Mathematical Programming 54 (1992) 41-56
North-Holland

New scaling algorithms for the assignment and minimum mean cycle problems

James B. Orlin and Ravindra K. Ahuja[*]

Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA 02139, USA

Received 31 May 1988
Revised manuscript received July 1989

In this paper we suggest new scaling algorithms for the assignment and minimum mean cycle problems. Our assignment algorithm is based on applying scaling to a hybrid version of the recent *auction* algorithm of Bertsekas and the successive shortest path algorithm. The algorithm proceeds by relaxing the optimality conditions, and the amount of relaxation is successively reduced to zero. On a network with $2n$ nodes, m arcs, and integer arc costs bounded by C, the algorithm runs in $O(\sqrt{n}\, m \log(nC))$ time and uses very simple data structures. This time bound is comparable to the time taken by Gabow and Tarjan's scaling algorithm, and is better than all other time bounds under the *similarity assumption*, i.e., $C = O(n^k)$ for some k. We next consider the minimum mean cycle problem. The *mean cost* of a cycle is defined as the cost of the cycle divided by the number of arcs it contains. The *minimum mean cycle problem* is to identify a cycle whose mean cost is minimum. We show that by using ideas of the assignment algorithm in an approximate binary search procedure, the minimum mean cycle problem can also be solved in $O(\sqrt{n}\, m \log nC)$ time. Under the similarity assumption, this is the best available time bound to solve the minimum mean cycle problem.

Key words: Minimum cycle mean, minimum mean cycle, assignment problem, bipartite matching, shortest path problem, scaling.

1. Introduction

In this paper we propose new scaling algorithms for the assignment problem and the minimum mean cycle problem. The (*linear*) *assignment* problem consists of a set N_1 denoting persons, another set N_2 denoting jobs for which $|N_1| = |N_2|$, a collection of pairs $A \subseteq N_1 \times N_2$ representing possible person to job assignments, and an *integer* cost c_{ij} associated with each element (i, j) in A. The objective is to assign each person to exactly one job and minimize the cost of the assignment. This problem can be stated as the following linear program:

$$\text{Minimize} \quad \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{1a}$$

$$\text{subject to} \quad \sum_{\{j:(i,j)\in A\}} x_{ij} = 1 \quad \text{for all } i \in N_1, \tag{1b}$$

$$\sum_{\{i:(i,j)\in A\}} x_{ij} = 1 \quad \text{for all } j \in N_2, \tag{1c}$$

$$x_{ij} \in \{0, 1\} \quad \text{for all } (i, j) \in A. \tag{1d}$$

[*] Now at the Department of Industrial and Management Engineering, Indian Institute of Technology, Kanpur 208 016, India.

The assignment problem can be considered as a network optimization problem on the graph $G=(N, A)$ consisting of the node set $N = N_1 \cup N_2$ and the arc set A. In the network context, this problem is also known as the *weighted bipartite matching* problem. We shall assume without any loss of generality that $c_{ij} \geqslant 0$ for all $(i, j) \in A$, since a suitably large constant can be added to all arc costs without changing the optimum solution. Let $C = \max\{c_{ij}\} + 1$. We also assume that the assignment problem has a feasible solution. The infeasibility of the assignment problem can be detected in $O(\sqrt{n}\, m)$ time by the bipartite cardinality matching algorithm of Hopcroft and Karp (1973), or by using the maximum flow algorithms of Even and Tarjan (1975) and Orlin and Ahuja (1987).

Instances of the assignment problem arise not only in practical settings, but also are encountered as a subproblem in algorithms for hard combinatorial optimization problems such as the quadratic assignment problem (Francis and White, 1974), the traveling salesman problem (Karp, 1977), crew scheduling and vehicle routing problems (Bodin et al., 1983). In view of these applications, the development of faster algorithms for the assignment problem deserves special attention.

The literature devoted to the assignment problem is very rich. There exist a variety of algorithms to solve the assignment problem and we refer the reader to the paper of Ahuja et al. (1989) for a detailed survey of these algorithms. The successive shortest path algorithm is one of the more popular algorithms to solve the assignment problem. This algorithm solves the shortest path problem as a sequence of n shortest path problems each with nonnegative arc lengths. If we use Fredman and Tarjan's (1984) implementation of Dijkstra's algorithm to solve these shortest path problems, then this assignment algorithm runs in $O(nm + n^2 \log n)$ time. This is the best strongly polynomial bound to solve the assignment problem. (A time bound is called strongly polynomial if it is polynomial in the dimension of the problem, i.e., it is polynomial in n and m). The best (weakly) polynomial algorithm to solve the assignment is due to Gabow and Tarjan (1988) which runs in $O(\sqrt{n}\, m \log(nC))$ time. This algorithm uses a cost-scaling technique, based on the approach of Goldberg and Tarjan (1987) for the minimum cost flow problem. Under the similarity assumption, i.e., that $C = O(n^k)$ for some k, this algorithm runs in $O(\sqrt{n}\, m \log n)$ time, and dominates the $O(nm + n^2 \log n)$ bound for all problem classes. Further, the scaling algorithm uses far simpler data structures.

Bertsekas (1979, 1987) developed a new approach for the assignment problem, called the *auction algorithm*, which assigns jobs to persons using *auction*. The original version of this algorithm (Bertsekas, 1979) ran in pseudo-polynomial time, but by incorporating scaling in this method, Bertsekas and Eckstein (1988) obtained an $O(nm \log(nC))$ algorithm. Their computational results find the auction algorithm to be substantially faster than the best other method for the assignment problem for several classes of networks.

In this paper we suggest a hybrid version of the auction and successive shortest path algorithms running in time $O(\sqrt{n}\, m \log(nC))$ which substantially improves the running times obtained by using either technique alone. In fact, by combining these two approaches in a rather natural way, we obtain a time bound that is comparable to the best available time bound. This algorithm is a cost scaling algorithm, and successively obtains assignments that are closer and closer to an optimum solution. The algorithm performs $O(\log(nC))$ scaling phases, and computations within each phase can be decomposed into two parts: the first part applies the auction algorithm, and the second part applies the successive shortest path algorithm. The auction algorithm starts with a null assignment and assigns all but at most $\sqrt{n}$ nodes; these unassigned nodes are subsequently assigned by the scuccessive shortest path algorithm. We show that each of these parts takes $O(\sqrt{n}\, m)$ time and, consequently, this approach runs in $O(\sqrt{n}\, m \log(nC))$ time, which is comparable to the best other polynomial bound to solve the assignment problem under the assumption of similarity.

We anticipate that the running time of our hybrid algorithm will be comparable in practice to the running time of the auction algorithm. The reason is relatively straightforward. The second phase of our algorithm "kicks in" at a point in which the auction algorithm is doing very poorly, and in particular it kicks in after $O(\sqrt{n})$ "bidding cycles". In recent testing on the auction on the auction algorithm (Bertsekas and Castanon, 1989) for dense large scale problems (with more than 20 000 nodes), over 99% of the nodes are assigned after only two "bidding cycles" of the auction algorithm. Although the percentage of unassigned nodes increases for sparse problems, we anticipate that the almost all of the nodes will be assigned within $\sqrt{n}$ "bidding cycles". Thus, the hybrid algorithm will perform essentially the same steps as the auction algorithm, except for possibly a few successive augmenting path iterations at the end.

Hence the contribution of our paper is to develop an algorithm that obtains the best running time in theory and has an attractive empirical performance. Our hybrid algorithm is a cost-scaling algorithm, and in that sense is similar to the algorithm of Gabow and Tarjan (1988). Computationally, our algorithm is quite different, except possibly in the second phase of our algorithm. Gabow and Tarjan's algorithm uses a primal-dual approach within each scaling phase. Their algorithm is essentially an efficient implementation of a successive approximate shortest path computation. The second phase of our algorithm relies on a sequence of successive approximate shortest path computations which is similar to Gabow and Tarjan's algorithm.

We also use a variant of our assignment algorithm to solve the minimum mean cycle problem efficiently. The *mean cost* of a directed cycle W is defined as $\sum_{(i,j)\in W} c_{ij}/|W|$. The *minimum mean cycle* of a network is a directed cycle whose mean cost is minimum. (The minimum mean cycle problem is sometimes referred to as the minimum cycle mean problem, but we prefer the former name.) Some applications of the minimum mean cycle problem can be found in periodic optimization (Orlin, 1981), and in the minimum cost flow algorithm of Goldberg and Tarjan (1988). The minimum mean cycle problem can be solved in $O(nm)$ time by the algorithm of Karp (1978), or in $O(nm \log n)$ time by the algorithm of Karp and Orlin (1981). The minimum mean cycle problem is a special case of the minimum cost-to-time ratio cycle problem, and by using binary search can be solved as a sequence of $O(\log(nC))$ shortest path problems with arbitrary arc lengths (see Lawler, 1976). Since any assignment algorithm can be used to solve the shortest path problem with arbitrary arc lengths (see, e.g., Ahuja et al., 1989), this approach gives an $O(\sqrt{n}\, m \log^2(nC))$ algorithm for the minimum mean cycle problem. We show that the scaling algorithm for the assignment problem can be used in "approximate binary search" to obtain an $O(\sqrt{n}\, m \log(nC))$ algorithm for the minimum mean cycle problem, i.e., the same running time as for the assignment problem. The subroutine for the approximate binary search can be either our hybrid algorithm for the assignment problem, or the algorithm developed by Gabow and Tarjan (1988).

The idea of approximate binary search is due to Zemel (1987). Typically, in the (exact) binary search algorithm one knows that the optimum objective function value λ lies in a search interval $[L, U]$. Also, for a given value x, there is a method P for determining whether $\lambda \leqslant x$ or $\lambda \geqslant x$. In binary search, one guesses the value $x = \frac{1}{2}(U - L)$, applies method P and subsequently reduces the length of the interval $[L, U]$ by a factor of 2. In the case of the minimum mean cycle problem, the method P relies on the solution to an assignment problem and binary search consists of solving $O(\log(nC))$ assignment problems. However, in the approximate binary search algorithm, we need a method P' for determining whether $\lambda \leqslant x + \varepsilon$ or $\lambda \geqslant x - \varepsilon$. We show that an application of the cost scaling phase in the hybrid assignment algorithm constitutes the method P'. We further show that if we use $\varepsilon = \frac{1}{4}(U - L)$, then in each iteration we reduce the length of the search interval by 25%. Thus the number of iterations increase by a constant factor over the exact binary search but the computational time within each iteration is improved by a factor of $O(\log(nC))$. Consequently, this approach obtains an improvement of $O(\log(nC))$ in the running time over the exact binary search.

Although the ideas underlying the approximate binary search are quite general, the success of approximate binary search relies on determining an approximate method P' which is asymptotically faster than the exact method P. In fact, our primary contribution is to show that each iteration of our approximate binary search method is asymptotically faster than that in the exact binary search. In general, one cannot expect approximate binary search to be more than a constant times faster than exact binary search, even when P relies on scaling. For example, it is still an open question as to whether the minimum cost-to-time ratio cycle problem can be solved faster than $O(\sqrt{n}\, m \log^2(nC))$ time, even though it too can be solved using binary search as a sequence of assignment problems.

2. Exact and approximate optimality

A 0-1 solution x of (1) is called an *assignment.* If $x_{ij} = 1$ then i is *assigned* to j and j is *assigned* to i. A 0-1 solution x for which $\sum_{\{j:(i,j)\in A\}} x_{ij} \leqslant 1$ for all $i \in N_1$ and $\sum_{\{i:(i,j)\in A\}} x_{ij} \leqslant 1$ for all $j \in N_2$ is called a *partial assignment.* In other words, in a partial assignment, some nodes may be *unassigned*. Associated with any partial assignment x is an index set X defined as $X=\{(i,j)\in A\colon x_{ij}=1\}$. We associate the dual variable $\pi(i)$ with the equation corresponding to node i in (1b) and the dual variable $-\pi(j)$ with the equation corresponding to node j in (1c). We subsequently refer to the $2n$-vector π as the vector of *node potentials*. We define the *reduced cost* of any arc $(i,j)\in A$ as $\bar{c}_{ij}=c_{ij}-\pi(i)+\pi(j)$. It follows from the linear programming duality theory that a partial assignment assignment x is optimum for (1) if there exist node potentials π so that the following (*exact*) *optimality conditions* are satisfied:

(C1) (*Primal feasibility*) x is an assignment.

(C2) (*Dual feasibility*) (a) $\bar{c}_{ij}=0$ for all $(i,j)\in X$; and
(b) $\bar{c}_{ij}\geqslant 0$ for all $(i,j)\in A$.

The concept of *approximate optimality* as developed by Bertsekas (1979) and Tardos (1985) is essential to our algorithms. This is a relaxation of the exact optimality conditions given above. A partial assignment x is said to be *ε-optimal* for some $\varepsilon\geqslant 0$ if there exist node potentials π so that the following *ε-dual feasibility conditions* are satisfied:

(C3) (*ε-Dual feasibility*) (a) $\bar{c}_{ij}\leqslant\varepsilon$ for all $(i,j)\in X$; and
(b) $\bar{c}_{ij}\geqslant -\varepsilon$ for all $(i,j)\in A$.

We associate with each partial assignment x a *residual network* $G(x)$. The residual network $G(x)$ has N as the node set, and the arc set consists of A plus an arc (j,i) of cost $-c_{ij}$ for every arc (i,j) for which $x_{ij}=1$. Observe that $\bar{c}_{ij}=-\bar{c}_{ji}$ whenever both (i,j) and (j,i) are in the residual network.

The above conditions can be slightly simplified if presented in terms of the residual network. Observe that for all $(i,j)\in X$, (j,i) is also in the residual network $G(x)$, and $\bar{c}_{ij}\leqslant\varepsilon$ is equivalent to $\bar{c}_{ji}=-\bar{c}_{ij}\geqslant-\varepsilon$. Hence, with respect to the residual network, the condition (C3) is equivalent to the following condition:

(C4) (*ε-Dual feasibility*) $\bar{c}_{ij}\geqslant-\varepsilon$ and for all (i,j) in $G(x)$.

Clearly, for $\varepsilon=0$, the condition (C4) reduces to (C2) and hence any ε-optimal assignment is an optimum assignment. In fact, when costs are integer, then ε need not necessarily be equal to 0 for (C4) to become equivalent to (C2) as proved by Bertsekas (1979).

Lemma 1. *Any assignment is ε-optimal for $\varepsilon>C$ and any ε-optimal assignment with $\varepsilon<1/(2n)$ is an optimum assignment.*

Proof. See Bertsekas and Eckstein (1988) or Ahuja et al. (1989). □

3. The assignment algorithm

Our algorithm for the assignment problem proceeds by obtaining ε-optimal assignments for successively smaller values of ε. In other words, the algorithm obtains solutions that are closer and closer to being optimum until $\varepsilon < 1/(2n)$, at which point the solution is optimum. The algorithm performs a number of *cost scaling phases*. In a cost scaling phase, the major subroutine of the algorithm is the *improve-approximation* procedure, which starts with a $k\varepsilon$-optimal assignment and produces an ε-optimal assignment. The parameter k is a constant and will typically have values 2, 3 or 4. A high-level description of the scaling algorithm is given below.

```
algorithm assignment;
begin
    set $\pi := 0$ and $\varepsilon := C$;
    $L := 2(k+1)\lceil\sqrt{n}\,\rceil$
    while $\varepsilon \geqslant 1/(2n)$ do
    begin
        $\varepsilon := \varepsilon/k$;
        improve-approximation($c$, $k$, $\varepsilon$, $L$, $\pi$, $x$);
    end;
    the solution $x$ is an optimum assignment;
end;
```

```
procedure improve-approximation($c$, $k$, $\varepsilon$, $L$, $\pi$, $x$);
begin
    auction;
    successive-shortest-path;
end;
```

Clearly, the procedure *improve-approximation* is executed $O(\log(nC))$ times because initially $\varepsilon = C$, finally $\varepsilon < 1/(2n)$, and in each scaling phase ε decreases by a factor of k, which is at least 2. The *improve-approximation* procedure consists of two subprocedures: *auction* and *successive-shortest-path*. The procedure *auction* starts with a $k\varepsilon$-optimum assignment. It first converts it into an ε-optimum partial assignment in which every node is unassigned. Then it uses a variation of the auction algorithm by Bertsekas and Eckstein (1988) to assign the unassigned nodes while throughout maintaining the ε-optimality of the partial assignment. At the termination of this method all but at most $\sqrt{n}$ nodes are unassigned. The procedure *successive-shortest-path* converts this ε-optimum partial assignment into an ε-optimum assignment. We shall show that each of these procedures runs in $O(\sqrt{n}\, m)$ time, which would yield an overall running time of $O(\sqrt{n}\, m \log(nC))$ for the assignment algorithm.

We give in Section 3.1 an algorithmic description of the procedure *auction*. In this procedure, an arc (i, j) is called *admissible* if $-\varepsilon \leqslant \bar{c}_{ij} < 0$. The procedure performs assignments only on *admissible* arcs. A node that is relabeled $L+k$ times during an execution of the *auction* is not assigned anymore and is called *ineligible*. A node that is not ineligible is called an *eligible* node. The procedure *auction* assigns all but (possibly) a small number of nodes. Let x^0 and π^0 respectively denote the partial assignment and the node potentials at the end of the *auction* procedure. We use the method described in Section 3.2 to assign the remaining unassigned nodes.

3.1. *Procedure auction*

```
procedure auction($c$, $k$, $\varepsilon$, $L$, $\pi$, $x$);
begin
  $X := \emptyset$;
  $\pi(j) := \pi(j) + k\varepsilon$ for all $j \in N_2$;
  while there is an eligible unassigned node $i$ in $N_1$ do
    if there is an admissible arc $(i, j)$ then
    begin
      $\pi(j) := \pi(j) + \varepsilon$;
      if node $j$ was already assigned to some node $l$ then
      begin
        deassign node $l$ from node $j$;
        $X := X - \{(l, j)\}$;
      end;
      assign node $i$ to node $j$;
      $X := X + \{(i, j)\}$;
    end;
    else $\pi(i) := \pi(i) + \varepsilon$;
end;
```

We first analyse the procedure *auction*. In the above description, we left out the detail on how to identify an admissible arc (i, j) incident to the node i. We use the following data structure to perform this operation. We maintain with each node $i \in N_1$, a list $A(i)$ of arcs emanating from it. Arcs in this list can be arranged arbitrarily, but the order once decided remains unchanged throughout the algorithm. Each node i has a *current arc* (i, j) which is the next candidate for assignment. To determine an admissible arc emanating at node i, the algorithm examines this list sequentially and whenever the current arc is found to be inadmissible, the next arc in the arc list is made the current arc. When the arc list is completely examined, the node potential is increased and the current arc is again the first arc in its arc list. Note that if any arc (i, j) is found inadmissible during a scan of the arc list, it remains inadmissible until $\pi(i)$ is increased. This follows from the fact that all node potentials are non-decreasing.

The procedure *auction* iteratively performs three basic operations: (i) *assignment*, i.e., assigning a node i to a node j; (ii) *deassignment*, i.e., deassigning a node l from a node j; and (iii) *relabel*, i.e., increasing a node potential by ε units. We shall use the following observation about the *auction* procedure.

Observation 1. The potential of an unassigned node $j \in N_2$ never changes until it is assigned (except at the beginning when it increases by $k\varepsilon$ units). Further, if node $j \in N_2$ is assigned to a node in N_1, then in all subsequent iterations of the *auction* procedure it remains assigned to some node in N_1.

We need the following lemmas to prove correctness of the algorithm and obtain its complexity results.

Lemma 2. *The partial assignment maintained by the* auction *procedure is always ε-optimal.*

Proof. This result is shown by performing induction on the number of iterations. At the beginning of the procedure, we have $c_{ij} - \pi(i) + \pi(j) \geqslant -k\varepsilon$ for all $(i,j) \in A$. Alternatively, $c_{ij} - \pi(i) + \pi(j) + k\varepsilon \geqslant 0$, for all $(i,j) \in A$. Hence when the potentials of all nodes in N_2 are increased by $k\varepsilon$, then condition (C3)(b) is satisfied for all arcs in A. The intial solution x is a null assignment and (C3)(a) is vacuously satisfied. Thus this solution is ε-optimal (in fact, it is 0-optimal).

Each iteration of the **while** loop either assigns some unassigned node $i \in N_1$ to a node $j \in N_2$, or increases the node potential of node i. In the former case, $-\varepsilon \leqslant \bar{c}_{ij} < 0$ by the criteria of admissibility, and hence the arc satisfies (C3)(a). Then $\pi(j)$ is increased by ε and we get $0 \leqslant \bar{c}_{ij} < \varepsilon$. If node j we already assigned to some node l, then deassigning node l from node j does not affect the ε-optimality of the partial assignment. In the latter case, when the node potential of node i is increased, we have $\bar{c}_{ij} \geqslant 0$ for all $(i,j) \in A(i)$ before the increase, and hence $\bar{c}_{ij} \geqslant -\varepsilon$ for all $(i,j) \in A(i)$ after the increase. It follows that each iteration of the procedure preserves ε-optimality of the partial assignment. $\square$

Lemma 3. *The* auction *procedure runs in $O((k+L)m)$ time.*

Proof. Each iteration of the procedure results in at least one of the following outcomes: (i) assignment; (ii) deassignment; and (iii) relabel of a node in N_1. Clearly, the number of relabels of nodes in N_1 is bounded by $O((k+L)n)$. Initially, no node is assigned and, finally, all nodes may be assigned. Thus the number of assignments are bounded by n plus the number of deassignments. The deassignment of node l from node j causes the current arc of node l to advance to the next arc. After $|A(l)|$ such advancements for node l, a relabel operation takes place. Since any node $l \in N_1$ can be relabeled at most $(k+L)$ times, we get a bound of $O(\sum_{l \in N_1} |A(l)|(k+L)) = O((k+L)m)$ on the number of deassignments, and the lemma follows. $\square$

Lemma 4. *At the termination of the* auction *procedure, at most $2n(k+1)/L$ nodes in N_1 are unassigned in the partial assignment x^o.*

Proof. Let x' denote some $k\varepsilon$-optimal assignment. Let π (resp., π') and $\bar{c}_{ij}$ (resp., $\bar{c}'_{ij}$) denote the node potentials and reduced costs corresponding to x^o (resp., x').

Let $i_0 \in N_1$ be an unassigned node in x^o. We intend to show that there exists a sequence of nodes $i_0 - j_0 - i_1 - j_1 - \cdots - i_l - j_l$ such that $j_l \in N_2$ and is unassigned in x^o. Further, this sequence satisfies the property that $(i_0, j_0), (i_1, j_1), \ldots, (i_l, j_l)$ are in the assignment x' and the arcs $(i_1, j_0), (i_2, j_1), \ldots, (i_l, j_{l-1})$ are in the assignment x^o.

Suppose that node $i_0 \in N_1$ is assigned to node $j_0 \in N_2$ in x'. If node j_0 is unassigned in x^o, then we have discovered such a path; otherwise let node $i_1 \in N_1$ be assigned to the node j_0 in x^o. Observe that $i_1 \neq i_0$. Now let node i_1 be assigned to node $j_1 \in N_2$ in x'. Observe that $j_1 \neq j_0$. If node j_1 is unassigned in x^o, then we are done; else repeating the same logic yields that eventually a sequence satisfying the above properties would be discovered. Now observe that the path P consisting of the node sequence $i_0 - j_0 - i_1 - j_1 \cdots - i_l - j_l$ is in the residual network $G(x^o)$ and its reversal $\bar{P}$ consisting of node sequence $j_l - i_l - \cdots - j_1 - i_1 - j_0 - i_0$ is in the residual network $G(x')$. The ε-optimality of the solution x^o yields that

$$\sum_{(i,j) \in P} \bar{c}_{ij} = \pi(j_l) - \pi(i_0) + \sum_{(i,j) \in P} c_{ij} \geq -l^o \varepsilon, \tag{2}$$

where $l^o = 2l + 1$ and represents the number of arcs in P. Further, from the $k\varepsilon$-optimality of the solution x' it follows that

$$\sum_{(i,j) \in \bar{P}} \bar{c}'_{ij} = \pi'(i_0) - \pi'(j_l) + \sum_{(i,j) \in \bar{P}} c_{ij} \geq -k l^o \varepsilon. \tag{3}$$

Note that $\pi(j_l) = \pi'(j_l) + k\varepsilon$, since the potential of node j_l increases by $k\varepsilon$ at the beginning of the scaling phase and then remains unchanged. Also note that $\sum_{(i,j) \in \bar{P}} c_{ij} = -\sum_{(i,j) \in P} c_{ij}$. Using these facts in (2) and (3), we get

$$\pi(i_0) \leq \pi'(i_0) + \{k + (k+1) l^o\} \varepsilon. \tag{4}$$

Finally, we use the fact that $\pi(i_0) = \pi'(i_0) + (L + k)\varepsilon$, i.e., the potential of each unassigned node in N_1 has been increased $(L + k)$ times. This gives $l^o \geq L/(k+1)$. To summarize, we have shown that for every unassigned node i_0 in N_1 there exists an unassigned node j_l in N_2, and a path P between these two nodes consisting of at least $L/(k+1)$ nodes such that P is in the $G(x^o)$ and its reversal $\bar{P}$ is in $G(x')$. The facts that x' is an assignment and x^o is a partial assignment imply that these paths corresponding to two different unassigned nodes in N_1 are node disjoint. Consequently, there are at most $2n(k+1)/L$ unassigned nodes in N_1. □

Since in the *auction* procedure, we set $L = 2(k+1)\lceil \sqrt{n} \rceil$, Lemmas 3 and 4 immediately yield the following result:

Lemma 5. *For fixed value of k, the* auction *procedure terminates in* $O(\sqrt{n}\, m)$ *time and yields an ε-optimal partial assignment in which at most* $\lceil \sqrt{n} \rceil$ *nodes are unassigned.* □

This lemma also shows the necessity of the two phase approach in the *improve-approximation* procedure. By setting $L = 2(k+1)\lceil \sqrt{n} \rceil$ in the *auction* procedure, we obtain an assignment in which at most $\lceil \sqrt{n} \rceil$ nodes are unassigned. If we want all nodes to be assigned then we shall have to set $L = 2(k+1)n + 1$. In this case, the procedure would give an ε-optimal assignment and would take $O(nm)$ time. This implies that if $n = 10000$, then the *auction* procedure would assign the first 99% of the nodes in 1% in the overall running time, and would assign the remaining 1% of the nodes in the remaining 99% of the running time. By using a successive shortest path procedure described next we can improve the running time for assigning the last 1% of the nodes by a factor of nearly 100 in the worst case.

3.2. Procedure successive-shortest-path

```
procedure successive-shortest-path;
begin
  while there are unassigned nodes in $N_1$ do
  begin
    let $\bar{c}_{ij}$ be the reduced costs with respect to the potentials $\pi^{\circ}$;
    define $d_{ij} := \max\{0, \lfloor \bar{c}_{ij}/\varepsilon \rfloor + 1\}$;
    select an unassigned node $i_0$ in $N_1$;
    apply Dijkstra's algorithm with $d_{ij}$ as arc lengths starting at node $i_0$ until some
      unassigned node $j_l$ in $N_2$ is permanently labeled;
    let $w(i)$ denote the distance labels of nodes that are permanently labeled;
    set $\pi^{\circ}(i) := \pi^{\circ}(i) + \varepsilon(w(j_l) - w(i))$ for all permanently labeled nodes $i \in N$;
    augment one unit of flow along the shortest path from node $i_0$ to $j_l$ and update
      $x^{\circ}$;
  end;
  set $x := x^{\circ}$ and $\pi := \pi^{\circ}$;
end;
```

We now analyse the complexity of the *successive-shortest-path* procedure. This procedure solves a shortest path problem for every unassigned node in x°. We show that each shortest path problem can be solved in $O(m)$ time. As the partial assignment x° has at most $\lceil \sqrt{n} \rceil$ unassigned nodes, this would imply that this procedure too would run in $O(\sqrt{n}\, m)$ time.

We can easily make the following observations about the procedure *successive-shortest-path*:

Observation 2. $d_{ij} \leq (\bar{c}_{ij}/\varepsilon) + 1$.

Observation 3. The node potentials π° are nondecreasing through the procedure.

Observation 4. The procedure does not change the potential of any unassigned node in N_2.

Observation 2 is immediate from the definition of d_{ij}. Observation 3 follows from the fact that the distance labels made permanent by the algorithm are nondecreasing. Observation 4 follows from the facts that the algorithm does not change the potential of any temporarily labeled node and an unassigned node in N_2 is never permanently labeled.

We now prove the correctness of the algorithm. It is easily seen that if the solution x° is ε-optimal with respect to the node potentials π° then each d_{ij} is nonnegative. In the first iteration, x° is ε-optimal with respect to the potential π° (by Lemma 5), and we shall show later that the algorithm preserves this property in each subsequent iteration. Hence each d_{ij} is nonnegative throughout, and Dijkstra's algorithm can be used to solve the shortest path problems. We now show that the distance labels that are permanently labeled by Dijkstra's algorithm are *small.*

We have already shown in the proof of Lemma 4 that in x° for every unassigned node $i_0 \in N_1$ there exists an unassigned node $j_l \in N_2$, and a path P of length l° between these two nodes satisfying some properties. Let us consider the inequality (3). Using the facts that (i) $\pi^\circ(i_0) \geq \pi'(\mathrm{i}_0)$ (since potentials are nondecreasing in both the *auction* and *successive-shortest-path* procedures), and (ii) $\pi^\circ(j_l) = \pi'(j_l) + k\varepsilon$ (since potentials of nodes in N_2 increase by at the beginning of *auction* procedure and then remain unchanged) in (3), we get

$$\pi(j_l) - \pi(i_0) + \sum_{(i,j)\in P} c_{ij} \leq k\varepsilon + kl^\circ\varepsilon. \tag{5}$$

This yields that $\sum_{(i,j)\in P} \bar{c}_{ij}/\varepsilon \leq k + kl^\circ$. Using Observation 2 in this inequality we get $\sum_{(i,j)\in P} d_{ij} \leq k + l^\circ + kl^\circ = \mathrm{O}(n)$. The fact that this path has a *small* length is used to improve the complexity of Dijkstra's shortest path algorithm.

We use Dial's implementation of Dijkstra's algorithm (see Dial et al., 1979, or Ahuja et al., 1989) to solve the shortest path problems. Dial's implementation takes $\mathrm{O}(m+p)$ time to solve a shortest path problem where p is the largest distance label made permanent by the implementation. For our shortest path problems, $p = \mathrm{O}(n)$ because there exists a path of length $\mathrm{O}(n)$ between node i_0 and some unassigned node j_l in N_2. Hence each iteration of the procedure would take $\mathrm{O}(m)$ time. As there are at most $\lceil\sqrt{n}\,\rceil$ such iterations, the *successive-shortest-path* procedure would take $\mathrm{O}(\sqrt{n}\; m)$ time to terminate.

We now show that at the end of each iteration the partial assignment x° is ε-optimal with respect to the node potentials π°. We assume inductively that the solution satisfies condition (C4) at the begining of each iteration. Let S denote the set of nodes permanently labeled by Dijkstra's algorithm and $\bar{S} = N - S$. Since the algorithm increases the potentials of nodes in S only, it follows that (i) the reduced cost of any arc (i,j) for which $i \in \bar{S}$ and $j \in \bar{S}$ remains unchanged, and (ii) the reduced cost of any arc (i,j) for which $i \in \bar{S}$ and $j \in S$ does not decrease. Hence for these two categories of arcs (C4) remains satisfied. We next consider arcs (i,j) such that $i \in S$. We use the following observation about Dijkstra's algorithm.

Observation 5. For every node $i \in S$, Dijkstra's algorithm satisfies $w(j) \leq w(i) + d_{ij}$ for all $(i,j) \in A(i)$.

This observation can easily be shown by performing induction on the number of permanently labeled nodes. We arrange the preceding inequality as $-w(i) \leq -w(j) + d_{ij}$ and add $w(j_l)$ to both the sides to obtain $w(j_l) - w(i) \leq w(j_l) - w(j) + d_{ij}$.

Then using Observation 2 we get

$$w(j_l) - w(i) \leq w(j_l) - w(j) + (\bar{c}_{ij}/\varepsilon) + 1, \tag{6}$$

which can be arranged as

$$c_{ij} - \{\pi^{\circ}(i) + \varepsilon(w(j_l) - w(i))\} + \{\pi^{\circ}(j) + \varepsilon(w(j_l) - w(j))\} \geq \varepsilon. \tag{7}$$

This establishes that the arc (i, j) satisfies the condition $\bar{c}_{ij} \geq -\varepsilon$ with respect to the new node potentials. We have thus shown the following result:

Lemma 6. *The procedure* successive-shortest-path *obtains an ε-optimal assignment in* $O(\sqrt{n}\, m)$ *time.* □

Combining Lemma 5 and Lemma 6, we get the following theorem:

Theorem 1. *The procedure* improve-approximation *runs in* $O(\sqrt{n}\, m)$ *time. The scaling algorithm obtains an optimum assignment in* $O(\sqrt{n}\, m \log(nC))$ *time.* □

4. The minimum mean cycle problem

In this section we develop scaling algorithms for the minimum mean cycle problem. Recall that the mean cost of a directed cycle W is defined as $\sum_{(i,j)\in W} c_{ij}/|W|$ and the minimum mean cycle problem is to identify a cycle of minimum mean cost. The minimum mean cycle problem is a special case of the minimum cost-to-time ratio cycle problem (see Lawler, 1976) and we use its special structure to develop more efficient algorithms.

We consider the network $G = (N, A)$ where a (possibly negative) integer c_{ij} represents the cost of each arc $(i, j) \in A$. Let $n = |N|$ and $m = |A|$. Let $C = 1 + \max(|c_{ij}| : (i, j) \in A)$. We assume that the network contains at least one cycle; otherwise there is no feasible solution. Acyclicity of a network can be determined in $O(m)$ time (see, for example, Aho et al., 1974). We have already indicated in Section 1 that the minimum mean cycle problem can be solved in $O(\sqrt{n}\, m \log^2(nC))$ time by using using binary search and solving a shortest path problem at each search point. We now describe how the minimum mean cycle problem can be solved in $O(\sqrt{n}\, m \log(nC))$ time by using approximate binary search, as per Zemel (1987).

Our algorithm is based on a well known transformation using *node splitting*. We split each node $i \in N$ into two nodes i and i', and replace each arc (i, j) in the original network by the arc (i, j'). The cost of the arc (i, j') is same as that of arc (i, j). We also add arcs (i, i') of cost δ for each $i \in N$ in the transformed network. This gives us an assignment problem with $N_1 = \{1, 2, \ldots, n\}$ and $N_2 = \{1', 2', \ldots, n'\}$. We treat δ as a parameter in the transformation. We represent the cost vector of the assignment problem by $C(\delta)$ and refer to the problem as ASSIGN(δ). An optimum solution of ASSIGN(δ) either consists solely of the arcs (i, i') for all i, or does not contain (i, i') for some i. In the former case, the assignment is called a *uniform* assignment, and in the latter case it is called a *non-uniform* assignment. We represent the minimum mean cost in the network by μ^*. Our algorithm is based on the following result:

Lemma 7. (a) *If an ε-optimal assignment X of* ASSIGN(δ) *is uniform, then* $\delta - 2\varepsilon \leqslant \mu^*$.

(b) *If an ε-optimal assignment X of* ASSIGN(δ) *is non-uniform, then* $\mu^* \leqslant \delta + 2\varepsilon$.

Proof. (a) Let π be the node potentials corresponding to the ε-optimal assignment X. Suppose the cycle W^* in G consisting of the node sequence $i_1 - i_2 - i_3 - \cdots - i_r - i_1$ is a minimum mean cycle of cost μ^*. Let $I^* = \{(i_1, i_2'), (i_2, i_3'), \ldots, (i_{r-1}, i_r'), (i_r, i_1')\}$. It follows from condition (C3)(b) that

$$\sum_{(i,j)\in I^*} \bar{c}_{ij} = r\mu^* - \sum_{i=i_1}^{i_r} \pi(i) + \sum_{i=i_1'}^{i_r'} \pi(i) \geqslant -r\varepsilon. \tag{8}$$

Let $I = \{(i_1, i_1'), (i_2, i_2'), \ldots, (i_r, i_r')\}$. As I is part of the ε-optimal assignment X, it follows from condition (C3)(a) that

$$\sum_{(i,j)\in I} \bar{c}_{ij} = r\delta - \sum_{i=i_1}^{i_r} \pi(i) + \sum_{i=i_1'}^{i_r'} \pi(i) \leqslant r\varepsilon. \tag{9}$$

Combining (8) and (9) we get

$$\delta - 2\varepsilon \leqslant u^*. \tag{10}$$

(b) Since the assignment X is non-uniform there exists a node $j_1 \in N_1$ that is assigned to a node $j_2' \in N_2$ where $j_1 \neq j_2$. Suppose that node j_2 is assigned to some node j_3'. Note that $j_2 \neq j_3$. Extending this logic further indicates that eventually we would discover a node $j_r \in N_1$ which is assigned to the node $j_1' \in N_2$. The node sequence $j_1 - j_2 - j_3 - \cdots - j_r - j_1$ defines a directed cycle in G and we denote it by W. Let μ be the mean cost of W. Then $\mu^* \leqslant \mu$. Let $I = \{(j_1, j_2'), (j_2, j_3'), \ldots, (j_{r-1}, j_r'), (j_r, j_1')\}$ and $J = \{(j_1, j_1'), (j_2, j_2'), \ldots, (j_r, j_r')\}$. It follows from (C3)(b) and (C3)(a), respectively, that

$$\sum_{(i,j)\in J} \bar{c}_{ij} = r\delta - \sum_{j=j_1}^{j_r} \pi(j) + \sum_{j=j_1'}^{j_r'} \pi(j) \geqslant -r\varepsilon \tag{11}$$

and

$$\sum_{(i,j)\in I} \bar{c}_{ij} = r\mu - \sum_{j=j_1}^{j_r} \pi(j) + \sum_{j=j_1'}^{j_r'} \pi(j) \leqslant r\varepsilon. \tag{12}$$

Combining (11) and (12) we get

$$\mu^* \leqslant \mu \leqslant \delta + 2\varepsilon. \qquad \Box$$

We use this result in our algorithm for the minimum mean cycle problem. The algorithm maintains an interval [LB, UB] containing the minimum mean cycle value μ^* and in one application of the *improve-approximation* procedure with carefully selected values of δ and ε reduces the interval length by a constant factor. A formal description of this algorithm is given below followed by its justification.

```
algorithm minimum-mean-cycle;
begin
  set LB := $-C$ and UB := $C$;
  set $\pi(i) := -C/2$ for all $i \in N_1$ and $\pi(i') := 0$ for all $i' \in N_2$;
  let $x$ be the uniform assignment;
  while $(\mathrm{UB}-\mathrm{LB}) \geq 1/n^2$ do
  begin
    $\delta := \frac{1}{2}(\mathrm{UB}+\mathrm{LB})$;
    $\varepsilon := \frac{1}{8}(\mathrm{UB}-\mathrm{LB})$;
    $k := 3$;
    improve-approximation($c(\delta)$, $k$, $\varepsilon$, $L$, $\pi$, $x$);
    if $x$ is a uniform assignment then LB := $\delta - 2\varepsilon$
    else UB := $\delta + 2\varepsilon$ and $x^* := x$;
  end;
  use the non-uniform assignment $x^*$ to construct the minimum mean cycle of cost
    $\mu^*$ in the interval [LB, UB];
end;
```

Notice that $-C < \mu^* < C$ since the absolute values of all arc costs are strictly less than C. Furthermore, if the mean cycle costs of two cycles are unequal, then they must differ by at least $1/n^2$. This can be seen as follows. Let W_1 and W_2 be two distinct cycles and $L_k = \sum_{(i,j) \in W_k} c_{ij}$ and $r_k = |W_k|$ for each $k = 1, 2$. Then

$$\left|\frac{L_1}{r_1} - \frac{L_2}{r_2}\right| = \left|\frac{r_2 L_1 - r_2 L_2}{r_1 r_2}\right| \neq 0.$$

The numerator in the above expression is at least 1 and the denominator is at most n^2, thereby proving our claim. This observation shows that the algorithm terminates with a minimum mean cycle.

Theorem 2. *The minimum mean cycle algorithm correctly determines a minimum mean cycle in* $O(\sqrt{n}\, m \log(nC))$ *time.* □

Proof. The algorithm always maintains an interval [LB, UB] containing the minimum mean cost. If the execution of *improve-approximation* procedure yields a uniform assignment x, then the new lower bound is $\delta - 2\varepsilon = \frac{1}{4}(\mathrm{UB}+3\mathrm{LB})$; otherwise, the new upper bound is $\delta + 2\varepsilon = \frac{1}{4}(3\mathrm{UB}+\mathrm{LB})$. In either case, the interval length $(\mathrm{UB}-\mathrm{LB})$ decreases by 25%. Since, initially $\mathrm{UB}-\mathrm{LB} = 2C$, after at most $1 + \lceil \log_{4/3} n^2 C \rceil = O(\log(nC))$ iterations, $\mathrm{UB}-\mathrm{LB} < 1/n^2$ and the algorithm terminates. Each execution of the *improve-approximation* procedure takes $O(\sqrt{n}\, m)$ time and the algorithm runs in $O(\sqrt{n}\, m \log(nC))$ time.

To show the correctness of the algorithm, we need to show that the solution x is 3ε-optimal with respect to the costs $C(\delta)$ before calling *improve-approximation* procedure. Initially, x is the uniform assignment $\delta = 0$ and $\varepsilon = \frac{1}{4}C$. Hence for each arc (i, i') in the uniform assignment $\bar{c}_{ii'} = \frac{1}{2}C \leq \frac{3}{4}C$, thereby satisfying (C3)(a). Further, for each arc (i, j') in ASSIGN(δ), we have $\bar{c}_{ij'} = c_{ij'} + \frac{1}{2}C \geq -C + \frac{1}{2}C = -\frac{1}{2}C \geq -\frac{3}{4}C$, which is consistent with (C3)(a).

Now consider any general step. Let the assignment x be an ε-optimal solution for ASSIGN(δ) and let $\bar{c}_{ij}$ denote the reduced costs of arcs at this point. Further, let UB$'$, LB$'$, δ', ε' and $\bar{c}'_{ij}$ be the corresponding values in the next iteration just before calling the procedure *improve-approximation.* We need to show that the solution x is $3\varepsilon'$-optimal for ASSIGN(δ'). We consider the case when $\delta' > \delta$, The case when $\delta' < \delta$ can be proved similarly. We showed earlier that the interval length decreases by 25% at every iteration. Therefore, $\varepsilon' = \frac{3}{4}\varepsilon$. Further, observe that the case $\delta' > \delta$ occurs when the lower bound increases. Hence, $\delta' = \frac{1}{2}(\text{UB}' + \text{LB}') = \frac{1}{2}(\text{UB} + \frac{1}{4}(\text{UB} + 3\text{LB})) = \delta + \varepsilon$. Since arc costs do not decrease in ASSIGN(δ'), all arcs keep satisfying (C3)(b). For an arc $(i, j') \in X$, we have $\bar{c}'_{ij'} \leq \bar{c}_{ij'} + \varepsilon$ because the costs increase by at most ε units. We then use the fact that $\bar{c}_{ij'} \leq \varepsilon$ to obtain $\bar{c}_{ij'} \leq 2\varepsilon \leq 3\varepsilon'$, which is consistent with (C3)(a). The theorem now follows. $\square$

The minimum mean cycle algorithm can possibly be sped up in practice by using a better estimate of the upper bound UB. Whenever an application of the *improve-approximation* procedure yields a non-uniform assignment x, then a cycle is located in the original network using the assignment x and the upper bound UB is set to the mean cost of this cycle. Further, if it is desirable to perform all computations in integers, then we can multiply all arc costs by n^2, initially set $\text{UB} = k^{\lceil \log_k(n^2C) \rceil}$, and terminate the algorithm when $\varepsilon < 1$. The accuracy of this modified version can be easily established.

We close this paper by making a curious observation. The best known time to detect a negative cost cycle in a network is the same as the time to determine the minimum mean cycle of a network. However, this latter problem is more general because the minimum mean cycle value is negative if and only if there is a negative cost cycle. In addition, the minimum mean cycle problem seems as though it should be more difficult to solve. The best time bound for both problems under the similarity assumption is $O(\sqrt{n}\, m \log(nC))$. Under the assumption that C is exponentially large, the best time bound for each of these problems is $O(nm)$. Determining a better time bound for the problem of detecting a negative cost cycle is both of theoretical and practical importance.

Acknowledgements

We thank Michel Goemans and Hershel Safer for a careful reading of the paper and many useful suggestions. We also thank the referees for their insightful comments. This research was supported in part by the Presidential Young Investigator Grant 8451517-ECS of the National Science Foundation, by Grant AFOSR-88-0088 from the Air Force Office of Scientific Research, and by Grants from Analog Devices, Apple Computer, Inc., and Prime Computer.

References

A.V. Aho, J.E. Hopcroft and J.B. Ullman, *The Design and Analysis of Computer Algorithms* (Addison-Wesley, Reading, MA, 1974).

R.K. Ahuja, T.L. Magnanti and J.B. Orlin, "Network flows," in: *Handbooks in Operations Research and Management Science. Vol. I: Optimization* (North-Holland, Amsterdam, 1989).

D.P. Bertsekas, "A distributed algorithm for the assignment problem," Working Paper, Laboratory for Information and Decision Systems, MIT (Cambridge, MA, 1979).

D.P. Bertsekas, "The auction algorithm: a distributed relaxation method for the assignment problem," Technical Report LIDS-P-1653, MIT (Cambridge, MA, 1987).

D.P. Bertsekas and J. Eckstein, "Dual coordinate step methods for linear network flow problems," *Mathematical Programming* 42 (1988) 203-243.

D.P. Bertsekas and D.A. Castanon, "The auction algorithm for the minimum cost network flow problem," LIDS Technical Report, MIT (Cambridge, MA, 1989).

L.D. Bodin, B.L. Golden, A.A. Assad and M.O. Ball, "Routing and scheduling of vehicles and crews," *Computers and Operations Research* 10 (1983) 65-211.

R. Dial, "Algorithm 360: shortest path forest with topological ordering," *Communications of the ACM* 12 (1969) 632-633.

E. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik* 1 (1959) 269-271.

E.A. Dinic, "Algorithm for solution of a problem of maximum flow in networks with power estimation," *Soviet Mathematics Doklady* 11 (1970) 1277-1280.

S. Even and R.E. Tarjan, "Network flow and testing graph connectivity," *SIAM Journal of Computing* 4 (1975) 507-518.

R.L. Francis and J.A. White, *Facility Location and Layout: An Analytical Approach* (Prentice-Hall, Englewood Cliffs, NJ, 1974).

M.L. Fredman and R.E. Tarjan, "Fibonacci heaps and their uses in network optimization algorithms," *Proceedings 25th Annual IEEE Symposium on Foundation of Computer Science* (1984) pp. 338-346.

H.N. Gabow and R.E. Tarjan, "Almost-optimum speed-ups of algorithms for bipartite matchings and related problems," *Proceedings of the 20th Annual ACM Symposium on Theory of Computing* (1988) pp. 514-527.

A.V. Goldberg and R.E. Tarjan, "Solving minimum cost flow problem by successive approximation," *Proceedings of the 19th ACM Symposium on the Theory of Computing* (1987) pp 136-146.

A.V. Goldberg and R.E. Tarjan, "Finding minimum-cost circulations by canceling negative cycles," *Proceedings of the 20th ACM Symposium on Theory of Computing* (1988) pp. 388-397.

J.E. Hopcroft and R.M. Karp, "An $n^{5/2}$ algorithm for maximum matching in bipartite graphs," *SIAM Journal of Computing* 2 (1973) 225-231.

R.M. Karp, "Probabilistic analysis of partitioning algorithms for the traveling salesman problem in the plane," *Mathematics Operations Research* 2 (1977) 209-224.

R.M. Karp, "A characterization of the minimum cycle mean in a digraph," *Discrete Mathematiccs* 23 (1977) 309-311.

R.M. Karp and J.B. Orlin, "Parametric shortest path algorithms with an application to cyclic staffing," *Discrete Applied Mathematics* 3 (1981) 37-45.

E.L. Lawler, *Combinatorial Optimization: Networks and Matroids* (Holt, Rinehart and Winston, NY, 1976).

J.B. Orlin, "The complexity of dynamic languages and dynamic optimization problems," *Proceedings of the 13th Annual ACM Symposium on the Theory of Computing* (1981) pp. 218-227.

J.B. Orlin and R.K. Ahuja, "New distance-directed algorithms for maximum flow and parametric maximum flow problems," Sloan, W.P. No. 1908-87, Sloan School of Management, MIT (Cambridge, MA, 1987).

E. Tardos, "A strongly polynomial minimum cost circulation algorithm," *Combinatorica* 5 (1985) 247-255.

E. Zemel, "A linear time randomizing algorithm for searching ranked functions," *Algorithmica* 2 (1987) 81-90.