GLOBAL PRICE UPDATES HELP

ANDREW V. GOLDBERG
AND
ROBERT KENNEDY

COMPUTER SCIENCE DEPARTMENT
STANFORD UNIVERSITY
STANFORD, CA 94305

GOLDBERG@CS.STANFORD.EDU
ROBERT@CS.STANFORD.EDU

March 1994

Abstract. Periodic global updates of dual variables have been shown to yield a substantial speed advantage in implementations of push-relabel algorithms for the maximum flow and minimum cost flow problems. In this paper, we show that in the context of the bipartite matching and assignment problems, global updates yield a theoretical improvement as well. For bipartite matching, a push-relabel algorithm that matches the best bound when global updates are used achieves a bound that is worse by a factor of $\sqrt{n}$ without the updates. A similar result holds for the assignment problem.

Andrew V. Goldberg was supported in part by ONR Young Investigator Award N00014-91-J-1855. Robert Kennedy was supported by the above mentioned ONR grant.

1. Introduction.

The push-relabel method [10, 13] is the best currently known way for solving the maximum flow problem [1, 2, 18]. This method extends to the minimum cost flow problem using cost scaling [10, 14], and an implementation of this technique has proven very competitive on a wide class of problems [11]. In both contexts, the idea of periodic global updates of node distances or prices has been critical to obtaining the best running times in practice.

Several algorithms for the bipartite matching problem run in $O(\sqrt{n}m)$ time.[1] Hopcroft and Karp [15] first proposed an algorithm that achieves this bound. Karzanov [16] and Even and Tarjan [5] proved that the blocking flow algorithm of Dinitz [4] runs in this time when applied to the bipartite matching problem. Two phase algorithms based on a combination of the push-relabel method [13] and the augmenting path method [7] were proposed in [12, 19].

Feder and Motwani [6] give a "graph compression" technique that combines with the algorithm of Dinitz to yield an $O(\sqrt{n}m\frac{\log(n^2/m)}{\log n})$ algorithm. This is the best time bound known for the problem.

The most relevant theoretical results on the assignment problem are as follows. The best currently known strongly polynomial time bound of $O(n(m + n\log n))$ is achieved by the classical Hungarian method of Kuhn [17]. Under the assumption that the input costs are integers in the range $[-C, \ldots, C]$, Gabow and Tarjan [9] use cost scaling and blocking flow techniques to obtain an $O(\sqrt{n}m\log(nC))$ time algorithm. An algorithm using an idea similar to global updates with the same running time appeared in [8]. Two-phase algorithms with the same running time appeared in [12, 19]. The first phase of these algorithms is based on the push-relabel method and the second phase is based on the successive augmentation approach.

We show that algorithms based on the push-relabel method with global updates match the best bounds for the bipartite matching and assignment problems. Our results are based on new selection strategies: the *minimum distance* strategy in the bipartite matching case and the *minimum price change* in the assignment problem case. We also prove that the algorithms perform significantly worse without global updates. Similar results can be obtained for maximum and minimum cost flows in networks with unit capacities. Our results are a step toward a theoretical justification of the use of global update heuristics in practice.

This paper is organized as follows. Section 2 gives definitions relevant to bipartite matching and maximum flow. Section 3 outlines the push-relabel method for maximum flow and shows its application to bipartite matching. In Section 4, we present the time bound for the bipartite matching algorithm with global updates, and in Section 5 we show that without global updates, the algorithm performs poorly. Section 6 gives definitions relevant to the assignment problem and minimum cost flow. In Section 7, we describe the cost-scaling push-relabel method for minimum cost flow and apply the method to the assignment problem. Sections 8 and 9 gen-

[1] Here n and m denote the number of nodes and edges, respectively.

eralize the bipartite matching results to the assignment problem. In Section 10, we give our conclusions and suggest directions for further research.

2. Bipartite Matching and Maximum Flow

Let $\overline{G} = (\overline{V} = X \cup Y, \overline{E})$ be an undirected bipartite graph, let $n = |\overline{V}|$, and let $m = |\overline{E}|$. A *matching* in $\overline{G}$ is a subset of edges $M \subseteq \overline{E}$ that have no node in common. The *cardinality* of the matching is $|M|$. The *bipartite matching problem* is to find a maximum cardinality matching.

The conventions we assume for the maximum flow problem are as follows: Let $G = (\{s,t\} \cup V, E)$ be a digraph with an integer-valued *capacity* $u(a)$ associated with each arc[2] $a \in E$. We assume that $a \in E \Rightarrow a^R \in E$ (where a^R denotes the reverse of arc a). A *pseudoflow* is a function $f : E \to \mathbf{R}$ satisfying the following for each $a \in E$:

- $f(a) = -f(a^R)$ (*flow antisymmetry* constraints);
- $f(a) \le u(a)$ (*capacity* constraints).

The antisymmetry constraints are for notational convenience only, and we will often take advantage of this fact by mentioning only those arcs with nonnegative flow; in every case, the antisymmetry constraints are satisfied simply by setting the reverse arc's flow to the appropriate value. For a pseudoflow f and a node v, the *excess flow into* v, denoted $e_f(v)$, is defined by $e_f(v) = \sum_{(u,v)\in E} f(u,v)$. A *preflow* is a pseudoflow with the property that the excess of every node except s is nonnegative. A node $v \neq t$ with $e_f(v) > 0$ is called *active*.

A *flow* is a pseudoflow f such that, for each node $v \in V$, $e_f(v) = 0$. Observe that a preflow f is a flow if and only if there are no active nodes. The *maximum flow problem* is to find a flow maximizing $e_f(t)$.

3. The Push-Relabel Method for Bipartite Matching

We reduce the bipartite matching problem to the maximum flow problem in a standard way. For brevity, we mention only the "forward" arcs in the flow network; to each such arc we give unit capacity. The "reverse" arcs have capacity zero. Given an instance $\overline{G} = (\overline{V} = X \cup Y, \overline{E})$ of the bipartite matching problem, we construct an instance $(G = (\{s,t\} \cup V, E), u)$ of the maximum flow problem by

- setting $V = \overline{V}$;
- for each node $v \in X$ placing arc (s,v) in E;
- for each node $v \in Y$ placing arc (v,t) in E;

[2]Sometimes we refer to an arc a by its endpoints, *e.g.*, (v,w). This is ambiguous if there are multiple arcs from v to w. An alternative is to refer to v as the tail of a and to w as the head of a, which is precise but inconvenient.

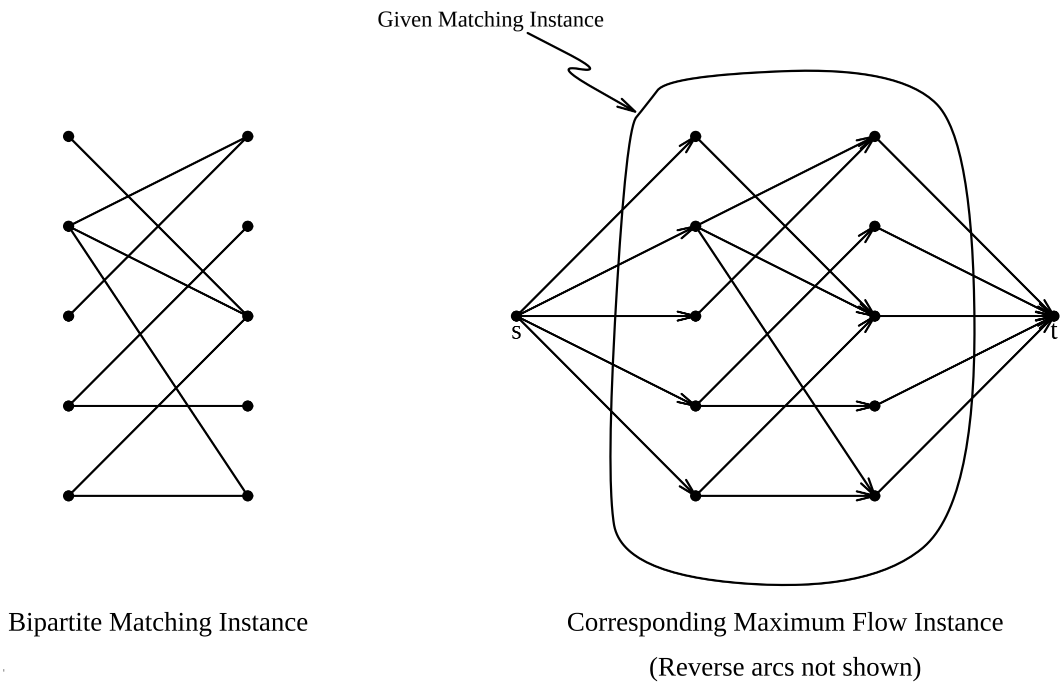


FIGURE 1. Reduction from Bipartite Matching to Maximum Flow

- for each edge $\{v, w\} \in \overline{E}$ with $v \in X$ placing arc (v, w) in E

A graph obtained by this reduction is called a *matching network*. Note that if G is a matching network, then for any integral pseudoflow f and for any arc $a \in E$, $u(a), f(a) \in \{0, 1\}$. Indeed, any integral flow in G can be interpreted conveniently as a matching in $\overline{G}$: the matching is exactly the edges corresponding to those arcs $a \in X \times Y$ with $f(a) = 1$. It is a well-known fact [7] that a maximum flow in G corresponds to a maximum matching in $\overline{G}$.

For a given pseudoflow f, the *residual capacity* of an arc $a \in E$ is $u_f(a) = u(a) - f(a)$. The set of *residual arcs* E_f contains the arcs $a \in E$ with $f(a) < u(a)$. The *residual graph* $G_f = (V, E_f)$ is the graph induced by the residual arcs.

A *distance labeling* is a function $d : V \to \mathbf{Z}^+$. We say a distance labeling d is *valid* with respect to a pseudoflow f if $d(t) = 0$, $d(s) = n$, and for every arc $(v, w) \in E_f$, $d(v) \leq d(w) + 1$. Those residual arcs (v, w) with the property that $d(v) = d(w) + 1$ are called *admissible* arcs.

We begin with a high-level description of the generic push-relabel algorithm for maximum flow specialized to the case of matching networks. The algorithm starts with the zero flow, then sets $f(s, v) = 1$ for every $v \in X$. For an initial distance labeling, the algorithm sets $d(s) = n$ and $d(t) = 0$, and for every $v \in V$, sets $d(v) = 0$. Then the algorithm applies *push* and *relabel* operations in any order until the current pseudoflow is a flow. The *push* and *relabel* operations, described below, preserve the properties that the current pseudoflow f is a preflow and that the current distance labeling d is valid with respect to f.

PUSH(v, w).
send a unit of flow from v to w.
end.

RELABEL(V).
replace $d(v)$ by $\min_{(v,w)\in E_f}\{\, d(w)+1 \,\}$
end.

FIGURE 2. The *push* and *relabel* operations

The *push* operation applies to an admissible arc (v, w) whose tail node v is active. It consists of "pushing" a unit of flow along the arc, *i.e.*, increasing $f(v, w)$ by one, increasing $e_f(w)$ by one, and decreasing $e_f(v)$ by one. The *relabel* operation applies to an active node v that is not the tail of any admissible arc. It consists of changing v's distance label so that v is the tail of at least one admissible arc, *i.e.*, setting $d(v)$ to the largest value that preserves the validity of the distance labeling. See Figure 2.

Our analysis of the push-relabel method is based on the following facts. See [13] for details; note that arcs in a matching network have unit capacities and thus PUSH(v, w) saturates the arc (v, w).

(1) $\forall v \in V$, $0 \le d(v) \le 2n - 1$.
(2) Distance labels do not decrease during the computation.
(3) *relabel*(v) increases $d(v)$.
(4) The number of *relabel* operations during the computation is $O(n)$ per node.
(5) The work involved in *relabel* operations is $O(nm)$.
(6) If a node v is relabeled t times during a computation segment, then the number of pushes from v is at most $(t + 1) \times$ *degree*(v).
(7) The number of *push* operations during the computation is $O(nm)$.

The above lemma implies that any push-relabel algorithm runs in $O(nm)$ time given that the work involved in selecting the next operation to apply does not exceed the work involved in applying these operations. This can be easily achieved using simple data structures described in [13].

4. Global Updates and the Minimum Distance Discharge Algorithm

In this section, we specify an ordering of the *push* and *relabel* operations that yields certain desirable properties. We also introduce the idea of a *global distance update* and show that the algorithm resulting from our operation ordering and global update strategy runs in $O(\sqrt{n}m)$ time.

For any nodes v, w, let $d_w(v)$ denote the breadth-first-search distance from v to w in the residual graph of the current preflow. If w is unreachable from v in the residual graph, $d_w(v)$ is infinite. Setting $d(v) = \min\{\, d_t(v), n + d_s(v)\,\}$ for every node v is called a *global update operation*. Such an operation can be accomplished with $O(m)$ work that amounts to two breadth-first-search computations.

The ordering of operations we use is called *Minimum Distance Discharge*, and it consists of repeatedly choosing an active node whose distance label is minimum among all active nodes and, if there is an admissible arc leaving that node, pushing a unit of flow along the admissible arc, otherwise relabeling the node. For convenience, we denote by $\Gamma(f,d)$ (or simply Γ) the minimum distance label of an active node with respect to the pseudoflow f and the distance labeling d. We let $\Gamma_{\max}$ denote the maximum value reached by Γ during the algorithm so far. Every time Γ attains a new maximum, we perform a global update operation.

Our analysis hinges on a parameter k in the range $2 \le k \le n$, to be chosen later. We divide the execution of the algorithm into four stages: In the first two stages, excesses are moved to t; in the final two stages, excesses that cannot reach the sink return to s. We analyze the first stage of each pair using the following lemma.

Lemma 4.1. *The Minimum Distance Discharge algorithm uses $O((j-i)m)$ work during the period beginning when Γ first exceeds $i-1$ and ending when Γ first exceeds j.*

Proof: The number of relabelings that can occur when $\Gamma_{\max}$ lies in the interval $[i,j]$ is at most $n(j-i+1)$. Thus the relabelings and pushes require $O((j-i)m)$ work. The observations that a global update requires $O(m)$ work and during the period there are $O(j-i)$ global updates complete the proof. ■

Lemma 4.1 allows us to account for the periods when $\Gamma_{\max} \in [0,k]$ and $\Gamma_{\max} \in [n, n+k]$. The algorithm expends $O(km)$ work during these periods. To study the behavior of the algorithm during the remainder of its execution, we introduce a combinatorial lemma that is a special case of a well-known decomposition theorem [7] (see also [5]).

Lemma 4.2. *Any integral pseudoflow f' in the residual graph of an integral preflow f in a matching network can be decomposed into cycles and simple paths that are pairwise node-disjoint except at the endpoints of the paths. Each path takes one of the following forms:*

- *from s to t;*
- *from a node v with $e_f(v) > 0$ to a node w with $e_{f+f'}(w) > 0$ (w can be t);*
- *from a node v with $e_f(v) > 0$ to s.*

Lemma 4.2 allows us to show that when $\Gamma_{\max}$ is outside the intervals covered by Lemma 4.1, the amount of excess the algorithm must process is small.

Lemma 4.3. *If $\Gamma(f,d) \ge k > 2$, the total excess that can reach the sink is at most $n/(k-1)$.*

Proof: Let f^* be a maximum flow in G, and let $f' = f^* - f$. f' is a pseudoflow in G_f, and therefore can be decomposed into paths as in Lemma 4.2. Because $\Gamma \geq k$ and d is a valid distance labeling with respect to f, any path from an active node to t in G_f must contain at least $k+1$ nodes. In particular, the excess-to-sink paths of Lemma 4.2 contain at least $k+1$ nodes each, and are node-disjoint except for their endpoints. Since G contains only $n+2$ nodes, there can be no more than $n/(k-1)$ such paths. Since f^* is a maximum flow, the amount of excess that can reach the sink in G_f is no more than $n/(k-1)$. ■

The proof of the next lemma is similar.

Lemma 4.4. *If $\Gamma(f, d) \geq n + k$, the total excess at nodes in V is at most $n/(k-1)$.*

Lemma 4.3 and Lemma 4.4 show that outside the intervals covered by Lemma 4.1, the total excess processed by the algorithm is at most $2n/(k-1)$. To complete the bound on the work expended by the algorithm outside these intervals, we use the following lemma and the fact that at most $O(m)$ work takes place between consecutive global updates to deduce that $O(nm/k)$ time suffices to process the excess outside the intervals covered by Lemma 4.1.

Lemma 4.5. *Between any two consecutive global update operations, at least one unit of excess reaches the source or the sink.*

Proof: For every node v, at least one of $d_s(v)$, $d_t(v)$ is finite. Therefore, immediately after a global update operation, at least one admissible arc leaves every node, by the definition of a global update. Hence the first unit of excess processed by the algorithm immediately after a global update arrives at t or at s before any relabeling occurs. ■

The time bound for the Minimum Distance Discharge algorithm is $O(km + nm/k)$. Choosing $k = \Theta(\sqrt{n})$ to balance the two terms, we see that the Minimum Distance Discharge algorithm with global updates runs in $O(\sqrt{n}m)$ time.

Feder and Motwani [6] give an algorithm that runs in $o(\sqrt{n}m)$ time and produces a "compressed" version $\overline{G}^* = (\overline{V} \cup W, \overline{E}^*)$ of a bipartite graph in which all adjacency information is preserved, but that has asymptotically fewer edges if the original graph $\overline{G} = (\overline{V}, \overline{E})$ is dense. This graph consists of all the original nodes of X and Y, as well as a set of additional nodes W. If an edge $\{x, y\}$ appears in $\overline{E}$, either $\{x, y\} \in \overline{E}^*$ or $\overline{G}^*$ contains a length-two path from x to y through some node of W. It is possible to show that an analogue to Lemma 4.2 holds in such a graph; the paths in the decomposition may not be node-disjoint at nodes of W, but remain so at nodes of X and Y, and this is enough to show that the Minimum Distance Discharge algorithm with graph compression runs in $O\left(\sqrt{n}m\frac{\log(n^2/m)}{\log n}\right)$ time. This bound matches the bound of Feder and Motwani for Dinitz's algorithm.

1. Initialization establishes $|X|$ units of excess, one at each node of X;
2. Nodes of X are relabeled one-by-one, so all $v \in X$ have $d(v) = 1$;
3. While $e_f(t) < |Y|$,
  3.1. a unit of excess moves from some node $v \in X$ to some node $w \in Y$ with $d(w) = 0$;
  3.2. w is relabeled so that $d(w) = 1$;
  3.3. The unit of excess moves from w to t, increasing $e_f(t)$ by one.
4. A single node, x_1 with $e_f(x_1) = 1$, is relabeled so that $d(x_1) = 2$.
5. $\ell \leftarrow 1$.
6. While $\ell \leq n$,
  Remark: All nodes $v \in V$ now have $d(v) = \ell$ with the exception of the one node $x_\ell \in X$, which has $d(x_\ell) = \ell + 1$ and $e_f(x_\ell) = 1$; all excesses are at nodes of X;
  6.1. All nodes with excess, except the single node x_ℓ, are relabeled one-by-one so that all $v \in X$ with $e_f(v) = 1$ have $d(v) = \ell + 1$;
  6.2. While some node $y \in Y$ has $d(y) = \ell$,
    6.2.1. A unit of excess is pushed from a node in X to y;
    6.2.2. y is relabeled so $d(y) = \ell + 1$;
    6.2.3. The unit of excess at y is pushed to a node $x \in X$ with $d(x) = \ell$;
    6.2.4. x is relabeled so that if some node in Y still has distance label ℓ,
      $d(x) = \ell + 1$;
    otherwise
      $d(x) = \ell + 2$ and $x_{\ell+1} \leftarrow x$;
  6.3. $\ell \leftarrow \ell + 1$;
7. Excesses are pushed one-by-one from nodes in X (labeled $n + 1$) to s.

FIGURE 3. The Minimum Distance Discharge execution on bad examples.

5. Minimum Distance Discharge Algorithm without Global Updates

In this section we describe a family of graphs on which the Minimum Distance Discharge algorithm *without* global updates requires $\Omega(nm)$ time (for values of m between $\Theta(n)$ and $\Theta(n^2)$). This shows that the updates improve the worst-case running time of the algorithm.

Given $\tilde{n}$ and $\tilde{m} < \tilde{n}^2/4$, we construct a graph $\overline{G}$ as follows: $\overline{G}$ is the complete bipartite graph with $\overline{V} = X \cup Y$, where

$$X = \left\{ 1, 2, \ldots, \left\lfloor \frac{\tilde{n} - \sqrt{\tilde{n}^2 - 4\tilde{m}}}{2} \right\rfloor \right\} \qquad \text{and} \qquad Y = \left\{ 1, 2, \ldots, \left\lceil \frac{\tilde{n} + \sqrt{\tilde{n}^2 - 4\tilde{m}}}{2} \right\rceil \right\}.$$

It is straightforward to verify that this graph has $n = \tilde{n} + O(1)$ nodes and $m = \tilde{m} + O(\tilde{m}/\tilde{n})$ edges. Note that $|X| > |Y|$.

Figure 3 describes an execution of the Minimum Distance Discharge algorithm on G, the matching network derived from $\overline{G}$, that requires $\Omega(nm)$ time. With more complicated analysis, it is possible to show that every execution of the Minimum Distance Discharge algorithm on G requires $\Omega(nm)$ time.

It is straightforward to verify that in the execution outlined, all processing takes place at active nodes with minimum distance labels among the active nodes. Another important fact is that during the execution, no relabeling changes a distance label by more than two. Hence the execution uses $\Theta(nm)$ work in the course of its $\Theta(n^2)$ relabelings.

6. Minimum Cost Circulation and Assignment Problems

Given a weight function $\overline{c} : \overline{E} \to \mathbf{R}$ and a set of edges M, we define the weight of M to be the sum of weights of edges in M. The *assignment problem* is to find a maximum cardinality matching of minimum weight. We assume that the costs are integers in the range $[0, \ldots, C]$ where $C \geq 1$. (Note that we can always make the costs nonnegative by adding an appropriate number to all arc costs.)

For the minimum cost circulation problem, we adopt the following framework. We are given a graph $G = (V, E)$, with an integer-valued capacity function as before. In addition to the capacity function, we are given an integer-valued *cost* $c(a)$ for each arc $a \in E$.

We assume $c(a) = -c(a^R)$ for every arc a. A *circulation* is a pseudoflow f with the property that $e_f(v) = 0$ for every node $v \in V$. (The absence of a distinguished source and sink accounts for the difference in nomenclature between a circulation and a flow.)

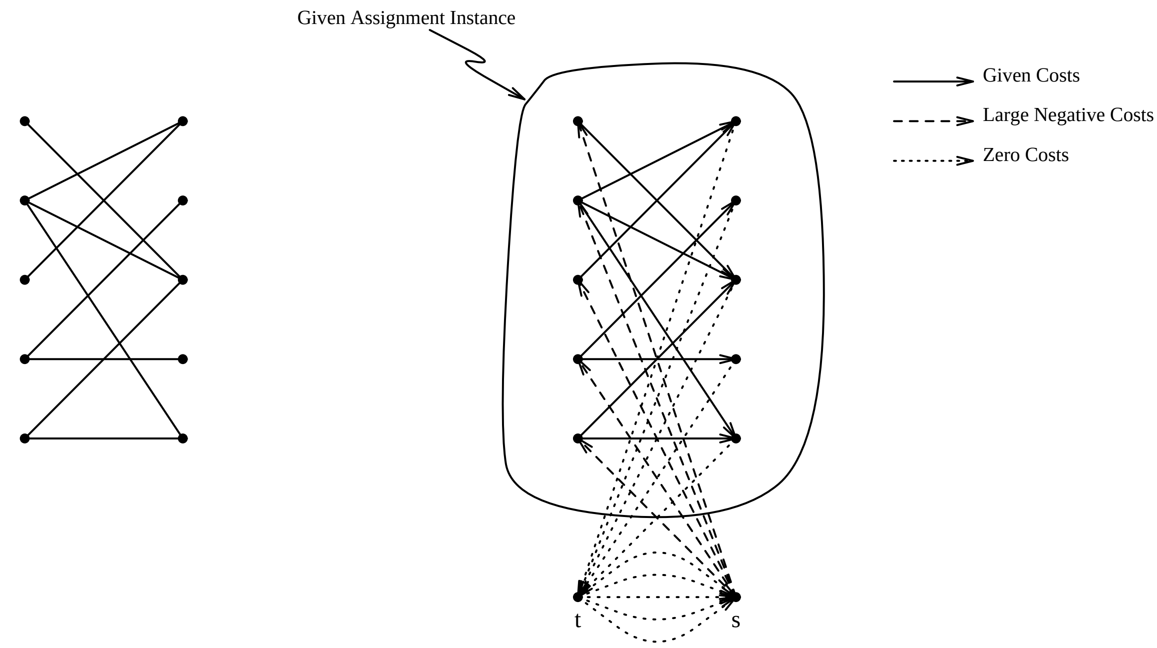
The cost of a pseudoflow f is given by $c(f) = \sum_{f(a)>0} c(a)f(a)$. The *minimum cost circulation problem* is to find a circulation of minimum cost.

7. The Push-Relabel Method for the Assignment Problem

We reduce the assignment problem to the minimum cost circulation problem as follows. As in the unweighted case, we mention only "forward" arcs, each of which we give unit capacity. The "reverse" arcs have zero capacity and obey cost antisymmetry. Given an instance $(\overline{G} = (\overline{V} = X \cup Y, \overline{E}), \overline{c})$ of the assignment problem, we construct an instance $(G = (\{s, t\} \cup V, E), u, c)$ of the minimum cost circulation problem by

- creating special nodes s and t, and setting $V = \overline{V} \cup \{s, t\}$;
- for each node $v \in X$ placing arc (s, v) in E and defining $c(s, v) = -nC$;
- for each node $v \in Y$ placing arc (v, t) in E and defining $c(v, t) = 0$;
- for each edge $\{v, w\} \in \overline{E}$ with $v \in X$ placing arc (v, w) in E and defining $c(v, w) = \overline{c}(v, w)$;
- placing $n/2$ arcs (t, s) in E and defining $c(t, s) = 0$.

If G is obtained by this reduction, we can interpret an integral circulation in G as a matching in $\overline{G}$ just as we did in the bipartite matching case. Further, it is straightforward to verify that a minimum cost circulation in G corresponds to a maximum matching of minimum weight in $\overline{G}$.



Assignment Problem Instance

Corresponding Minimum Cost Circulation Instance

FIGURE 4. Reduction from Assignment to Minimum Cost Circulation

A *price function* is a function $p : V \to \mathbf{R}$. For a given price function p, the *reduced cost* of an arc (v, w) is $c_p(v, w) = c(v, w) + p(v) - p(w)$.

Let $U = X \cup \{t\}$. Note that all arcs in E have one endpoint in U and one endpoint in its complement. Define E_U to be the set of arcs whose tail node is in U.

For a constant $\epsilon \geq 0$, a pseudoflow f is said to be *ϵ-optimal with respect to a price function p* if, for every residual arc $a \in E_f$, we have

$$\begin{cases} a \in E_U \Rightarrow c_p(a) \geq 0, \\ a \notin E_U \Rightarrow c_p(a) \geq -\epsilon. \end{cases}$$

A pseudoflow f is *ϵ-optimal* if f is ϵ-optimal with respect to some price function p. If the arc costs are integers and $\epsilon < 1/n$, any ϵ-optimal circulation is optimal.

For a given f and p, an arc $a \in E_f$ is *admissible* iff

$$\begin{cases} a \in E_U \text{ and } c_p(a) < \epsilon \quad \text{or} \\ a \notin E_U \text{ and } c_p(a) < 0. \end{cases}$$

The *admissible graph* $G_A = (V, E_A)$ is the graph induced by the admissible arcs.

Our asymmetric definitions of ϵ-optimality and admissibility are natural in the context of the assignment problem. They have the benefit that the complementary slackness conditions are violated on $O(n)$ arcs (corresponding to the matched arcs). For the symmetric definition, complementary slackness can be violated on $\Omega(m)$ arcs.

```
procedure MIN-COST(V, E, u, c);
    [initialization]
    ε ← C ; ∀v,  p(v) ← 0;   ∀a,  f(a) ← 0;
    [loop]
    while ε ≥ 1/n do
        (ε, f, p) ← refine(ε, f, p);
    return(f);
end.
```

FIGURE 5. The cost scaling algorithm.

```
procedure REFINE(ε, f, p);
    [initialization]
    ε ← ε/α;
    ∀a ∈ E with c_p(a) < 0,  f(a) ← u(a);
    [loop]
    while f is not a circulation
        apply a push or a relabel operation;
    return(ε, f, p);
end.
```

FIGURE 6. The generic *refine* subroutine.

First we give a high-level description of the successive approximation algorithm (see Figure 5). The algorithm starts with $\epsilon = C$, $f(a) = 0$ for all $a \in E$, and $p(v) = 0$ for all $v \in V$. At the beginning of every iteration, the algorithm divides ϵ by a constant factor α and saturates all arcs a with $c_p(a) < 0$. The iteration modifies f and p so that f is a circulation that is (ϵ/α)-optimal with respect to p. When $\epsilon < 1/n$, f is optimal and the algorithm terminates. The number of iterations of the algorithm is $\lceil \log_\alpha(nC) \rceil$.

Reducing ϵ is the task of the subroutine *refine*. The input to *refine* is ϵ, f, and p such that (except in the first iteration) circulation f is ϵ-optimal with respect to p. The output from *refine* is $\epsilon' = \epsilon/\alpha$, a circulation f, and a price function p such that f is ϵ'-optimal with respect to p. At the first iteration, the zero flow is not C-optimal with respect to the zero price function, but because every simple path in the residual graph has length of at least $-nC$, standard results about *refine* remain true.

The generic *refine* subroutine (described in Figure 6) begins by decreasing the value of ϵ, and setting f to saturate all residual arcs with negative reduced cost.

This converts f into an ϵ-optimal pseudoflow (indeed, into a 0-optimal pseudoflow). Then the subroutine converts f into an ϵ-optimal circulation by applying a sequence of *push* and *relabel* operations, each of which preserves ϵ-optimality. The generic algorithm does not specify the order in which these operations are applied. Next, we describe the *push* and *relabel* operations

PUSH(v, w).
    send a unit of flow from v to w.
end.

RELABEL(V).
    if $v \in U$
        then replace $p(v)$ by $\max_{(v,w)\in E_f}\{\, p(w) - c(v, w) \,\}$
        else replace $p(v)$ by $\max_{(u,v)\in E_f}\{\, p(u) + c(u, v) - \epsilon \,\}$
end.

FIGURE 7. The *push* and *relabel* operations

for the unit-capacity case.

As in the maximum flow case, a *push* operation applies to an admissible arc (v, w) whose tail node v is active, and consists of pushing one unit of flow from v to w. A *relabel* operation applies to an active node v. The operation sets $p(v)$ to the smallest value allowed by the ϵ-optimality constraints, namely $\max_{(v,w)\in E_f}\{p(w) - c(v, w)\}$ if $v \in U$, or $\max_{(v,w)\in E_f}\{p(w) - c(v, w) - \epsilon\}$ otherwise.

The analysis of cost scaling push-relabel algorithms is based on the following facts [12, 14]. During a scaling iteration

(1) no node price increases;
(2) every relabeling decreases a node price by at least ϵ;
(3) for any $v \in V$, $p(v)$ decreases by $O(n\epsilon)$.

8. Global Updates and the Minimum Change Discharge Algorithm

In this section, we generalize the ideas of minimum distance discharge and global updates to the context of the minimum cost circulation problem and analyze the algorithm that embodies these generalizations.

We analyze a single execution of *refine*, and to simplify our notation, we make some assumptions that do not affect the results. We assume that the price function is identically zero at the beginning of the iteration. Our analysis goes through without this assumption, but the required condition can be achieved at no increased asymptotic cost by replacing the arc costs with their reduced costs and setting the node prices to zero in the first step of *refine*.

Under the assumption that each iteration begins with the zero price function, the *price change* of a node v during an iteration is $-p(v)$. By analogy to the matching case, we define $\Gamma(f, p) = \min_{e_f(v)>0}(-p(v))$, and let $\Gamma_{\max}$ denote the maximum value attained by $\Gamma(f, p)$ so far in this iteration. The *minimum change discharge* strategy consists of repeatedly choosing a node v with $p(v) = -\Gamma$ and applying a *push* or *relabel* operation at v.

In the weighted context, a global update takes the form of setting each node price so that there is a path in G_A from every excess to some deficit (a node v with $e_f(v) < 0$) and every node reachable in G_A from a node with excess lies on such a path. This amounts to a modified shortest-paths computation, and can be done in $O(m)$ time using ideas from Dial's work [3]. We perform a global update every time $\Gamma_{\max}$ has increased by at least ϵ since the last global update. We developed global updates from an implementation heuristic for the minimum cost circulation problem [11], but in retrospect, they prove similar in the assignment context to the one-processor Hungarian Search technique developed in [8].

We use essentially the same argument as for the unweighted case to analyze the part of the algorithm's execution when $\Gamma_{\max}$ is small.

Lemma 8.1. *The Minimum Change Discharge algorithm uses $O((j-i)m)$ work during the period beginning when Γ first exceeds $i-1$ and ending when Γ first exceeds j.*

Proof: Similar to Lemma 4.1. ■

When $\Gamma_{\max}$ is large, the argument we used in the unweighted case does not generalize because it is not true that $-p(v)$ gives a bound on the breadth-first-search distance from v to a deficit in the residual graph. Let $\mathcal{E}(f)$ denote the total excess in pseudoflow f, *i.e.*, $\sum_{e_f(v)>0} e_f(v)$.

The following lemma is analogous to Lemma 4.2.

Lemma 8.2. *Given a matching network G and a circulation g, any pseudoflow f in G_g can be decomposed into*

- *cycles and*
- *paths, each from a node u with $e_f(u) < 0$ to a node v with $e_f(v) > 0$,*

where all the elements of the decomposition are pairwise node-disjoint except at the endpoints of the paths, and each element carries one unit of flow.

We denote a path from node u to node v in such a decomposition by $(u \rightsquigarrow v)$.

The following lemma is similar in spirit to those in [8] and [12], although the single-phase push-relabel framework of our algorithm changes the structure of the proof.

Lemma 8.3. *At any point during* refine, *$\mathcal{E}(f) \times \Gamma_{max} \leq ((3+\alpha)n+2)\epsilon$.*

Proof: Let c denote the (reduced) arc cost function at the beginning of this execution of *refine*, and let $G = (V, E)$ denote the residual graph at the same instant. For simplicity in the following analysis, we view a pseudoflow as an entity in this graph G. Let f, p be the current pseudoflow and price function at the most recent point during the execution of *refine* when

$\Gamma(f,p) = \Gamma_{\max}$. Then we have

$$\mathcal{E}(f) \times \Gamma_{\max} \leq -\sum_{e_f(v)>0} p(v)e_f(v).$$

We will complete our proof by showing that

$$\sum_{e_f(v)>0} p(v)e_f(v) = c_p(f) - c(f)$$

and then deriving an upper bound on this quantity.

By the definition of the reduced costs,

$$c_p(f) - c(f) = \sum_{f(v,w)>0} (p(v) - p(w))f(v,w).$$

Letting $\mathcal{P}$ be a decomposition of f into paths and cycles according to Lemma 8.2 and noting that cycles make no contribution to the sum, we can rewrite this expression as

$$\sum_{(u \rightsquigarrow v)\in\mathcal{P}} (p(u) - p(v)).$$

Since nodes u with $e_f(u) < 0$ are never relabeled, $p(u) = 0$ for such a node, and we have

$$c_p(f) - c(f) = -\sum_{(u \rightsquigarrow v)\in\mathcal{P}} p(v).$$

Because the decomposition $\mathcal{P}$ must account for all of f's excesses and deficits, we can rewrite

$$c_p(f) - c(f) = -\sum_{e_f(v)>0} p(v)e_f(v).$$

Now we derive an upper bound on $c_p(f) - c(f)$. It is straightforward to verify that for any matching network G and integral circulation g, G_g has exactly n arcs $a \notin E_U$, and so from the fact that the execution of *refine* begins with the residual graph of an $(\alpha\epsilon)$-optimal circulation, we deduce that there are at most n negative-cost arcs in E. Because each of these arcs has cost at least $-\alpha\epsilon$, we have $c(f) \geq -\alpha n\epsilon$. Hence $c_p(f) - c(f) \leq c_p(f) + \alpha n\epsilon$.

Now consider $c_p(f)$. Clearly, $f(a) > 0 \implies a^R \in E_f$, and ϵ-optimality of f with respect to p says that $a^R \in E_f \implies c_p(a^R) \geq -\epsilon$. Since $c_p(a^R) = -c_p(a)$, we have $f(a) > 0 \implies c_p(a) \leq \epsilon$. Now by Lemma 8.2, f can be decomposed into cycles and paths from deficits to excesses. Let $\mathcal{P}$ denote this decomposition, and observe that $c_p(f) = \sum_{P\in\mathcal{P}} c_p(P)$. Let $\nu(P)$ denote the interior of a path P, *i.e.*, the path minus its endpoints and initial and final arcs, and let $\partial(P)$ denote the set containing the initial and final arcs of P. If P is a cycle, $\nu(P) = P$ and $\partial(P) = \emptyset$. We can write

$$c_p(f) = \sum_{P\in\mathcal{P}} c_p(\nu(P)) + \sum_{P\in\mathcal{P}} c_p(\partial(P)).$$

The total number of arcs in the cycles and path interiors is at most $n+2$, by node-disjointness. Also, the total excess is never more than n, so the initial and final arcs of the paths number no more than $2n$. And because each arc carrying positive flow has reduced cost at most ϵ, we have $c_p(f) \leq (3n+2)\epsilon$.

Therefore, $c_p(f) - c(f) \leq ((3+\alpha)n+2)\epsilon$. ■

Now to complete our time bound, we use the following lemma.

Lemma 8.4. *Between any two consecutive global update operations, at least one unit of excess reaches a deficit.*

Proof: This lemma is a simple consequence of the ϵ-optimality of f with respect to p. In particular, the definition of ϵ-optimality implies that no *push* operation can move a unit of excess from a node to another node with higher price change, and indeed, two consecutive *push* operations on any given unit of excess suffice to move the excess to some node with strictly lower price change. By the definition of a global update operation, these properties suffice to ensure that a unit of excess reaches some deficit immediately after a global update, and before any relabeling occurs. ■

Lemma 8.3 shows that when $\Gamma_{\max} \geq k$, the total excess remaining is $O(n/k)$. Lemma 8.4 shows that $O(m)$ work suffices to cancel each unit of excess remaining. As in the unweighted case, the total work in an execution of *refine* is $O(mk + nm/k)$, and choosing $k = \Theta(\sqrt{n})$ gives a $O(\sqrt{n}m)$ time bound on an execution of *refine*. The overall time bound follows from the $O(\log(nC))$ bound on the number of scaling iterations.

Graph compression methods [6] do not apply to graphs with weights because the compressed graph preserves only adjacency information and cannot encode arbitrary edge weights. Hence the Feder-Motwani techniques do not apply in the assignment problem context.

9. Minimum Change Discharge Algorithm without Global Updates

We present a family of assignment instances on which we show *refine* without global updates performs $\Omega(nm)$ work in the first scaling iteration, under the minimum distance discharge selection rule. Hence this family of matching networks suffices to show that global updates account for an asymptotic difference in running time.

The family of assignment instances on which we show *refine* without global updates takes $\Omega(nm)$ time is structurally the same as the family of bad examples we used in the unweighted case, except that they are have two additional nodes and one additional edge. The costs of the edges present in the unweighted example are zero, and there are two extra nodes connected only to each other, sharing an edge with cost α.

At the beginning of the first scaling iteration, $\epsilon = \alpha$. The execution starts by setting $\epsilon = 1$. From this point on, the execution of *refine* restricted to the nodes and arcs present in the unweighted example parallels the execution of the maximum flow algorithm detailed in Section 5.

10. Conclusions and Open Questions

We have given algorithms that achieve the best time bounds known for bipartite matching, namely $O\left(\sqrt{n}m\frac{\log(n^2/m)}{\log n}\right)$, and for the assignment problem in the cost scaling context, namely $O\left(\sqrt{n}m\log(nC)\right)$. We have also given examples to show that without global updates, the algorithms perform worse. Hence we conclude that global updates can be a useful tool in theoretical development of algorithms.

We have shown a family of assignment instances on which *refine* performs poorly, but our proof seems to hinge on details of the reduction, and so it applies only in the first scaling iteration. An interesting open question is the existence of a family of instances of the assignment problem on which *refine* uses $\Omega(nm)$ time in *every* scaling iteration.

References

1. R. J. Anderson and J. C. Setubal. Goldberg's Algorithm for the Maximum Flow in Perspective: a Computational Study. In D. S. Johnson and C. C. McGeoch, editors, *Network Flows and Matching: First DIMACS Implementation Challenge*, pages 1–18. AMS, 1993.
2. U. Derigs and W. Meier. Implementing Goldberg's Max-Flow Algorithm — A Computational Investigation. *ZOR — Methods and Models of Operations Research*, 33:383–403, 1989.
3. R. B. Dial. Algorithm 360: Shortest Path Forest with Topological Ordering. *Comm. ACM*, 12:632–633, 1969.
4. E. A. Dinic. Algorithm for Solution of a Problem of Maximum Flow in Networks with Power Estimation. *Soviet Math. Dokl.*, 11:1277–1280, 1970.
5. S. Even and R. E. Tarjan. Network Flow and Testing Graph Connectivity. *SIAM J. Comput.*, 4:507–518, 1975.
6. T. Feder and R. Motwani. Clique Partitions, Graph Compression and Speeding-up Algorithms. In *Proc. 23st Annual ACM Symposium on Theory of Computing*, pages 123–133, 1991.
7. L. R. Ford, Jr. and D. R. Fulkerson. *Flows in Networks.* Princeton Univ. Press, Princeton, NJ, 1962.
8. H. N. Gabow and R. E. Tarjan. Almost-Optimal Speed-ups of Algorithms for Matching and Related Problems. In *Proc. 20th Annual ACM Symposium on Theory of Computing*, pages 514–527, 1988.
9. H. N. Gabow and R. E. Tarjan. Faster Scaling Algorithms for Network Problems. *SIAM J. Comput.*, pages 1013–1036, 1989.
10. A. V. Goldberg. *Efficient Graph Algorithms for Sequential and Parallel Computers.* PhD thesis, M.I.T., January 1987. (Also available as Technical Report TR-374, Lab. for Computer Science, M.I.T., 1987).
11. A. V. Goldberg. An Efficient Implementation of a Scaling Minimum-Cost Flow Algorithm. In *Proc. 3rd Integer Prog. and Combinatorial Opt. Conf.*, pages 251–266, 1993.
12. A. V. Goldberg, S. A. Plotkin, and P. M. Vaidya. Sublinear-Time Parallel Algorithms for Matching and Related Problems. *J. Algorithms*, 14:180–213, 1993.
13. A. V. Goldberg and R. E. Tarjan. A New Approach to the Maximum Flow Problem. *J. Assoc. Comput. Mach.*, 35:921–940, 1988.

14. A. V. Goldberg and R. E. Tarjan. Finding Minimum-Cost Circulations by Successive Approximation. *Math. of Oper. Res.*, 15:430–466, 1990.
15. J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ Algorithm for Maximum Matching in Bipartite Graphs. *SIAM J. Comput.*, 2:225–231, 1973.
16. A. V. Karzanov. O nakhozhdenii maksimal'nogo potoka v setyakh spetsial'nogo vida i nekotorykh prilozheniyakh. In *Matematicheskie Voprosy Upravleniya Proizvodstvom*, volume 5. Moscow State University Press, Moscow, 1973. In Russian; title translation: On Finding Maximum Flows in Network with Special Structure and Some Applications.
17. H. W. Kuhn. The Hungarian Method for the Assignment Problem. *Naval Res. Logist. Quart.*, 2:83–97, 1955.
18. Q. C. Nguyen and V. Venkateswaran. Implementations of Goldberg-Tarjan Maximum Flow Algorithm. In D. S. Johnson and C. C. McGeoch, editors, *Network Flows and Matching: First DIMACS Implementation Challenge*, pages 19–42. AMS, 1993.
19. J. B. Orlin and R. K. Ahuja. New Scaling Algorithms for the Assignment and Minimum Cycle Mean Problems. *Math. Prog.*, 54:41–56, 1992.