SIAM J. COMPUT.
Vol. 7, No. 2, May 1978

AN $O(N \cdot \log N)$ ALGORITHM FOR A CLASS OF MATCHING PROBLEMS*

NIMROD MEGIDDO† AND ARIE TAMIR‡

Abstract. The following class of matching problems is considered. The vertices of a complete undirected graph are indexed $1, \cdots, n$, where $n = 2m$. Every vertex i is assigned two numbers a_i, b_i. The length of every edge (i, j), where $i < j$, is $d(i, j) = a_i + b_j$. This class of weighted graphs is applicable to scheduling and optimal assignment problems. A maximum weighted (perfect) matching is found in $O(n \cdot \log n)$ operations.

Key words. matching, assignment, scheduling, polynomial-time algorithm, 2-3 trees

1. Introduction. The maximum matching problem has many applications in operations research. The first polynomial-time bounded algorithm for the maximum weighted matching problem is Edmonds' [2]. The most efficient algorithm for the maximum (cardinality) matching, known to the authors, is Even and Kariv's [3]. Gabow [4] has the most efficient algorithm for the weighted matching. In this paper we focus on a subclass of maximum weighted matching problems (see § 2 for a precise definition). Our study is motivated by the following two problems which are easily shown to belong to our class.

In the first problem, a group of individuals, ordered by seniority, is to be partitioned into teams, having the same mission. Each team consists of two positions—a senior position and a junior one. The senior position must be manned by the more senior individual between the members of the team. Assuming that we know the effectiveness of each individual in both the senior and the junior positions, we wish to maximize the total effectiveness of the teams.

The second problem is to schedule $2m$ jobs to m identical processors, two jobs to each processor, preserving the arrival ordering. The objective is to minimize the total flow time, or equivalently, the average waiting time of a job.

Using a dynamic programming approach, these two models can be solved in $O(m^2)$ time. In this paper we present an algorithm which solves the above problems in $O(m \cdot \log m)$ operations.

2. Preliminaries. Our goal is to develop an efficient algorithm for the following problem.

Problem 1. Given numbers a_i, b_i, $i = 1, \cdots, n$ $(n = 2m)$, find a perfect matching $(i_1, j_1), \cdots, (i_m, j_m)$, where $i_k < j_k$, $k = 1, \cdots, m$, which maximizes $\sum_{k=1}^{m} (a_{i_k} + b_{j_k})$.

We may assume without loss of generality that a maximum matching $(i_1, j_1), \cdots, (i_m, j_m)$ satisfies $i_k < i_{k+1}$, $j_k < j_{k+1}$, $k = 1, \cdots, m-1$. In view of this we shall restrict our attention to matchings $(i_1, j_1), \cdots, (i_m, j_m)$ which satisfy $i_k < j_k$, $k = 1, \cdots, m$, and $i_k < i_{k+1}$, $j_k < j_{k+1}$, $k = 1, \cdots, m-1$. These can be handled by introducing the following notation.

Let $x = (x_1, \cdots, x_n)$ be a vector whose components are either 1 or -1. Denote $H_i(x) = \sum_{k=1}^{i} x_k$, $i = 1, \cdots, n$. Let X be the set of all vectors x $(x = (x_1, \cdots, x_n)$, $x_i \in \{1, -1\})$ such that $H_i(x) \geqq 0$, $i = 1, \cdots, n-1$ and $H_n(x) = 0$. Consider the following problem.

---

* Received by the editors April 7, 1977.

† Department of Statistics, Tel-Aviv University, Tel-Aviv, Israel. Now at Department of Business Administration, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801.

‡ Department of Statistics, Tel-Aviv University, Tel-Aviv, Israel.

Problem 2. Maximize $c(x) \equiv \sum_{i=1}^{n} (a_i - b_i) \cdot x_i$ over X.

We claim that Problems 1 and 2 are equivalent. Specifically, if $(i_1, j_1), \cdots, (i_m, j_m)$ solves Problem 1 then the vector x, where $x_k = 1$ if $k = i_q$ and $x_k = -1$ if $k = j_q$, solves Problem 2. Conversely, if x solves Problem 2 then a solution to Problem 1 is defined recursively as follows. Let $i_1 = 1$. Suppose that $i_1, \cdots, i_q$ and $j_1, \cdots, j_r$ $(0 \leqq r \leqq q)$ have been defined, and $\{i_1, \cdots, i_q, j_1, \cdots, j_r\} = \{1, \cdots, q+r\}$. Then, if $x_{q+r+1} = 1$ let $i_{q+1} = q+r+1$ and if $x_{q+r+1} = -1$ let $j_{r+1} = q+r+1$. Thus, we shall henceforth be dealing with Problem 2. We note that Problem 2 can be transformed to a linear program with a totally unimodular matrix whose basic solutions yield solutions to our problem. Thus, the theory of linear programming suffices for solving Problem 2. However, we shall present an algorithm which is more suitable. Our algorithm is based on the following theorem.

THEOREM. *A vector x solves Problem 2 if and only if the following condition holds. For every pair i, j, $1 \leqq i < j \leqq n$,* (i) *if $x_i = -1$, $x_j = 1$ then $a_i - b_i \leqq a_j - b_j$ and* (ii) *if $x_i = 1$, $x_j = -1$ and $H_k(x) \geqq 2$, for $i \leqq k < j$, then $a_i - b_i \geqq a_j - b_j$.*

Necessity is obvious, since if the condition does not hold, then by defining $y_i = -x_i$, $y_j = -x_j$, and $y_k = x_k$ for $k \neq i, j$ we have $y \in X$ and $c(y) > c(x)$. We shall now prove the sufficiency of the condition. For $x, y \in X$ define a metric $D(x, y) = \#\{i: x_i \neq y_i\}$. Suppose that $x \in X$ does not solve Problem 2 and let $y \in X$ be a solution to Problem 2, which is nearest (with respect to (w.r.t.) D) to x. Let i be the smallest index such that $x_i = 1$ and $y_i = -1$. Let j be the smallest index such that $x_j = -1$ and $y_j = 1$. If $i > j$ then for every $k, j \leqq k < i$, $H_k(y) \geqq 2$. It follows that $a_j - b_j > a_i - b_i$ (equality cannot occur since it implies $D(x, z) < D(x, y)$, $c(z) = c(y)$, $z \in X$, where $z_i = 1$, $z_j = -1$, $z_k = y_k$ for $k \neq i, j$). Thus, part (i) of the condition does not hold. If $i < j$ then for every $k, i \leqq k < j$, $H_k(x) \geqq 2$. Similar arguments imply $a_j - b_j > a_i - b_i$ and in this case part (ii) does not hold.

3. The algorithm. We shall first describe our algorithm in general terms and then elaborate on its details. In this section we concentrate on the validity of the algorithm; an estimate of the number of operations is given in § 4.

Let $M_1 = \{1, \cdots, m\}$, $M_2 = \{m+1, \cdots, n\}$. For every $x \in X$ let

$$I(x) = \min \{i \in M_1: H_k(x) \geqq 2 \text{ for all } k, i \leqq k \leqq m\},$$

$$J(x) = \max \{j \in M_2: H_k(x) \geqq 2 \text{ for all } k, m+1 \leqq k \leqq j-1\}.$$

Our algorithm generates a sequence $x^0, \cdots, x^r$ of vectors in X such that $D(x^{k-1}, x^k) = 2$. This sequence develops according to the following scheme.

Scheme.

0. Initiate with $x = (1, \cdots, 1, -1, \cdots, -1) \in X$.
1. Find an $i \in M_1$ such that $x_i = 1$, $i \geqq I(x)$ and $a_i - b_i = \min \{a_k - b_k: I(x) \leqq k \leqq m, x_k = 1\}$; find a $j \in M_2$ such that $x_j = -1$, $j \leqq J(x)$ and $a_j - b_j = \max \{a_k - b_k: m+1 \leqq k \leqq J(x), x_k = -1\}$.
2. If $a_i - b_i \geqq a_j - b_j$ then terminate; otherwise, set $x_i = -1$, $x_j = 1$ and go to 1.

Let x^i $(i = 0, 1, \cdots)$ denote the vector x stored after i executions of step 2, and suppose that the scheme terminates after r iterations. It can be easily verified that $c(x^{k-1}) < c(x^k)$, $k = 1, \cdots, r$. Moreover, since $H_m(x^k) = m - 2k$ and $H_m(x^k) \geqq 0$ for $k = 0, 1, \cdots, r$, it follows that $r \leqq m/2$.

We shall now prove that upon termination the vector $x = x^r$ is a solution to Problem 2. This is done by verifying that the condition stated in the theorem is satisfied. Let $i < j$ be any pair $(1 \leqq i, j \leqq n)$. Distinguish cases: (i) $x_i = -1$, $x_j = 1$. If $i, j \in M_1$ then there is $q < r$ such that $x_i^q = 1$, $i \geqq I(x^q)$ and $a_i - b_i =$

$\min\{a_k - b_k : I(x^q) \leqq k \leqq m,\ x_k^q = 1\}$. This implies $a_i - b_i \leqq a_j - b_j$. Analogous arguments hold in case $i, j \in M_2$. The case $i \in M_1$, $j \in M_2$ can be handled by applying this type of arguments twice. (ii) $x_i = 1$, $x_j = -1$, and $H_k(x) \geqq 2$ for $i \leqq k < j$. If $i, j \in M_1$ then there is $q < r$ such that $x_j^q = 1$, $j \geqq I(x^q)$ and $a_j - b_j = \min\{a_k - b_k : I(x^q) \leqq k \leqq m, x_k^q = 1\}$. However, since $H_k(x^q) \geqq H_k(x)$ $(k = 1, \cdots, m)$, it follows that $i \geqq I(x^q)$ and hence $a_i - b_i \geqq a_j - b_j$. A similar argument holds in case $i, j \in M_2$. If $i \in M_1$ and $j \in M_2$ then termination implies $a_i - b_i \geqq a_j - b_j$.

In fact, the sequence $x^0, \cdots, x^r$ can be generated without calculating the values $I(x)$, $J(x)$, $H_k(x)$ explicitly. This can be performed as follows. First, the elements i of M_1 are sorted according to increasing magnitude of $a_i - b_i$ and the elements j of M_2 are sorted according to decreasing magnitude of $a_j - b_j$. Let x^q be a vector in the sequence generated by the scheme. Let A_1 denote the ordered (by the natural order on M_1) q-tuple of the indices $i \in M_1$ such that $x_i^q = -1$. An index $i \in A_1$ is called a right minimum if $H_i(x^q) < H_k(x^q)$ for every $k \in M_1$ such that $k > i$. Let B_1 denote the ordered set of right minima. Linearly ordered sets A_2, B_2 are defined in analogous manner with respect to the elements in M_2; A_2 is the ordered tuple of the indices $j \in M_2$ such that $x_{j+1}^q = 1$ and B_2 consists of those $j \in A_2$ such that $H_j(x^q) < H_k(x^q)$ for every $k < j$ $(k \in M_2)$. Once the lists A_1, B_1, A_2, B_2 (w.r.t. a vector x) are known, it is easy to execute step 1 of the scheme. The following algorithm generates the same sequence as that generated by the scheme, and at the same time maintains the lists A_1, B_1, A_2, B_2. Our algorithm operates symmetrically on the sets M_1, M_2. Hence we shall describe in detail only the part concerning M_1.

ALGORITHM.

Phase I: Sort the elements i of M_1 to form a list L_1 arranged in order of increasing magnitude of $a_i - b_i$; sort M_2 to form a list L_2 arranged in decreasing order.

Phase II:

0. Initiate with $x = (1, \cdots, 1, -1, \cdots, -1) \in X$ and $A_1 = B_1 = A_2 = B_2 = \emptyset$.
1. Let i be the first element in L_1 and let $s = \#\{k : k \in A_1, k < i\}$.
2. If $i - 2s < 2$ then delete i from L_1 and go to 1; otherwise go to 3.
3. If there is no $k \in B_1$ such that $k > i$ then set $i^* = \infty$, $s^* = 0$ and go to 5; otherwise let i^* be the smallest element of B_1 such that $i^* > i$ and let $s^* = \#\{k : k \in A_1, k < i^*\}$.
4. If $i^* - 2(s^* + 1) < 2$ then delete i from L_1 and go to 1; otherwise go to 5.
5. Pick an element $j \in M_2$ in a manner similar to that by which i is picked from M_1 (see steps 1–4; j is the first in L_2 such that $2m - (j-1) - 2t \geqq 2$, where $t = \#\{k : k \in A_2, k \geqq j\}$, and either there is no $k \in B_2$ such that $k < j$, or $2m - j^* - 2t^* \geqq 2$, where j^* is the largest element of B_2 such that $j^* < j - 1$ and $t^* = \#\{k : k \in A_2, k \geqq j^*\}$).
6. If $a_i - b_i \geqq a_j - b_j$ then terminate; otherwise, set $x_i = -1$, $x_j = 1$ and go to 7.
7. Delete i from L_1 and insert i into A_1.
8. If $i - 2(s+1) \geqq i^* - 2(s^* + 2)$ then set $i = i^*$, $s = s^* + 1$ and go to 9; otherwise insert i into B_1.
9. If there is no $k \in B_1$ such that $k < i$ then go to 11; otherwise let i' be the largest element of B_1 such that $i' < i$ and let $s' = \#\{k : k \in A_1, k < i'\}$.
10. If $i' - 2(s' + 1) < i - 2(s + 1)$ then go to 11; otherwise delete i' from B_1 and go to 9.
11. Perform on j, A_2, B_2 operations similar to those performed on i, A_1, B_1 in steps 7–10 (delete j from L_2; insert $j - 1$ into A_2, if $j > m + 1$; insert $j - 1$ into B_2 if it has become a "left minimum" and delete from B_2 those elements that have ceased from being left minima).
12. Go to 1.

4. The efficiency of the algorithm. We may employ the device of a 2-3 tree (see [1, p. 146] for a precise definition) for handling the linearly ordered sets A_1, B_1, A_2, B_2 in our algorithm. Again, the symmetry enables us to restrict our attention to A_1 and B_1. Let T be a 2-3 tree which represents A_1. For every vertex v of T which is not a leaf, $L[v]$ is the largest element of A_1 assigned to the subtree whose root is the leftmost son of v; $M[v]$ is the largest element of A_1, assigned to the subtree whose root is the second son of v. For every vertex v of T let $a(v)$ denote the number of leaves of the subtree rooted in v, and let $b(v)$ denote the number of leaves of this subtree storing an element of B_1.

It can be easily verified (see [1]) that each one of the following operations can be executed in at most $O(\log n)$ steps: (a) Find the smallest element of A_1 which is greater than a given $i \in M_1$. (b) Find the smallest [largest] element of B_1 which is greater [smaller] than a given $i \in M_1$. (c) Insert an element into A_1. (d) Insert an element of A_1 into B_1. (e) Calculate s, s^*, s'.

Since each one of the operations listed above can be executed no more than $O(n)$ times in Phase II of our algorithm, and since these are essentially all the operations executed during Phase II, it follows that Phase II requires no more than $O(n \cdot \log n)$ steps. It is well-known that Phase I can also be executed in $O(n \cdot \log n)$ steps (see [1]).

REFERENCES

[1] A. V. AHO, J. E. HOPCROFT AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[2] J. EDMONDS, *Paths, trees and flowers*, Canad. J. Math., 17 (1965), pp. 449–467.

[3] S. EVEN AND O. KARIV, *An $O(n^{5/2})$ algorithm for maximum matching in general graphs*, presented at the 16th Annual Symposium on Foundations of Computer Science, Univ. of California at Berkeley, October 1975.

[4] H. N. GABOW, *An efficient implementation of Edmonds' algorithm for maximum matching on graphs*, J. Assoc. Comput. Mach., 23 (1976), pp. 221–234.