AN ALGORITHM FOR SOLVING THE ASSIGNMENT PROBLEM

1969

SovietRxiv

Abstract

Full Text

UDC 518.5

E. A. DINITS, M. A. KRONROD

AN ALGORITHM FOR SOLVING THE ASSIGNMENT PROBLEM

(Presented by Academician M. V. Keldysh on 11 III 1969)

Let there be a square matrix (a_{ij}) of order n. A solution of the assignment problem for this matrix is a set of n elements of the matrix, one in each column and in each row, such that the sum of these elements is minimal among all such sets. Algorithms are known that solve the assignment problem in Cn^4 operations*, for example, the Bradford method.

In the present paper an algorithm is constructed that solves the problem faster: one modification in $Cn^3 \lg n$, another in Cn^3 operations.

Definition. Let there be some vector $\Delta = (\Delta_1, \Delta_2, \dots, \Delta_n)$. An element a_{ij} is called Δ-minimal if $a_{ij} - \Delta_j \leqslant a_{ik} - \Delta_k$ for all k.

Lemma. Suppose that for some Δ there is a set of n Δ-minimal elements $a_{1j(1)}, a_{2j(2)}, \dots, a_{nj(n)}$, one in each column and in each row. Then this set is a solution of the assignment problem.

Proof.

$$\sum_{i=1}^{n} a_{ij(i)} = \sum_{k=1}^{n} \Delta_k + \sum_{i=1}^{n} (a_{ij(i)} - \Delta_{j(i)}).$$

The first sum on the right-hand side is constant over all sets, and the second is minimal.

Suppose that for some vector Δ we have selected in each row one of the Δ-minimal elements (we shall call these selected elements bases). The defect of a set of bases is the number of free columns (columns without bases). It follows from the lemma that a set of bases with defect equal to zero is a solution of the problem.

Below we give an algorithm that makes it possible to carry out an iteration: given a vector Δ and a set of bases with defect not equal to zero, find a new vector and a set of bases having a smaller defect.

Taking the zero vector Δ and some set of bases for it, we can solve the problem by successive iterations.

Iteration. Suppose that for some vector Δ we have a set of bases $a_{1j(1)}, a_{2j(2)}, \ldots, a_{nj(n)}$ with defect $m \neq 0$. Mark some free column. Let its number be s_1. Increase Δ_{s_1} by the maximal δ such that all bases remain Δ-minimal elements (the method for finding δ is indicated below). We obtain that for some i, $a_{ij(i)} - \Delta_{j(i)} = a_{is_1} - \Delta_{s_1}$, i.e., the element a_{is_1} has become Δ-minimal. We shall call it an alternative base and mark the column with number $s_2 = j(i)$, containing the base of this row (we now have two marked columns). Increase Δ_{s_1} and Δ_{s_2} by the maximal δ such that all bases remain Δ-minimal, find a new alternative base in one of the columns and mark the new column, and so on, until we mark a column with two or more bases.

Now we shall construct a new set of bases. Note that in each row there is no more than one alternative base.

By replacement of the base in a row we shall mean the following operation: the alternative base in this row is declared to be the base, and the old base ceases

* What is meant is the number of operations performed by a computing machine when running a program implementing the algorithm. C is a multiplier independent of n.

be a basis. We perform a change of basis in the row in which the last alternative basis lies. At the same time, in the last marked column the number of bases will decrease by 1. In the column where the new basis has appeared, we again take the old basis and perform a change of bases in the corresponding row, and so on, until a basis appears in the column with number s_1. Thus we obtain the vector Δ that we need and a set of bases for it with defect equal to $m - 1$.

It is easy to see that each iteration of the described algorithm requires Cn^2 operations, not counting the operations involved in computing δ at each step of the iteration. Since the number of iterations does not exceed n, we are interested only in the time for computing δ.

We shall now show various ways of computing δ. Consider an arbitrary step of some iteration.

Let S be the set of numbers of the marked columns, and let R be the set of numbers of those rows in which there are no alternative bases. Then we have

$$\delta = \min_{i \in R,\ k \in S} [(a_{ik} - \Delta_k) - (a_{ij(i)} - \Delta_{j(i)})].$$

If at each step of the iteration δ is computed directly from this formula as the minimum of an array of length of order n^2, then the whole algorithm will require Cn^4 operations.

We can write

$$\delta = \min_{k \in S} \left(\min_{i \in R} [(a_{ik} - \Delta_k) - (a_{ij(i)} - \Delta_{j(i)})] \right)$$

—this formula is used by the first modification of the algorithm, which makes it possible to find

$$\min_{i\in R}[(a_{ik}-\Delta_k)-(a_{ij(i)}-\Delta_{j(i)})]$$

quickly and requires $Cn^3\lg n$ operations, or

$$\delta=\min_{i\in R}\left[\min_{k\in S}(a_{ik}-\Delta_k)-(a_{ij(i)}-\Delta_{j(i)})\right]$$

—this formula is used by the second modification, which makes it possible to find $\min_{k\in S}(a_{ik}-\Delta_k)$ quickly and requires Cn^3 operations.

First modification. We introduce a certain method of recording the ordering of an array of numbers without permuting its elements, making it possible to react quickly to the deletion (without renumbering) of its elements.

Suppose we have an array of n numbers. To record the ordering, an array of n information cells is required. In the cell corresponding to a certain element of the array, we place pointers to the preceding and following elements (in the sense of the ordering), i.e., their numbers.

Suppose we need to delete a certain element of the array. From its information cell we learn the numbers of its neighbors in the ordering. We change their information cells, replacing the pointers to the element being deleted by pointers to its opposite neighbor. Obviously, the record of the remaining array will be a record of its ordering in the sense indicated above.

The initial ordering requires $Cn\lg n$ operations; changes upon deletion require C operations.

We use this method for rapidly finding

$$\min_{i\in R}[(a_{ik}-\Delta_k)-(a_{ij(i)}-\Delta_{j(i)})]$$

for each $k\in S$.

Let

$$b_{ik}=[(a_{ik}-\Delta_k)-(a_{ij(i)}-\Delta_{j(i)})].$$

We order each of the columns (b_{ik}) and record this ordering by columns in a table of $n\times n$ information cells. Separately, we write out a row of the column minima, equal to $\min_i b_{ik}$ at the beginning of the iteration.

In the course of carrying out the iteration, we must delete from the matrix B those rows in which alternative bases have appeared, so that the minimum over the columns of the remaining matrix is the desired $\min_{i\in R} b_{ik}$. The corresponding modification of the information table and of the row of minima, as well as finding δ from the formula

$$\delta=\min_{k\in S}\left(\min_{i\in R} b_{ik}\right),$$

requires Cn operations at each step of the iteration. But because the construction of the information table for each iteration requires $Cn^2 \lg n$ operations, the total length of the algorithm is $Cn^3 \lg n$ operations.

Second modification. We have

$$\delta = \min_{i \in R} \left[\min_{k \in S} (a_{ik} - \Delta_k) - (a_{ij(i)} - \Delta_{j(i)}) \right].$$

Consider the vector $\mathbf{q} = (q_1, q_2, \ldots, q_n)$, where

$$q_i = \min_{k \in S} (a_{ik} - \Delta_k).$$

We have

$$\delta = \min_{i \in R} \left[q_i - (a_{ij(i)} - \Delta_{j(i)}) \right].$$

It follows from this formula that, in order to solve the problem in Cn^3 operations, it is enough to be able to compute, at each step, the vector $\mathbf{q}$ in Cn operations. We show how this can be done.

At the first iteration step the vector $\mathbf{q}$ coincides with the first marked column. At each subsequent step the vector $\mathbf{q}$ is obtained from the old vector $\mathbf{q}$ and from the new marked column by the following formula:

$$q_i' = \min(a_{is_m}; q_i - \delta),$$

where s_m is the number of the newly marked column.

Moscow State University
named after M. V. Lomonosov

Received
27 II 1969

Note: Figure translations are in progress. See original paper for figures.