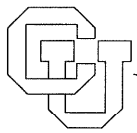
An Improved Method For Finding The K Smallest Cost Assignments in Order

Frank Fussenegger & Harold N. Gabow

CU-CS-124-78 September 1977



University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

AN IMPROVED METHOD FOR FINDING THE K SMALLEST COST ASSIGNMENTS IN ORDER

by

Frank Fussenegger
Martin Marietta Corporation
Data Systems
Denver, Colorado

Harold N. Gabow
Department of Computer Science
University of Colorado at Boulder
Boulder, Colorado

#CU-CS-124-77 September, 1977

Abstract

An assignment is a perfect matching on a bipartite graph. An algorithm is given that outputs the K smallest cost assignments in order of increasing cost. The time is $O(K \min(V^3, VE \log V))$ and the space is $O(E+K)$, where V and E are the number of vertices and edges. This compares favorably to a previous algorithm with run time $O(KV^4)$. The speed-up is achieved by using a shortest path calculation to generate one optimal assignment from another. Two special cases of the problem are also discussed: finding all assignments in order, and finding all minimum cost assignments. The latter is done in $O(\min(V^3, VE \log V) + ME)$ time and $O(E)$ space, where M is the number of minimum cost assignments. A previous algorithm for finding all perfect matchings is used. The algorithms extend to ranking maximum matchings. The closely related problems of finding the K^{th} smallest assignment and finding an assignment of given cost C are shown NP-hard.

1. Introduction

In the classic assignment problem, n men must be paired with n jobs, so the cost is minimum; here the cost of assignment is the sum of the costs of each man-job pair. This paper gives an algorithm for finding the K smallest assignments, in order of increasing cost (where K is some given number). The algorithm can be used to find an assignment satisfying certain constraints, with minimum cost. To do so, just generate assignments in order of increasing cost, until one satisfies the constraints. This approach can be attractive, if the constraints are so complex they cannot be incorporated directly into a minimum cost assignment algorithm.

The basic algorithm for K assignments, given by Murty [M], runs in time $O(KV^4)$. We improve this to $O(K \min(V^3, VE\log V))$, using space $O(E+K)$. Here V and E denote the number of vertices and edges. The speed-up is achieved by using one assignment to find the next.

We make further improvements for two special cases. The first is finding all assignments in order. Our algorithm is not necessarily faster than an alternate approach, based on an algorithm of Itai, Rodeh, and Tanimoto [IRT] for finding all perfect matchings. However, it does afford other advantages, such as a smaller guaranteed time delay between consecutive assignments. The second special case is finding all assignments of minimum cost. Here the time is $O(\min(V^3, VE\log V) + ME)$, and the space is $O(E)$, where M is the number of minimum cost assignments. This algorithm incorporates the algorithm of [IRT]. All three algorithms are shown to extend to ranking maximum matchings.

We show the problem of finding just the K^{th} smallest assignment is NP-hard. Similarly, the problem of finding an assignment of given cost C is NP-complete. These results are similar to those in [JK].

Section 2 gives background definitions and results. Section 3 presents an algorithm for generating one assignment from another. Section 4 applies this algorithm to ranking assignments, and also matchings. Section 5 gives the NP-hard results.

2. Preliminaries

This section gives some basic definitions and results, emphasizing matchings. We are given a bipartite graph G, with vertex sets X and Y. (Thus an edge joins vertices in X and Y.) V and E denote the number of vertices and edges, respectively. We are also given a cost function assigning a real-valued cost c(e) to each edge e.

A matching is a set of edges, such that each vertex is on at most one edge. The cost of matching M is $c(m) = \sum_{e \in M} c(e)$. In a perfect matching, or assignment, each vertex is on exactly one edge. Throughout this paper, we assume G has a perfect matching. Fig. 1 shows a perfect matching with cost 4; this is a minimum cost assignment. Matched edges are drawn wavy.

An alternating cycle is a sequence of edges (v_1,v_2), $(v_2,v_3)\ldots$, (v_{2n},v_{2n+1}), such that $v_1=v_{2n+1}$, all other vertices v_j are distinct, and $(v_{2i-1},v_{2i}) \in M$, $(v_{2i},v_{2i+1}) \notin M$. We usually write this cycle as $(v_1,v_2,\ldots,v_{2n})$. The cost of an alternating cycle C is

$$c(C) = \sum_{e \in C-M} c(e) - \sum_{e \in C \cap M} c(e).$$

If M is a perfect matching and $C_1,\ldots,C_n$ are vertex-disjoint alternating cycles, then $N = M \oplus C_1 \ldots \oplus C_n$ is another perfect matching. Further, $c(N) = c(M) + \sum_{i=1}^{n} c(C_i)$. Conversely, if M and N are perfect matchings, then N can be written as $M \oplus C_1 \ldots \oplus C_n$, where $C_1,\ldots,C_n$ are vertex disjoint alternating cycles. [L]. In Fig. 1, let

M be the matching shown. Another matching is N = {(1,6),(2,5),(3,4)}; an alternating cycle is C = (1,4,3,6,1). We have N = M $\oplus$ C, and c(N) = 6 = 4 + 2 = c(M) + c(C).

3. A Matching Problem

This section presents an algorithm for a matching problem that arises in ranking assignments. We are given a minimum cost perfect matching M, and an edge $e \in M$. We must find a perfect matching not containing e, with smallest cost possible. An approach is indicated by the following result.

Lemma 1: Let C be an alternating cycle containing e, with smallest cost possible. Then M $\oplus$ C is a perfect matching not containing e, with smallest cost possible.

Proof: A perfect matching not containing e can be written in the form $N = M \oplus C_1 \oplus C_2 \ldots \oplus C_n$, where for $1 \le i \le n$, C_i is an alternating cycle with non-negative cost, all cycles are vertex-disjoint, and $e \in C_1$. Thus $c(N) = c(M) + c(C_1) + \sum_{i=2}^{n} c(C_i) \ge c(M) + c(C) = c(M \oplus C)$, as desired. □

The Lemma is illustrated in Fig. 1, where for e = (3,6), the cycle C = (3,6,2,5,3), with cost 0.

Thus we seek a smallest cost alternating cycle C containing e. We treat this as a shortest path problem on a directed graph, as follows. For a vertex $v \in X \cup Y$, let M(v) be the vertex matched to v, i.e., $(v,m(v)) \in M$. The derived graph D is a directed graph with vertices X, and edges (x,x'), where (x,m(x')) is an unmatched edge in G. The derived graph for Fig. 1 is shown in Fig. 2.

There is a 1-1 correspondence between alternating cycles in G and directed cycles in D: a cycle in G,

$$C_G = (x_1, m(x_1), x_2, m(x_2), \ldots, x_n, m(x_n), x_1)$$

corresponds to a cycle in D,

$$C_D = (x_n, x_{n-1}, \ldots, x_1, x_n).$$

Further, let edge e = (z,m(z)). Then it is easy to see the desired cycle C (containing e) corresponds to a cycle in D containing z.

We wish to define edge lengths $\ell(x,x')$ in D, so C corresponds to a shortest cycle containing z. One possibility is to let $\ell(x,x')$ be $c(x,m(x')) - c(x',m(x'))$. In this case C corresponds to a shortest length cycle, since $c(C_G) = \ell(C_D)$. However some edges may have negative length. (For example in Fig. 2, edge (2,3) would have length -1.)

It is desirable that all edge lengths be non-negative. Then a shortest cycle containing z can be found in time $O(V^2)$. (The time is $O(V^3)$ if some lengths can be negative.) To achieve this, we make use of dual variables.

Originally the minimum cost matching M is found by the Hungarian method [L]. This method defines a dual variable u(v) for each vertex $v \in X \cup Y$, satisfying this condition:

(1) For any edge (x,y), $c(x,y) \geq u(x) + u(y)$. If $(x,y) \in M$, then $c(x,y) = u(x) + u(y)$.

To define edge lengths, set

(2) $\ell(x,x') = c(x,m(x')) - u(x) - u(m(x'))$.

These lengths are non-negative, by (1). Further, it is easy to check that for corresponding cycles C_G and C_D, $c(C_G) = \ell(C_D)$. So ℓ has the desired

properties. In Fig. 2, edge lengths are derived from the dual variables u of Table I.

We use a version of Dijkstra's algorithm [AHU] to find the shortest cycle containing z. As usually presented, this algorithm finds the shortest path from z to each vertex x, and computes the corresponding distance, w(x). For x = z, this path is empty, so w(z) = 0. It is easy to extend the algorithm to compute the shortest non-empty path to z. We do so. The algorithm still sets w(z) = 0, but also sets W to the length of the shortest cycle containing z.

Depending on the context, it may be necessary for the algorithm to compute dual variables for the new matching $M \oplus C$. There is an optional step for this. Note $M \oplus C$ is a minimum cost matching on $G - (z,m(z))$, so the new dual variables should satisfy (1) for all edges (x,y) except (z,m(z)).

Now we state the algorithm. We use a style similar to that of [AHU].

Algorithm EXCLUDE

Input: A bipartite graph G with edge costs c; a minimum cost perfect matching M, and dual variables u satisfying (1); also a matched edge (z,m(z)).

Output: A perfect matching N, not containing (z,m(z)), with cost as small as possible; also if desired, new dual variables v, satisfying (1). (If N does not exist, this is indicated.)

Method: 1. Form the derived graph D, with edge lengths given by (2).

2. Find the shortest path from z to each vertex x, of length w(x), and also, the shortes cycle containg z, of length W. (Use Dijkstra's algorithm as modified above.)

3. Let C by the alternating cycle corresponding to the cycle found in Step 2. If no such cycle exists, indicate this and stop. Otherwise set $N \leftarrow M \oplus C$.
4. (This step is optional.) Set $v(z) \leftarrow u(z) + W$. For each vertex $x \in X - \{z\}$, if $w(x) < W$, set $v(x) \leftarrow u(x) + W - w(x)$, $v(m(x)) \leftarrow u(m(x)) + w(x) - W$; otherwise if $w(x) \geq W$, set $v(x') \leftarrow u(x')$, $v(m(x')) \leftarrow u(m(x'))$.

For Fig. 1, with $(z,m(z)) = (1,4)$, the cycle $(1,3,2,1)$ is found in Fig. 2, so $C = (1,6,3,5,2,4,1)$. Matching $N = \{(1,6),(2,4),(3,5)\}$. Updated dual variables are shown in Table I.

Lemma 2: EXCLUDE satisfies its input-output specifications. It uses $O(\min(V^2, E \log V))$ time and $O(E)$ space.

Proof: Lemma 1 implies N is constructed correctly. Now we show the dual variables v computed in Step 4 satisfy (1) for all edges of $G = (z,m(z))$. Let (x,y) be such an edge. Let $y = m(x')$, so edges (x,y) and (x',y) correspond to edge (x,x') in D. By the definition of shortest paths,

$$w(x') \leq w(x) + \ell(x,x').$$

Substituting (2) gives

$$c(x,y) \geq u(x) - w(x) + u(y) + w(x').$$

Now we proceed by a case analysis. First suppose $x,x' \neq z$ and $w(x), w(x') < W$. Then by Step 4, the above inequality is equivalent to

$$c(x,y) \geq v(x) + v(y).$$

Thus (1) is true if $(x,y) \notin N$. If $(x,y) \in N - M$, then (x,x') is on

the shortest cycle; equality holds in the three above inequalities, so (1) is true. Last, if $(x,y) \in N \cap M$, (1) is true by inspection of step 4. The remaining cases (x or x' = z, or one or both of $w(x)$, $w(x') \geq$ $\geq w(z)$) follow similarly. Thus the dual variables v are correct.

For the time bound, note step 2 is $O(\min(V^2, E \log V))$ [AHU], while the remaining steps are $O(E)$. The space bound follows since D uses $O(E)$ space, and $O(V)$ further auxiliary storage is needed. □

4. Ranking Assignments

This section describes an algorithm for ranking assignments. The algorithm is similar to Murty's [M], and incorporates the algorithm of the previous section. Further improvements are discussed for two special cases.

The basic approach is branch and bound: the algorith partitions all assignments into disjoint sets, and keeps track of the minimum assignment in each set. Suppose the algorithm has output the first i-1 assignments, and has partitioned the remaining assignments into sets S. Then the set S associated with the smallest minimum cost assignment A is found. A is the i^{th} assignment, so it is output. Then the partition is updated: S is replaced by sets S_j, $r < j < t$, that partition S - {A}. Associated with each S_j is its minimum assignment. Now this procedure is repeated to output the $i + 1^{st}$ and subsequent assignments.

A set S is defined by requiring that an assignment in it includes certain edges and excludes others. More precisely, let $I(e_1,\ldots,e_r)$ denote the set of all assignments including each of the edges $e_1,\ldots,e_r$; let $X(f_1,\ldots,f_s)$ denote the set of all assignments excluding each of the edges $f_1,\ldots,f_s$. Then a set S in the partition has the form

$$S = I(e_1,\ldots,e_r) \cap X(f_1,\ldots,f_s),$$

where edges $f_1,\ldots,f_s$ are all incident to some vertex v. Thus $r,s < V$. Associated with S is the minimum cost assignment $A \in S$.

The set S-{A} is partitioned into sets S_j, using the edges in A-$\{e_1,\ldots,e_r\}$. (Note $e_1,\ldots,e_r$ are the edges in the above equation.) We denote the edges used as $e_{r+1},\ldots,e_t$, where $t = V/2$, and edge e_{r+1} is incident to v. The new sets S_j, $r < j < t$, are

$$S_{r+1} = I(e_1,\ldots,e_r) \cap X(f_1,\ldots,f_s,e_{r+1}),$$

$$S_j = I(e_1,\ldots,e_{j-1}) \cap X(e_j),\ r+1 < j < t.$$

(Note for $r+1 < j < t$, e_{r+1} as an included edge, so it is unnecessary to exclude $f_1,\ldots,f_s$ explicitly, since all these edges are incident to v.) These sets are disjoint, since if $i < k$, S_i excludes e_i while S_k includes it. Also, $S - \bigcup_{j=r+1}^{t-1} S_j = I(e_1,\ldots,e_t) \cap X(f_1,\ldots,f_s) = \{A\}$, so these sets partition S-{A}.

For example, in Table II, set 1 is the set of all assignments in Fig. 1. After the minimum assignment is output, it is partitioned into sets 2 and 3.

Now we state the algorithm.

Algorithm RANK

Input: A bipartite graph G, with edge costs c, and a positive integer K.

Output: The K smallest cost assignments, in order of increasing cost. If G has fewer than K assignments, all assignments are output, in order.

Method: A partition P is maintained, as described above. A set $S = I(e_1,\ldots,e_r) \cap X(f_1,\ldots,f_s)$ is associated with the following information: a list of included edges e_j; a list of excluded edges f_j; the minimum cost assignment $A \in S$; its cost $c(A)$; the dual variables u. This information may be explicitly stored, or computed (see Lemma 5 below).

1. Find a minimum cost assignment A_1. (Use the Hungarian method.)
2. Initialize P to contain one set, the set of all assignments, with minimum assignment A_1, cost $c(A_1)$, and dual variables from step 1. Set $i \leftarrow 1$.
3. If P is empty, stop. Otherwise, remove a set $S \in P$, with minimum assignment A, such that $c(A)$ is smallest among all sets in P.
4. Let $S = I(e_1,\ldots,e_r) \cap X(f_1,\ldots,f_s)$, with minimum assignment A, and dual variables u. Let $A = \{e_k \mid 1 \le k \le t\}$, where edge $e_k = (x_k,y_k)$, and $e_{r+1},f_1,\ldots,f_s$ are all incident to v.
5. Output A as the i^{th} assignment. If $i = K$, stop.
6. Form the graph $H = G - \{x_k,y_k \mid 1 \le k \le r\} - \{f_k \mid 1 \le k \le s\}$, with perfect matching $A - \{e_k \mid 1 \le k \le r\}$. Set $j \leftarrow r+1$.
7. Let S_j be the set $I(e_1,\ldots,e_{j-1}) \cap X(f_1,\ldots,f_s,e_j)$. (Note if $j > r+1$, excluding $f_j,\ldots,f_s$ is redundant.) Find a minimum assignment $B \in S_j$. (Use EXCLUDE on graph H, with e_j as the edge to exclude. Form B by by adding edges $e_1,\ldots,e_{j-1}$ to the assignment found). If no B exists, go to 9.
8. Add the set S_j, with minimum assignment B, cost $c(B)$, and dual variables from step 7, to P.
9. Delete vertices x_j,y_j, from H. Set $j \leftarrow j+1$. If $j < t$, go to 7. Otherwise set $i \leftarrow i+1$ and go to 3.

Table II illustrates the algorithm. As already noted, set 1 is partitioned into sets 2 and 3; the former is partitioned into set 4.

Lemma 3: RANK satisfies its input-output specifications.

Proof: First we check EXCLUDE works correctly in step 7. An assignment in graph H corresponds to an assingment in S_j, by adding edges $e_1,\ldots,e_{j-1}$. This correspondence is 1-1, onto, and preserves rank by cost. So EXCLUDE performs as desired.

The partition P is maintained according to the remarks preceding the algorithm. Thus the RANK works correctly. □

Lemma 4: RANK requires $O(K \min(V^3, VE \log V))$ time.

Proof: In step 1, the Hungarian method uses $O(\min(V^3, VE \log V))$ time. (An $O(V^3)$ bound is shown in [L]; an $O(VE \log V)$ bound is obtained by using a priority queue, as in [AHU,p.220,ex.21].)

Next we discuss the time for operation on P. These include initialization (step 2), finding and removing the smallest set (step 3), and adding a set (step 8). We implement P as a priority queue [AHU]. If P contains k sets, each operation can be done in $O(\log k)$ time.

At any point in the algorithm, P need not contain more than K elements, since only the K smallest assignments are of interest. So in step 8, if adding a set to P causes it to contain $K+1$ sets, the maximum cost set is removed. Finding and removing the maximum set requires $O(\log K)$ time. Thus the total time for the $O(K)$ operations on P is $O(K \log K)$.

Note K is at most V!, the greatest number of assignments possible. So the time for operations on P is $O(K\ V \log V)$.

The rest of the time is dominated by steps 6-7. Step 6 is done once in $O(E)$ time. Step 7 is done in $O(\min(V^2, E \log V))$ time (Lemma 2).

Since steps 6-7 are repeated at most V times for every assignment output, the total time is $O(K \min (V^3, V\ E \log V))$. □

Now we examine the space requirement. For a set $S \in P$, O(V) space is needed to represent S explicitly. Thus P may require O(KV) space. We reduce this, without increasing the aymptotic run time, as follows.

Define the partition history tree T to contain a node corresponding to every set ever in P. The root of T corresponds to the initial set (see step 2). The sons of the node corresponding to S correspond to the sets $S_j, r < j \leq t$, that partition S - {A} (see step 8). We say S_j is a younger brother of S_k if $j < k$. The node for S_j is labelled with the edge e_j that S_j excludes (see step 7). (The root of T is unlabelled.) Fig. 3 gives the partition history tree for the example.

From T, we can construct the lists of included and excluded edges for a set S. Let P be the path in T from the node for S to the root. Let S' correspond to the first node in P that is not a youngest son. The excluded edges for S are the labels of nodes in the path from S to S'. The included edges are the labels of younger brothers of nodes in the path from S' to the root.

The partition history tree gives this result.

Lemma 5: RANK uses O(E+K) space.

Proof: Graphs G and H use O(E) space. The remaining space is dominated by sets in P. For each set, we store the cost of a minimum assignment. We also store the tree T, using father, son, and younger brother pointers. Now we must check that T can be maintained in O(K) space, and further, the information for a set S can be computed without increasing the time bound of Lemma 4.

First we show how T is maintained to contain a node for each set that may include one of the K smallest assignments, but no others. In step 8, when S_j is added to P, a corresponding node is added to T. Note if S_j is empty, step 8 is skipped. In this case it is unnecessary to add a node for S_j to T. For $S_j = \emptyset$ implies e_j is included in any older brother of S_j (and in any older brother's descendents). Thus e_j need not be put on the included list for any of these nodes.

Also in step 8, when a set S is removed from P (as indicated in Lemma 4) the corresponding node is removed from T. (Note this node is a leaf, since it corresponds to the maximum set.)

Maintaining T in this way, no more than K nodes are needed. Thus the space is O(K).

Next we check the time. In step 4, the included and excluded edges for S are computed from T, as described above. In T, a path from a node S to the root contains at most E nodes, since no edge label is repeated. Also, there are less than V included edges for S. These facts imply the time to construct the included and excluded edge lists is O(E). The minimum perfect matching A is computed, using the Hungarian method on graph H of step 6. This also gives the dual variables. The time is $O(\min(V^3, V\ E \log V))$. Thus the additional time for computing the information for S is $O(K \min(V^3, V\ E \log V))$. Hence the time bound is unchanged. □

We summarize Lemmas 3-5.

Theorem 1: RANK outputs the K smallest assignments, in order of increasing cost. The time is $O(K \min(V^3, V\ E \log V))$, and the space is $O(E + K)$. □

Now we consider two special cases of our problem. The first is ranking all assignments of G, rather than K. We describe an algorithm, RANK-ALL, derived from RANK by making three main changes. Let N be the number of assignments. The first change is simple: set K to a value of N or more, e.g., K = V!, so all assignments are generated.

The purpose of the second change is to avoid fruitless searches. In RANK, some sets S_j are empty, i.e., in step 7, graph $H-e_j$ contains no assignment. The time spent by EXCLUDE searching for such an assignment is wasted. RANK-ALL avoids this, using an approach first proposed in [IRT]: Given the graph H, we search for an alternating cycle. If one is found, any matched edge in it is chosen as e_j, the next edge to exclude; then EXCLUDE is executed, to find the minimum cycle containing e_j. To find an alternating cycle in H, we use the derived graph.

Thus, for the second change we insert the following steps after the current step 6 of RANK:

6.1 For the derived graph D of H.

6.2 Find a cycle in D (use depth-first search [AHU, pp. 189-195 or p. 218, ex. 7(c)] or topological sort [K, pp. 258-65, and p. 268, ex. 23].). If none exists, set $i \leftarrow i+1$ and go to 3. Otherwise, let x be a vertex in the cycle; set $e_j \leftarrow (x,m(x))$.

Note the time to execute steps 6.1-2 is O(E) (see references). Steps 6.1-2 become part of the loop in steps 7-9; thus in step 9, "go to 7" becomes "go to 6.1".

The third and final change trades space for time. In Lemma 5, we save space for a set S by using the Hungarian method to recompute assignments and dual variables. For the purposes of RANK-ALL, this recomputation takes too much time. So the third change is to store

all information associated with a set in the partition. This increases the space for the partition from $O(N)$ to $O(NV)$.

We summarize the characteristics of the algorithm RANK-ALL defined by these three changes.

Theorem 2: RANK-ALL outputs all assignments in order of increasing cost, in time $O((V+N) \min (V^2, E \log V))$ and space $O(E+NV)$.

Proof: It is easy to check the changes preserve correctess of the algorithm. For the time, we proceed as in Lemma 4, except we account for the time in steps 6.1-2 and 7 in a slightly different way, as follows.

In the course of the algorithm, each assignment is found once as B in step 7. The time for this execution of EXCLUDE, and for the corresponding execution of steps 6.1-2, $(O(\min(V^2, E \log V)))$ is charged to assignment B. Also, when each assignment is A in step 3, steps 6.1-2 indicate no edge e_j exists exactly once. The time for this, $O(E)$, is charged to A. This accounts for all the time in steps 6.1-2 and 7. The total charge to an assignment is $O(\min(V^2, E \log V))$, except A_1 is charged $O(\min(V^3, E \log V))$ in step 1. The time bound follows.

The space bound is obvious. □

An alternate approach to the problem of ranking all assignments is the following: First, generate all assignments of G_1, disregarding cost; then sort the assignments in order of increasing cost. Itai, Rodeh and Tanimoto [IRT] show the first step can be done in $O(V^{1/2}E + NE)$ time (also, see Corollary 1 below). Then the sort requires $O(N \log N) = O(N V \log V)$ additional time. So the total time is $O(V^{1/2}E + N(E + V \log V))$. The space is $O(E+NV)$, since each assignment must be stored for the sort. (Note if this were not necessary, the algorithm of [IRT] uses only $O(E)$ space.)

In general, the alternate approach uses less time and the same space as RANK-ALL. However, two features may make RANK-ALL preferable:

1. For the common special case of a dense graph, where $E = \Omega(V^2)$ and $N >> V$, RANK-ALL is as fast as the alternate approach. Also for sparse graphs ($E = O(V)$), the set-up time for RANK-ALL is slightly more ($O(\min(V^3, V E \log V))$ versus $O(V^{1/2}E)$), but then assignments are generated at the same rate.

2. In RANK-ALL, the maximum time delay between outputting two consecutive assignments is $O(\min(V^3, V E \log V))$ (the time for the loop in steps 6.1-9.). In the first approach, $O(V^{1/2}E + NE)$ time elapses before the first assignment is output.

The second special case we consider is ranking all assignments that have the minimum cost $c(A_1)$. To analyze this problem, let $u(v)$ be the dual variables corresponding to assignment A_1. Define a graph H to contain all edges (x,y) of G where $c(x,y) = u(x) + u(y)$.

Lemma 6: Every minimum cost assignment in G is a perfect matching in H, and vice versa.

Proof: A minimum assignment B in G can be written as $B = A_1 \oplus C_1 \oplus \oplus \ldots \oplus C_n$, where each C_i is an alternating cycle of cost 0, and all cycles are vertex-disjoint. An edge $(x,y) \in C_i$ satisfies $c(x,y) = u(x) + u(y)$ (otherwise, it is easy to see C_i has positive cost). Thus each C_i is a cycle in H, and B is a perfect matching in H.

The argument on the reserve direction is similar. □

Thus the problem is reduced to finding all perfect matchings on a graph H. This can be done by RANK-ALL, by just ignoring costs. In this case, step 7 is unnecessary and can be deleted; the assignment B

can be taken as $A \oplus C$, where C is the alternating cycle found in step 6.2. This modfication reduces the time to find the next assignment, from $O(\min(V^2, E \log V))$ to $O(E)$ (see Theorem 2).

The space can also be reduced, to $O(E)$. The idea is to take advantage of the fact that assignments can be output in any order. An efficient order is given by traversing the partition history tree T depth-first [AHU,p.176-179], outputting an assignment at each node of T. Then instead of maintaining the complete partition P, the algorithm need only keep track of the edges included and excluded at the current node. A stack can be conviently used for this. The size of the stack never exceeds E. Hence the space is $O(E)$.

Filling in the details of this approach gives an algorithm equivalent to that of Itai, Rodeh and Tanimoto [IRT]. We refer the reader to their paper.

To summarize, let M be the number of perfect matchings of G.

Theorem 3: All minimum cost perfect matchings of G can be output in time $O(\min(V^3, V\ E \log V\ ) + ME)$, and space $O(E)$. □

Corollary 1: In a bipartite graph without costs, all perfect matchings can be output in time $O(V^{1/2}E + ME)$ and space $O(E)$.

Proof: Using the algorithm of Hopcraft and Karp [HK], the first perfect matching can be found in time $O(V^{1/2}E)$. □

We close this section by extending the algorithms to rank maximum matchings on an arbitrary bipartite graph. First we give some definitions. A matching covers a vertex v if v is on a matched edge.

A maximum matching covers the greatest number of vertices possible. An even alternating path P is a sequence of edges (v_1,v_2), (v_2,v_3), $\ldots,(v_{2n},v_{2n+1})$, where all vertices v_j are distinct, v_1 is not covered, and $(v_{2i-1},v_{2i}) \notin M$, $(v_{2i},v_{2i+1}) \in M$. The cost of an even alternating path P is $c(P) = \sum_{e \in P-M} c(e) - \sum_{e \in P \cap M} c(e)$.

The following subproblem arises in ranking matchings: We are given a maximum matching M covering vertices $v_1,\ldots,v_p$, with smallest cost possible, and a vertex z not covered by M. We must find a maximum matching covering $z,v_1,\ldots,v_p$, with smallest cost possible.

Lemma 7: Let P be an even alternating path from z to a vertex different from $v_1,\ldots,v_p$, with smallest cost possible. Then $M \oplus P$ is a maximum matching covering z, v_1, ..., v_p, with smallest cost possible.

Proof: Similar to Lemma 1. □

Path P can be found using the derived graph D. An even alternating path P_G from z to z' in G corresponds to a directed path P_D from z to z' in D. Also, when the Hungarian method is used to find a maximum matching of smallest cost, the dual variables satisfy (1) (in addition to other linear programming duality conditions). So non-negative edge lengths can be defined by (2). Thus we have $c(P_G) = \ell(P_D) + u(z) - u(z')$. It is easy to modify EXCLUDE so it finds the desired path P, and forms the matching $M \oplus P$, and the new dual variables.

The partition is defined as in RANK, but we also require matchings to cover certain vertices and not cover others. More precisely, let $C(v_1,\ldots,v_p)$ denote the set of all maximum matchings covering vertices $v_1,..,v_p$; let $U(w_1,\ldots,w_q)$ denote the set of all maximum matchings not covering any of the vertices $w_1,\ldots,w_q$. A set S in the partition has the form

$$S = I(e_1,\ldots,e_r) \cap X(f_1,\ldots,f_s) \cap C(v_1,..,v_p) \cap U(w_1,\ldots,w_q).$$

As in Section 4, edges $f_1,..,f_s$ are incident to some vertex v, and S is associated with the minimum cost matching $M \in S$.

When M is output, the set S - {M} of remaining matchings is partitioned, as follows. Let $w_{q+1},\ldots,w_a$ be the uncovered vertices of $X-\{w_1,\ldots,w_q\}$; let $e_{r+1},..,e_t$ be the edges in M - $\{e_1,\ldots,e_r\}$, where e_{r+1} is incident to v. Then the new sets are $T_i, q<i\leq a$, and $S_j, r<j<t$, where

$$T_i = I(e_1,\ldots,e_r) \cap X(f_1,\ldots,f_s) \cap C(v_1,..,v_p,w_j) \cap U(w_1,..,w_{j-1}), q<i\leq a,$$
$$S_{r+1} = I(e_1,..,e_r) \cap X(f_1,..,f_s,e_{r+1}) \cap U(w_1,..,w_a),$$
$$S_j = I(e_1,..,e_{j-1}) \cap X(e_j) \cap U(w_1,..,w_a), r+1<j<t.$$

The smallest matching in S_j can be found as in RANK, using the original version of EXCLUDE; the smallest matching in T_i can be found using the modified version of EXCLUDE.

We obtain the following result:

Theorem 4: The K smallest maximum matchings can be output, in order of increasing cost, in the same time and space bounds as Theorems 1-3.

Proof: The details of the algorithms, beyond those given above, are left as an exercise. □

5. Relation to NP-hard problems

This Section relates ranking assignments to some intractable problems. The results are similar to those of Johnson and Kashden [JK] for ranking spanning trees and circuits.

We begin by defining two problems that can be solved by RANK:

COST C MATCHING: Given a bipartite graph, integer edge costs, and an integer C, is there a perfect matching with cost exactly C?

K^{th}LARGEST MATCHING: Given a bipartite graph, integer edge costs, and integers K,C, are there at least K perfect matchings with cost C or less?

To solve COST C MATCHING, apply RANK, until a matching with cost at least C is found. Unfortunately the time is exponential, since each of (V/2)! assignments may be generated. For K^{th} LARGEST MATCHING, RANK can be used to solve the problem, but again the time is exponential, since it is proportional to K rather than log K.

However it is easy to show a polynomial algorithm for either problem is unlikely, since it would imply P=NP. To do this, recall these two problems:

SUBSET SUM: Given integers a_i, $1 \le i \le n$, and S, is there a subset of the integers a_i that sums to exactly S?

K^{th}LARGEST SUBSET: Given integers a_i, $1 \le i \le n$, and K,S, are there at least K different subsets of the integers a_i, each summing to at most S?

SUBSET SUM is NP-complete [AHU]; K^{th} LARGEST SUBSET is NP-hard [GJ,JK].

From a set of integers a_i, it is easy to construct a graph, so that subsets of integers a_i correspond 1-1 to perfect matchings of the graph, and corresponding integer sums and matching costs are identical.

To do this, just represent each integer a_i by a cycle of length 4, where one edge has cost a_i and the others have cost 0. (If desired, the graph can be made connected, by choosing a vertex from each cycle, and joining these vertices by a path.) The graph can clearly be constructed in polynomial time. This polynomial transformation gives following result.

Theorem 4: COST C MATCHING is NP-complete, and K^{th} LARGEST MATCHING is NP-hard. □

References

[AHU] A. Aho, J. Hopcroft and J. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Mass., 1974.

[GJ] M.R. Garey and D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, to be published by W.H. Freeman & Co., San Francisco, Calif.

[HK] J. Hopcroft and R. Karp, "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs." SIAM J. on Comp. 2, 1973, pp. 225-231.

[IRT] A Itai, M. Rodeh and S. L. Tanimoto, "Some matching problems for bipartite graphs", J. ACM, to appear.

[JK] D.B. Johnson and S. D. Kashdan, "Lower bounds for selection in X + Y and other multisets", SIAM J. On Computing, to appear.

[K] D.E. Knuth, The Art of Computer Programming Vol. 1, Ed. 2, Pr. 2, Addison-Wesley, Reading, Mass., 1975.

[L] E. L. Lawler, Combinatorial Optimization: Networks and Matroids, Holt, Rinehart and Winston, New York, 1976.

[M] K. G. Murty, "An algorithm for ranking all assignments in order of increasing cost", Op. Res. 163, 1968, pp. 682-687.

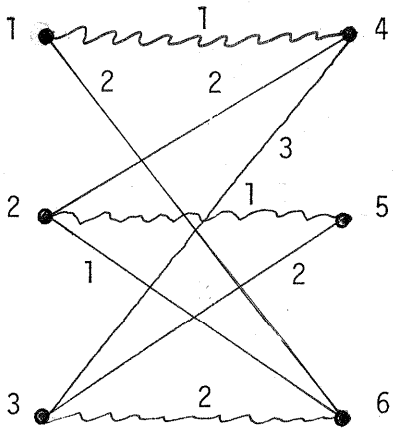


Fig. 1 Matched graph

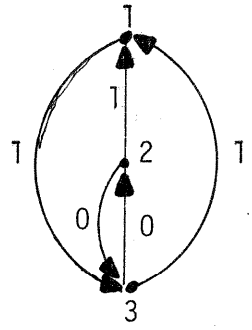


Fig. 2 Derived graph

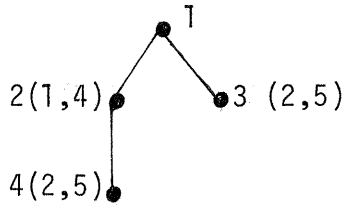


Fig. 3 Partition
History Tree.

i	1	2	3	4	5	6
u	1	1	2	0	0	0
v	3	2	3	0	-1	-1

Table I. Dual variables

set	c(A)	minimum assignment A	I edges	X edges
1	4	(1,4),(2,5),(3,6)		
2	6	(1,6),(2,5),(3,4)		(1,4)
3	4	(1,4),(2,6),(3,5)	(1,4)	(2,5)
4	6	(1,6),(2,4),(3,5)		(1,4),(2,5)

Table II. Partition sets