AN ALGORITHM TO SOLVE THE $m \times n$ ASSIGNMENT PROBLEM IN EXPECTED TIME $O(mn \log O)$

by

R.M. Karp

Memorandum No. UCB/ERL M78/67

27 September 1978

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

AN ALGORITHM TO SOLVE THE $m \times n$ ASSIGNMENT PROBLEM IN EXPECTED TIME $O(mn \log n)$[†]

Richard M. Karp[††]

Abstract

We give an algorithm to solve the m-source, n-destination assignment problem in expected time $O(mn \log n)$ under the assumption that the edge costs are independent random variables and the costs of the edges incident with any given source are identically distributed. The algorithm uses a new implementation of priority queues.

Key Words. combinatorial optimization, network flow, matching, assignment problem, priority queue, analysis of algorithms, data structure

[†]Research supported by National Science Foundation Grant MCS77-09906.

[††]Computer Science Division, Department of Electrical Engineering and Computer Sciences and the Electronics Research Laboratory, University of California, Berkeley, CA 94720.

1. Introduction

An instance of the assignment problem is specified by a complete undirected bipartite graph, together with an assignment of a nonnegative cost to each edge. The graph is denoted $G = (V,E)$, where the vertex set V is the union of two disjoint sets, X (the sources) and Y (the destinations), such that $|X| \leq |Y|$; the edge set E consists of all two-element sets $\{x,y\}$ such that $x \in X$ and $y \in Y$. The cost of the edge $e = \{x,y\}$ is denoted $c(e)$ or $c(x,y)$. A matching in G is a set $M \subseteq E$ such that every vertex is incident with at most one edge in M. The cost of M, denoted $c(M)$, is $\sum_{e \in M} c(e)$. A matching M is full if every source is incident with some edge in M. The assignment problem asks for a full matching of minimum cost.

The assignment problem can be solved in time $O(|X|^2 \cdot |Y|)$ by an algorithm based on the repeated computation of minimum-cost augmenting paths ([4],[5],[9]). In the present paper we show that a variant of this algorithm has expected execution time $O(|X| \cdot |Y| \cdot \log|Y|)$, provided the costs of the edges are independent random variables and the costs of the edges incident with any given source are identically distributed.

2. Review of Augmenting Path Methods

We review the augmenting path approach to the assignment problem. Further background material on augmenting path methods can be found in [5] and [8].

Given a matching M, a vertex v is called free if it is incident with no edge in M. A path in G is called alternating if its edges are alternately in M and not in M. A simple alternating path between free vertices is called an augmenting path. If P is an augmenting path then one of its end points is a source and the other is a destination. If M is a matching and P is an augmenting path then their symmetric difference is also a matching, and $|M \oplus P| = |M| + 1$. The cost of the augmenting path P is $c(M \oplus P) - c(M)$, which can be expressed as $\sum_{e \in P-M} c(e) - \sum_{e \in P \cap M} c(e)$.

The following well-known lemma ([5]) is fundamental.

Lemma 1. If M is of minimum cost among matchings of cardinality k and P is of minimum cost among augmenting paths relative to M, then $M \oplus P$ is of minimum cost among matchings of cardinality $k+1$.

Lemma 1 is the basis of the following algorithm to solve the assignment problem.

ASSIGNMENT ALGORITHM - VERSION 1

```
begin
M ← ∅;
while |M| < |X| do
    begin
    let P be a minimum-cost augmenting path relative to M;
    M ← M ⊕ P
    end;
end
```

We will arrive at our eventual algorithm by refining and specifying in greater detail the process of finding the augmenting path P. The augmenting paths relative to a matching M can be determined using an associated directed graph $\tilde{G} = (V, \tilde{E})$ called an augmentation graph. The set of directed edges $\tilde{E}$ consists of forward edges and backward edges, specified as follows:

if $\{x,y\} \in E$, $x \in X$ and $y \in Y$ then:

if $\{x,y\} \notin M$ then $\langle x,y \rangle$ is a forward edge;

if $\{x,y\} \in M$ then $\langle y,x \rangle$ is a backward edge.

A minimum-cost augmenting path P is found as follows:

To each forward edge $<x,y>$ assign cost $c(x,y)$; to each backward edge $<y,x>$ assign cost $-c(x,y)$; let $\tilde{P}$ be a minimum-cost directed path from a free source to a free destination; then $P = \{\{x,y\} \mid <x,y> \in \tilde{P} \text{ or } <y,x> \in \tilde{P}\}$.

References [4] and [9] give an improvement on VERSION 1. This improvement depends on the fact that a minimum-cost path in a digraph from one given set of vertices to another given set can be computed rapidly provided all edge costs are nonnegative. The idea of the improvement is to modify the costs of the edges in the augmentation network so that:

(i) the identity of the minimum-cost directed path (or paths) from a free vertex in X to a free vertex in Y is unchanged

and (ii) all edge costs are nonnegative.

This is achieved by associating with each vertex v a "potential" $\alpha(v)$ which affects additively the costs of all edges of $\tilde{G}$ incident with v. The details of the scheme are given in the next version of the algorithm.

ASSIGNMENT ALGORITHM - VERSION 2

```
begin
M ← ∅
for v∈V do α(v) ← 0
while |M| < |X| do
    begin
    form the augmentation network G̃;
    for each {x,y}∈E-M assign directed edge <x,y> the cost
        $\bar{c}(x,y) = c(x,y)+\alpha(x)-\alpha(y)$;
    for each {x,y}∈M assign directed edge <y,x> the cost 0;
    for all v∈V do
        begin
        let γ(v) be the minimum cost of a directed path in G̃ which begins
            at a free source and ends either at v or at a free destination;
        α(v) ← α(v)+γ(v)
        end
        let $\tilde{P}$ be a minimum-cost directed path in $\tilde{G}$ from a free source to a
            free destination;
        let P be the set of edges in G corresponding to edges in $\tilde{P}$;
        $M \leftarrow M \oplus P$
        end
    end
```

It can be proven inductively that the following properties hold at the beginning of each iteration of the while loop:

(i) for each free source x, $\alpha(x) = 0$;

(ii) for each free destination y, $\alpha(y) = \alpha^* = \max_{v \in V} \alpha(v)$;

(iii) for each edge $\{x,y\}$ in E, $\bar{c}(x,y) = c(x,y) + \alpha(x) - \alpha(y) \geq 0$;

(iv) for each edge $\{x,y\}$ in M, $\bar{c}(x,y) = c(x,y) + \alpha(x) - \alpha(y) = 0$;

(v) if P is a directed path in $\tilde{G}$ from a free source to a free destination, then

cost of P relative to the edge costs in VERSION 1
- cost of P relative to the edge costs in VERSION 2
$= \alpha^*$

Properties (iii) and (iv) ensure that all shortest-path computations are performed on networks with nonnegative costs. Property (v) ensures that, at each iteration, the algorithm indeed computes a minimum-cost augmenting path; hence the algorithm solves the assignment problem correctly.

3. A Minimum-Cost Path Algorithm Using Priority Queues

Efficient algorithms to compute minimum-cost paths from a single source to all the vertices of a network with nonnegative edge costs were suggested two decades ago by Dantzig [2] and Dijkstra [3]. Many refinements have been suggested since (see, for example, [6]). The computation of the path

$\tilde{P}$ and the quantities $\gamma(v)$ involves a variant of the single-source minimum-cost path problem having the following input data: a digraph $\bar{G} = \langle V,\bar{E}\rangle$; a set $S \subseteq V$ of <u>start vertices</u>; a set $T \subseteq V$ of <u>target vertices</u> such that $S \cap T = \phi$ and there exists a path from S to T; for each edge $\langle u,v\rangle$ a nonnegative cost $d(u,v)$. For each $v \in V$, let $OUT(v)$ denote the set of edges of $\bar{G}$ directed out of v, and let $\gamma(v)$ denote the minimum cost of a path from a vertex in S to a vertex in $T \cup \{v\}$. The following algorithm computes $\gamma(v)$ and the associated minimum-cost paths.

MINIMUM-COST PATH ALGORITHM - VERSION 1

```
begin
PATHSET ← ∅;
EDGE ← ∅;
R ← S;
for u∈R do γ(u) ← 0; EDGE ← EDGE ∪ OUT(u);
while R∩T = ∅ do
   begin
   choose <x,y>∈EDGE so that γ(x)+d(x,y) =      min       [γ(u)+d(u,v)];
   EDGE ← EDGE - {<x,y>};                 <u,v>∈EDGE
   if y∉R then
      begin
      PATHSET ← PATHSET ∪ {<x,y>};
      R ← R ∪ {y};
      γ(y) ← γ(x)+d(x,y);
      EDGE ← EDGE ∪ OUT(y)
      end
   end;
for v∉R do γ(v) ← γ(y);
end
```

At the beginning of each execution of the body of the while loop R denotes the set of vertices v for which $\gamma(v)$ has been determined. The set PATHSET contains, for each $v \in R-S$, the last edge in a minimum-cost path from S to v. The set EDGE contains those edges which are directed out of vertices in R and have not been examined during the execution of the algorithm.

The bulk of the execution time of the algorithm is spent in performing the following operations on the set EDGE:

(i) EDGE $\leftarrow$ EDGE $\cup$ OUT(y); this operation is performed whenever a a vertex y enters R.

(ii) selecting an edge $\langle x,y \rangle \in$ EDGE to minimize $\gamma(x) + c(x,y)$.

The complexity of these operations depends critically on the data structure chosen to represent EDGE. A natural choice is a data structure called a <u>priority queue</u> ([1],[7]). A priority queue represents a set of items of the form $\langle x,\theta \rangle$, where x is an arbitrary data object and θ is a real number called the <u>value</u> of the item; it supports the operations of initialization, insertion and deletion, which are defined as follows:

Operation	Execution Time
Initialize the queue by inserting N items	O(N)
Insert a new item into a queue containing N items	O(log N)
Delete an item of minimum value from a queue containing N items	O(log N)

The total execution time of operations on the priority queue EDGE during the execution of the minimum-cost path algorithm is $O(|E|\log|E|)$, since each edge is inserted at most once and deleted at most once. The total time for other operations is $O(|V|)$, and hence the over-all execution time of the algorithm is $O(|V| + |E|\log|E|)$.

The computation of an optimum assignment requires $|X|$ executions of the minimum-cost path algorithm. At each execution the digraph is the augmentation digraph with respect to the current matching, the start vertices are the free sources and the target vertices are the free destinations. The cost of a forward edge $<x,y>$ is $\bar{c}(x,y) = c(x,y) + \alpha(x) - \alpha(y)$, and the cost of a backward edge is zero.

It will be convenient to modify the minimum-cost path algorithm in two ways when it is used as a subroutine within an assignment algorithm.

(i) Backward edges of the form $<y,x>$ are not explicitly inserted into EDGE. Instead, whenever a destination y enters R and $\{x,y\} \in M$, x enters R immediately and $\gamma(x)$ is set equal to $\gamma(y)$. This is valid since $<y,x>$ has cost zero.

(ii) When a source x enters R, each pair $<x,y>$ such that $y \in Y$ is inserted into EDGE. This insertion is erroneous in case $\{x,y\} \in M$; in such a case $<x,y>$ is not an edge of $\tilde{G}$. To compensate, every time an edge $<x,y>$ is selected from EDGE, it is discarded if $\{x,y\} \in M$.

Since each augmentation graph has $|X| + |Y|$ vertices and at most $|X| \cdot |Y|$ edges, the execution time for each augmenting path computation is $O(|X| \cdot |Y| \cdot \log |Y|)$, and the over-all execution time of the assignment algorithm is $O(|X|^2 \cdot |Y| \cdot \log |Y|)$. A different implementation of the minimum-cost path algorithm is possible [3] which leads to a worst-case execution time of $O(|X|^2 \cdot |Y|)$, for the assignment algorithm, but does not lend itself to the modification which is examined in the next two sections, and is the key point of the present paper.

4. A Modified Assignment Algorithm

When executed on typical examples the assignment algorithm we have presented spends most of its time pouring edges into the set EDGE; very few of these edges are ever selected from EDGE. In this section we present a trick which tends to reduce drastically the number of priority queue insertion operations executed by the algorithm, at the cost of a modest increase in the number of deletions. The trick is based on the use of "surrogate items" in the priority queue; the insertion of one surrogate item will take the place of a large number of insertions of conventional items.

In the next section we show that, under suitable assumptions about the joint probability distribution of the costs $c(x,y)$, the average execution time of the modified assignment algorithm is $O(|X|\cdot|Y|\cdot\log|Y|)$.

Consider a point in the execution of the assignment algorithm (Version 2) when a matching M has just been determined and the node potentials $\alpha(v)$ have been readjusted. Let $\alpha^* = \max_v \alpha(v)$. Recall that $\bar{c}(x,y) = c(x,y) + \alpha(x) - \alpha(y)$. Define $c^*(x,y)$ by $c^*(x,y) = c(x,y) + \alpha(x) - \alpha^*$. Then

$$c^*(x,y) \leq \bar{c}(x,y)$$
$$\gamma(x) + c^*(x,y) \leq \gamma(x) + \bar{c}(x,y) ,$$

with equality if y is a free destination. Also, $c^*(x,y) \leq c^*(x,y')$ if and only if $c(x,y) \leq c(x,y')$.

The modified assignment algorithm has a preprocessing phase in which, for each source x, a list LIST(x) is formed consisting of all ordered pairs $<x,y>$ such that y is a destination, sorted in increasing order of the cost $c(x,y)$.

During the execution of the modified assignment algorithm, information about the set EDGE is kept in a priority queue Q containing two types of items: <u>regular items</u> of the form $<<x,z>,\gamma(x)+\bar{c}(x,z)>$ and <u>special items</u> of the form $<<x,y>,\gamma(x)+c^*(x,y)>$. Such a special item is called a <u>surrogate</u> for the regular item $<<x,z>,\gamma(x)+\bar{c}(x,z)>$ if $c(x,y) \leq c(x,z)$ (or, equivalently, $\gamma(x)+c^*(x,y) \leq \gamma(x)+c^*(x,z) \leq \gamma(x)+\bar{c}(x,z)$). The relation of the queue Q to the set of edges EDGE is as follows: $\langle x,z\rangle \in \text{EDGE} \Leftrightarrow$ Q contains either the regular item $\langle\langle x,z\rangle,\gamma(x)+\bar{c}(x,z)\rangle$ or a surrogate for that item.

Using the structure Q we can implement all the operations on EDGE required by the assignment algorithm. These operations are:

(A) $\text{EDGE} \leftarrow \emptyset$

(B) Test if $\text{EDGE} = \emptyset$

(C) $\text{EDGE} \leftarrow \text{EDGE} \cup \text{OUT}(x)$

(D) choose $\langle x,y\rangle$ such that

$$\gamma(x)+\bar{c}(x,y) = \min_{\langle u,v\rangle \in \text{EDGE}} [\gamma(u)+\bar{c}(u,v)] .$$

The implementations of these operations using Q are as follows:

(A) $Q \leftarrow \emptyset$

(B) $Q = \emptyset$?

```
(C) begin
    <x,y> ← first element of LIST(x);
    insert into Q the special entry $\langle\langle x,y\rangle,\gamma(x)+c^*(x,y)\rangle$
    end
(D) Procedure SELECT
    begin
    do
       begin
       q ← the item of least value in Q;
       delete q from Q;
       <x,y> ← the edge to which q corresponds;
       if q is special
       then
```

```
                begin
                if <x,y> is not the last element of LIST(x)
                then
                   begin
                   <x,w> ← the successor of <x,y> in LIST(x);
                   insert into Q the special item <<x,y>,γ(x)+c*(x,y)>;
                   end
                insert into Q the regular item <<x,y>,γ(x)+c̄(x,y)>
                end
             end
          until a regular item <<x,y>,γ(x)+c̄(x,y)> has been selected such
             that {x,y} ∉ M
          end
```

The following version of the assignment algorithm incorporates all our modifications.

ASSIGNMENT ALGORITHM - VERSION 3

```
    begin
    for x∈X do LIST(x) ← an array containing the set of elements {<x,y>|
       y∈Y} in increasing order of c(x,y);
    M ← ∅;
    for v∈V do α(v) ← 0;
    while |M| < |X| do
       begin
       PATHSET ← ∅;
       Q ← ∅;
       R ← {free sources};
       for x∈R do
          begin
          γ(x) ← 0;
          <x,y> ← first element of LIST(x);
          insert into Q the special item <<x,y>,γ(x)+c*(x,y)>;
          end
       while R∩{free destinations} = ∅ do
```

```
        begin
        Execute Procedure SELECT
        if $y \notin R$ then
            begin
            PATHSET ← PATHSET $\cup \{<x,y>\}$
            $R \leftarrow R \cup \{y\}$
            $\gamma(y) \leftarrow \gamma(x)+\bar{c}(x,y)$
            if y is not free then
                begin
                $\{y,v\} \leftarrow$ the edge of M incident with y;
                PATHSET ← PATHSET $\cup \{<y,v>\}$;
                $R \leftarrow R \cup \{v\}$;
                $\gamma(v) \leftarrow \gamma(y)$;
                $<v,\ell> \leftarrow$ first element of LIST(v);
                insert into Q the special item $<<v,\ell>,\gamma(v)+c^*(v,\ell)>$
                end
            end
        end
        for $v \notin R$ do $\gamma(v) \leftarrow \gamma(y)$;
        for $v \in V$ do $\alpha(v) \leftarrow \alpha(v)+\gamma(v)$;
        Let $\tilde{P}$ be the unique directed path from a free source to y whose
            edges are all in PATHSET;
        Let P be the set of edges in G corresponding to directed edges in $\tilde{P}$;
        $M \leftarrow M \oplus P$
    end
end
```

5. Average Execution Time of the Assignment Algorithm (VERSION 3)

We make the following assumptions:

(i) the costs $c(x,y)$ are independent random variables;

(ii) for each fixed source x, the costs $c(x,y)$, for all $y \in Y$, are drawn independently from a common distribution.

The sorting operations involved in forming the list LIST(x) for each $x \in X$ require $O(|X|\cdot|Y|\cdot\log|Y|)$ steps.

Next we derive an upper bound on the expected time spent in operations on the priority queue Q. At any point in the execution of the algorithm call $\langle x,y\rangle$ a virgin edge if no entry previously selected from Q during the entire computation contains $\langle x,y\rangle$ in its edge field. In particular, $\langle x,y\rangle$ is a virgin edge whenever Y is a free destination. Call the selection of an item corresponding to a virgin edge an initial selection. Each time an initial selection occurs involving an edge directed out of x, all virgin edges $\langle x,y\rangle$ are equally likely to be involved. This is so because the costs $c(x,y)$ (and hence $c^*(x,y)$) associated with edges out of x are drawn independently from a common distribution, and none of the previous selections of edges from Q have given information about the relative costs of the virgin edges directed out of x.

For any k, phase k of the algorithm refers to the period when the matching M is of size k. We place an upper bound on the expected number of initial selections during phase k. The selection of a virgin edge directed into a free destination will trigger an augmentation and end the phase. For each x, there are at most $|Y|$ virgin edges directed out of x, and $|Y|-k$ of these are directed into free destinations. Hence the probability that a given initial selection will end the phase is $\geq \frac{|Y|-k}{|Y|}$, and the expected number of initial selections in phase k is $\leq \frac{|Y|}{|Y|-k}$.

Now we bound the expected number of selections of all kinds. If an edge ceases to be virgin in phase k, then the number of selections involving it is $\leq 2(|X|-k+1)$, since at most one special item and one regular item corresponding to that edge can be selected at any phase. Hence the expected total number of selections is bounded above by

$$2\sum_{k=0}^{|X|-1} \frac{|Y|}{|Y|-k}\cdot(|X|-k+1) = 2|Y|\cdot|X| + O(|Y|) .$$

Since each selection takes time $O(\log|Y|)$, the expected time for all selections is bounded above by $O(|X|\cdot|Y|\cdot\log|Y|)$.

The time for all operations not associated with preprocessing or selection from Q is bounded above by $O(|X|\cdot|Y|)$. Hence the expected execution time of the algorithm is $O(|X|\cdot|Y|\cdot\log|Y|)$.

References

[1] Aho, A.V., Hopcroft, J.E. and Ullman, J.D., The Design and Analysis of Computer Algorithms, Addison-Wesley (1974).

[2] Dantzig, G.B., "On the shortest route through a network," Manag. Sci. 6, 187-190 (1960).

[3] Dijkstra, E.W., "A note on two problems in connexion with graphs," Numer. Math. 1, 269-271 (1959).

[4] Edmonds, J. and Karp, R.M., "Theoretical improvements in algorithmic efficiency for network flow problems," J. ACM 19, 248-264 (1972).

[5] Ford, L. and Fulkerson, D.R., Flows in Networks, Princeton University Press (1962).

[6] Johnson, D.B., "Efficient algorithms for shortest paths in sparse networks," J. ACM 24 (1977).

[7] Knuth, D.E., The Art of Computer Programming, Vol. 1: Fundamental Algorithms, Addison-Wesley (1968).

[8] Lawler, E.L., Combinatorial Optimization: Networks and Matroids, Holt, Rinehart and Winston (1976).

[9] Tomizawa, N., "On some techniques useful for solution of transportation problems," Networks 1, 173-194 (1972).