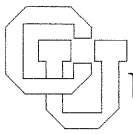
Finding the Best Solutions: The Cost $\leq$ C Problem and Finding Matchings

Hal Gabow

CU-CS-139-78 October 1978



University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

FINDING THE BEST SOLUTIONS: THE COST ≤ C PROBLEM AND FINDING MATCHINGS

HAROLD N. GABOW
Department of Computer Science
University of Colorado at Boulder
Boulder, Colorado 80309

ABSTRACT

For a given combinatorial optimization problem, the corresponding COST ≤ C problem is to find all solutions of cost at most C. This is closely related to the often-studied K BEST problem, which is to find the K smallest solutions in order. Typical COST ≤ C algorithms for graph problems (e.g., paths, matchings, spanning trees) use less space (O(m+n) instead of O(K+m+n)) than K BEST, and are slightly faster.

As an illustration, several COST ≤ C (and K BEST) algorithms for matchings on bipartite graphs are presented. The times are $O(K \min(n^3, mn \log n))$, improving previous bounds. The approach is to reduce the matching problems to path problems on directed graphs.

Lastly, a justification is given for the relatively high time bound on COST ≤ C for paths: A single-source problem arising in its solution is shown to be equivalent to a multi-source problem.

1. INTRODUCTION

When a real-world problem is formulated as a graph optimization problem, various aspects may be lost. These include unstated constraints or desiderata, approximate numerical parameters, and others. One way to deal with this is to find a number of solutions of small cost, rather than just one optimum solution. The best of these solutions can then be selected.

This approach is usually formulated as the K BEST problem: find the K smallest solutions, in order of increasing cost. Efficient algorithms have been given for K BEST in the cases of paths [L72,Y], cycles [JK], matchings [M], and spanning trees [CFM,G,KIM]. These algorithms use time $O(K \cdot P(m,n))$ and space O(K+m+n); here n and m are the number of vertices and edges, and P is a slow-growing (e.g., low order polynomial) function.

An alternate approach is the COST ≤ C problem: find all solutions of cost at most C. This problem is easier than K BEST, since the output need not be sorted. Real-world situations may make sorted output unnecessary or of limited significance. The K BEST algorithms mentioned above can be adapted to COST ≤ C. The space bounds reduce to O(m+n). The constant of proportionality in the time bounds decrease. So for reasons of applicability and efficiency, COST ≤ C may be a better problem formulation than K BEST.

Section 2 gives general remarks on COST ≤ C. These include #P-completeness results [V77a-b], and the space savings mentioned above. Section 3 studies a specific example, COST ≤ C for matchings on bipartite graphs. Algorithms are obtained by reducing the matching problems to path problems. A K BEST matching algorithm improves previous results. Section 4 studies the complexity of COST ≤ C for paths. A possible justification for the relatively high time bound for paths is given, by showing a multi-source path problem is involved.

2. THE COST ≤ C PROBLEM: GENERAL REMARKS

This Section discusses these general complexity properties of COST ≤ C: Counting COST ≤ C solutions is usually #P-complete; a COST ≤ C algorithm

usually requires less space than the corresponding K BEST algorithm.

We start with some terminology. Let S-subgraph denote a fixed family of subgraphs of graphs, e.g., cycle, spanning tree, etc. In a graph G, let c(e) be a real-valued cost assigned to each edge e; the cost of a subgraph H is $c(H) = \sum_{e \in H} c(e)$. The COST ≤ C(S) problem is, for a given graph G with cost function c and cost bound C, find all S-subgraphs of cost at most C. The #COST ≤ C(S) problem is to find only the number of such subgraphs. This number is denoted by K, which is used as a parameter in time estimates for COST ≤ C.

We first study #COST ≤ C. Define these related problems: #(S) is the problem of counting all S-subgraphs of a given graph. #(S) problems are studied in [V77b]. #COST = C(S) is defined analogous to #COST ≤ C.

Let α denote reducibility by oracle Turing machines in the sense of Valiant [V77b]. Then the three counting problems are related as follows:

Lemma 2.1: For a fixed S, #(S) α #COST = C(S) α #COST ≤ C(S).

Proof: In the first reduction the cost function c can be restricted to positive integers: Let all edge costs be 1, and compute #COST = i, for i = 0, ..., m.

In the second reduction a general real-valued cost function can be allowed, by computing the number of S-subgraphs with cost ≤ C and ≥ C. □

Let #SUBSET SUM be the problem of, given numbers $s_1,\ldots,s_n$ and a desired sum S, find the number of subsets of numbers summing exactly to S. This problem is #P-complete (The reduction in [MY,pp.239-243] leads to a simple proof). This is the basis of our hardness result.

Lemma 2.2: #COST ≤ C(S) is #P-hard if S-subgraphs have these two properties:

(a) For any graphs G_i, i = 1,2, we can find (in polynomial time) a graph G containing the G_i as edge-disjoint subgraphs, such that the S-subgraphs of G are precisely the unions of S-subgraphs of G_i;

(b) Some graph G_0 has a positive even number of S-subgraphs, with some edge e_o in exactly half of them.

Proof: By Lemma 2.1, it suffices to show #COST = C(S) is #P-hard. Use (a) to construct a graph containing n copies of G_0. Let all edge costs be 0, except the i^{th} copy of e_0 has cost s_i. Take C = S. Then if G_0 has 2k S-subgraphs, the answer to #SUBSET SUM is the answer to #COST = C divided by k^n. □

This reduction uses cost 0 edges. However, by scaling up the numbers s_i, we see COST ≤C is #P-hard when costs are all positive integers.

#(S), and hence #COST ≤ C(S), is known to be #P-complete when an S-subgraph is a maximum matching or a path joining designated start and end vertices [V77b]. However the Lemma gives interesting results in these cases:

Theorem 2.1: #COST ≤ C(S) is #P-complete when an S-subgraph is a maximal matching or spanning tree.

Proof: For a maximal matching, G_0 is a cycle of length 4. For a spanning tree, G_0 is a cycle of length 4 plus a diagonal; the diagonal is in 4 of the 8 spanning trees. □

The status of #(S) is unknown for maximal matchings [V77b]. For spanning trees #(S) can be solved in polynomial time [E].

Now we turn our attention to COST ≤ C algorithms. In spite of the previous results, these algorithms give feasible computations if K is not huge. The usual approach is backtracking: edges are systematically included and excluded until all desired S-subgraphs are found.

As an example, suppose in a directed graph, an S-subgraph is an s-t path, i.e., a simple path from a designated start vertex s to an end vertex t. K BEST (s-t path) has been solved in $O(K \min(n^3, mn \log n))$ time and $O(K+m+n)$ space (see[L72,L76,Y] for time, [G] for space). This approach gives a COST ≤ C algorithm using the same time but less space, $O(m+n)$.

The algorithm is given below. P is a global data structure for a path. The recursive procedure EXTEND (v) is called with P as the last s-t path found so far, and v on P. Let P = (s,...,v,w,...,t). Then EXTEND (v) outputs all paths (including P) of cost ≤ C that begin with (s,...,v). This is done by first (recursively) outputting the paths that include edge (v,w), and then the paths that exclude it. EXTEND (v) returns with P set to (s,...,v).

```
procedure PATHS; comment finds all s-t paths of cost ≤ C, where vertices
s,t and cost C have already been specified; begin
procedure EXTEND (v);
begin let EXCLUDE be a list of edges local to EXTEND;
1.  if v = t then begin output (P); return end;
2.  let w be the vertex following v in P;
3.  EXTEND (w); comment now P ends at w;
4.  remove w from P; let EXCLUDE consist of the edge (v,w);
5.  while there is a v-t path avoiding vertices in P-v and edges in EXCLUDE,
        with cost ≤ C-c(P) do
        begin
6.      let (v=w_0,w_1,...,w_k=t) be such a path;
7.      for i ← 1 to k do add w_i to P;
8.      EXTEND (w_1);
9.      remove w_1 from P; add (v,w_1) to EXCLUDE;
10. end end EXTEND;
11. let d(v) be the length of a shortest s-v path, for each vertex v;
12. if d(t) > C then return comment no paths of desired cost exist;
13. if G has negative edges then
14.     begin for each edge (v,w) do c(v,w) ← c(v,w) + d(v) - d(w);
15.         C ← C-d(t);
        end;
16. let P be a shortest s-t path;
17. EXTEND (s);
    end PATHS;
```

Lemma 2.3 PATHS solves COST ≤ C(s-t path) in a directed graph with no negative cycles, in time $O(K \min(n^3, m n \log n))$ and space $O(m+n)$.

Proof: The time is dominated by lines 5-6: finding an acceptable v-t path. This is a single-source shortest path problem with non-negative edge costs (by lines 13-15). So Dijkstra's algorithm does lines 5-6 once in time $O(\min(n^2, m \log n))$[AHU]. These lines are done ≤ n times per output path. □

This result illustrates the general space savings for COST ≤ C.

Theorem 2.2: COST ≤ C (S) can be solved in $O(m+n)$ space when an S-subgraph is an s-t path, a cycle, a maximum matching, or a spanning tree in an undirected or directed graph.

Proof: As with s-t paths, other K BEST algorithms [JK,M,G,KIM,CFM] can be modified for backtracking. (Section 3 gives more details for cycles and matchings). □

Regarding time, corresponding COST ≤ C and K BEST algorithms have the same asymptotic time bound. But a simple comparison of code shows COST ≤ C saves time, by replacing a priority queue with a stack.

3. FINDING BEST MATCHINGS

This Section presents algorithms for several COST ≤ C problems for matchings on bipartite graphs. Two previous algorithms are improved or extended. The approach is to reduce the matching problems to path problems on directed graphs.

We first recall some fundamental facts. A matching M is a set of disjoint edges. A vertex not on an edge of M is exposed. A path is alternating if its edges are alternately in M and $\bar{M}$, and further, an end edge is in $\bar{M}$ only if the corresponding end vertex is exposed. (This restriction is not standard.) An alternating path is augmenting if both end edges are in G -M, and de-augmenting if these edges are in M. The cost of an alternating path P with respect to M is $c(P,M) = c(P \cap \bar{M}) - c(P \cap M)$.

The following "direct sum principle" is well-known [L76]: Fix a match- M of cardinality k; let N be another cardinality k-matching. Then $N \oplus M$ consists of disjoint alternating cycles, even length alternating paths, and pairs of disjoint augmenting and de-augmenting paths ("Alternating" means alternating with respect to M). Further $c(N \oplus M, M) = c(N) - c(M)$.

The direct sum principle gives a 1-1 correspondence between cardinality k matchings and sets of alternating paths described above. Note when k gives a perfect matching, the sets contain only alternating cycles; when k gives a maximum matching, the sets contain only alternating cycles and even paths.

Now we recall a well-known correspondence between bipartite graphs G with a distinguished perfect matching M and directed graphs D. Specifically, let G have vertices u_i, v_i, i=1, ..., n, and let $M = \{(u_i,v_i) \mid i=1,\ldots,n\}$. The corresponding directed graph D has vertices w_i, i=1, ..., n, and edges (w_i,w_j), where (u_i,v_j) is an edge of G-M.

Alternating cycles in G correspond to directed cycles in D. If edge costs satisfy $c(w_i,w_j) = c(u_i,v_j) - c(u_j,v_j)$, then the costs of an alternating cycle (with respect to M) and the corresponding directed cycle are equal.

Thus by the direct sum principle, a perfect matching N on G corresponds

to a set of disjoint cycles S on D, and c(N) - c(M) = c(S). This implies the following result.

Lemma 3.1: Let F(n,m) be the time to find a minimum cost perfect matching on a bipartite graph, M(n,m,K) be the time for COST ≤ C (perfect matching) on a bipartite graph, and S(n,m,K) be the time for COST ≤ C (set of disjoint cycles) on a directed graph with no negative cycles. Then

$$M(n,m,K) = O(F(n,m) + Kn + S(n,m,K)),$$

$$S(n,m,K) = O(Kn + M(n,m,K)).$$

The space for both COST ≤ C problems is equal.

Proof: Given G, find a minimum cost perfect matching M. The corresponding graph D has no negative cycles, by the minimality of M. The equation for M follows. □

Thus COST ≤ C (perfect matching) reduces to COST ≤ C (set of disjoint cycles). This in turn reduces to COST ≤ C (cycle), by the following general technique.

For a fixed S-subgraph, let an S-set be a set of disjoint S-subgraphs. Let FIND (C) be an algorithm for COST ≤ C (S). Often we can convert FIND to an algorithm for COST ≤ C (S-set), by these changes: First, introduce a list LIST to store the S-set. LIST is initially empty. For each call output (H) when an S-subgraph H is found, substitute these lines:

1. begin add H to LIST; remove the vertices of H from the graph;
2. output (LIST);
3. FIND (C-c(H));
4. remove H from LIST; add the vertices of H to the graph; end;

Lemma 3.2: Let FIND, an algorithm for COST ≤ C (S), satisfy this property:

(*) Let H_1 and H_2 be disjoint S-subgraphs, with H_1 output before H_2. When H_2 is output, FIND has modified the graph so H_1 is not an S-subgraph.

Then the above modifications to FIND give an algorithm for COST ≤ C (S-set) having the same time and space bounds as FIND.
Proof: Use induction on the number of S-subgraphs. Property (*) insures that an S-set is listed only once. □

Note (*) often holds. For instance if FIND works by finding min cost S-subgraphs, H_1 must be removed before H_2 can be discovered.

We use these results to get algorithms for two versions of COST ≤ C. First consider finding all perfect matchings (i.e., all costs are 0). Lemmas 3.1-2 reduce this to finding all directed cycles. The most efficient algorithms for all cycles use time O(Km+m+n) and space O(m+n) [J75,RT,SL]. This gives the following result.

Theorem 3.1: All perfect matchings on a bipartite graph can be found in $O(n^{1/2}\ m+n+Km)$ time and O(n+m) space.

Proof: The first perfect matching is found in time $O(n^{1/2}m+n)$ [HK]. □

This result has been previously obtained by Itai et al [IRT]. Our algorithm is essentially equivalent.

Next consider arbitrary cost functions. COST ≤ C (cycle) on a directgraph with no negative cycles can be solvedin $O(K \min(n^3, mn \log n))$ time and O(m+n) space [JK]: The algorithm just repeatedly chooses a vertex s, uses

PATHS of Section 2 to find all s-s paths of cost ≤ C, and then deletes s.

Theorem 3.2: COST ≤ C (perfect matching) on a bipartite graph can be solved in $O(K \min(n^3, mn \log n))$ time and $O(m+n)$ space.

Proof: A min cost perfect matching M is found in time $O(\min(n^3, mn \log n))$ [L76]. The directed graph corresponding to G,M has no negative cycles by the minimality of M. □

A corresponding K BEST algorithm, using the same time and O(K+m+n) space, is easily constructed. This improves the $O(Kn^4)$ time algorithm given in [M].

Next we consider COST ≤ C (maximum matching). First we specify how a graph with an arbitrary matching corresponds to a directed graph. Let G be bipartite, with vertex set (U,V), and arbitrary matching M. The corresponding directed graph is found by enlarging G to a graph G' with a perfect matching, finding the corresponding directed graph D', and enlarging D' to the desired graph D.

To form G', add a vertex x' for each exposed vertex x of G, and add the matched edge (x,x'). G' has a perfect matching, so let D' correspond to G'. Note the vertices of D' correspond naturally to the matched edges of G', and so partition into 3 sets:

W_M (vertices of D' corresponding to matched edges of M);

W_U (vertices corresponding to matched edges (x,x'), $x \in U$)

W_V (vertices corresponding to matched edges (x,x'), $x \in V$).

To form D, add a fourth set of vertices $W_{M'}$ containing a vertex w' for each $w \in W_M$, and add the edge (w',w) for each corresponding pair w',w.

As with perfect matchings, alternating cycles of G correspond to directed cycles of D. An even length alternating path starting at an exposed vertex of U corresponds to a directed path from W_U to W_M. An even alternating path starting in V corresponds to a directed path from $W_{M'}$ to W_V.

Let the edges (w',w) from $W_{M'}$ to W_M have cost -c(u,v), where w corresponds to $(u,v) \in M$. Then the correspondence between cycles and paths preserves cost.

This correspondence shows COST ≤ C (maximum matchings) reduces to COST ≤ C (set of disjoint cycles and S_i-T_i paths, i=1,...,k). (An S-T path, for sets of vertices S, T is a path from some $s \in S$ to some $t \in T$). We allow a number of corresponding sets S_i, T_i). It is easy to see COST ≤ C (S-T path) reduces to COST ≤ C (s-t path). Thus the directed graph problem reduces to COST ≤ C (set of disjoint cycles and s_i-t_i paths, i=1, ..., k). This problem is solved using the following general technique.

Let an $S_1,\ldots,S_k$-set be a set of disjoint S_1-subgraphs, ..., and S_k-subgraphs. If FIND i(C) is an algorithm for COST ≤ C (S_i-set), i=1, ..., k, then an algorithm for COST ≤ C($S_1,\ldots,S_k$-set) can usually be formed as

follows: Let FIND1 be the main routine. Assume all FINDi use LIST to store the S_i-set. In FINDi, i=2, ..., k, remove the initialization of LIST. In FINDi, i=1, ..., k-1, add a call, FINDi+1 (C), on entry; also replace each occurence of output (LIST) by these lines:

1. begin output (LIST);
2. remove the vertices in subgraphs of LIST from the graph;
3. FINDi+1 (C-c(LIST));
4. add the vertices in subgraphs of LIST to the graph; end;

Lemma 3.3: Let FINDi be an algorithm for COST ≤ C(S_i-set), i=1, ..., k. Suppose no S_i-subgraph has negative cost, for i=2, ..., k. Then the above modifications to FINDi give an algorithm for COST ≤ C(S_1,...,S_k-set), whose time and space bounds are the maximum of those for FINDi, i=1, ..., k. □

Now we extend Theorems 3.1-2 to maximum matchings. When costs are 0, all s-t paths can be found using essentially the same algorithm as for cycles [RT].

Theorem 3.3: All maximum matchings on a bipartite graph can be found in time $O(n^{1/2}m+n+Km)$ time and $O(m+n)$ space. □

An equivalent algorithm is given by Itai et al [IRT].
For arbitrary costs, we use PATHS of Section 2 for s-t paths.

Theorem 3.4: COST ≤ C (maximum matching) on a bipartite graph can be solved in $O(K \min(n^3, mn \log n))$ time and $O(m+n)$ space. □

Finally we consider COST ≤ C (cardinality k matching). Let G be a bipartite graph with a matching M of cardinality k; let D be the corresponding directed graph defined above. Augmenting and de-augmenting paths in G corresponds to W_U-W_V and W_M,-W_M paths in D, respectively. Thus COST ≤ C (cardinality k matching) reduces to COST ≤ C (set of disjoint cycles, S_1-T_1 paths, S_2-T_2 paths, and S_1-T_2, S_2-T_1 path pairs). (An S_1-T_2, S_2-T_1 path pair consists of an S_1-T_2 path and an S_2-T_1 path, disjoint from each other.) Vertex sets S_1, T_1, S_2, T_2 are disjoint; they correspond to W_U, W_M, W_M,, W_V, respectively.

To solve the COST ≤ C problem on D, we use a graph D*. Form D* by adding two vertices z_i, i=1,2. Add cost 0 edges (w,z_i), where $w \in T_i$, and (z_i,w), where $w \in S_i$, i=1,2. Note the cycles in D* correspond in D to cycles, S_1-T_1 paths, S_2-T_2 paths, and S_1-T_2, S_2-T_1 path pairs.

Unfortunately, the problem in D does not reduce to all cycle-sets in D*. In the direct sum principle, a cardinality k matching corresponds to a set containing an equal number of augmenting and de-augmenting paths, i.e., these paths are not naturally paired. This means in D, sets of path pairs composed of the same paths are not distinct. We will generate sets of S_1-T_2, S_2-T_1 path pairs so (numbers of) the start vertices of both paths increase in successive pairs. This way, a set of path pairs is generated only once.

Another misfortune is that to generate all path pairs, there seems to

be no way to avoid generating all cycles of D*. Although this involves extra work, we shall see it is not excessive.

The following algorithm, executed on D*, solves COST ≤ C (S_1-T_2,S_2-T_1 path pair-set) for D.

```
procedure PAIRS (C);
1.  begin for each cycle of cost ≤ C do
2.          if the cycle corresponds to an S1-T2, S2-T1 path pair P then
3.          begin add P to LIST;
4.          remove from D* the vertices of P;
5.          let si be the start vertices of P, si,ε Si, i=1,2;
6.          remove from D* all edges (zi,s), for s ε Si with s < si, i=1,2;
7.          output (LIST);
8.          PAIRS (C-c(P));
9.          add the vertices and edges removed in lines 4 and 6 to D*;
10.         remove P from LIST;
    end     end PAIRS;
```

Lemma 3.4: PAIRS solves COST ≤ C (S_1-S_2,S_2-T_1 path pair-set). If COST ≤ C (cycle) can be solved in time $O(K \cdot T(n,m))$, and K_{D*} is the number of cycle sets of cost ≤ C in D*, then PAIRS uses time $O(K_{D*} \cdot T(n,m))$ and space $O(m+n)$.

Proof: Each cycle found in line 1 corresponds to a distinct cycle-set of D*. □

To solve the original COST ≤ C problem on D, we use PAIRS (on D*) to find all desired pair sets. For each pair set, we find all sets of disjoint cycles, S_1-T_1 paths, and S_2-T_2 paths, using previous algorithms (on D). Note if K is the number of solutions to the entire COST ≤ C problem, then in Lemma 3.4, $K_{D*} \leq K$. Thus the pair sets are found in acceptable time.

Theorem 3.4: All cardinality k matchings on a bipartite graph can be found in $O(n^{1/2}m+n+Km)$ time and $O(m+n)$ space. □

Theorem 3.5: COST ≤ C (cardinality k matching) on a bipartite graph can be solved in $O(K \min(n^3, mn \log n))$ time and $O(m+n)$ space. □

These two results are the most general of this Section. Theorem 3.4 extends the results of [IRT]. Theorem 3.5 is easily modified for K BEST (cardinality k matching); the time remains the same and the space increases to $O(K+m+n)$. This extends and improves the $O(K n^4)$ time algorithm for K BEST (perfect matching) of [M].

4. THE SECOND SHORTEST PATH

The COST ≤ C (s-t path) problem is essentially equivalent to finding a second shortest s-t path. (A COST ≤ C algorithm can be constructed from a second path algorithm, using the approach of PATHS of Section 2; the time per path for COST ≤ C is the time for the second path.) This Section shows finding a second shortest path is equivalent to a variant of the all pairs shortest path problem. The best algorithms for the latter use $O(\min(n^3, mn \log n))$ time [J77]. So this Section gives some justification for

the seemingly high time bound for PATHS of Section 2.

We start by formally defining two problems on directed graphs with no negative cycles.

SP: Given vertices s, t, and a value L, does the second shortest s-t path have length $\leq$ L?

AP: Given values $\ell(v,w)$ for each pair of distinct vertices v, w, does some v-w path have length $\leq \ell(v,w)$?

Let S(n) and A(n) denote the time to solve SP and AP, respectively, on an n vertex graph. (Note the restriction in AP that v and w are distinct is inessential. Allowing bounds $\ell(v,v)$ on non-trivial cycles does not change the time complexity A.)

Now we show S(n) and A(n) are essentially equal. First note the structure of a second shortest s-t path:

Lemma 4.1: If P is a shortest s-t path, there are vertices v and w on P, with v preceding w, and a v-w path Q disjoint from P-v-w, such that a second shortest s-t path consists of Q plus the s-v and w-t portions of P. (Possibly s=v or w=t). □

Lemma 4.2: $S(n) = O(n^2+A(n))$.

Proof: Let an instance of SP be given on a directed graph G. Let P be a shortest s-t path; if v precedes w on P, let p(v,w) be the length of the portion of P from v to w. Let M be the largest magnitude of an edge length. Define an instance of AP on a graph G', as follows. G' is G with the edges of P deleted. The bound $\ell(v,w)$ is $L-p(s,v) - p(w,t)$, if v precedes w on P; otherwise it is -nM. It is easy to see the instances of SP and AP have the same answer. □

Lemma 4.3: $A(n) = O(n^2+S(n))$.

Proof: Let an instance of AP be given on a graph G, having vertices v_i, i=1, ..., n. Without loss of generality, assume G has no negative edges [J77]. Let M be greater than the largest magnitude of a path length bound $\ell(v_i,v_j)$.

Define an instance of SP on a graph G' with edge lengths given by c, and length bound L = nM, as follows. G' contains a copy of G. The additional vertices are u_i, i=1, ..., n+1, where $s = u_1$, $t = u_{n+1}$. The additional edges are in a shortest path from s to t: (u_i,u_{i+1}), i=1, ..., n, with $c(u_i,u_{i+1}) = 0$; also edges leaving the path (u_i,v_i), i=1, ..., n, with $c(u_i,v_i) = (n-i)M$; and edges entering the path: (v_j,u_{i+1}), i,j=1,..., n with $c(v_j,u_{i+1}) = iM-\ell(v_i,v_j)$.

Note all edge lengths are non-negative. Thus a shortest s-t path is $(u_1,\ldots,u_{n+1})$. Now we show the instances of AP and SP have the same answer.

If AP has answer "yes" then some v_i-v_j path has length $\leq \ell(v_i,v_j)$. Adding on the paths $(u_1,\ldots,u_i,v_i)$ and $(v_j,u_{i+1},\ldots,u_{n+1})$ gives an s-t path of length $\leq$ nM. So SP has answer "yes".

Conversely, suppose SP has answer "yes". A second shortest s-t path consists of a path in G from v_i to v_j, plus paths $(u_1,\ldots,u_i,v_i)$ and $(v_j,u_{k+1},u_{k+2},\ldots,u_{n+1})$, where $k \geq i$. Its length is $(n-i+k)M - \ell(v_k,v_j) + p$, where p is the length of the v_i-v_j path. For this to be $\leq$ nM, we must have k = i, and $p \leq \ell(v_i,v_j)$. Thus AP has answer "yes". □

Combining Lemmas 4.2-3 gives the desired result.

Theorem 4.1 The time for SP is $O(n^2)$ if and only if the time for AP is $O(n^2)$. Otherwise, the time for SP is the same order as the time for AP. □

ACKNOWLEDGMENT

The author thanks Frank Fussenegger of Martin Marietta Coroporation for his inspiring discussions on matching.

REFERENCES

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

[CFM] P. M. Camerini, L. Fratta, and F. Maffioli, "The K best spanning arborescences of a network", Istituto di Elettrotecnica ed Elettronica, Politecnico di Milano, Milano, Italy, preprint.

[E] S. Even, Algorithmic Combinatorics, Macmillan Co., New York, 1973.

[G] H. N. Gabow, "Two algorithms for generating weighted spanning trees in order", SIAM J. Comp. 6, 1977, pp. 139-150.

[HK] J. Hopcroft and R. Karp, "An $n^{5/2}$ algorithn for maximum matchings in bipartite graphs," SIAM J. 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.

[J75] D. B. Johnson, "Finding all the elementary circuits of a directed graph", SIAM J. Comp. 4, 1975, pp. 77-84.

[J77] D. B. Johnson, "Efficient algorithms for shortest paths in sparse networks", J.ACM 24, 1977, pp.1-13.

[JK] D. B. Johnson and S. D. Kashdan, "Lower bounds for selection in X + Y and other multisets", SIAM J. Comp. 7, 1978, pp. 147-153.

[KIM] N. Katoh, T. Ibaraki, H. Mine, "An algorithm for finding K minimum spanning trees", Dept. of Appl. Math. and Physics, Kyoto Univ., Kyoto, Japan, preprint.

[L72] E. L. Lawler, "A procedure for computing the K best solutions to discrete optimization problems and its application to the shortest path problem", Mgt. Sci. 18, 1972, pp. 401-405.

[L76] E. L. Lawler, Combinatorial Optimization: Networks and Matroids, Holt, Rinehart and Winston, New York, 1976.

[M] K. G. Murty, "An algorithm for ranking all the assignments in increasing order of cost", Op.Res.16, 1968, pp. 682-687.

[MY] M. Machtey and P. Young, An Introduction to the General Theory of Algorithms, North-Holland Co., New York, 1978.

[RT] R. C. Read and R. E. Tarjan, "Bounds on backtrack algorithms for listing cycles, paths, and spanning trees", Networks 5, 1975, pp. 237-252.

[SL] J. L. Szwarcfiter and P. E. Lauer, "A search strategy for the elementary cycles of a directed graph", BIT 16, 1976, pp. 192-204.

[V77a] L. G. Valiant,"The complexity of computing the permanent", Int. Rep.CSR-14-77, Dept. Comp. Sci., Univ. of Edinburgh, Edinburgh,Scotland, Oct.1977.

[V77b] L. G. Valiant, "The complexity of enumeration and reliability problems", Int. Rep. CSR-15-77, Dept. Comp. Sci., Univ. of Edinburgh, Edinburgh, Scotland, Oct. 1977.

[Y] J. Y. Yen, "Finding the K shortest loopless paths in a network", Mgt. Sci. 17, 1971, pp. 712-716.