a point $z \in [x,y]$ such that $g(z) = 0$.

(iii) If $g(x) \cdot g(y) > 0$, return an error message.

The basic framework presented here is able to handle any set of continuous functions F. However, we need some conventions regarding our notion of a computational step. These are summarized by the following two assumptions:

(a) For each function $f_i \in F$, and each $x \in U$ we can compute $f_i(x)$ in constant time. (In fact, computing the sign of $f_i(x)$ would suffice.)

(b) For each $f_i \in F$ and each interval $[x,y] \subseteq U$, we can find a local root of f_i in [x,y] in constant time (with the convention on the sign of $f_i(x) \cdot f_i(y)$ provided above).

If the functions in F are linear, assumptions (a) and (b) require that the four basic arithmetic operations be carried out in constant time. If F are quadratic, then we require that square roots can also be taken in constant time, etc.

The following two additional assumptions are necessary for the analysis:

(c) There exists a constant s (independent of n), such that for each pair $(f_i,f_j) \in F$, the equation $f_i(x) = f_j(x)$ has at most s roots which can be found in constant time.

(d) There exists a constant t (independent of n) such that ℓ has at most t roots in the interval U.

Assumption (c) would clearly hold if the fucntions in F are polynomials of bounded degree. Assumption (d) holds with t = 1 if the functions in F are monotone, or if these functions are convex, and one searches for the minimum of L. Under the convexity assumption, (d) holds with t = 2 for the problem of searching a root of L.

III. The Test

In the squeal we consider the only interesting case $\ell(a)\cdot\ell(b) < 0$, say $\ell(a) < 0$, $\ell(b) > 0$. Let x be any point in the interval (a,b). We call the operation of computing the sign of $\ell(x)$, a test at x. Assumption (a) implies that this can be accomplished in O(n) time. Clearly, if $\ell(x) > 0$ then a root exists in the interval (a,x); if $\ell(x) < 0$ then a root exists in (x,b); and if $\ell(x) = 0$, then x is the required root. In a typical binary search algorithm, a test enables us to reduce the interval of interest U. A key idea in the parametric method of Megiddo is to perform the test in selected positions which enable us to reduce the size of F too.

IV. The Rank of an Element

Let x^* be the root of ℓ which is eventually found by the algorithm. The rank of a function f_i in F is defined relative to this point. Let $F_i^1 \subseteq F$, $F_i^2 \subseteq F$ and $F_i^3 \subseteq F$ be the sets of functions whose values at x^* are greater than, equal to, and less than $f_i(x^*)$, respectively. The rank of f_i in F is the ordered triplet of cardinalities $(|F_i^1|, |F_i^2|, |F_i^3|)$. Similarly, the weighted rank is the triplet of total weight inside these sets. Clearly, if the partition F_i^1, F_i^2, F_i^3 is known, we can examine the rank of f_i relative to k and discard from further considerations two of these three subsets. Unfortunately, as x^* is not known in advance, neither is the rank of f_i. However, we can still rank f_i in a roundabout way by performing several tests

at key points.

Indeed, let $X^j = \{x_1^j,\ldots,x_{p_j}^j\}$ with $p_j \leq s$, be the set of roots of the equation $f_i(x) = f_j(x)$ inside U. These roots induce a partition of U such that f_i alternates, being above and below f_j on subsequent intervals of this partition. Thus, we can find whether $f_i(x^*)$ is above or below $f_j(x^*)$ by locating x^* within the set X^j. Let $X = \{x_1,\ldots,x_p\}$ be the union of the sets X^j, $j=1,\ldots,n$, $j\neq i$, with $p \leq (n-1)\cdot s$. The complete partition of F into the three subsets F_i^1, F_i^2, F_i^3 can be identifed if the location of x^* is known relative to all the points in X. But this can be found by performing log(p) tests at suitably chosen points of X. Since $p = O(n)$, and since each test costs O(n) steps, this can be the basis of an O(n log n) randomizing algorithm for our problem. An alternative O(n log n) randomizing algorithm can be obtained by using Reischuk's randomizing parallel algorithm for selection [6] within the framework of Megiddo's method [2]. To achieve <u>linear time</u> performance, we have to economize further. This can be accomplished by using a relaxed form of ranking which we call ε-ranking.

V. <u>ε-ranking and ε-partitioning</u>.

Let $0 < \varepsilon < 1$ be arbitrary but fixed. An ε-partition of F by f_i is <u>any</u> partition of F into <u>four</u> subsets, G_i^α, $\alpha=1,2,3,4$ such that for $\alpha=1,2,3$

$$G_i^\alpha \subseteq G_i^\alpha$$

while for $\alpha = 4$

$$\left|G_i^4\right| \leq \varepsilon \left|F\right|.$$

Clearly, there are numerous ε-partitions associated with a given function f_i

as the only requirement on G_i^4 is on its cardinality. An ε-rank of f_i is the ordered triplet $(|G_i^1|,|G_i^2|,|G_i^3|)$ associated with one of these ε-partitions. A weighted ε-partition is similar to an ε partition, but with the requirement that the weight of G_i^4 does not exceed ε times the total weight. Although not unique, an ε-rank gives some useful information on the "real" rank of a function f_i as for $\alpha=1,2,3$, one has

$$|G_i^\alpha| \leq |F_i^\alpha| \leq |G_i^\alpha| + \varepsilon|F|.$$

A quick and easy way to compute an ε rank of an element f_i is the following. Let $X = \{x_1,\ldots,x_p\}$, $p \leq (n-1)s$ be the set of intersections of f_i with the other $n-1$ functions of F. By performing a test at the median element of X, we establish the location of x^* in relation to one half of the points in X. A second test at the median of the remaining set establishes the location of x^* with respect to an additional quarter of these points. Continuing in this fashion for a total of $\log(s/\varepsilon)$ tests, we are left with a subset of at most $\varepsilon(n-1) \leq \varepsilon n$ of the points of X. But then the number of functions $f_j \in F$ which cannot be put in one of the sets G_i^α, $\alpha=1,2,3$ is at most εn. For any prespecified $\varepsilon > 0$, this amounts to $O(n)$ overall effort, since s is a constant. We note that a similar effort is needed for deriving an ε-weighted partition. The latter can be achieved by performing tests at the weighted medians of the appropriate sets.

VI. The Algorithm and Its Analysis

We are now in a position to state our algorithm. Recall that we denote by x^* the root which is eventually produced by the algorithm. At each iteration, we pick up a function $f_i \in F$ randomly and generate the appropriate ε-partition and ε-rank. As we demonstrate below, this will enable us to

eliminate a subset of F from further consideration. When F is reduced to a single function, we can conclude the search by finding a root of that function. Although the set of active (uneliminated) functions varies from iteration to iteration, we abuse notation and refer to this set as F, with a similar convention with respect to the sets defining a true or an ε-partition, and to n and k.

We now examine the process of elimination. Assume that at a given point we are able to conclude, for a given function f_i, that $f_i(x^*) > 0$. Then clearly f_i can be removed from the set F without any effect on our ability to find x^*. Similarly, if we can conclude that $f_i(x^*) < 0$, then f_i can be removed provided k is decremented by 1. In order to understand how elimination can take place, assume first that the true rank and partition associated with f_i are known. Clearly, if $|F_1| > k$ then each function f_j in $F_i^2 \cup F_i^3$ must satisfy $f_j(x^*) < 0$ and can be eliminated. Similarly, if $|F_i^3| > n - k$, then the functions in $F_i^1 \cup F_i^2$ can be eliminated as these functions must be positive at x^*. Finally, if neither conditions hold, then $f_i(x^*)$ must be zero and we can conclude our search. In each of the former two cases, if $f_i(x^*)$ is located somewhere in the central region of the set $F(x^*)$, then the number of functions eliminated is a significant fraction of n.

In actuality, we only have information about the ε-rank of f_i, rather than about its true rank. However, we can still go through with the elimination process with only small modifications. For instance, assume that $|G_i^1| > k$. Then obviously $|F_i^1|$ must satisfy this inequality and we can safely eliminate $G_i^2 \cup G_i^3$. Note that the difference between the magnitudes of $G_i^2 \cup G_i^3$ and $F_i^2 \cup F_i^3$ is at most $\varepsilon \cdot n$. A similar situation arises when $|G_i^3| > n - k$. If neither condition holds but $|G_i^1 \cup G_i^2| > k$ then G_i^3 can be eliminated. Symetrically, if $|G_i^2 \cup G_i^3| > n - k$ then G_i^1 can be eliminated. Finally, if

none of the above mentioned conditions holds, i.e., $k > |G_i^1| \geq k - \varepsilon n$ and $n - k > |G_i^3| \geq n - k - \varepsilon n$, we cannot infer whether $f_i(x^*)$ is positive or negative and no elimination can take place. (Note that this situation never arises if the true rank of f_i is known.)

Let $\alpha > 0$ be a constant to be specified later. Call an iteration a success if we eliminate at this iteration at least $\alpha \cdot n$ variables. We wish to assess the probability of success. Throughout this analysis we ignore the issues of rounding and treat expressions like $\alpha \cdot n$ as integers. The reader should have no difficulty realizing that the errors introduced by this approximation are negligible and do not change the conclusions of this analysis.

Assume first that x^* is the unique root of ℓ in U. Consider any permutation of the indices of $F(x^*)$ which is consistent with the order of the set $F(x^*)$, i.e., $f_{j_1}(x^*) \geq f_{j_2}(x^*) \geq \ldots \geq f_{j_n}(x^*)$. Let $i = j_r$. Obviously, since f_i is chosen randomly from F, r is uniform over the integers 1 to n. Denote the integers in the set

$$\{j_r \mid r \in [1,(\alpha+\varepsilon)n] \cup [n - (\alpha + \varepsilon)n, n] \cup [k - \varepsilon n, k + \varepsilon n]\},$$

forbidden. Clearly, the probability that r falls in the forbidden range is at most $2\alpha + 4\varepsilon$. It is easy to see that any choice of f_i which is in the complement of the forbidden set must yield a success even for the worst possible choice of the set G_i^4.

Consider now the case of several local roots. Since the order of the functions in F may be different for each root, the forbidden regions with respect to each root may be disjoint. For each given root, the probability of falling in the forbidden range is as calculated in the previous section.

However, x^* can not be assumed fixed since its identity depends on the random steps of the algorithm. To overcome this difficulty, let A be the event: [f_i is in the forbidden range for a given root of ℓ in U] and let B be the event: [f is in the forbidden range for at least one of the t roots of ℓ in U]. Clearly, $Pr[A] \leq 2\alpha + 4\varepsilon$ and $Pr[B] \leq t \cdot Pr[A] \leq t(2\alpha + 4\varepsilon) \equiv q$. Choose the constants α and ε so that $q < 1$. This is an upper bound on the probability of failure. We recall that the work per iteration is $c \log (s/\varepsilon)n \equiv c_1 n$ for some small constant c, independent of success or failure.

Let S_n be the running time of the algorithm, with n functions left to go. S_n is a random variable whose distribution depends on the data (i.e., on F and U) in a very complex way. However, we can analyze the running time, T_n, of a slightly modified version of the algorithm. The modified version proceeds precisely as the original algorithm, except that whenever success occurs, exactly $\alpha \cdot n$ functions are eliminated (ever if more could have been). Analyzing T_n is also complex, but it can be easily observed to be stochastically dominated by X_n, defined recursively by the equation

$$X_n = c_1 \cdot n + Y \cdot X_n + (1 - Y)X_{(1-\alpha)n}$$

where Y is a zero-one random variable, independent of X and n and such that $Pr[Y = 1] = q$. (In the process defining X success occurs with probability exactly $1 - q$ while in T the probability is <u>at least</u> that.) X_n can be analyzed easily. In fact, one can show that

$$E(x_n) = \frac{c_1}{(1 - q)\alpha} \cdot n \equiv c_2 n$$

and

$$Var(X_n) = d^2n^2$$

for some easily obtained constant d. The first condition immediately yields that $E(T_n) \leq c_2 n$. Furthermore, using the second condition and Chebyshev's inequality, we get that for any prespecified level of confidence, p, there exists a constat c(p) such that

$$Pr[T_n > c(p) \cdot n] \leq p,$$

i.e., our algorithm is linear time to any desired level of confidence.

VII. Extensions

We consider here some extensions of our method. Since the techniques employed in this section are close in spirit to those of the previous one, we treat here a simplified version of our problem. Specifically, whenever a comparison is made between elements (functions) we ignore the possibility of a tie so we can say that the rank of an element is simply the number of elements larger than it and the weighted rank is the weight of these elements. Similarly, an ε-rank and a weighted ε-rank are defined as the number and weight, respectively, of the elements known to be above an element in an ε or a weighted ε-partition. As seen in the previous section, the existence of ties can only speed up the algorithm so the results derived are valid for the general case. Also, we consider here the case of a unique root x^* of ℓ in U. In the applications which follow, this condition is met. However, by the same reasoning applied in the previous section, the order of the running time is not affected, provided the number of roots is bounded by a constant. Finally, we ignore the issue of rounding and regard expressions like $\alpha \cdot n$ as integers. Again, the implications of this simplification are trivial.

Consider first the original problem of finding a local root of ℓ but with some additional structure imposed on the set F. Specifically, assume that the set F is partitioned into m subsets $F^1,\ldots,F^m$ such that the sorted permutation at x^* of the functions in each subset F^i, i=1,...,m, is known in advance. This additional information allows us to speed up the solution, yielding, for appropriate values of m/n a sublinear algorithm.

Assume that $k \leq n/2$. Let $0 < \beta, \gamma < 1$ to be specifed later. Let $n_i = |F^i|$. Denote by h_i, i=1,...,m, the element whose rank in F^i is βn_i and let $H = \{h_1,\ldots,h_m\}$. Let $h \in H$ be a given element and let its weighted rank in H be γn, where the weight of h_i is taken to be n_i, i=1,...,m. Clearly there are at least $\beta\gamma n$ functions $f_i \in F$ which are larger than h at x^*, and at least $(1 - \beta)(1 - \gamma)n$ functions which are smaller. If $\beta\gamma n \geq n/2$, then these $(1 - \beta)(1 - \gamma)n$ functions can be eliminated, being negative at x^*. Now, assume that the ε-weighted rank of h in H is $\gamma'n$. Then, clearly $\gamma - \varepsilon \leq \gamma' \leq \gamma$, and if $\beta\gamma'n \geq n/2$ then at least $[(1 - \beta)((1 - \gamma' - \varepsilon)]n$ functions can be eliminated. Note that $\beta\gamma' \geq \beta(\gamma - \varepsilon)$ and that $(1 - \beta)(1 - \gamma' - \varepsilon) \geq (1 - \beta)(1 - \gamma - \varepsilon)$. Thus, if $\beta(\gamma - \varepsilon) \geq 1/2$ we eliminate at least $(1 - \beta)(1 - \gamma - \varepsilon)n$ functions.

Let $\alpha > 0$ be a constant to be specified later. Call an iteration a "success" if we eliminate at least αn variables. Clearly, a success occurs whenever $\gamma \in [1/2\beta - \varepsilon, 1 - \varepsilon - (\alpha/(1 - \beta)] \equiv T$. Choose constants α, β, ε such that $T \neq \phi$. We wish to compute the probability p that $\gamma \in T$. Let δ be the size of the interval T. Pick $h_i \in H$ randomly, with probability n_i/n. Then, p is approximately equal to γ, the degree of approximation depending on the relative sizes of the sets F^i, i = 1,...,n. In fact, let

$$\Delta = \max_{i=1,\ldots,n} n_i/n$$

then $p \geq \delta - 2\Delta$. To achieve a small Δ, break each subset F^i with $n_i > \delta/10$ into $10n_i/\delta$ separate pieces, each of size $\delta/10$. This ensures that $\Delta \leq \delta/10$ and thus $p \geq 8\delta/10$. Note that after elimination is accomplished, the different pieces which make F^i can be put back together, so that throughout the algorithm, m is increased by at most a constant, $10/\delta$. Since the work per iteration is $O(m \log (n/m))$, we get by an analysis similar to that of Section VI that the running time is $O(m \log^2 (n/m))$ to any level of confidence. A specific example is the Cost Effective Resource Allocation Problem:

$$\max \sum_{i=1}^{m} f_i(x_i)/g_i(x_i)$$

subject to

$$\sum_{i=1}^{m} x_i \leq k$$

$$x_i \geq 0, \quad \text{integer}$$

where $F = \{f_1,\ldots,f_m\}$ and $G = \{g_1,\ldots,g_m\}$ are sets of concave and convex functions respectively. The algorithm proposed here can solve this problem in $O(m \log^2 m)$ steps with high probability as opposed to the $O(m \log^2 m \log^2 k)$ worst case deterministic complexity, [1].

As another extension consider weighted problems where a weight, $w_i \geq 0$, is attached to each function $f_i \in F$. A case in point is the max ratio knapsack problem:

$$\max \frac{\sum_{j \in N} c_j x_j + c_o}{\sum_{j \in N} d_j x_j + d_o}$$

subject to

$$\sum_{j \in N} a_j x_j \leq ao$$

$$0 \leq x_j \leq 1, \quad j \in N$$

with

$$d_j > 0, \quad j=0,1,\ldots,n.$$

or the parametric knapsack problem:

$$\min_{\lambda} \max_{x} \sum_{j \in N} (c_j - \lambda d_j)x$$

subject to

$$\sum_{j \in N} a_j x_j \leq a_0$$

$$0 \leq x_j \leq 1, \quad j \in N.$$

Both these problems can be solved in $O(n \log^2 n)$ deterministically by the methods of [2]. By picking a variable x_i, $i \in N$ randomly (with probability $\frac{1}{n}$) and computing its weighted rank, we can get an $O(n \log n)$ randomizing running time. Also, by picking x_i, $i \in N$ with probability proportional to a_i and computing its ε-weighted rank, we get an $o(n \log \lambda)$ randomized running time, where λ is the ratio of the largest to smallest weight coefficient, i.e.,

$$\lambda = \max_{i,j} \left|\frac{a_i}{a_j}\right|.$$

References

[1] Megiddo, N., "An Application of Parallel Computation to Sequential Computation: The Problem of Cost Effective Resource Allocation." TWISK 202 CSIR-NRIMS, Pretoria, S. Africa, 1981.

[2] _____, "Applying Parallel Computation Algorithms in the Design of Serial Algorithms." SACM, 30, 4 (1983), pp. 852-865.

[3] _____, "The Weighted Euclidean 1-Center Problem." Math. of O. R., 8, 4 (1983), pp. 498-505.

[4] _____, and E. Zemel. "An O(n log n) Randomized Algorithm for the Weighted Euclidian One Center Problem." Forthcoming.

[5] Rabin, M. O. "Probablistic Algorithms." In Algorithms and Complexity: New Directions and Recent Results, J. F. Traub (ed.). Academic Press, 1976.

[6] Reischuk, R., "A Fast Probabilistic Parallel Sorting Algorithm." 22nd Annual Symposium of Foundations of Computer Sciences, pp. 212-219, 1981.

[7] Zemel, E., "A Parallel Randomized Algorithm for Selection with Applications." Working Paper No. 761/82, Israel Institute for Business Research, Tel Aviv University, November 1982.