An Implementation of a Shortest Augmenting Path Algorithm for the Assignment Problem

Jianxiu Hao and George Kocur

August 1991
Revised September 1992

Abstract

We consider the successive shortest augmenting path approach for solving the assignment problem. We propose an algorithm, in which we develop a new initialization procedure and a technique to compute the shortest augmenting paths by using a small subset of the arcs. Computational results show that our algorithm solves dense assignment problems efficiently.

1 Introduction

The *assignment problem* (AP) is defined on a bipartite graph $G(N_1, N_2, A)$. N_1 and N_2 are sets of *right* and *left* nodes, also known as the *row* and *column* nodes. A is set of arcs which connect left and right nodes. Associated with each arc $(u, v) \in A$ there is an integer-valued cost $c_{u,v}$. The objective is to assign each left node to a right node such that the total assignment cost is minimized.

Applications for the AP arise not only in real-world assignment problems, but also as subproblems in algorithms for other combinatorial optimization problems such as the quadratic assignment problem, the traveling salesman problem, crew scheduling and vehicle routing problems (See Ahuja, Magnanti and Orlin [1]).

The literature devoted to the assignment problem is very rich. Kuhn [17] developed the first algorithm for this problem, which is widely known as the *Hungarian algorithm*. The subsequent algorithms for the AP can be roughly classified into the following three basic approaches: the *primal-dual approach*, which includes Kuhn [17], Balinski and Gomory [4], Engquist [10], Bertsekas [8] and Glover et al. [12]; the *dual simplex approach*, which includes Weintraub and Barahona [20], Hung and Rom [15], Balinski [5], Goldfarb [13] and Akgul [3]; and the *primal simplex approach*, which includes Barr et al. [6], Roohy-Laleh [19], Hung [14], Akgul [2] and Orlin [18].

Ahuja, Magnanti and Orlin [1] have noted that, in general, the algorithms based on the primal-dual and dual simplex approaches have better running times in the worst case, while implementations based on the primal-dual, successive shortest augmenting path (SSAP) algorithms tend to be the most efficient approach in average case. The SSAP implementations of Glover et al. [12] and

[0] The authors wish to thank the referees for several helpful comments.

Jonker and Volgenant [16] seem to be the fastest to date. Glover et al. use a threshold shortest path algorithm while Jonker and Volgenant use a label-setting algorithm.

The SSAP solves the assignment problem in stages. Each stage is a *shortest path augmentation* which consists of two fundamental steps. The first step is to find *shortest augmenting paths* from a subset of left nodes to a subset of right nodes. Here the subset of nodes are *unmatched* or *free* nodes. The second step, often called the *flow augmentation* step, consists of matching and unmatching arcs on the shortest augmenting path(s) found in the first step. Here *an augmenting path* from a node u to another node v in the network is a simple path such that forward arcs on this path are *free* and backward arcs are *matched* . An augmenting cycle is an augmenting path such that the starting node is the same as the end node. The cost of an augmenting path (cycle) is the sum of costs of forward arcs minus the sum of costs of backwards arcs on the path (cycle). If the cost of an augmenting cycle is negative, then we call it *a negative augmenting cycle*. If the cost of an augmenting path from a node u to another node v is the minimum among all the augmenting paths from u to v, then we call it the shortest augmenting path from u to v. Note that if the direction of all backward arcs on an augmenting path is reversed, then it is a simple directed path. Hence, finding the shortest augmenting paths is equivalent to finding the shortest paths.

One of the main reasons that the SSAP algorithm for the AP is so successful in practice is effective initialization. Jonker and Volgenant [16] showed that their initialization heuristics match about 90% of nodes in a full dense network. Shortest path augmentations are only used to match the remaining 10% of the nodes.

In this study, we implement a successive shortest augmenting path algorithm. We propose a new initialization procedure. We also propose a technique to compute the shortest augmenting paths using a very small subset of the arcs, in which the shortest augmenting path of the whole network is guaranteed to be contained. When our algorithm is applied to the dense $n \times n$ assignment problem, our initialization procedure, which is simpler than others, also matches a total of about 90% of the nodes in the network. Our shortest augmenting path procedure finds the shortest augmenting paths by examining only $n \log n$ out of a total of n^2 arcs on the average. The algorithm solves the $n \times n$ dense assignment problems in a empirical running time proportional to something between n^2 and $n^2 \log n$. However, for sparse networks, our implementation is not as efficient as Jonker and Volgenant's. One of the main reasons is that our initialization is, though still efficient, not as effective as theirs. We note that an efficient algorithm that solves the dense assignment problem is very important for solving large real TSP problems using the assignment relaxation method.

2 The Initialization Phase

We present a different initialization method which consists of two steps. In step 1 *row* and *column reductions* are performed. The costs of all arcs connected to a left node or a right node are decreased (increased) by the same amount so all costs are greater than or equal to zero, and so there is at least one arc with cost exactly zero connecting each left node and each right node. Then a bipartite matching is found in the zero cost arcs. We use a bipartite matching algorithm developed by Chang and McCormick [16], which is very efficient. The overall time spent on solving the bipartite matching problems is negligible. Step 1 consists essentially of shortest path augmentations, however, all paths consist of one arc and have zero cost. Experimental results show that this matches more than 80% of the nodes in a dense network. Step 1 of the initialization phase is as follows:

Procedure Step1

1. *For each left node* $u \in N_1$*: let* $\delta_u = \min\{c_{u,v} | (u,v) \in A\}$*; set* $c_{u,v} \leftarrow c_{u,v} - \delta_u$*,* $\forall (u,v) \in A$*.*
2. *For each right node* $v \in N_2$*: let* $\delta_v = \min\{c_{u,v} | (u,v) \in A\}$*; set* $c_{u,v} \leftarrow c_{u,v} - \delta_v$*,* $\forall (u,v) \in A$*.*
3. *Find a maximum matching in* $(N_1, N_2, \bar{A})$*, where* $\bar{A} = \{(u,v) | c_{u,v} = 0 \text{ and } (u,v) \in A\}$*.*

Step 2 of the initialization phase is to make more zero cost paths from free left nodes to free right nodes by adjusting the costs of arcs. Then the matching algorithm is used to make more assignments.

The left and right nodes are partitioned into four groups: *Free* nodes (F), *Zero* nodes (Z), *Matched-Zero* nodes (MZ) and *Remaining* nodes (R). A node is in F if it is not assigned to any node. A node is in Z if it is connected to at least one node in F by a zero cost arc. A node is in MZ if it is matched to one of the nodes in Z. R contains all the remaining nodes. Note that these four sets of nodes are mutually exclusive.

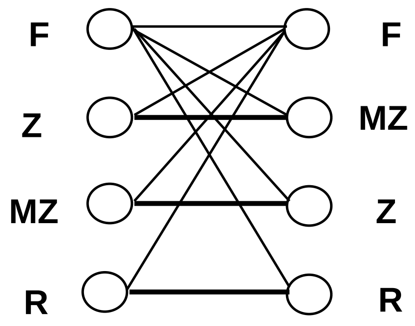

Figure 1: Node partitions

This node partition has several interesting properties:

1. The number of nodes in left F equals that in the right F and the number of nodes in left R equals that in the right R

2. The number of nodes in left Z equals that in the right MZ and the number of nodes in right Z equals that in the left MZ

3. All zero cost arcs out of left F connect to right Z and all zero cost arcs out of right F connect to left Z

4. No zero cost arcs connect left MZ to right MZ

In step 2 of the initialization, operations on MZ and F are used to further reduce costs. For each node i in left MZ, the maximum amount is subtracted from the costs of arcs connecting to i such that the new costs are greater than or equal to zero, except for the matched arc. Then the cost of all arcs connecting to the node's matched node in Z is increased by the same amount such that the new cost of the matched arc is still zero.

A similar operation is then applied to each node in right MZ.

Finally, for each node i in right F and left F, the maximum amount is subtracted from the cost of arcs connecting to i so the new costs of arcs are greater than or equal to zero. This ensures that each node in F will still have at least one zero cost arc connecting to it.

The maximum bipartite matching algorithm is then used to find more matches in the new subnetwork of zero cost arcs.

The heuristic in step 2 will probably produce some zero cost augmenting paths with length either one or three. Augmenting paths with length one will consist of new zero cost arcs connecting nodes in left F and nodes in right F.

Augmenting paths with length three will consist of new zero cost paths which indirectly connect nodes in left F with nodes in right F. These includes paths from: F-MZ-Z-F, F-Z-MZ-F and F-R-R-F. This step may also produce some zero cost augmenting paths with length greater than 3 and it is also possible that this step may not produce any new zero cost augmenting paths.

Procedure Step2

1. *For each node u in the left MZ: let v be the right node to which u is matched and* $\delta_u = \min\{c_{u,j} | (u,j) \in A, j \neq v\}$*, set* $c_{u,j} \leftarrow c_{u,j} - \delta_u$, $\forall (u,j) \in A$ *and* $c_{i,v} \leftarrow c_{i,v} + \delta_u$, $\forall (i,v) \in A$.

2. *For each node v in the right MZ: let u be the left node to which v is matched and* $\delta_v = \min\{c_{i,v} | (i,v) \in A, i \neq u\}$*, set* $c_{i,v} \leftarrow c_{i,v} - \delta_v$, $\forall (i,v) \in A$ *and* $c_{u,j} \leftarrow c_{u,j} + \delta_v$, $\forall (u,j) \in A$.

3. *For each node u in the left F: let* $\delta_u = \min\{c_{u,v} | (u,v) \in A\}$*, set* $c_{u,v} \leftarrow c_{u,v} - \delta_u$, $\forall (u,v) \in A$.

4. *For each node v in the right F: let* $\delta_v = \min\{c_{u,v} | (u,v) \in A\}$*, set* $c_{u,v} \leftarrow c_{u,v} - \delta_v$, $\forall (u,v) \in A$.

5. *Find a maximum matching in* $(N_1, N_2, \bar{A})$*, where* $\bar{A} = \{(u,v) | c_{u,v} = 0$ *and* $(u,v) \in A\}$.

Our computational experiments indicate the heuristic used in step 2 will match about another 10% of the nodes in a random dense bipartite graph, resulting in a total of 90% of the nodes being matched during the initialization stage. The following table shows the computational results on random dense assignment problems with large cost range(1-100,000).

Problem	DHK		DJV	
	After-Step1	After-Step2	After-Step1	After-Step2
128x128	25	15	47	1
256x256	49	29	93	3
512x512	98	58	190	7
1024x1024	197	117	374	18
2048x2048	389	233	755	47

Table 1. Free Node-pairs for Assign-Xhi-Degree Network

Here DHK (SHK) is our shortest augmenting path algorithm for dense (sparse) network and DJV (SJV) is the Jonker and Volgenant's algorithm for dense (sparse) network. For more detailed definitions on the algorithms and problems, see section 4.

On random dense assignment problems with small cost range, our initialization procedures perform even better; the following table is for the random dense problem instances with cost range (1-100).

Problem	DHK		DJV	
	After-Step1	After-Step2	After-Step1	After-Step2
128x128	16	9	51	10
256x256	11	7	115	27
512x512	1	0	295	96
1024x1024	0	0	731	382
2048x2048	0	0	1691	1215

Table 2. Free Node-pairs for Assign-Xhi-Degree-Lo-Cost Network

We note that the initialization procedures of Jonker and Volgenant are performing better on the random instances with large cost range than those with small cost range. This is consistent with what is reported in [16]. We also note that their procedures achieve more matches than ours. But, for dense problems with small cost range, ours are better than theirs.

On random sparse assignment problems, for both small cost range and large cost range, our initialization procedures also achieve about a total of 90% of the matches.

Problem	SHK		SJV	
Nodes(Arcs)	After-Step1	After-Step2	After-Step1	After-Step2
512(4608)	45	27	90	10
1024(10240)	91	54	185	25
2048(22528)	180	106	371	48
4096(49152)	361	213	737	102
8192(106496)	720	421	1,485	208
16384(229376)	1,424	844	2,970	429

Table 3. Free Node-pairs for Assign-Lo-Cost Network

Problem Nodes(Arcs)	SHK After-Step1	SHK After-Step2	SJV After-Step1	SJV After-Step2
512(4608)	50	30	90	0
1024(10240)	96	58	184	0
2048(22528)	195	116	367	0
4096(49152)	391	232	733	0
8192(106496)	783	463	1481	1
16384(229376)	1567	922	2957	2

Table 4. Free Node-pairs for Assign-High-Cost Network

However, the performance of our initialization procedures are not as good on the geometric-matching problems. The geometric-matching problem instances are converted from the 2-dimensional random points matching problem instances. Half the points are red and half blue, arcs connect red points to blue, and arc costs correspond to Euclidean distances. About 80% of the nodes in those problem instances get matched by the procedures.

Problem	DHK After-Step1	DHK After-Step2	DJV After-Step1	DJV After-Step2
128x128	36	26	53	11
256x256	71	51	107	22
512x512	141	101	214	47
1024x1024	282	194	424	96

Table 5. Free Node-pairs for Assign-Geometric Network

Two-cost random assignment problem is an sparse assignment problem with two costs either 100 or 100,000,000. See Section 4, for more details on how the problem instances are generated. For the two-cost random assignment problem instances, our initialization procedures solve the problem completely. In fact, procedure Step 1 of the initialization is enough.

Problem Nodes(Arcs)	SHK After-Step1	SHK After-Step2	SJV After-Step1	SJV After-Step2
512(4608)	0	0	179	44
1024(10240)	0	0	360	93
2048(22528)	0	0	740	186
4096(49152)	0	0	1514	375
8192(106496)	0	0	3089	758
16384(229376)	0	0	6274	1523

Table 6. Free Node-pairs for Assign-Two-Cost Network

The following proposition shows that Procedures Step1 and Step2 are valid.

Proposition 1. If there is a feasible solution to the AP, then the optimal assignment will not change if the costs of all arcs connecting to a left(right) node are changed by the same amount. ∎

3 Shortest Path Augmentation Phase

To solve the assignment problem by the SSAP algorithm, we introduce an artificial node s. Artificial arcs of cost zero are added connecting s to those nodes in left F which are unmatched after the initialization phase.

The SSAP algorithm for the assignment problem is a primal-dual approach in which dual feasibility is always maintained so there is no negative augmenting cycle in the solution. This means that the shortest augmenting paths could be found in a polynomial time. We use a label-correcting shortest path algorithm since it is more efficient than the label-setting algorithm for sparse networks [1]. As we have shown from our computational results on random dense networks in the last section, when the input network is dense, a sparse subnetwork could be identified for the shortest path calculations. The algorithm used in this implementation is essentially a Bellman-Ford's algorithm [7] [11] using a list array to store the candidate nodes whose out-arcs are eligible to be scanned in the next stage. This would reduce the unnecessary scan of nodes and their out-arcs if their labels are not reduced since the last scan.

Since the costs of arcs can be negative in the label-correcting shortest path algorithm, the reduced costs of arcs need not be updated after each shortest path augmentation. Instead we use the final costs from our initialization phase in the shortest path calculations. This is another advantage of the label-correcting algorithm.

To simplify proofs as well as the algorithm, we define a residual network where the directions of all matched arcs are reversed and the costs of these arcs are equal to the negative of their original values. We will also remove artificial arcs connecting to node s where the left node adjacent to the arc is no longer a free left node. The directions and costs of unmatched arcs remain unchanged. In the residual network, an augmenting path (augmenting cycle) in the original network corresponds to a directed path (directed cycle) in the residual network.

For a given dual feasible flow, shortest paths from s to all other nodes in the network are found using the shortest path algorithm. A free right node is chosen with the smallest shortest path label among all free right nodes. This shortest path (not counting the artificial arc) is called an *augmentation path*.

The augmentation path contains either one arc connecting a free left node to the free right node, or contains an alternating sequence of matched and unmatched arcs starting from a free left node and ending with a free right node. An augmentation is performed along the augmentation path by pushing one unit of flow from the free left node to the free right node along this augmentation path. Equivalently, all the unmatched arcs are changed into matched arcs and all the matched arcs are changed into unmatched arcs along the augmentation path.

We must now show that the new flow is still dual feasible after the shortest path augmentation.

Lemma 1: A shortest path augmentation on a dual feasible flow produces another dual feasible flow.

Proof. A flow is dual feasible if and only if it contains no negative augmenting cycle with respect to it. Hence showing that the new flow is dual feasible is equivalent to showing that there is no negative augmenting cycle after the shortest path augmentation.

Since we reverse the direction of an arc if it is matched, an augmenting cycle is a directed cycle. Then to show there is no negative augmenting cycle, we can show that there are no negative directed cycles.

Let π_u be the shortest path label of node u before the augmentation and $\bar{c}_{u,v} = \pi_u + c_{u,v} - \pi_v \geq 0$ (the reduced cost of arc (u,v) before the augmentation) for all arcs $(u,v) \in A$.

We know $\bar{c}_{u,v} = 0$ for all arcs (u,v) on the augmentation path. After the augmentation, the directions of arcs on the augmentation path are reversed while the directions of arcs not on the augmentation path remain unchanged. Since there were no directed negative cycles before augmentation, and since $\bar{c}_{u,v} \geq 0$ for all arcs $(u,v) \in A$, there is no negative directed cycle with respect to $\bar{c}$ after the augmentation. ∎

Before continuing to the shortest path augmentation procedure that produces an optimal assignment, we prove another lemma:

Lemma 2: Let π^i and π^{i+1} be the shortest path labels of stage i and stage $i+1$, then $\pi_t^{i+1} \geq \pi_t^i$ for all nodes t in the network.

Proof. There are two cases:

1. The shortest path from s to t in stage $i+1$ contains no arcs which are in the augmentation path of stage i.
2. The shortest path from s to t in stage $i+1$ contains some arcs which are in the augmentation path of stage i.

In case 1, $\pi_t^{i+1} \geq \pi_t^i$ since the shortest augmenting path in stage $i+1$ is also a valid augmenting path in stage i.

In case 2, let $\ell^i(u,v)$ and $\ell^{i+1}(u,v)$ denote the shortest path cost from node u to node v in stages i and $i+1$, respectively. Clearly $\ell^i(s,t) = \pi_t^i$. Let the shortest augmenting path from s to t in stage $i+1$ be $s, ..., v_1, u_1, ..., v_2, u_2, ..., v_p, u_p, ..., t$, where (u_k, v_k), $k = 1, 2, ...p$ is an arc on the augmentation path in stage i. We first show that $\ell^i(s,u_p) \leq \ell^{i+1}(s,u_p)$ by induction.

Since arc (u_1, v_1) is on the shortest augmentation path in stage i, we have $\ell^i(s,u_1) \leq \ell^i(s,v_1) - c_{u_1,v_1} = \ell^{i+1}(s,u_1)$. Suppose $\ell^i(s,u_{p-1}) \leq \ell^{i+1}(s,u_{p-1})$. Then:

$$\begin{aligned}
\ell^i(s,u_p) &\leq \ell^i(s,v_p) - c_{u_p,v_p} \\
&\leq \ell^i(s,u_{p-1}) + \ell^i(u_{p-1},v_p) - c_{u_p,v_p} \\
&\leq \ell^{i+1}(s,u_{p-1}) + \ell^{i+1}(u_{p-1},v_p) - c_{u_p,v_p} \\
&= \ell^{i+1}(s,u_p)
\end{aligned}$$

The first inequality follows from the fact that arc (u_p, v_p) is on the shortest augmentation path at stage i, the second inequality follows from the triangle inequality for shortest path labels, and the last inequality follows from our induction assumption. Hence $\pi_t^i = \ell^i(s,t) \leq \ell^i(s,u_p) + \ell^i(u_p,t) \leq \ell^{i+1}(s,u_p) + \ell^{i+1}(u_p,t)$ $= \ell^{i+1}(s,t) = \pi_t^{i+1}$. ∎

Procedure Short-Path-Augmentation (SPA)

Input: A partial matching such that all matched arcs' costs are equal to zero and unmatched arcs' costs are greater than or equal to zero.

Output: An optimal assignment.

1. *Define threshold costs thresh[1], thresh[2], thresh[3] and thresh[4], where thresh[4] is set to infinity. Construct an initial subnetwork* $\bar{G} = (N_1, N_2, \bar{A})$ *of good arcs with* $\bar{A} = \{(u,v) \in A | \pi_u^o + c_{u,v} \leq thresh[1]\}$*, where* π^o *is the set of initial shortest path labels. Add an artificial node s and arcs from s to the free left nodes to* $\bar{G}$ *and set the costs of the artificial arcs to zero. Reverse the directions of matched arcs. Set* $thresh\_label[v] \leftarrow thresh[1]$ *for all nodes* $v \in N_2$*. Set stage* $i = 0$*;*

2. *Find shortest paths from s to all nodes in the network* $(N_1, N_2, \bar{A})$ *and let* π^i *be the shortest path labels. If* $\pi_v^i \leq thresh\_label[v]$ *for all* $v \in N_2$ *go to (4). Otherwise go to (3).*

3. *For each right node v such that* $\pi_v^i > thresh\_label[v]$*, set* $thresh\_label[v] \leftarrow thresh[k]$ *such that* $thresh[k] > \pi_v^i$ *and* $\bar{A} \leftarrow \bar{A} \cup \{(u,v)\}$ *for all* $(u,v) \in A$ *and* $\pi_u^i + c_{u,v} \leq thresh\_label[v]$*. Go to (2).*

4. *Let* $t \in N_2$ *be the right node with the minimum shortest path label among all the free right nodes and let P be the shortest path from s to t. For all arcs a in P, if a is marked matched, mark it unmatched; if a is marked unmatched, mark it matched. Reverse the direction of arc a and set the cost* $c_{u,v} = -c_{u,v}$ *for arc a. If all nodes are matched, stop. Otherwise set* $i \leftarrow i + 1$ *and go to (2).*

Theorem 1: Procedure SPA produces an optimal assignment.

Proof. Lemma 1 shows that the flow is always dual feasible if the augmentation is made along a shortest augmenting path. Then when all nodes are matched, we have a solution which is both primal and dual feasible. Therefore, we only need to show that the shortest augmentation path produced using the good arcs is indeed the shortest augmentation path in the original network.

For this purpose we need to show that those arcs which are not in the set of good arcs will not be on the shortest augmenting paths immediately before the execution of step (4). In other words, $\pi_u^h + c_{u,v} \geq \pi_v^h$ at stage h for all arcs (u,v) not in the set of good arcs.

If the last time that arc (u,v) is examined (in step (1) or step (3)) was in stage j with $j \leq h$ and $\pi_u^j + c_{u,v} > thresh\_label[v]$, then this arc (u,v) is not in the set of good arcs at stage h. We have $\pi_u^h + c_{u,v} \geq \pi_u^j + c_{u,v} > thresh\_label[v] \geq \pi_v^h$, where the first inequality follows from Lemma 2 and last inequality follows from step (2) of the procedure SPA. Hence, all arcs not in the set of good arcs at stage h will not be part of the shortest augmentation path at stage h. ∎

The threshold costs are chosen in the following way. Let the cost vector c be the final cost vector from the initialization phase. For each free left node the minimum cost of its out arcs is calculated. Let c_{max} be the maximum of these minimum cost values and c_{min} be the minimum. Then $thresh[1]$ is set to $c_{min} + c_{max}/h$, $thresh[2]$ to $thresh[1] \times 2$, $thresh[3]$ to $thresh[2] \times 2$, ..., and and *thresh*[*k*] is set to infinity. In this implementation, we chose h as 6 and k as 4. *thresh*[1] is the most important parameter. If *thresh*[1] is too big, unnecessary arcs are put in the set of good arcs. If *thresh*[1] is too small, many corrections in the set of good arcs will be needed later.

This set of threshold costs may not be the best in all cases, but it seems to be effective according to our computational results. For random dense assignment problem instances, shortest path augmentations are performed only on a very small subset of arcs. The following table shows that only about $n \log n$ out of a total of n^2 arcs are used on the average.

Problem	End-of-Step1	End-of-Step2	End-of-SPA
128x128	175	231	1,022
256x256	350	472	2,292
512x512	703	954	5,256
1024x1024	1,404	1,905	10,445
2048x2048	2,828	3,839	22,875

Table 7. Good Arcs in DHK for Assign-Xhi-Degree Network

We also note that the threshold labels used here are different from what are used by Glover et al. [12]. In their algorithm, they use threshold labels to choose the eligible nodes to be scanned in their shortest path algorithm. In our algorithm, all the nodes are eligible to be scanned but only a portion of arcs out of each node are eligible to be scanned, which are determined by their reduced costs and the threshold labels.

Since the label of a left node can be obtained from the label of its matched right node, we need only keep track of the labels of right nodes. Furthermore, if the label of a node is not affected by the augmentation path, then its label need not be updated in the next shortest path calculation. For this purpose, we need to keep track of the nodes whose labels are not going to change after an augmentation.

We do not calculate π^o in step (1) since this is expensive. Instead we set $\pi^o_u = 0$ for all $u \in N_1$ for the purpose of selecting the initial set of good arcs. Since the costs of all unmatched arcs are greater than or equal to zero, and the costs of all matched arcs are equal to zero with respect to the final costs of the initialization phase, $\pi^o_u \geq 0$ for all $u \in N_1$. Hence, setting $\pi^o_u = 0$ for all $u \in N_1$ does not affect the correctness of procedure SPA nor of Theorem 1.

We note that the subnetwork of good arcs generated for the random geometric matching problem instances are not so sparse (Table 8). We also note that the idea of good arcs do not apply well for the problems which are already very sparse(Table 9).

Problem	End-of-Step1	End-of-Step2	End-of-SPA
128x128	182	267	5286
256x256	364	547	17431
512x512	728	1142	72968
1024x1024	1454	2375	226337
1024x1024	1461	2368	247293

Table 8. Good Arcs in DHK for Assign-Geometric Network

Nodes(Arcs)	End-of-Step1	End-of-Step2	End-of-SPA
512(4608)	349	384	4,608
1024(10240)	700	767	10,240
2048(22528)	1,396	1,533	22,528
4096(49152)	2,800	3,069	49,152
8192(106496)	5,599	6,135	106,496
16384(229376)	11,205	12,280	229,376

Table 9. Good Arcs in SHK for Assign-High-Cost Network

4 Computational Results

Our computational results are based on averages of over 30 runs for each problem instance. The random problem instances are generated by DIMACS *assign.c* and *dcube.c*. We have run our algorithm on an HP-9000/720 work-station and the CPU time do not include problem setup or input and output time.

All algorithms were coded in C and were compiled with the default optimization. Arrays inside the program are dynamically allocated using malloc.

4.1 Algorithms:

DHK: Algorithm presented in this paper using matrix format to store network data. This algorithm is for dense assignment problems.

SHK: Algorithm presented in this paper using linked-list format to store sparse network data. This algorithm is for sparse assignment problems.

DJV: Jonker and Volgenant's algorithm using matrix format to store network data. This algorithm is for dense assignment problems.

SJV: Jonker and Volgenant's algorithm using linked-list format to store sparse network data. This algorithm is for sparse assignment problems.

4.2 Assignment Problems:

This describes the testbed instances for assignment problem instances tested in this experiment.

1. **Assign-Xhi-Degree.** Use `assign.c` with the following commands for problem size $N = 256, 512, 1024, 2048, 4096$.

   nodes (N)

   sources (N/2)

   complete

   maxcost 100000

   seed (X)

2. **Assign-Xhi-Degree-Lo-Cost.** Use `assign.c` with the following commands for problem size $N = 256, 512, 1024, 2048, 4096$.

   nodes (N)

   sources (N/2)

   complete

   maxcost 100

   seed (X)

3. **Assign-Lo-Cost.** Use `assign.c` with the following commands for problem size $N = 512, 1024, 2048, 4096, 8192, 16384$.

   nodes (N)

   sources (N/2)

   degree (2 log_2 N)

   maxcost 100

   seed (X)

4. **Assign-Hi-Cost.** Use `assign.c` with the following commands for problem size $N = 512, 1024, 2048, 4096, 8192, 16384$.

   nodes (N)

   sources (N/2)

   degree (2 log_2 N)

   maxcost 100000000

5. **Assign-Two-Cost.** Use the instances generated for Assign-Lo-Cost for problem size $N = 512, 1024, 2048, 4096, 8192, 16384$. A graph `g.asn` is transformed using the "awk" command `awk -f twocost.a <g.asn >gt.asn`. The resulting graphs have all costs either 100 or 100000000. These instances are hard for auction algorithms.

6. **Assign-Geometric.** Use `dcube.c` with the following commands to generate random points in 2-space with coordinates in the range $1..10^6$.

   nodes (N)

   dimension 2

   maxloc 1000000

   seed (X)

   The resulting graph `g.geom` can be converted to an assignment problem with `awk -f geomasn.a <g.geom >g.asn`. Half the points are red and half blue, arcs connect red points to blue, and arc costs correspond to Euclidean distances.

4.3 Results of the experiments

In all the tables, the column "Init" indicates the average CPU time (in milliseconds) in initialization stage. Column "SPA" indicates the average CPU time spent on shortest path augmentations. Column "DHK/DJV"("SHK/SJV") indicates the time ratio of algorithm DHK(SHK) to DJV(SJV).

Problem	DHK			DJV			DHK/DJV
	Init	SPA	Total	Init	SPA	Total	
128x128	64	46	110	128	7	134	0.82
256x256	258	162	420	359	56	416	1.00
512x512	1,218	612	1,880	1,450	526	1,976	0.95
1024x1024	5,679	2,336	8,015	5,525	4,161	9,686	0.82
2048x2048	25,193	9,959	35,153	21,696	29,000	50,696	0.69

Table 10. CPU time (in ms) for Assign-Xhi-Degree network

Problem	DHK			DJV			DHK/DJV
	Init	SPA	Total	Init	SPA	Total	
128x128	46	48	95	31	24	55	1.72
256x256	143	146	289	125	134	259	1.11
512x512	493	729	1,222	660	578	1,238	0.98
1024x1024	1,915	0	1,915	3,203	1,972	5174	0.37
2048x2048	7,666	0	7,666	16,196	11,562	27,759	0.27

Table 11. CPU time (in ms) for Assign-Xhi-Degree-Lo-Cost network

Problem	SHK			SJV			SHK/SJV
Nodes(Arcs)	Init	SPA	Total	Init	SPA	Total	
512(4608)	12	31	44	4	18	22	2.00
1024(11264)	43	188	232	22	92	114	2.03
2048(22528)	115	689	805	62	366	428	1.88
4096(49152)	285	2,439	2,725	158	1,461	1,620	1.68
8192(106496)	669	8,117	8,786	364	5,518	5,882	1.49
16384(229376)	1,515	28,284	29,799	823	23,309	24,132	1.23

Table 12. CPU time (in ms) for Assign-Lo-Cost network

Problem	SHK			SJV			SHK/SJV
Nodes(Arcs)	Init	SPA	Total	Init	SPA	Total	
512(4608)	11	37	49	27	1	29	1.68
1024(10240)	42	186	228	133	1	134	1.70
2048(22528)	129	838	967	523	11	535	1.80
4096(49152)	318	2,988	3,306	1,437	56	1,493	2.21
8192(106496)	724	11,514	12,239	4,956	1,782	6,739	1.81
16384(229376)	1,712	41,676	43,399	13,535	13,502	27,038	1.60

Table 13. CPU time (in ms) for Assign-Hi-Cost network

Problem Nodes(Arcs)	SHK Init	SHK SPA	SHK Total	SJV Init	SJV SPA	SJV Total	SHK/SJV
512(4608)	22	0	23	2	9	11	2.09
1024(10240)	57	0	58	16	39	56	1.03
2048(22528)	136	0	136	56	132	188	0.72
4096(49152)	320	0	320	146	500	647	0.49
8192(106496)	710	0	710	330	1,873	2,204	0.32
16384(229376)	1,671	0	,1671	744	7,192	7,937	0.21

Table 14. CPU time (in ms) for Assign-Two-Cost network

Problem	DHK Init	DHK SPA	DHK Total	DJV Init	DJV SPA	DJV Total	DHK/ DJV
128x128	377	402	779	1,469	322	1,790	0.43
256x256	1,571	2,025	3,597	4,663	2,103	6,766	0.53
512x512	6,876	14,531	21,407	15,990	15,957	31,947	0.67
1024x1024	29,704	73,935	103,639	51,609	89,885	141,449	0.73

Table 15. CPU time (in ms) for Assign-Geometric network

4.4 Discussion:

We have considered the successive shortest augmenting path algorithm for solving the assignment problem. We have proposed an algorithm, in which we developed a new initialization procedure and a technique to compute the shortest augmenting paths by using a small subset of the arcs. This approach is relatively efficient for dense assignment problems as we could see from the running times shown in Tables 10 and 11. This algorithm solves the dense assignment problem in an empirical running time proportional to something between n^2 and $n^2 \log n$ as shown in the following table.

Problem	CPU	CPU$\times 10^3/(n^2)$	CUP$\times 10^4/(n^2 \log n)$
128x128	110	6.47	9.58
256x256	420	6.41	8.01
512x512	1,880	7.17	7.96
1024x1024	8,015	7.64	7.64
2048x2048	35,153	8.38	7.62

Table 16. Algorithm DHK running time analysis on Assign-Xhi-Degree network

However, our algorithm for sparse random networks(SHK) is not as efficient as Jonker and Volgenant's(SJV) as shown in Tables 12 and 13. But the performance of our algorithm improves as the problem size increases.

Our initialization procedure matches about 90% of the nodes for both dense and sparse networks. We note that our procedure performs better than Jonker and Volgenant's on random problems with small cost range while theirs performs better on those with large cost range.

References

[1] Ahuja, R. K., T. L. Magnanti, and J. B. Orlin. 1988. Network Flows, in *Handbooks in Operations Research and Management Science, Vol 1. Optimization.*

[2] Akgul, M. 1985. A genuinely polynomial simplex algorithm for the assignment problem, Research Report, Department of Computer Science and Operations Research, North Carolina State University, Raleigh, NC.

[3] Akgul, M. 1988. A Sequential Dual Simplex Algorithm for the Linear Assignment Problem. *Operations Research Letters* 7, 155-158.

[4] Balinski, M. L. and R. E. Gomory. 1964. A Primal Method for the Assignment and Transportation Problem. *Management Programming* 13, 1-13.

[5] Balinski, M. L. 1986. A Competitive (Dual) Simplex Method for the Assignment and Transportation Problem. *Mathematical Programming* 34, 125-141.

[6] Barr, R., F. Glover, and D. Klingman. 1977. The Alternating path basis algorithm for the assignment problem. *Math. Prog.* 12, 1 -13.

[7] Bellman, R. 1958. On a Route Problem. *Quart. of Appl. Math.16,* 87-90.

[8] Bertsekas, D. 1981. A New Algorithm for the Assignment Problem. *Mathematical Programming* 21, 152-157.

[9] Chang, S.F. and S.T. McCormick 1989. A fast implementation of a bipartite matching algorithm. Technical Report, Columbia University, New York.

[10] Engquist, M. 1982. A Successive Shortest Path Algorithm for the Assignment Problem. *INFOR* 20, 370-384.

[11] Ford, L. R. 1956. *Network Flow Theory.* The Rand Corporation Report P-923, Santa Monica, CA.

[12] Glover, F., R. Glover, and D. Klingman. 1986. The Threshold assignment algorithm. *Math. Prog. Study* 26, 12-37.

[13] Goldfarb, D. 1985. Efficient Dual Simplex Algorithms for the Assignment Problem. *Math. Prog.* 33, 187-203.

[14] Hung, M. S. 1983. A Polynomial Simplex Method for The Assignment Problem. *Oper. Res.* 31,595-600.

[15] Hung, M. S. and W. O. Rom. 1980. Solving the assignment problem by relaxation. *Oper. Res.* 28, 969-892.

[16] Jonker, R. and A. Volgenant. 1987. A Shortest Augmenting Path Algorithm for Dense and Sparse Linear Assignment Problems. *Computing* 38, 325-340.

[17] Kuhn, H. W. 1955. The Hungarian Method for Assignment Problem. *Naval Research Logistics Quarterly* 2, 83-97.

[18] Orlin, J. B. 1985. On the Simplex Algorithm for Networks and Generalized Networks. *Mathematical Programming Study* 25, 166-178.

[19] Roohy-Laleh, E. 1980. Improvements to the theoretical efficiency of the network simplex method, Ph.D. Thesis, Carleton University, Ottawa, Canada.

[20] Weintraub, A., and F. Barahona. 1979. A dual algorithm for the assignment problem. Departmente de Industrias Report No. 2, Universidad de Chile-Sede Occidente.

GTE Laboratories Incorporated, 40 Sylvan Road, Waltham, MA 02254
E-mail: jh04@gte.com for Jianxiu Hao and *gak0@gte.com* for George Kocur.