A FAST ALGORITHM FOR DETERMINING THE DISTANCES OF THE POINTS OF A GIVEN SET IN AN INTEGRAL LATTICE FROM ITS COMPLEMENT

A. V. Karzanov

UDC 519.854

A linear-time algorithm is proposed for determining the distances of the points of a given set in an r-dimensional integral lattice from its complement.

1. STATEMENT OF THE PROBLEM AND RESULTS

Analysis of cartographic data for predicting the location of natural resources involves determination of the distances from the points of a section of a given trajectory to the nearest points within the scope of a certain geological factor. Knowledge of these distances for a sufficiently large number of factors makes it possible to develop valid hypotheses concerning the structure and the boundaries of the deposit. The problem of determining a set of distances has to be solved repeatedly (for many factors and many sections), and it is therefore relevant to develop a fast solution algorithm.

We start with a formal statement of the problem for the general multidimensional case. The formulation of the problem for the two-dimensional case and the cartographic application described above have been communicated to the author by É. A. Nemirovskii.

In the r-dimensional Euclidean space R^r consider an integral lattice Γ of dimension $n_1 \times n_2 \times \ldots \times n_r$ which consists of all integer vectors x such that $1 \leq x(l) \leq n_l$, $l = 1,\ldots,r$, where $x(l)$ is the l-th component of the vector x. A subset B is given in Γ; the points of B are called "black" and the points of $W := \Gamma - B$ are called "white." Let $d(x, X)$ be the Euclidean distance of the point $x \in R^r$ from a finite subset $X \subset R^r$, i.e., $d(x, X) = \min\{d(x, y) | y \in X\}$, where $d(x, y)$ is the Euclidean distance between the points x and y (if $X = \varnothing$, then $d(x, X) = \infty$). For each black point $x \in B$ it is required to find the distance $d(x, W)$ from the set of white points W.

We propose an effective algorithm to find these distances (more precisely, squared distances) with an upper bound of $O(rN)$ on the number of operations (the running time) and $O(N + n)$ on the number of memory locations of length not exceeding $\omega = C + \max\{\log_2 N, \log_2 rn^2\}$, where $N = |B|$, $n = \max(n_1,\ldots,n_r)$, and C is a constant. The algorithm is realized on a random-access computer, using standard logical and arithmetic operations on words of length not exceeding ω and addressable read/write operations. The input of the algorithm are the numbers $n_1,\ldots,n_r$ and the set B; we assume that B is defined in the form of an N-element list or array, and each element in turn is defined as a list or array of coordinates of the corresponding point. The size of the input data is thus $r + rN$ and the algorithm runs in linear time.

The proposed algorithm can be easily modified to solve a similar problem in a fairly wide class of metrics, in particular, for an arbitrary metric l^p, $1 \leq p \leq \infty$. This class of metrics is described in Sec. 3, where two generalizations of our problem are also considered. The proposed algorithm can be used in computer graphics and other areas.

2. THE ALGORITHM

For $x \in \Gamma$ and $l \in \{1,\ldots,r\}$, we denote:

$W_i(x)$ is the set of points y in W such that $y(j) = x(j)$ for $j = i + 1,\ldots,r$;

$\xi_i(x)$ is the line formed by the vectors $y \in R^r$ for which $y(j) = x(j)$, $j = 1,\ldots,i - 1,i + 1,\ldots,r$;

$\tau_i(x)$ is the point y in $\xi_i(x)$ for which $y(i) = x(i) + 1$;

$d_i(x)$ is the distance $d(x, W_i(x))$.

Translated from Kibernetika i Sistemnyi Analiz, No. 2, pp. 177-181, March-April, 1992. Original article submitted April 11, 1988.

1060-0396/92/2802-0316$12.50

We also assume that $W_0(x) = \{x\} \cap W$ and $d_0(x) = d(x, W_0(x))$, i.e., $d_0(x) = 0$ if $x \in W$ and $d_0(x) = \infty$ if $x \in B$. For $x \in B$ the sequence T of elements in $\xi_i(x) \cap B$ ordered by increasing values of the i-th component is called a complete i-segment and the inclusionwise maximal subsequence $S = (x_1,\ldots,x_m)$ in T such that $x_{j+1}(i) = x_j(i) + 1, j = 1,\ldots,m-1$, is called an i-segment; the i-segment S is called left if $x_1(i) = 1$ and right if $x_m(i) = n_i$. Let $z \in \xi_i(x)$ and $y, y_1,\ldots,y_k \in \xi_i(x) \cap \Gamma$. We say that y dominates (strictly dominates) $y_1,\ldots,y_k$ for z if $d_i(z, W_{i-1}(y))$ is not greater than (resp. is less than) $d_i(z, W_{i-1}(y_j))$ for all $j = 1,\ldots,k$.

The idea of the algorithm is the following. Consider some i-segment S. Let $\xi = \xi_i(x)$ for $x \in S$ and let a and b respectively be the first and the last elements in S. Let $a' = \tau_i^{-1}(a)$ and $b' = \tau_i(b)$. Let $D = (y_1,\ldots,y_k)$ be a sequence of elements in $S \cup \{a', b'\}$ such that $1 \le y_1(i) < y_2(i) < \ldots < y_k(i) \le n_i$ and $d_{i-1}(y_j) < \infty$ (i.e., $W_{i-1}(y_j)$ is nonempty), $j = 1,\ldots,k$. Consider the element $x \in S$. Clearly

$$d_i^2(x) = \min\{d^2(x, W_{i-1}(y)) = (x(i) - y(i))^2 + d_{i-1}^2(y) \mid y \in \xi \cap \Gamma\}.$$

Moreover, if S is not a left segment (not a right segment), then $d_{-1}(a') = 0$ and $d(x, W_{-1}(a')) < d(x, W_{-1}(y))$ for all $y \in \xi \cap \Gamma$ such that $y(i) < a'(i)$ (resp., $d_{-1}(b') = 0$ and $d(x, W_{-1}(b')) < d(x, W_{-1}(y))$ for all $y \in \xi \cap \Gamma$ such that $y(i) > b'(i)$). Thus,

$$d_i^2(x) = \min\{(x(i) - y_j(i))^2 + d_{i-1}^2(y_j) \mid j = 1,\ldots,k\}. \tag{1}$$

Also note that for $1 \le j < j' \le k$ there is a point z on the line ξ such that

$$\begin{aligned} &y_j \text{ strictly dominates } y_{j'} \text{ for all } z' \in \xi \text{ with } z'(i) < z(i),\\ &y_{j'} \text{ strictly dominates } y_j \text{ for all } z' \in \xi \text{ with } z'(i) > z(i). \end{aligned} \tag{2}$$

Assume that $d_{i-1}^2(x)$ have been determined for $x \in B$ (in particular, $d_{i-1}^2(y)$ are known for $y \in D$). Then, using (1) and (2), we can compute $d_i^2(x)$ for all $x \in S$ by the following procedure. Moving along the segment $S' = S \cup \{a', b'\}$, successively identify the elements $y_1, y_2,\ldots$ that constitute the sequence D introduced above. If $k = |D| = 0$, then $d_i(x) = \infty$ for all $x \in S$. Assume that by the time we reach the next element y_j we have already constructed the subsequence $V = (v_1,\ldots,v_l)$, $v_1 < \ldots < v_l$, of elements from $\{y_1,\ldots,y_{j-1}\}$ and the subsequence $Z = (a = z_1, z_2,\ldots,z_l)$, $z_1(i) < \ldots < z_l(i)$, of elements from S such that:

$$\begin{gathered} \text{for each } q = 1,\ldots,l, \text{ the point } v_q \text{ dominates } y_1,\ldots,y_{j-1} \text{ for all } x \in \xi \text{ with}\\ z_q(i) \le x_i \le z_{q+1}(i) \end{gathered} \tag{3}$$

(by definition, $z_{l+1} = b$). If $j = 1$, then set $l := 1$, $v_l := v_j$, and $z_l := a$. The processing of the element y_j consists of a sequence of steps. The current step considers the pair $\{y_j, v_l\}$. Approximately solving the corresponding quadratic equation, we find the integer point z on the line ξ such that y_j strictly dominates v_l for the points $x \in \xi$ with $x(i) \ge z(i)$ and v_l dominates y_j for the points $x \in \xi$ with $x(i) \le z(i) - 1$. Three cases are possible.

1. $z(i) > b(i)$. By (2) and (3), this means that y_j does not strictly dominate $y_1,\ldots,y_{j-1}$ for any point from S. Go to find the next point y_{j+1} in D.

2. $z_l(i) < z(i) \le b(i)$. This means that y_j strictly dominates $y_1,\ldots,y_{j-1}$ for $x \in S$ with $x(i) \ge z(i)$. Set $l := 1 + 1$, $v_l := y_j$, $z_l := z$ and go to find the next point y_{j+1}.

3. $z(i) \le z_l(i)$. This means that v_l does not strictly dominate $v_1,\ldots,v_{l-1},y_j$ for any point from S. Remove v_l from V and z_l from Z. If $l = 1$, then set $v_l := y_j$ and $z_l := a$ and go to the next element y_{j+1}. If $l > 1$, then set $l := l - 1$ and go to the next step of processing y_j, examining the pair (y_j, v_l) for the new l.

As a result of the processing of y_j, the current sets V and Z still satisfy (3) (with $j - 1$ replaced by j). The final sequences V and Z obtained by processing y_k enable us to find quickly (in time $O(|S|)$) the sought distances $d_i^2(x)$, $x \in S$; specifically, these distances are calculated as $d_i^2(x) := (x(i) - v_q(i))^2 + d_{i-1}^2(v_q)$, where q is such that $z_q(i) \le x(i) < z_{q+1}(i)$ (and $x(i) \le b(i)$ for $q = l$).

The total number of steps when processing the elements y_j obviously does not exceed D_l plus the number of changes in V. Every change in V involves either removing the current v_j or adding the current y_j. Since every element y_j may be added at most once to V and removed at most once from V, the number of changes in V does not exceed $2|D|$. The other operations used in calculating the distances $d_i^2(x)$, $x \in S$, are bounded by a constant for each element in S (assuming that the segment S is given and the values $d_{i-1}^2(x)$, $x \in S$, are already known). We thus obtain a running time of the order $O(|S|)$ for the proposed procedure.

Let us now describe the general scheme of the algorithm. It consists of r iterations. The current iteration l runs in two stages. In the first stage, the algorithm identifies the set of all complete l-segments and then the set of all i-segments. In the second stage, for each l-segment S, the algorithm calculates $d_i^2(x)$, $x \in S$, by the method described above. The last iteration r thus produces the squares of the sought distances $d(x, W) = d_r(x)$ for all $x \in B$. To ensure more efficient execution of the first stages of the iterations, the original array B is preprocessed as described below before starting the first iteration.

The algorithm uses a certain number of work arrays. The elements of one array M can be identified in another array M' ("secondary" in relation to M) by their indices ("addresses") in M. In particular, when we say that the point $x \in B$ is an element of the work array M', we mean that this element in M' is the index of the point x in the original array B.

Preprocessing. The preprocessing procedure creates the auxiliary arrays M_1 and M_2 and also determines the numbers $\alpha(x)$ and $\beta(x)$, $x \in B$. The array M_1 consists of the elements (indices) of B arranged in natural lexicographic order: x precedes y in M_1 if for some $i \in \{1,\ldots,r\}$ we have $x(j) = y(j)$, $j = 1,\ldots,i-1$, and $x(i) < y(i)$. The array M_2 consists of the elements of B in reverse lexicographic order: x precedes y in M_2 if for some $i \in \{1,\ldots,r\}$ we have $x(j) = y(j)$, $j = i+1,\ldots,r$, and $x(i) < y(i)$. For $x \in B$, the numbers $\alpha(x)$ and $\beta(x)$ are defined as follows. Let y (y') be the point in B that directly follows x in the array M_1 (M_2). Then $\alpha(x)$ ($\beta(x)$) is the minimal (resp., maximal) index i for which $x(i) \neq y(i)$ (resp., $x(i) \neq y'(i)$).

These arrays can be constructed by distribution (coordinatewise) sorting of the set B, which requires $O(rN)$ operations on words of length $C + \log_2 N$ and a work space of $O(N + n)$ words (for details of distribution sorting, see [1, Sec. 5.2.5]). The set of numbers $\alpha(x)$, $\beta(x)$, $x \in B$, is also constructed in time $O(rN)$.

An i-fragment of the array M_1 (M_2) is an inclusionwise maximal subarray M' of consecutive elements, e.g., $x_1,\ldots,x_m$, such that $\alpha(x_j) \geq i$ (resp., $\beta(x_j) \leq i$) for $j = 1,\ldots,m-1$. It is easy to see that i-fragments define a partition of the array M_1 (M_2) and that the complete i-segments are in one-to-one correspondence with the nonempty intersections $M' \cap M''$, where M' (M'') is an i-fragment of the array M_1 (M_2).

First stage of iteration i. Successively scanning the elements of the array M_1 and using the numbers $\alpha(x)$, we identify the i-fragments in M_1. To each element $x \in B$ we assign the index $\gamma(x)$ of the i-fragment that contains x. We similarly identify the i-fragments in M_2 and assign to the elements $x \in B$ the indices $\delta(x)$ of the corresponding fragments in M_2. Sorting M_2 by ascending values of λ, we construct an array M of the elements $x \in B$ which is lexicographically ordered by the pair of indices $(\gamma(x), \delta(x))$. Identify in M the subarrays T of elements x with fixed $\gamma(x)$ and $\delta(x)$. For the given sorting, the elements x in each T are arranged in the order of ascending values of $x(l)$ and form a complete l-segment. Finally, successively scanning the elements of each complete l-segment T, we identify its component i-segments S.

It is easy to see that the identification of all i-segments requires $O(N)$ operations on words of length $C + \log_2 N$ and a memory space of $O(N)$ words. Using the previous bounds, we see that the algorithm on the whole performs $O(rN)$ operations on words of length $C + \min\{\log_2 N, \log_2 rm^2\}$ and uses a memory space of $O(N + n)$ words (words of length $[\log_2 rm^2]$ are needed for storing the distances $d_i^2(x)$). Q.E.D.

3. GENERALIZATIONS

1. For $x \in \Gamma$ and $i \in \{1,\ldots,r\}$, denote by $\Gamma^i(x)$ and $B^i(x)$ the sets of points $\Gamma \cap \xi_i(x)$ and $B \cap \xi_i(x)$, respectively. Consider the class $\mathscr{K}$ of nonnegative functions d: $B \times W \to R_+$ such that for each $i = 1,\ldots,r$ we have:

(i) if $x \in B$, $x' \in B^i(x)$ and $y, y' \in W_{i-1}(x)$, then $d(x', y) \leq d(x', y')$ if and only if $d(x, y) \leq d(x, y')$;

(ii) if $x, y \in W$, $z \in B^i(x)$ and either $y(i) \leq x(i) \leq z(i)$ or $y(i) \geq x(i) \geq z(i)$, then $d(z, x) \leq d(z, y)$;

(iii) if $x \in B$, $x' \in B^i(x)$, $x'(i) > x(i)$, $y \in W_{i-1}(x)$ and $y' \in W_{i-1}(x')$, then one of the following three holds:

a) $d(z', y) \leq d(z', y')$ for all $z' \in B^i(x)$,

b) $d(z', y) \geq d(z', y')$ for all $z' \in B^i(x)$,

c) there is a point $z \in B^i(x)$ such that $d(z', y) \leq d(z', y')$ for all $z' \in B^i(x)$ with $z'(i) \leq z(i)$ and $d(z', y) \geq d(z', y')$ for all $z' \in B^i(x)$ with $z'(i) \geq z(i)$.

We can check that the class $\mathscr{K}$ includes functions induced by the metrics l^p, $1 \leq p \leq \infty$ (in particular, the functions $d(x, y) = |x(1) - y(1)| + \ldots + |x(r) - y(r)|$ and $d(x, y) = \max\{|x(1) - y(1)|,\ldots,|x(r) - y(r)|\}$). $\mathscr{K}$ is the class of "distances" for which the proposed algorithm is applicable. Conditions (i) and (ii) combined are equivalent to (1): they show that in order to compute the distances $d_i(x)$, $x \in S$ (where S is an i-segment) it is sufficient to consider the white points w that satisfy $a'(i) \leq w(i) \leq b'(i)$, and for each $y \in S \cup \{a', b'\}$ it is sufficient to take one point in $W_{-1}(y)$ that is closest to y. Condition (iii) is a weaker form of condition (2) (cases a) and b) may hold in the metric l^1). The algorithm should be augmented with an oracle that produces the values of the function d (with prescribed accuracy) and another oracle that recognizes the cases a), b), and c) in (iii) and determines the corresponding "separating" point z. It is left to the reader to work out the details of the algorithm for the general case and for particular functions d.

2. Assume that the lattice Γ is the Cartesian product of r ordered numerical sets $I_i = (a_i^1, a_i^2, \ldots, a_i^{n(i)})$, $a_i^1 < \ldots < a_i^{n(i)}$, $i = 1,\ldots,r$. Form the lattice Γ^* by associating to each point $x \in \Gamma$ the vector x^* with the components $x^*(i) = j$, where $x(i) = a_i^j$, $i = 1,\ldots,r$. Consider the problem for the set $B \subset \Gamma$ and the function d: $B \times (\Gamma - B) \to R_+$ that satisfies (i)-(iii) (e.g., for the Euclidean metric d). Conditions (i)-(iii) are obviously preserved on passing from Γ to Γ^* (and to the induced B^* and d^*). We can thus pass from a problem on the "nonhomogeneous" lattice Γ to a problem on Γ^* and solve the latter by the proposed algorithm with virtually the same time and memory bounds.

3. Let us consider the following generalization of the original problem: for each point in $B \subset \Gamma$, find the Euclidean distance from some set $W \subset \Gamma$ disjoint with B (but not necessarily $W = \Gamma - B$). If the set $B' = \Gamma - W$ is "not too large," this problem can be reduced to the original problem with the set B' and solved by the proposed algorithm.

LITERATURE CITED

1. D. Knuth, The Art of Computer Programming [Russian translation], Vol. 3, Mir, Moscow (1978).

AN ITERATIVE METHOD FOR COMPUTING CLOSED QUEUEING NETWORKS

D. M. Sologub

UDC 519.8

An iterative method is proposed for calculating closed queueing networks. The method successively calculates the state probabilities of each server treated as a closed queueing system.

The formula for the state probabilities of a closed queueing network under steady conditions has the form [1]

$$P_{m^{(1)}-i_1,\ldots,m^{(k)}-i_k,\ldots,m^{(R)}-i_R} = \prod_{k=1}^{R} \delta_k^{m^{(k)}-i_k} \frac{\alpha_k^{i_k}}{(m^{(k)}-i_k)!} \times$$

$$\times \left[\sum_{m^{(k)}=0}^{m} \sum_{i_k=0}^{m^{(k)}} \prod_{k=1}^{R} \delta_k^{m^{(k)}-i_k} \frac{\alpha_k^{i_k}}{(m^{(k)}-i_k)!} \right]^{-1}, \qquad (1)$$

$$m^{(1)} + \ldots + m^k + \ldots + m^R = m, \quad 0 \leqslant i_1 \leqslant m^{(1)}, \ldots, 0 \leqslant i_k \leqslant$$

$$\leqslant m^{(k)}, \ldots, 0 \leqslant i_R \leqslant m^{(R)},$$

where R is the number of server nodes in the closed network, m is the total number of jobs circulating in the system, δ_k is the transportation time of jobs from node $(k - 1)$ to node k as a proportion of the total transportation time in the network, α_k is the load factor of node k.

Calculations using formula (1) involve solution of complex combinatorial problems, which is impossible without a computer. We propose an approximate iterative method for calculating closed queueing networks, which produces a result with prescribed accuracy using a hand-held calculator. The proposed method relies on the following property of closed queueing networks.

Translated from Kibernetika i Sistemnyi Analiz, No. 2, pp. 181-183, March-April, 1992. Original article submitted July 10, 1991.