AD-A091 312 TEXAS UNIV AT AUSTIN CENTER FOR CYBERNETIC STUDIES F/G 12/1
A SUCCESSIVE SHORTEST PATH ALGORITHM FOR THE ASSIGNMENT PROBLEM--ETC(U)
AUG 80 M ENGQUIST N00014-75-C-0569
UNCLASSIFIED CCS-375 NL

1 OF 1
AD
A091312

END
DATE
FILMED
11-80
DTIC

AD A091312

LEVEL ~~II~~ (12)

CENTER FOR CYBERNETIC STUDIES

The University of Texas
Austin, Texas 78712

DTIC
ELECTE
NOV 4 1980
S D
C

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

SEAL OF THE UNIVERSITY OF TEXAS
PRAESIDIUM
DISCIPLINA CIVITATIS

80 10 31 182

12

Research Report CCS 375

A SUCCESSIVE SHORTEST PATH ALGORITHM FOR THE ASSIGNMENT PROBLEM

by

Michael Engquist*

[illegible]

DTIC
ELECTE
NOV 4 1980
C

August 1980

*Visiting Associate Professor of Statistics and Operations Research,
The University of Texas at Austin, 1979-80
Associate Professor of Mathematics and Computer Science,
Eastern Washington University, 1980-81

This research is partly supported by ONR Contracts N00014-75-C-0569 and N00014-80-C-0242 with the Center for Cybernetic Studies, The University of Texas at Austin. Reproduction in whole or in part is permitted for any purpose of the United States Government.

CENTER FOR CYBERNETIC STUDIES

A. Charnes, Director
Business-Economics Building, 203E
The University of Texas at Austin
Austin, Texas 78712
(512) 471-1821

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

ABSTRACT

In this paper a new successive shortest path (SSP) algorithm for solving the assignment problem is introduced. A computer implementation of this algorithm has been developed and a discussion of the details of this implementation is provided. Computational results are presented which show this implementation of SSP to be substantially more efficient than several recently developed codes including the best primal simplex code. Also, some new theoretical results are presented which are useful in the implementation of SSP, and it is shown that the algorithm has a computational bound of $O(n^3)$, where n is the number of origins.

Accession For
NTIS GRA&I ✓
DTIC TAB ☐
Unannounced ☐
Justification

By
Distribution/
Availability Codes

Dist	Avail and/or Special
A	

1. INTRODUCTION

Recently, there has been quite a lot of activity in the development of new algorithms and computer codes for solving assignment problems [2], [16], [19]. In this study, we compare these approaches to a successive shortest path algorithm (SSP) which is a refinement of the Dinic-Kronrod algorithm [7]. We have used SSP to develop a computer code which is very efficient for solving large, sparse assignment problems, and we introduce some new theoretical results which are useful in our implementation. Furthermore, it is shown that SSP has a computational bound of $O(n^3)$, where n is the number of origins. SSP goes through a series of modified assignment problems in which some destinations are permitted to have demands greater than one, while some demands are set to zero. The algorithm proceeds from the optimal solution of one of these modified problems to the optimal solution of the next via a related shortest path problem. The algorithm terminates when the modified problem coincides with the original assignment problem. This algorithm is closely related to those developed independently by Hung and Rom [16] and Gribov [14].

Weintraub and Barahona [19] have based their work on the minimum cost flow algorithm of Edmonds and Karp [9]. Although this approach has some similarities to SSP, it is evidently a different algorithm.

Barr, Glover, and Klingman [2] developed a new version of the primal simplex algorithm called the AB algorithm which examines only certain bases (called the alternating path bases) representing a given extreme point. Even with this improvement to the primal simplex algorithm, over 90% of the pivots are degenerate.

The algorithms tested, other than the AB algorithm, substitute a more complicated procedure for the primal simplex pivot so that nondegenerate flow change and progress toward optimality are guaranteed at each iteration.

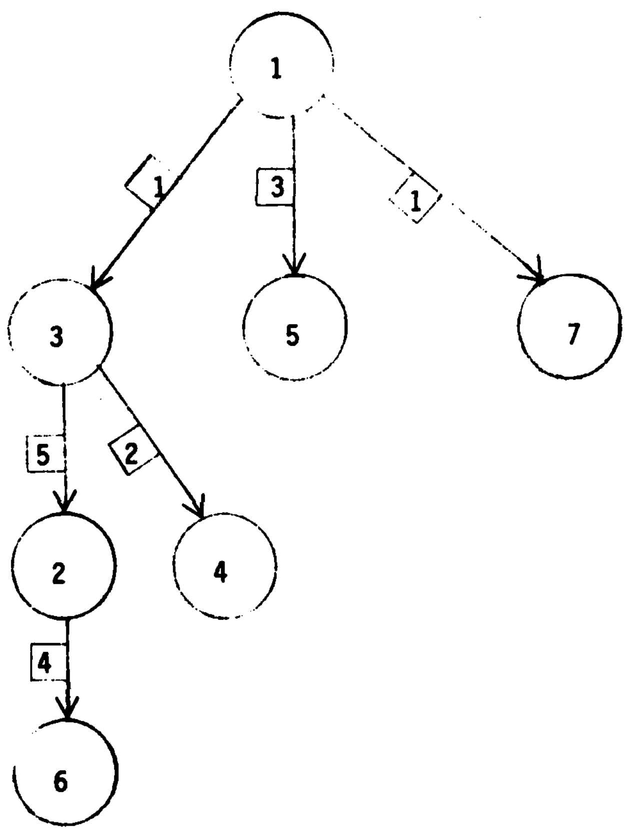
2. BACKGROUND ON SHORTEST PATH PROBLEMS

Since our implementation of SSP is based on a label-setting approach for solving the related shortest path problems we give a very general outline of this approach. Further details may be found in [5], [8], or [10]. The type of shortest path problem we wish to solve involves a directed network, a special node r (called the root) and a set of special nodes (called abundant nodes) such that r is not abundant. It is desired to find the shortest path from r to some abundant node.

The label setting algorithm begins with a shortest path tree consisting only of r, and it assigns a distance of zero to r and a distance of infinity to all other nodes in the shortest path network. Roughly speaking, at each iteration, the node closest to the existing tree is adjoined to the tree and the shortest distance from the root to that node is computed (when this happens, the node is said to be <u>permanently labelled</u>). This process is terminated when one of the abundant nodes is permanently labelled.

The solution of a shortest path problem is given in terms of the rooted tree T of permanently labelled nodes. In dealing with such trees, the predecessor list is useful. The predecessor of a node $v \neq r$ in the tree is the starting node u of the single arc in the tree terminating at v. We illustrate these ideas by giving an example of such a rooted tree in

Figure 1. In this example, the arcs are labelled with their lengths as given in the shortest path problem, and the root r equals 1.



Node, i	Predecessor, P_i	Distance, D_i
1	none	0
2	3	6
3	1	1
4	3	3
5	1	3
6	2	10
7	1	1

Fig. 1. A shortest path tree.

In a shortest path tree, the unique sequence of nodes beginning with a node $v \neq r$ and leading to r is called the path from v to the root and is denoted S(v). For our purposes, it is convenient to exclude r from S(v). Thus, in Figure 1, S(4) is 4,3. We note that S(v) may be generated by taking successive predecessors.

The label-setting approach we employ is based on the Dijkstra algorithm [6] and we use Dial's implementation [4] of that algorithm. Dial's implementation involves a modular sort list which is used for obtaining the node closest to the existing tree. Each position on this list represents a distance (reduced by the modulus) from the root. The length of the sort list (i.e., the modulus) is one greater than the maximum arc length for the shortest path problem. A complication which arises in the implementation of SSP is that the maximum arc length in the related shortest path problems increases as the algorithm proceeds. This complication is resolved by employing a single radix sort as described in [5]. For the single radix sort, each position on the modular sort list represents a range of distances from the root, and the length of the sort list remains fixed for all related shortest path problems in our implementation of SSP.

3. DESCRIPTION OF SSP

We begin this section with a review of some terminology. A directed network, or for simplicity, a network, consists of a finite set of nodes and a finite set of arcs. Each arc may be identified with an ordered pair of distinct nodes. That is, we can visualize an arc as beginning at the first node in the ordered pair and terminating at the second. The set of arcs emanating from a node u is called the forward star of u and is denoted FS(u). Similarly, the set of arcs entering a node u is called the reverse star of u and is denoted RS(u).

A statement of the nxn assignment problem follows.

$$\text{Minimize} \sum_{(i,j)\in E} c_{ij}\, x_{ij}$$

$$\text{subject to} \sum_{(i,j)\in FS(i)} x_{ij} = 1,\ i \in I = \{1,2,3\ldots,n\}$$

$$\sum_{(i,j)\in RS(i)} x_{ij} = 1,\ j \in J = \{1,2,3\ldots,n\}$$

$$x_{ij} \geq 0,\ (i,j) \in E$$

where I is the set of origin nodes, J is the set of destination nodes, E is the set of arcs, and c_{ij} is the cost of a unit flow on arc (i,j). That is, c_{ij} is the cost of assigning origin node i to destination node j. We define C to be $\{c_{ij}: (i,j) \in E\}$. The nodes $I \cup J$ together with the arcs E form the assignment network.

In order to avoid difficulties with the definition of the assignment problem, we will assume that FS(i) and RS(j) are not empty for $i \in I$ and $j \in J$.

Next, we introduce a concept which is central to the development of SSP. Whenever a mapping $A: I \rightarrow J$ is given, we say that A defines a tentative assignment provided that $(i, A_i) \in E$ for $i \in I$.

Since the assignment network will remain fixed in the following discussion, we denote the assignment problem equipped with a tentative assignment A by (C,A). We say that (C,A) is in standard form if $c_{ij} \geq 0$ for all $(i,j) \in E$ and $c_{ij} = 0$ when $j = A_i$. We note that when (C,A) is in standard form and A is one-to-one, then A determines an optimal solution of the assignment problem.

In the starting procedure for SSP a tentative assignment A is defined as follows. First,

$$\hat{c}_i = \min_{(i,p) \in FS(i)} \{c_{ip}\}$$

must be determined for $i \in I$. Next, for $i \in I$, A_i is defined to be some j such that $c_{ij} = \hat{c}_i$. For this A, (C,A) may not be in standard form. However, the forward star of each origin i may be scaled by setting

$$c_{ip} \longleftarrow c_{ip} - \hat{c}_i$$

for $(i,p) \in FS(i)$. The resulting (C,A) is in standard form. This technique is also used in the starting procedure for SSP. Of course, such scaling does not affect the solution of the original assignment problem.

For a given tentative assignment A, we let a_j denote the number of elements in $\{i\colon j = A_i\}$.

The modified assignment problem relative to (C,A) is defined as follows:

$$\text{Minimize} \sum_{(i,j)\in E} c_{ij}\, x_{ij}$$

$$\text{subject to} \sum_{(i,j)\in FS(i)} x_{ij} = 1,\ i \in I$$

$$\sum_{(i,j)\in RS(j)} x_{ij} = a_j,\ j \in J$$

$$x_{ij} \geq 0,\ (i,j) \in E$$

We note that when (C,A) is in standard form, A provides an optimal solution to the modified assignment problem relative to (C,A).

A destination node j is said to be abundant relative to A when $a_j > 1$. Likewise, j is said to be deficient relative to A when $a_j = 0$.

Suppose that (C,A) is in standard form and d is some deficient node with respect to A. Then the shortest path problem relative to (C,A) and d is denoted SP(C,A,d) and is defined as follows. The network for SP(C,A,d), which we refer to as the shortest path network, is derived from the assignment network. We proceed by describing how this is done and how the arc lengths for SP(C,A,d) are defined. The nodes of the shortest path network can be identified with arcs (i,j) of the assignment network which satisfy $j = A_i$. We denote such a shortest path node by $(i \longrightarrow A_i)$ or $(i \longrightarrow j)$ where $j = A_i$. Also, we refer to $(i \longrightarrow A_i)$ as the ith shortest path node.

Clearly, there n such nodes. We introduce one more node $(n+1 \rightarrow d)$ for the shortest path network and make this consistent with previous notation by extending A so that $A_{n+1} = d$. For SP(C,A,d), the root node is $(n+1 \rightarrow d)$ while a node $(i \rightarrow A_i)$ is abundant provided A_i is an abundant destination relative to A. An arc exists in the shortest path network from $(i \rightarrow A_i)$ to $(p \rightarrow A_p)$ in case $(p, A_i) \in E$. If this arc exists, its length is c_{pj} where $j = A_i$.

In order to fix ideas, we present an example showing how SP(C,A,d) is defined in a particular case. Let the assignment problem be as shown in Figure 2, where arc (i,j) is labelled with cost c_{ij}. We let $A_1 = 1$, $A_2 = 1$, $A_3 = 2$ and note that (C,A) is in standard form. Destination 3 is deficient and we let d = 3. The network for SP(C,A,d) is shown in Figure 3 where a shortest path node $(i \rightarrow A_i)$ is shown as a node with an upper label (i) and a lower label (A_i). The arcs of the shortest path network are labelled with their lengths.

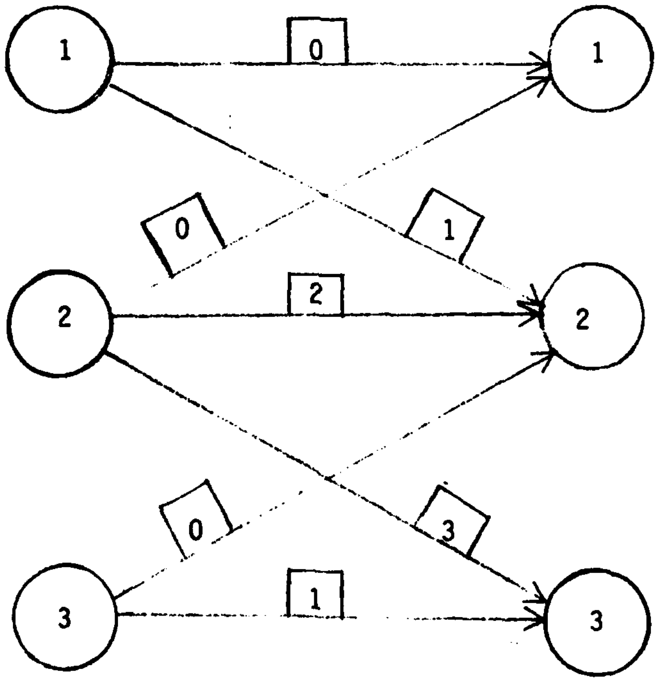


Fig. 2. An assignment problem

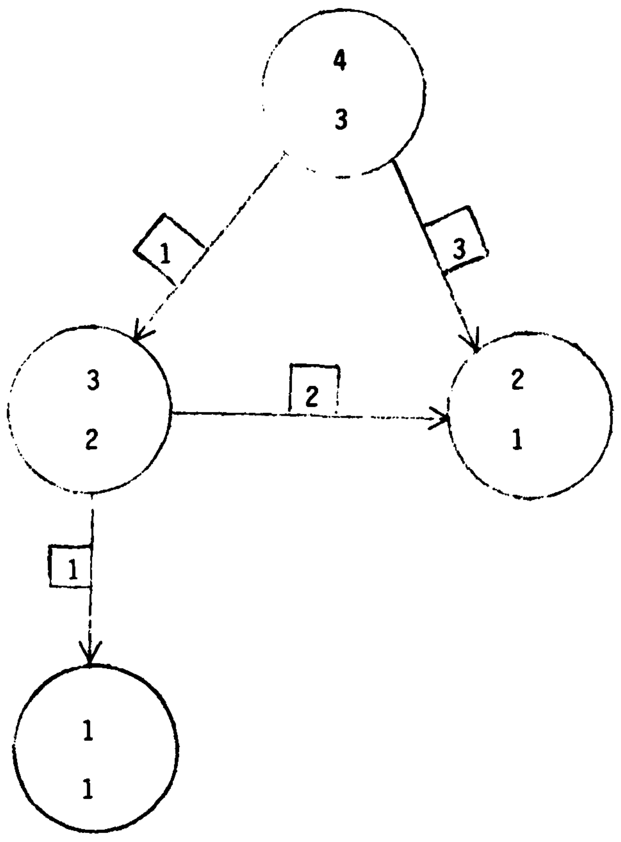


Fig. 3. An example of SP(C,A,d).

We remark that the arcs in the shortest path network are oppositely directed from their counterparts in the assignment network. This is why, in our implementation of SSP, the problem data for the assignment problem is stored in reverse star form. That is, the data for the arcs is stored consecutively in computer memory so that RS(j) appears immediately after the data for RS(j-1), where j is a destination. A pointer list is employed to indicate the entry position for the block of memory locations assigned to each reverse star.

The major steps of SSP will be listed after we set up some notation.

R_i = node potential for the ith origin node

K_j = node potential for the jth destination node

D_i = distance of the ith shortest path node from the root

P_i = predecessor of the ith shortest path node in the shortest path tree

We use R to denote the mapping whose value at i is R_i. The use of K, D, and P is similar. When a shortest path problem is solved, we will denote the first abundant node to be permanently labelled by v, and we will denote the distance of v from the root by L. We let $C^0 = \{c^0_{ij}: (i,j) \in E\}$ denote the costs for the original (unmodified) assignment problem.

0. Define A^1 and transform C^0 to C^1 by scaling as described above so that (C^1, A^1) is in standard form. Set $k = 1$.
1. Choose a destination node d^k which is deficient relative to A^k. If no deficient nodes exist, stop since A^k defines an optimal solution.
2. Solve $SP(C^k,A^k,d^k)$. The shortest path algorithm is terminated as soon as an abundant shortest path node is permanently labelled. If the shortest path algorithm fails to permanently label an abundant node, stop since the assignment problem is infeasible. Otherwise, the results of this step are D^k, P^k, v^k and L^k.
3. For each permanently labelled shortest path node $(i \longrightarrow j)$ from step 2, set $R^k_i = D^k_i - L^k$ and $K^k_j = L^k - D^k_i$. For any remaining origins i or destinations j, set $R^k_i = K^k_j = 0$.

4. Set $c_{ij}^{k+1} = c_{ij}^{k} - R_i^k - K_j^k$ for $(i,j) \in E$.

5. Whenever the ith shortest path node is in $S(v^k)$, set $A_i^{k+1} = A_\ell^k$ where $\ell = P_i^k$. For all other origins i, $A_i^{k+1} = A_i^k$. Set $k \leftarrow k+1$ and go to 1.

We claim that after the updates in steps 4 and 5, (C^k, A^k) is in standard form. This is verified in section 4.

4. THEORETICAL RESULTS

In this section, certain theoretical properties of SSP are examined. Theorem 1 is a convergence result, while Theorem 2 deals with the computational complexity of SSP. Theorem 3 is useful in obtaining the optimal value of the objective function, and Theorem 4 provides a verification of the infeasibility criterion in step 2 of the algorithm.

A node p of the assignment network (either an origin or a destination) is said to be on a shortest path tree T provided that p = i or p = j holds for some node $(i \to j)$ of T.

An origin node i and a destination node j of the assignment network are said to be adjacent on a shortest path tree T provided that either

(a) $(i \to j)$ is a node of T

or

(b) $(p \to j)$ and $(i \to q)$ are nodes of T for some p and q and there is an arc of T from $(p \to j)$ to $(i \to q)$.

We will let T^k denote the kth shortest path tree generated when $SP(C^k,A^k,d^k)$ is solved in step 2 of SSP.

Next, we present three preliminary results.

(1.) Suppose that (C^k,A^k) is in standard form and that step 4 of the algorithm has been completed. Then $c_{ij}^{k+1} = 0$ whenever i and j are adjacent on T^k.

Proof of (1.): In case $(i \rightarrow j)$ is a node of T^k, $c_{ij}^k = 0$. Hence,

$$c_{ij}^{k+1} = -(D_i^k - L^k) - (L^k - D_i^k) = 0.$$

In the other case, there exist p and q such that $(p \rightarrow j)$ is the predecessor of $(i \rightarrow q)$ in T^k. Hence, $c_{ij}^k + D_p^k = D_i^k$. It follows that

$$c_{ij}^{k+1} = c_{ij}^k - (D_i^k - L^k) - (L^k - D_p^k) = 0.$$

(2.) Suppose that (C^k,A^k) is in standard form and that step 4 of the algorithm has been completed. Then $c_{ij}^{k+1} \geq 0$ for $(i,j) \in E$.

Proof of (2.): We consider four cases based on whether i,j are on T^k.

(a) Both i and j are on T^k. If i and j are adjacent on T^k, result (1.) applies and we are done. Otherwise, there exists $\ell \neq i$ such that $(\ell \rightarrow j)$ is on T^k. We have $c_{ij}^k + D_\ell^k \geq D_i^k$ by the way T^k is constructed. It now follows readily that $c_{ij}^{k+1} \geq 0$.

(b) Suppose i is on T^k but j is not. Since $K_j^k = 0$, we have

$$c_{ij}^{k+1} = c_{ij}^k - (D_i^k - L^k) = c_{ij}^k + (L^k - D_i^k).$$

Now, $c_{ij}^{k} \geq 0$ by hypothesis and $L^k - D_i^k \geq 0$ by the way T^k is constructed.

(c) Suppose that i is not on T^k but j is. We have $R_i^k = 0$ and there exists ℓ such that $(\ell \rightarrow j)$ is on T^k. Since the ith shortest path node is not permanently labelled, $c_{ij}^{k} + D_{\ell}^{k} \geq L^k$. Since $K_j^k = L^k - D_{\ell}^{k}$, $c_{ij}^{k+1} = c_{ij}^{k} - K_j^k \geq 0$.

(d) Neither i nor j is on T^k. We have $c_{ij}^{k+1} = c_{ij}^{k} \geq 0$.

(3.) Suppose (C^k, A^k) is in standard form. After the update of A in step 5 of the algorithm, (C^{k+1}, A^{k+1}) is in standard form.

Proof of (3.): By result (2.) above, $c_{ij}^{k+1} \geq 0$ for $(i,j) \in E$. Hence, we need only verify that $j = A_i^{k+1}$ implies $c_{ij}^{k+1} = 0$. In case $j = A_i^{k+1} = A_i^k$, we have $c_{ij}^{k+1} = c_{ij}^{k} = 0$. If $j = A_i^{k+1} \neq A_i^k$, it follows that i and j are adjacent on T^k. By result (1.), $c_{ij}^{k+1} = 0$.

Theorem 1. If SSP does not stop because of the infeasibility test in step 2, it reaches optimality in at most n-1 iterations.

Proof: We proceed by induction. We have that (C^1, A^1) is in standard form with at most n-1 deficient destinations. If we assume that (C^k, A^k) is in standard form with $q \geq 1$ deficient destinations, it follows (using result (3.)) that (C^{k+1}, A^{k+1}) is in standard form with q-1 deficient destinations.

Theorem 2. SSP has a $O(n^3)$ computational bound.

Proof: When the algorithm does not indicate an infeasible assignment problem, it requires at most n-1 iterations by Theorem 1. This result, together with the $O(n^2)$ computational bound for the Dijkstra shortest path algorithm, implies the $O(n^3)$ bound in this case.

On the other hand, if the algorithm indicates an infeasible assignment problem at some iteration m, then $m \leq n-1$ by Theorem 1. The Dijkstra algorithm can be modified in an obvious way to detect a failure to permanently label an abundant node, and the modified Dijkstra algorithm will have an $O(n^2)$ bound as before. Consequently, the $O(n^3)$ bound holds in this case as well.

Next, we introduce some notation which is needed for Theorem 3.

$$R_i^0 = \min_{(i,j) \in FS(i)} \{c_{ij}^0\}, \quad i \in I$$

$$K_j^0 = 0, \quad j \in J$$

$$L^0 = \sum_{i=1}^{n} R_i^0$$

$$\bar{R}_i^m = \sum_{k=0}^{m-1} R_i^k, \quad i \in I$$

$$\bar{K}_j^m = \sum_{k=0}^{m-1} K_j^k, \quad j \in J$$

We refer to $\bar{R}_i^m$ and $\bar{K}_j^m$ as the <u>accumulated node potentials</u>, where m is some iteration of SSP.

Theorem 3. Assume that the conditions of Theorem 1 hold. Then the optimal objective value for the modified assignment problem relative to (C^0, A^m) is

$$\sum_{k=0}^{m-1} L^k.$$

Proof: Since A^m provides an optimal solution to the modified assignment problem relative to (C^m, A^m), it also provides an optimal solution to the modified assignment problem relative to (C^0, A^m). This follows since C^m can be obtained from C^0 by scaling using the accumulated node potentials. The optimal objective value for the latter problem is therefore

$$\sum_{j=A_i^m,\, i \in I} c_{ij}^0$$

In the remainder of the proof "$i \in I$" will hold for all summations involving i and will not be written in order to simplify the notation.

We will prove the following proposition by induction:

$$\sum_{k=0}^{m-1} L^k = \sum_{j=A_i^m} c_{ij}^0$$

The proposition is clearly true when m = 1.

We assume the proposition for a general m. Then,

$$\sum_{k=0}^{m} L^k = \sum_{j=A_i^m} c_{ij}^0 + L^m$$

$$= \sum_{\substack{j=A_i^m \\ i \notin S(v^m)}} c_{ij}^0 + \sum_{\substack{j=A_i^m \\ i \in S(v^m)}} c_{ij}^0 + \sum_{\substack{j=A_i^{m+1} \\ i \in S(v^m)}} (c_{ij}^0 - \bar{R}_i^m - \bar{K}_j^m)$$

where the latter equality uses the definition of L^m. Thus,

$$\sum_{k=0}^{m} L^k = \sum_{\substack{j=A_i^m \\ i \notin S(v^m)}} c_{ij}^0 + \sum_{\substack{j=A_i^m \\ i \in S(v^m)}} (\bar{R}_i^m + \bar{K}_j^m) + \sum_{\substack{j=A_i^{m+1} \\ i \in S(v^m)}} (c_{ij}^0 - \bar{R}_i^m - \bar{K}_j^m)$$

using preliminary result (1.). Hence,

$$\sum_{k=0}^{m} L^k = \sum_{\substack{j=A_i^m \\ i \notin S(v^m)}} c_{ij}^0 + \sum_{\substack{j=A_i^{m+1} \\ i \in S(v^m)}} c_{ij}^0 + \bar{K}_a^m - \bar{K}_d^m$$

where $\bar{K}_a^m$ is the accumulated potential for the abundant destination a such that $(v^m \rightarrow a)$ is on T^m and $\bar{K}_d^m$ is the accumulated potential for d^m. Clearly, $\bar{K}_a^m = \bar{K}_d^m = 0$. Finally,

$$\sum_{k=0}^{m} L^k = \sum_{j=A_i^{m+1}} c_{ij}^0$$

which is the proposition we wish to prove with m replaced by m+1.

It follows from Theorem 3 that $\sum_{k=0}^{m-1} L^k$ is a lower bound on the optimal objective value for the original assignment problem for any iteration m.

Theorem 4. The original assignment problem is infeasible if and only if the shortest path algorithm fails to permanently label an abundant node of $SP(C^k,A^k,d^k)$ on some iteration k.

Proof: Half the proof follows from Theorem 1. Regarding the other half, we note that whenever a path from the root to some abundant node exists, the shortest path algorithm will eventually permanently label an abundant node. We proceed by assuming that there is no path from the root to an abundant node in $SP(C^k,A^k,d^k)$ and that the desired conclusion--infeasibility of the assignment problem--does not hold. Thus, there is a one-to-one mapping $A:I \rightarrow J$ such that $(i,A_i) \in E$ for $i \in I$.

Now, let i_1 be chosen so that $A:i_1 \rightarrow d^k$. Let $A^k:i_1 \rightarrow j_1$. Since d^k is deficient relative to A^k, $j_1 \neq d^k$. Let i_2 be chosen so that $A:i_2 \rightarrow j_1$. Since A is one-to-one, $i_2 \neq i_1$. Let $A^k:i_2 \rightarrow j_2$. We have that $j_2 \neq d^k$ as before. Also, $j_2 \neq j_1$ since otherwise $(i_1 \rightarrow j_1)$ would be an abundant node for $SP(C^k,A^k,d^k)$ and there would be a path to this abundant node from the root. This process may be repeated indefinitely so that a sequence of distinct nodes of the assignment network-- $d^k,i_1,j_1,i_2,j_2,\ldots$ -- is created. This contradicts the finiteness of the assignment network and the proof of the theorem is complete.

5. COMPUTATIONAL ASPECTS

We have developed a FORTRAN code called SPAN which is an implementation of SSP. In this section we will discuss some of the details of this implementation.

The major steps of SSP listed in section 3 were formulated for ease of exposition and not for computational efficiency. For this reason, there is a difference between steps 3 and 4 as listed and what is done in the SPAN code. In step 3, node potentials are defined for all nodes of the assignment network at each iteration, while in SPAN, the accumulated node potentials introduced in section 4 are maintained. Thus, at iteration k, only the accumulated node potentials corresponding to assignment nodes on T^k need to be updated. In step 4, the cost data for all arcs is updated; however, this is not done in SPAN. Instead, whenever a cost c_{ij}^k is needed in the solution of $SP(C^k,A^k,d^k)$, it is computed using the relation $c_{ij}^k = c_{ij}^0 - \bar{R}_i^k - \bar{K}_j^k$. Next, we describe how the details of some SSP steps were handled in SPAN.

In the starting procedure A^1 is defined by setting A_i^1 equal to j where c_{ij}^0 is minimal over all costs of arcs in FS(i). There may be more than one j which could be chosen. We developed a heuristic for breaking ties in such a way that the number of deficient destinations relative to A^1 is decreased. However, on the basis of limited computational testing, we concluded that our heuristic was of benefit only for assignment problems in which the cost range is small. For this reason, we did not include the heuristic in SPAN. In SPAN the smallest j such that c_{ij}^0 is minimal over costs of arcs in FS(i) is chosen.

In step 1 of SSP, there may be more than one d^k which could be chosen. We did some experimentation but were unable to develop a more efficient strategy than simply choosing the smallest j such that j is deficient. This is the strategy used in SPAN.

When shortest path problems are solved using the Dijkstra algorithm, temporary distances from the root are assigned to all nodes not yet in the tree. Several nodes with a common temporary distance may simultaneously become eligible to be permanently labelled. In SPAN, such ties are broken by examining the nodes in the opposite order from that in which they received the common temporary distance. No other option was tested.

It is possible to develop an artificial start for SSP. This is done by introducing artificial arcs with costs of negative infinity beginning at each origin and terminating at an artificial destination. Then, SSP is applied. After a limited amount of computational testing, we concluded that the artificial start increased total solution times. We were motivated to do this testing by the fact that a procedure equivalent to the artificial start is included in the algorithm of Gribov [14].

We mentioned in the introduction that SSP is closely related to the so-called relaxation algorithm of Hung and Rom [16]. One major difference between SSP and the relaxation algorithm is that a basis for the modified assignment problem is maintained in the relaxation algorithm while we keep track of only the unit flow arcs in SSP. Maintaining a basis opens up the possibility that more than one deficient destination can receive an assignment on a given iteration. Of course, a certain amount of extra work is required to maintain the basis. In the recent computational study [5], a label correcting shortest path code was found to be the most efficient on large, sparse shortest path problems. Since the shortest path trees constructed by a label correcting must contain all nodes of the shortest path network, it seems that maintaining a basis as in the relaxation algorithm and using a label correcting shortest path code might be a good combination. We have begun an investigation of this approach.

As mentioned in section 2, the length of the sort list employed in solving the shortest path problem $SP(C^k,A^k,d^k)$ depends on the maximum arc length in the problem. Making a complete pass through the arc data to determine the maximum shortest path arc length at each iteration of SSP would be very inefficient. In SPAN we simply maintain an upper bound on the maximum shortest path arc length and use this upper bound in determining the length of the sort list or the size of the radix. If we let

$$\bar{c} = \max_{(i,j)\in E} \{c_{ij}^0\}$$

and

$$\hat{K}^k = \min_{j\in J} \{\bar{K}_j^k\}$$

then the upper bound on arc lengths for $SP(C^k,A^k,d^k)$ is $\bar{c} - \hat{K}^k$. That this is an upper bound can be deduced readily using the fact that $R_i^k \geq 0$. This upper bound is easily updated along with the accumulated node potentials $\bar{K}^k$.

6. COMPARATIVE COMPUTATIONAL TESTS

We have tested SPAN against implementations of several other algorithms. These include the codes of Weintraub and Barahona [19], Hung and Rom [16], and Barr, Glover, and Klingman [2]. The first two codes we call DUAL and RELAX, respectively. The third is known as AP-AB. The four codes tested are written in FORTRAN. In SPAN, there is a single parameter called NBUC which equals the length of the sort list. For all the tests reported in this section, NBUC was set at 200. In DUAL, a parameter called NSQR was set, as suggested in [19], to be about $\sqrt{n}$ (we recall that n is the number of origins). We did not set any other parameter values for the codes tested. All computer runs were carried out on the CDC Dual Cyber 170/750 using the FTN compiler during periods of comparable computer use. All solution times reported are exclusive of input and output. The problems used in the tests were randomly generated using NETGEN [18]. Each time reported in Tables 1-4 is the average of times for three runs on a single problem. The actual run times varied from the time reported by as much as 14% for the smallest problems tested, but such variation was generally much less.

Based on total solution times for the problems shown in Tables 1 and 2, SPAN is about 3 times faster than AP-AB and roughly 6 times faster than RELAX. The closest competing code is DUAL; however, because DUAL did not achieve optimality on some problems, we must be somewhat cautious with regard to the solution times reported. It appears that there may be some defect in the code, and this could cause solution times to increase when it is corrected. Nevertheless, we have run SPAN and DUAL on some additional problems with the results shown in Table 3. Based on the total solution time for all the problems on which DUAL achieved optimality, we conclude that SPAN is 25 to 30 percent faster than DUAL.

Although SPAN was much more efficient than RELAX for the tests on sparse problems reported in Tables 1 and 2, it is important to point out that RELAX was developed for totally dense problems. For this reason, we conducted further tests with the results shown in Table 4. These results indicate that RELAX is more efficient than SPAN on totally dense problems. Since assignment problems encountered in practice are invariably sparse, the question arises as to whether some new implementation of the relaxation algorithm might be more efficient than SPAN on sparse problems. We note that the feature of the relaxation algorithm which allows more than one deficient node to receive an assignment on a given iteration may benefit from the topology of dense problems. As we mentioned in section 5, work is underway to develop a code based on the relaxation algorithm which is designed to solve sparse problems.

Next, we compare the number and size of arrays required by the various codes. SPAN uses 2 arc length and 10 n-length arrays along with the sort list. DUAL requires 6 arc length arrays, 21 n-length arrays, and 3 arrays for which we were unable to determine the length except that it must be more than n. RELAX requires an nxn matrix and 9 n-length arrays. AP-AB requires 2 arc length and 6 n-length arrays.

For sparse problems AP-AB uses the least array space with SPAN running a close second. Both RELAX and DUAL use a lot of array space for sparse problems; however, RELAX is considerably better off when it comes to dense problems. When the arc density of a problem is about 50%, SPAN and RELAX require roughly the same amount of space. We remark that, as indicated by the results in Table 4, SPAN is more efficient than RELAX on such 50% dense problems.

TABLE 1.

SOLUTION TIMES IN SECONDS

FOR 200 x 200 ASSIGNMENT

PROBLEMS WITH COST RANGE 1-100

	NUMBER OF ARCS				
CODE	1500	2250	3000	3750	4500
SPAN	.085	.182	.159	.280	.187
DUAL	.178	did not achieve optimality	.272	.292	.342
RELAX	1.364	1.459	1.117	1.053	1.154
AP-AB	.490	.604	.631	.685	.921

TABLE 2.

SOLUTION TIMES IN SECONDS

FOR 200 x 200 ASSIGNMENT

PROBLEMS WITH COST RANGE 1-10000

	NUMBER OF ARCS				
CODE	1500	2250	3000	3750	4500
SPAN	.126	.194	.191	.265	.383
DUAL	did not achieve optimality	.273	did not achieve optimality	.433	.450
RELAX	.882	1.152	1.027	1.148	1.353
AP-AB	.513	.570	.630	.663	.945

TABLE 3.

SOLUTION TIMES IN SECONDS

FOR 300 x 300 ASSIGNMENT

PROBLEMS WITH COST RANGE 1-100

	NUMBER OF ARCS			
CODE	3000	3500	4000	4500
SPAN	.278	.279	.355	.314
DUAL	.330	.342	.477	.448

TABLE 4.

SOLUTION TIMES IN SECONDS

FOR 100 x 100 ASSIGNMENT

PROBLEMS WITH COST RANGE 1-100

	NUMBER OF ARCS			
CODE	2500	5000	7500	10000
SPAN	.083	.165	.216	.305
RELAX	.238	.257	.212	.258

7. SUMMARY AND CONCLUSIONS

Applications of minimal cost network flow problems are widespread [11], [12], [17], and new solution algorithms and implementations have stimulated further applications. A number of studies including [3], [13], and [15] have concluded that implementations based on the primal simplex algorithm are the most efficient for solving such problems. In this study, we have introduced a successive shortest path algorithm--SSP--and an implementation of this algorithm--SPAN. We have verified through computational testing that SPAN is substantially more efficient for solving assignment problems than AP-AB, which is currently the best primal simplex code for this type of problem.
SPAN was also found to be more efficient than an implementation of the Edmonds and Karp algorithm speicalized to assignment problems.

Our computational results raise the question of whether some extension of successive shortest path methods will enjoy similar success on transportation and transshipment problems. An extension of this type already exists [14], and it seems that the question will ultimately be settled through computational testing.

We have developed some new theoretical results specifically for SSP. However, it would be of interest if a more general framework could be set up which would relate our work and [7], [14], [16] to the family of simplex algorithms. Such a framework might follow along the lines of what Adolphson [1] has done in providing a theoretical basis for his so-called nondegenerate network simplex method.

ACKNOWLEDGEMENTS

The author wishes to thank Professor Darwin Klingman and Professor Fred Glover for many helpful discussions concerning network algorithms and their implementation. Also, thanks are due to Professor Andrés Weintraub and Mr. F. Barahona; Professors Ming Hung and Walter Rom; and Professors Richard Barr, Fred Glover, and Darwin Klingman for providing us with their computer codes.

Also, we wish to acknowledge the cooperation of the staff of The University of Texas Computation Center and the editorial assistance of Dr. John Mote.

Some of this research was done while the author was on sabbatical leave from Eastern Washington University. This support is sincerely acknowledged.

REFERENCES

1. D. Adolphson, "A Nondegenerate Network Simplex Method." Working paper, Graduate School of Business, University of Washington, Seattle, Washington, 1980.

2. R. Barr, F. Glover, and D. Klingman, "The Alternating Basis Algorithm for Assignment Problems." Mathematical Programming, vol. 13, 1977, 1-13.

3. G. Bradley, G. Brown, and G. Graves, "Design and Implementation of Large-Scale Primal Transshipment Algorithms." Management Science, vol. 24, no. 1, 1977, 1-34.

4. R. Dial, "Algorithm 360 Shortest Path Forest with Topological Ordering." Communications of the ACM, vol. 12, 1969, 632-633.

5. R. Dial, F. Glover, D. Karney, and D. Klingman, "A Computational Analysis of Alternative Algorithms and Labeling Techniques for Finding Shortest Path Trees." Networks, vol. 9, 1979, 215-248.

6. E. Dijkstra, "A Note on Two Problems in Connexion with Graphs." Numerical Mathematics, vol. 1, 1959, 269-271.

7. E. Dinic and M. Kronrod, "An Algorithm for the Solution of the Assignment Problem." Soviet Math. Doklady, vol. 10, no. 6, 1969.

8. S. Dreyfus, "An Appraisal of Some Shortest-Path Algorithms." Operations Research, vol. 17, 1969, 395-412.

9. J. Edmonds and R. Karp, "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems." Journal of the Association for Computing Machinery, vol. 19, 1972, 248-264.

10. J. Gilsinn and C. Witzgall, "A Performance Comparison of Labeling Algorithms for Calculating Shortest Path Trees." NBS Technical Note 772, U.S. Department of Commerce, 1973.

11. F. Glover, J. Hultz, and D. Klingman, "Improved Computer-Based Planning Techniques, Part I." Interfaces, vol. 8, no. 4, 1978, 16-24.

12. F. Glover, J. Hultz, and D. Klingman, "Improved Computer-Based Planning Techniques, Part II." Interfaces, vol. 9, no. 4, 1979, 12-20.

13. F. Glover, D. Karney, and D. Klingman, "Implementation and Computational Comparisons of Primal, Dual, and Primal-Dual Computer Codes for Minimum Cost Network Flow Problems." Networks, vol. 4, no. 3, 1974, 191-212.

14. A. Gribov, "Recursive Solution for the Transportation Problem of Linear Programming." Vestnik of Leningrad University, vol. 33, no. 19, 1978 (in Russian).

15. M. Grigoriadis and T. Hsu, "The Rutgers Minimum Cost Network Flow Subroutine." Sigmap, vol. 26, 1979, 17-18.

16. M. Hung and W. Rom, "Solving the Assignment Problem by Relaxation." To appear in Operations Research.

17. E. Johnson, "Flows in Networks." in Handbook of Operations Research: Foundations and Fundamentals. S. Elmaghraby and J. Moder, eds., Van Nostrand Reinhold Company, New York, 1978.

18. D. Klingman, A. Napier, and J. Stutz, "NETGEN--A Program for Generating Large-Scale (Un)Capacitated Assignment, Transportation, and Minimum Cost Flow Network Problems." Management Science, vol. 20, no. 5, 1974, 814-821.

19. A. Weintraub and F. Barahona, "A Dual Algorithm for the Assignment Problem." Publication No. 79/02/C, Departamento de Industrias, Universidad de Chile-Sede Occidente, Santiago, Chile, 1979.

Unclassified
Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Center for Cybernetic Studies The University of Texas at Austin	2a. REPORT SECURITY CLASSIFICATION Unclassified
	2b. GROUP

3. REPORT TITLE
(16) A Successive Shortest Path Algorithm for the Assignment Problem,

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

5. AUTHOR(S) (First name, middle initial, last name)
(10) Michael/Engquist (11) Aug 80 (12) 34

6. REPORT DATE August 1980	7a. TOTAL NO. OF PAGES 31	7b. NO. OF REFS 19
8a. ~~CONTRACT OR GRANT NO.~~ (15) N00014-75-C-0569 & N00014-80-C-0242	9a. ORIGINATOR'S REPORT NUMBER(S) (14) CCS-375	
b. ~~PROJECT NO.~~		
c. NR047-021 NR047-071	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)	
d. (9) Research rept.		

10. DISTRIBUTION STATEMENT
This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Office of Naval Research (Code 434) Washington, DC

13. ABSTRACT

In this paper a new successive shortest path (SSP) algorithm for solving the assignment problem is introduced. A computer implementation of this algorithm has been developed and a discussion of the details of this implementation is provided. Computational results are presented which show this implementation of SSP to be substantially more efficient than several recently developed codes including the best primal simplex code. Also, some new theoretical results are presented which are useful in the implementation of SSP, and it is shown that the algorithm has a computational bound of $O(n^3)$, where n is the number of origins.

DD FORM 1 NOV 65 1473 (PAGE 1) 406 777 Unclassified
S/N 0101-807-6811 Security Classification
A-31408

Unclassified
Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Networks						
Assignment problem						
Successive shortest path						
Label setting						
Label correcting						

DD FORM 1 NOV 65 1473 (BACK)
S/N 0102-014-6800

Unclassified
Security Classification

A-31409