GLOBAL PRICE UPDATES HELP*

ANDREW V. GOLDBERG† AND ROBERT KENNEDY‡

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 uses global updates runs in $O\left(\sqrt{n}m\frac{\log(n^2/m)}{\log n}\right)$ time (matching the best bound known) and performs worse by a factor of $\sqrt{n}$ without the updates. A similar result holds for the assignment problem, for which an algorithm that assumes integer costs in the range $[\,-C,\ldots,C\,]$ and that runs in time $O\big(\sqrt{n}m\log(nC)\big)$ (matching the best cost-scaling bound known) is presented.

Key words. assignment problem, cost scaling, bipartite matching, dual update, push-relabel algorithm, zero-one flow

AMS subject classifications. 90C08, 68Q20, 68R10

1. Introduction. The push-relabel method [10, 13] is the best currently known way for solving the maximum flow problem [1, 2, 19]. 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] The first algorithm proved to achieve this bound was proposed by Hopcroft and Karp [15]. Karzanov [17, 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, 20].

Feder and Motwani [6] give a "graph compression" technique that combines with the algorithm of Dinitz to yield an $O\left(\sqrt{n}m\frac{\log(n^2/m)}{\log n}\right)$ 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 $O\big(n(m+n\log n)\big)$ is achieved by the classical Hungarian method of Kuhn [18]. 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 $O\big(\sqrt{n}m\log(nC)\big)$ 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, 20]. The first phase of these algorithms is based on the push-relabel method and the second phase is based on the successive augmentation approach. Our algorithm for the assignment problem runs in $O\big(\sqrt{n}m\log(nC)\big)$, and like the other algorithms

*This work was done while both authors were at the Computer Science Department, Stanford University, Stanford, California.

†NEC Research Institute, Inc., Princeton, NJ 08540, USA (avg@research.nj.nec.com). Supported in part by ONR Young Investigator Award N00014-91-J-1855 and NSF Grant CCR-9307045.

‡Silicon Graphics, Inc., 2011 North Shoreline Blvd., MS 178, Mountain View, CA 94043, USA (robert@cs.stanford.edu). Supported by the above-mentioned ONR and NSF grants.

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

with this time bound, it is based on cost-scaling, assumes the input costs are integers, and is not strongly polynomial.

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 *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 an $O(\sqrt{n}m)$ time bound for the bipartite matching algorithm with global updates, and in Section 5 we show how to apply Feder and Motwani's techniques to improve the algorithm's performance to $O\left(\sqrt{n}m\frac{\log(n^2/m)}{\log n}\right)$. Section 6 shows that without global updates, the bipartite matching algorithm performs poorly. Section 7 gives definitions relevant to the assignment problem and minimum cost flow. In Section 8, we describe the cost-scaling push-relabel method for minimum cost flow and apply the method to the assignment problem. Sections 9 and 10 generalize the bipartite matching results to the assignment problem. In Section 11, 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 $n = |\overline{V}| + 2$ (the additive constant being for notational convenience in the reduction to come), 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);
- $f(a) \leq u(a)$ (*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

[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.

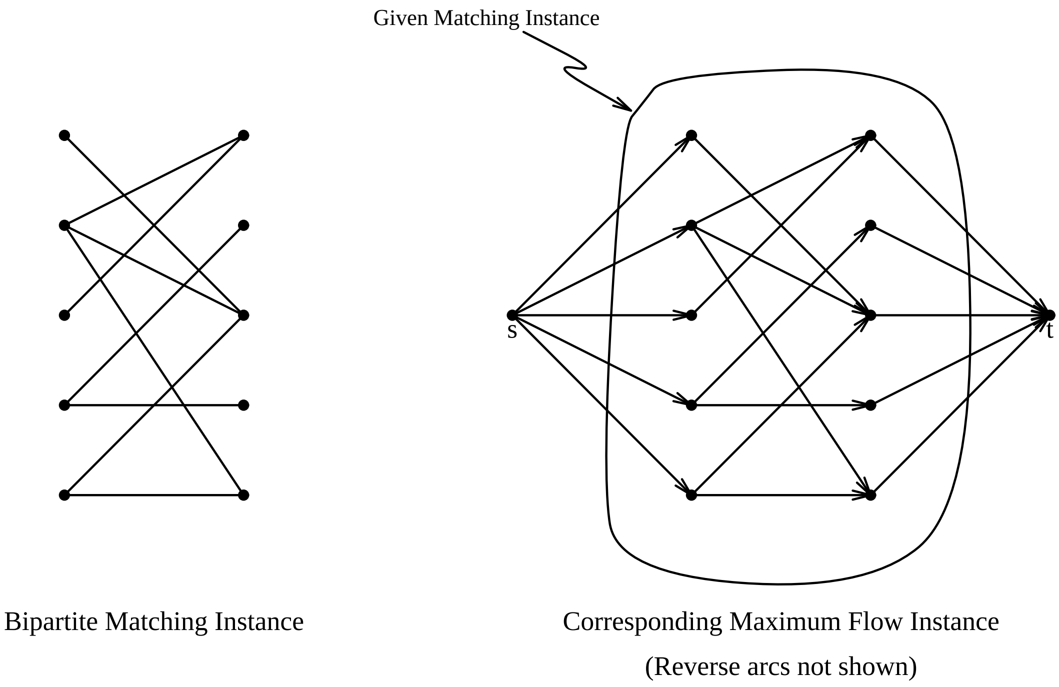

FIG. 3.1. *Reduction from Bipartite Matching to Maximum Flow*

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;
- for each edge $\{v, w\} \in \overline{E}$ with $v \in X$ and $w \in Y$ 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. The *augmented residual graph* $G_f^{=}$ has the same nodes and arcs as G, but is associated with the capacity function u_f. The point of defining $G_f^{=}$ is so we can meaningfully discuss pseudoflows that obey the residual capacity constraints. Since the residual graph lacks arcs a with $u_f(a) = 0$, it can lack reverse arcs that are assumed by the definition of a pseudoflow.

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

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.

FIG. 3.2. *The* push *and* relabel *operations*

called *admissible* arcs, and the *admissible graph* $G_A = (V, E_A)$ is the graph induced by the admissible arcs. It is straightforward to see that G_A is acyclic for any valid distance labeling.

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.

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 3.2.

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).

- For all nodes v, we have $0 \le d(v) \le 2n$.
- Distance labels do not decrease during the computation.
- *relabel*(v) increases $d(v)$.
- The number of *relabel* operations during the computation is $O(n)$ per node.
- The work involved in *relabel* operations is $O(nm)$.
- 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).
- The number of *push* operations during the computation is $O(nm)$.

The above facts imply 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 the following simple data structure (see [13] for details). We maintain a *current arc* for every node. Initially the first arc in the node's arc list is current. When pushing flow excess out of a node v, we push it on v's current arc if the arc is admissible, or advance the current arc to the next arc on the arc list. When there are no more arcs on the list, we relabel v and set v's current arc to the first arc on v's arc list.

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 (directed) 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 $v \in V$ is called a *global update operation.* This operation also sets the current arc of every node to the node's first arc. Such an operation can be accomplished with $O(m)$ work that amounts to two breadth-first-search computations. Validity of the resulting distance labeling is a straightforward consequence of the definition. Note that a global update cannot decrease any node's distance label [13].

The ordering of operations we use is called *Minimum Distance Discharge*; 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 the sake of efficient implementation and easy generalization to the weighted case, we formulate this selection strategy in a slightly different (but equivalent) way and use this formulation to guide the implementation and analysis. The intuition is that we select a unit of excess at an active node with minimum distance label, and process that unit of excess until a relabeling occurs or the excess reaches s or t. In the event of a relabeling, the new distance label may be small enough to guarantee that the same excess still has the minimum label; if so, we avoid the work associated with finding the next excess to process. This scheme's important properties generalize to the weighted case, and it allows us to show easily that the work done in active node selection is not too great.

To implement this selection rule, we maintain a collection of buckets, $b_0, \ldots, b_{2n}$; each b_i contains the active nodes with distance label i, except possibly one which is currently being processed. During the execution, we maintain μ, the index of the bucket from which we selected the most recent unit of excess. When we relabel a node, if the new distance label is no more than μ, we know that node still has minimum distance label among the active nodes, so we continue processing the same unit of excess.

In addition, we perform periodic global updates. The first global update is performed immediately after the preflow is initialized. After each *push* and *relabel* operation, the algorithm checks the following two conditions and performs a global update if both conditions hold:

- Since the most recent update, at least one unit of excess has reached s or t; and
- Since the most recent update, the algorithm has done at least m work in *push* and *relabel* operations.

Immediately after each global update, we rebuild the buckets in $O(n)$ time and set μ to zero. The following lemma shows that the algorithm does little extra work in selecting nodes to process.

LEMMA 4.1. *Between two consecutive global updates, the algorithm does $O(n)$ work in examining empty buckets.*

Proof. Immediate, because μ decreases only when it is set to zero after an update, and there are $2n + 1 = O(n)$ buckets. $\square$

We will 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. Note that $\Gamma_{\max}$ is often

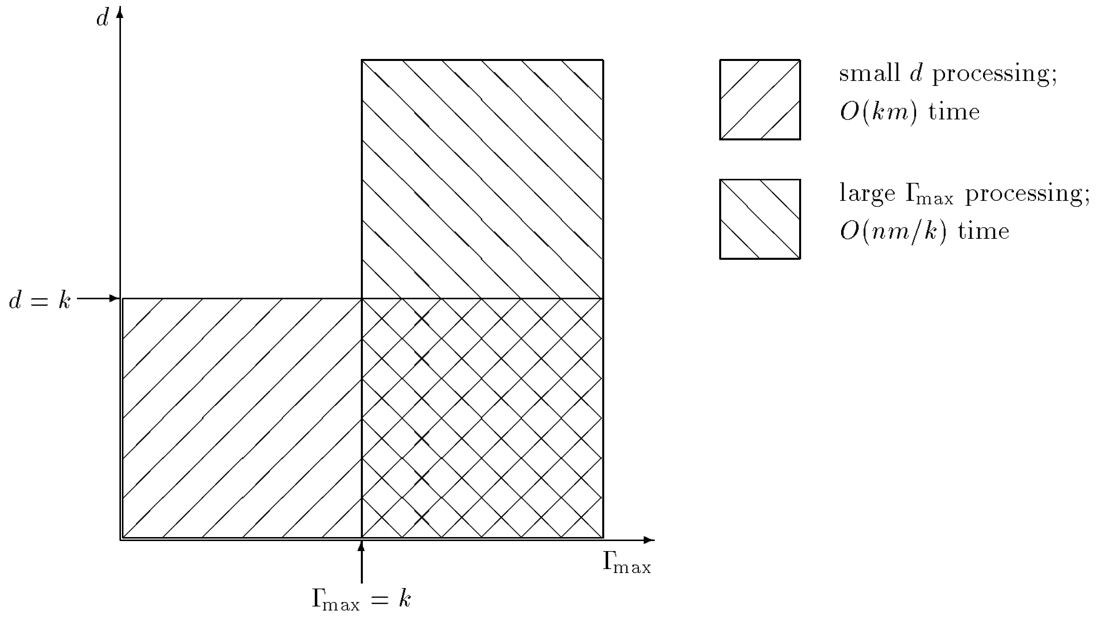

FIG. 4.1. *Accounting for work when* $0 \leq \Gamma_{\max} \leq n$.

equal to μ; we use the separate names mainly to emphasize that μ is maintained by the implementation, while $\Gamma_{\max}$ is an abstract quantity with relevance to the analysis regardless of the implementation details.

Figure 4.1 represents the structure underlying our analysis of the Minimum Distance Discharge algorithm. (Strictly speaking, the figure shows only half of the analysis; the part when $\Gamma_{\max} > n$ is essentially similar.) The horizontal axis corresponds to the value of $\Gamma_{\max}$ which increases as the algorithm proceeds, and the vertical axis corresponds to the distance label of the node currently being processed. Our analysis hinges on a parameter k in the range $2 \leq k \leq n$, 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 t return to s. We analyze the first stage of each pair using the following lemma.

LEMMA 4.2. *The Minimum Distance Discharge algorithm expends $O(km)$ work during the periods when $\Gamma_{\max} \in [0, k]$ and $\Gamma_{\max} \in [n, n+k]$.*

Proof. First, note that if $\Gamma_{\max}$ falls in the first interval of interest, Γ must lie in that interval as well. This relationship also holds for the second interval after a global update is performed, since $\Gamma_{\max} \geq n$ implies that no excess can reach t. Because the work from the beginning of the second interval until the price update is performed is $O(m)$, it is enough to show that the time spent by the algorithm during periods when $\Gamma \in [0, k]$ and $\Gamma \in [n, n+k]$ is in $O(km)$. Note that the periods defined in terms of Γ may not represent contiguous intervals during the execution of the algorithm.

Each node can be relabeled at most $k+1$ times when $\Gamma \in [0, k]$, and similarly for $\Gamma \in [n, n+k]$. Hence the relabelings and pushes require $O(km)$ work. The observations that a global update requires $O(m)$ work and during each period there are $O(k)$ global updates complete the proof. ∎

To study the behavior of the algorithm during the remainder of its execution, we exploit the structure of matching networks by appealing to a combinatorial lemma. The following lemma is a special case of a well-known decomposition theorem [7] (see also [5]). The proof depends mainly on the fact that for a matching network G, the in-degree of $v \in X$ in G_f is $1 - e_f(v)$ and the out-degree of $w \in Y$ in G_f is $1 + e_f(w)$ for any integral pseudoflow f.

LEMMA 4.3. *Any integral pseudoflow f in the augmented residual graph of an integral flow g in a matching network can be decomposed into cycles and simple paths that are pairwise node-disjoint except at the endpoints of the paths, such that each element in the decomposition carries one unit of flow. Each path is from a node v with $e_f(v) < 0$ (v can be t) to a node w with $e_f(w) > 0$ (w can be s).*

Lemma 4.3 allows us to show that when $\Gamma_{\max}$ is outside the intervals covered by Lemma 4.2, the amount of excess the algorithm must process is small.

Given a preflow f, we define the *residual flow value* to be the total excess that can reach t in G_f.

LEMMA 4.4. *If $\Gamma_{\max} \geq k > 2$, the residual flow value is at most $n/(k-1)$ if G is a matching network.*

Proof. Note that the residual flow value never increases during an execution of the algorithm, and consider the pair (f, d) such that $\Gamma(f, d) \geq k$ for the first time during the execution. Let f^* be a maximum flow in G, and let $f' = f^* - f$. Now $-f'$ is a pseudoflow in $G^{=}_{\overline{f^*}}$, and therefore can be decomposed into cycles and paths as in Lemma 4.3. Such a decomposition of $-f'$ induces the obvious decomposition on f' with all the paths and cycles reversed and excesses negated. Because $\Gamma \geq k$ and d is a valid distance labeling with respect to f, any path in G_f from an active node to t must contain at least $k+1$ nodes. In particular, the excess-to-t paths in the decomposition of f' contain at least $k+1$ nodes each, and are node-disjoint except for their endpoints. Since G contains only n nodes, there can be no more than $(n-2)/(k-1) < n/(k-1)$ such paths. Since f^* is a maximum flow, the amount of excess that can reach t in G_f is no more than $n/(k-1)$. ∎

The proof of the next lemma is similar.

LEMMA 4.5. *If $\Gamma_{\max} \geq n + k > n + 2$ during an execution of the Minimum Distance Discharge algorithm with global updates on a matching network, the total excess at nodes in V is at most $n/(k-1)$.*

The following lemma shows an important property of the rules we use to trigger global update operations, namely that during a period when the algorithm does $\Theta(m)$ work at least one unit of excess is guaranteed to reach s or t.

LEMMA 4.6. *Between any two consecutive global update operations, the algorithm does $\Theta(m)$ work.*

Proof. According to the two conditions that trigger a global update, it suffices to show that immediately after an update, the work done in moving a unit of excess to s or t is $O(m)$. For every node v, at least one of $d_s(v)$, $d_t(v)$ is finite. Therefore, immediately after a global update, at least one admissible arc leaves every node except s and t, by definition of the global update operation. Recall that the admissible graph is acyclic, so the first unit of excess processed by the algorithm immediately after a global update arrives at t or at s before any relabeling occurs, and does so along a simple path. Consider the path taken by the flow unit to s or t. The work performed while moving the unit along the path is proportional to the length of the path plus the number of times current arcs of nodes on the path are advanced. This $O(n+m) = O(m)$ work is performed before the the first condition for a global update is met.

Following an amount of additional work bounded above by $m + O(n)$, plus work proportional to that for a *push* or *relabel* operation, another global update operation will be triggered. Clearly a *push* or *relabel* takes $O(m)$ work and the lemma follows. $\square$

We are ready to prove the main result of this section.

THEOREM 4.7. *The Minimum Distance Discharge algorithm with global updates computes a maximum flow in a matching network (and hence a maximum cardinality bipartite matching) in $O(\sqrt{n}m)$ time.*

Proof. By Lemma 4.2, the total work done by the algorithm when $\Gamma_{\max} \in [0, k]$ and $\Gamma_{\max} \in [n, n+k]$ is $O(km)$. By Lemmas 4.4 and 4.5, the amount of excess processed when $\Gamma_{\max}$ falls outside these bounds is at most $2n/(k-1)$. From Lemma 4.6 we conclude that the work done in processing this excess is $O(nm/k)$. Hence 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. $\square$

5. Improved Performance through Graph Compression. Feder and Motwani [6] give an algorithm that runs in $o(\sqrt{n}m)$ time and produces a *compressed representation* $\overline{G}^* = (\overline{V} \cup W, \overline{E}^*)$ 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. An edge $\{x, y\}$ appears in $\overline{E}$ if and only if either $\{x, y\} \in \overline{E}^*$ or $\overline{G}^*$ contains a length-two path from x to y through some node of W.

The following theorem is slightly specialized from Feder and Motwani's Theorem 3.1 [6], which details the performance of their algorithm *Compress*:

THEOREM 5.1. *Let $\delta \in (0, 1)$ and let $\overline{G} = (\overline{V} = X \cup Y, \overline{E})$ be an undirected bipartite graph with $|X| = |Y| = n$ and $|\overline{E}| = m \geq n^{2-\delta}$. Then algorithm* Compress *computes a compressed representation $\overline{G}^* = (\overline{V} \cup W, \overline{E}^*)$ of $\overline{G}$ with $m^* = |\overline{E}^*| = O\left(m\delta^{-1}\frac{\log(n^2/m)}{\log n}\right)$ in time $O(mn^\delta \log^2 n)$. The number of nodes in W is $O(mn^{\delta-1})$.*

In particular, we choose a constant $\delta < 1/2$; then the compressed representation is computed in time $o(\sqrt{n}m)$ and has $m^* = O\left(m\frac{\log(n^2/m)}{\log n}\right)$ edges.

Given a compressed representation $\overline{G}^*$ of $\overline{G}$, we can compute a flow network G^* in which there is a correspondence between flows in G^* and matchings in $\overline{G}$. The only differences from the reduction of Section 3 are that each edge $\{x, w\}$ with $x \in X$ and $w \in W$ gives an arc (x, w), and each edge $\{w, y\}$ with $w \in W$ and $y \in Y$ gives an arc (w, y). As in Section 3, we have a relationship between matchings in the original graph $\overline{G}$ and flows in G^*, but now the correspondence is not one-to-one as it was before. Nevertheless, it remains true here that given a flow f with $e_f(t) = c$ in G^*, we can find a matching of cardinality c in $\overline{G}$ using only $O(n)$ time in a straightforward way.

The performance improvement we gain comes by using the graph compression step as preprocessing: we will show that the Minimum Distance Discharge algorithm with global updates runs in time $O(\sqrt{n}m^*)$ on the flow network G^* corresponding to the compressed representation $\overline{G}^*$ of a bipartite graph $\overline{G}$. In other words, the speedup results only from the reduced number of edges, not from changes within the Minimum Distance Discharge algorithm.

To prove the performance bound, we must generalize certain lemmas from Section 4 to networks with the structure of compressed representations. Let $n^* = n + |W|$

be the number of nodes in the maximum flow problem derived from the compressed representation of the input graph. Lemma 4.2 is independent of the input network's structure, as are Lemma 4.6 and Lemma 4.1. These three lemmas give us their conclusions for compressed representations where we substitute n^* for n and m^* for m in their statements and proofs. An analogue to Lemma 4.3 holds in a flow network derived from a compressed representation; this will extend Lemmas 4.4 and 4.5, allowing us to conclude the improved time bound.

LEMMA 5.2. *Any integral pseudoflow f in the augmented residual graph of an integral flow g in the flow graph of a compressed representation can be decomposed into cycles and simple paths that are pairwise node-disjoint at nodes of X and Y except at the endpoints of the paths, such that each element of the decomposition carries one unit of flow. Each path is from a node v with $e_f(v) < 0$ (v can be t) to a node w with $e_f(w) > 0$ (w can be s).*

Proof. As with matching networks, the in-degree of $v \in X$ is $1 - e_f(v)$ and the out-degree of $y \in Y$ is $1 + e_f(y)$, so the standard proof of Lemma 4.3 extends to this case. ☐

The following lemma is analogous to Lemma 4.4.

LEMMA 5.3. *If $\Gamma_{\max} \geq k > 2$, the residual flow value is at most $2n/(k-2)$ if G^* is a compressed representation.*

Proof. As in the case of Lemma 4.4, except that here an excess-to-t path in the decomposition of f' must contain at least $k/2$ nodes of $\overline{V}$. Since $\overline{V}$ contains only n nodes, there can be no more than $2n/(k-2)$ such paths, and so because f^* is a maximum flow, the amount of excess that can reach t in G_f^* is no more than $2n/(k-2)$. ☐

The following lemma is analogous to Lemma 4.5, and its proof is similar to the proof of Lemma 5.3.

LEMMA 5.4. *If $\Gamma_{\max} \geq n^* + k > n^* + 2$ during an execution of the Minimum Distance Discharge algorithm with global updates on a compressed representation, the total excess at nodes in $\overline{V} \cup W$ is at most $2n/(k-2)$.*

Using the same reasoning as in Theorem 4.7, we have:

THEOREM 5.5. *The Minimum Distance Discharge algorithm with global updates computes a maximum flow in the network corresponding to a compressed representation with m^* edges in $O(\sqrt{n}m^*)$ time.*

To complete our time bound for the bipartite matching problem we must dispense with some technical restrictions in Theorem 5.1, namely the requirements that $|X| = |Y| = n$ and that $m \geq n^{2-\delta}$. The former condition is easily met by adding nodes to whichever of X, Y is the smaller set, so their cardinalities are equal. These "dummy" nodes are incident to no edges. As for the remaining condition, observe that our time bound does not suffer if we simply forego the compression step and apply the result of Section 4 in the case where $m < n^{2-\delta}$. To see this, recall that we chose $\delta < 1/2$, and note that $1 \leq m < n^{2-\delta}$ implies $\frac{\log(n^2/m)}{\log n} = \Theta(1)$. So we have:

THEOREM 5.6. *The Minimum Distance Discharge algorithm with graph compression and global updates computes a maximum cardinality bipartite matching in $O\left(\sqrt{n}m\frac{\log(n^2/m)}{\log n}\right)$ time.* This bound matches that of Feder and Motwani for Dinitz's algorithm.

6. 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

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.

FIG. 6.1. *The Minimum Distance Discharge execution on bad examples.*

$\Theta(n)$ and $\Theta(n^2)$). This shows that the updates improve the worst-case running time of the algorithm. The goal of our construction is to admit an execution of the algorithm in which each relabeling changes a node's distance label by $O(1)$. Under this condition the execution will have to perform $\Omega(n^2)$ relabelings, and these relabelings will require $\Omega(nm)$ time.

Given $\tilde{n} \in \mathbf{Z}$ and $\tilde{m} \in [1, \tilde{n}^2/4]$, we construct a graph $\overline{G}$ as follows: $\overline{G}$ is the complete bipartite graph with $\overline{V} = X \cup Y$, where

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

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

Figure 6.1 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 but unilluminating 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 whose distance labels are minimum among the active nodes. The algorithm performs poorly because 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, and we have the following theorem:

THEOREM 6.1. *For any function $m(n)$ in the range $n \leq m(n) < n^2/4$, there exists an infinite family of instances of the bipartite matching problem having $\Theta(n)$ nodes and $\Theta(m(n))$ edges on which the Minimum Distance Discharge algorithm without global updates runs in $\Omega(nm(n))$ time.*

7. 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.) We will say that a node v with $e_f(v) < 0$ has a *deficit*.

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.

8. 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 $\big(\overline{G} = (\overline{V} = X \cup Y, \overline{E}), \overline{c}\big)$ of the assignment problem, we construct an instance $\big(G = (\{s,t\} \cup V, E), u, c\big)$ 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 easy to verify that a minimum cost circulation in G corresponds to a maximum matching of minimum weight in $\overline{G}$.

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

$$\begin{cases} a \in E_U \Rightarrow c_p(a) \geq 0, \\ a \notin E_U \Rightarrow c_p(a) \geq -2\epsilon. \end{cases}$$

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.

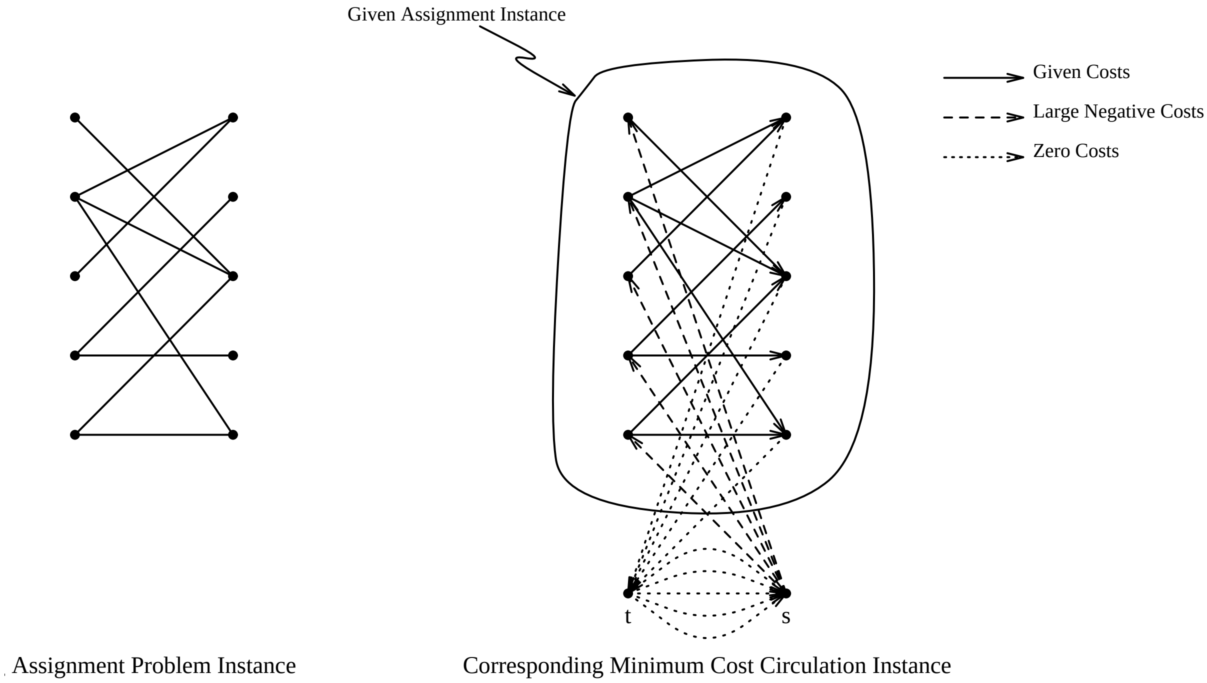

FIG. 8.1. *Reduction from Assignment to Minimum Cost Circulation*

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

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

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

These 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.

First we give a high-level description of the successive approximation algorithm (see Figure 8.2). 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 $1 + \lfloor \log_\alpha(nC) \rfloor$.

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 cost of at least $-nC$, standard results about *refine* remain true.

The generic *refine* subroutine (described in Figure 8.3) begins by decreasing the value of ϵ, and setting f to saturate all residual arcs with negative reduced cost.

```
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.
```

FIG. 8.2. *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.
```

FIG. 8.3. *The generic* refine *subroutine.*

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 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 that is not the tail of any admissible arc. 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 $\max_{(v,w)\in E_f}\{p(w) - c(v,w) - 2\epsilon\}$ otherwise.

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

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

9. 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

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 $\max_{(v,w)\in E_f}\{p(w) - c(v,w) - 2\epsilon\}$
end.

FIG. 8.4. *The* push *and* relabel *operations*

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 $\delta(v) = \lfloor -p(v)/\epsilon \rfloor$. By analogy to the matching case, we define $\Gamma(f,p) = \min_{e_f(v)>0}\{\delta(v)\}$, 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 selecting a unit of excess at an active node v with $\delta(v) = \Gamma$ and processing that unit until it cancels some deficit or a relabeling occurs. We implement this strategy as in the unweighted case. Observe that no node's price changes by more than $2\alpha n\epsilon$ during refine, so a collection of $2\alpha n + 1$ buckets $b_0, \ldots, b_{2\alpha n}$ is sufficient to keep every active node v in $b_{\delta(v)}$. As before, the algorithm maintains the index μ of the lowest-numbered nonempty bucket and avoids bucket access except immediately after a deficit is canceled or a relabeling of a node v sets $\delta(v) > \mu$.

In the weighted context, a global update takes the form of setting each node price so that G_A is acyclic, 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]. At every *refine*, the first global update is performed immediately after saturating all residual arcs with negative reduced cost. After each *push* and *relabel* operation, the algorithm checks the following two conditions and performs a global update if both conditions hold:

- Since the most recent update, at least one unit of excess has canceled some deficit; and
- Since the most recent update, the algorithm has done at least m work in *push* and *relabel* operations.

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].

Immediately after each global update, the algorithm rebuilds the buckets in $O(n)$ time and sets μ to zero. As in the unweighted case, we have the following easy bound on the extra work done by the algorithm in selecting nodes to process:

LEMMA 9.1. *Between two consecutive global updates, the algorithm does $O(n)$ work in examining empty buckets.*

Figure 9.1 represents the main ideas behind our analysis of an iteration of the Minimum Change Discharge algorithm. The diagram differs from Figure 4.1 because we must account for pushes and relabelings that occur at nodes with large values of

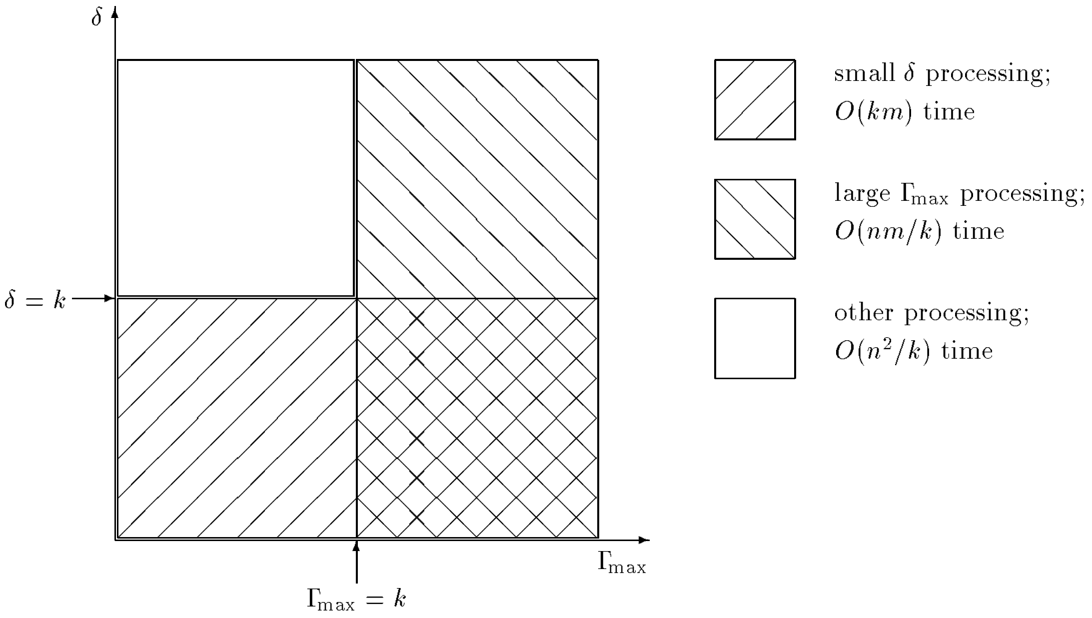

FIG. 9.1. *Accounting for work in the Minimum Change Discharge algorithm*

δ when $\Gamma_{\max}$ is small. Such operations could not arise in the matching algorithm, but are possible here.

We begin our analysis with a lemma that is essentially similar to Lemma 4.2.

LEMMA 9.2. *The algorithm does $O(km)$ work in the course of* relabel *operations on nodes v obeying $\delta(v) \le k$ and* push *operations from those nodes.*

Proof. A node v can be relabeled at most $k+1$ times while $\delta(v) \le k$, so the relabelings of such nodes and the pushes from them require $O(km)$ work. ☐

To analyze our algorithm for the assignment problem, we must overcome two main difficulties that were not present in the matching case. First, we can do *push* and *relabel* operations at nodes whose price changes are large even when $\Gamma_{\max}$ is small; this work is not bounded by Lemma 9.2 and we must account for it. Second, our analysis of the period when $\Gamma_{\max}$ is large in the unweighted case does not generalize because it is not true that $\delta(v)$ gives a bound on the breadth-first-search distance from v to a deficit in the residual graph.

Lemma 9.4 is crucial in resolving both of these issues, and to prove it we use the following standard result which is analogous to Lemma 4.3.

LEMMA 9.3. *Given a matching network G and an integral circulation g, any integral 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 s, t, and 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. Let $\mathcal{E}(f)$ denote the total excess in pseudoflow f, *i.e.*, $\sum_{e_f(v)>0} e_f(v)$. When no confusion will arise, we simply use $\mathcal{E}$ to denote the total excess in the current pseudoflow. The lemma depends on the $(\alpha\epsilon)$-optimality of the circulation produced by the previous iteration of *refine*, so it holds only in the second and subsequent scaling iterations. Because the zero circulation is not C-optimal with respect to the zero price function, we need different phrasing to accomplish the same task in the first iteration. The differences are mainly technical, so the first-iteration lemmas and their proofs are confined to Appendix A.

LEMMA 9.4. *At any point during an execution of* refine *other than the first,* $\mathcal{E} \times \Gamma_{\max} \leq 2\big((5+\alpha)n - 1\big)$.

Proof. Let c denote the (reduced) arc cost function at the beginning of this execution of *refine*, and let $G = (V, E)$ denote the augmented 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, and let f, p be the pseudoflow and price function at the most recent point during the execution of *refine* when $\Gamma(f,p) = \Gamma_{\max}$. Since $\mathcal{E}(f) \geq \mathcal{E}(f')$ and $\Gamma(f,p) \geq \Gamma(f',p')$, it is enough to prove the lemma for f, p. We have

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

From the definition of δ, then,

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

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,

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

Letting $\mathcal{P}$ be a decomposition of f into paths and cycles according to Lemma 9.3 and noting that cycles make no contribution to the sum, we can rewrite this expression as

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

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, the residual graph G_g has exactly n arcs $a \notin E_U$, and so from the fact that the execution of *refine* begins with the augmented 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 $-2\alpha\epsilon$, we have $c(f) \geq -2\alpha n\epsilon$. Hence $c_p(f) - c(f) \leq c_p(f) + 2\alpha n\epsilon$.

Now consider $c_p(f)$. Clearly, $f(a) > 0 \Rightarrow a^R \in E_f$, and ϵ-optimality of f with respect to p says that $a^R \in E_f \Rightarrow c_p(a^R) \geq -2\epsilon$. Since $c_p(a^R) = -c_p(a)$, we have $f(a) > 0 \Rightarrow c_p(a) \leq 2\epsilon$. Recalling our decomposition $\mathcal{P}$ into cycles and paths from deficits to excesses, 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

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

The total number of arcs not incident to s or t in the cycles and path interiors is at most n by node-disjointness, and the number of arcs incident to s or t is at most $2n-1$. 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 2ϵ, we have $c_p(f) \leq (n + 2n - 1 + 2n)2\epsilon = (5n-1)2\epsilon$.

Therefore, $c_p(f) - c(f) \leq 2\big((5+\alpha)n - 1\big)\epsilon$, and we have $\mathcal{E}(f) \times \Gamma_{\max} \leq 2\big((5+\alpha)n - 1\big)$. ☐

COROLLARY 9.5. *$\Gamma_{\max} \geq k$ implies $\mathcal{E} = O(n/k)$.*

We use the following lemma to show that when $\Gamma_{\max}$ is small, we do a limited amount of work at nodes whose price changes are large.

LEMMA 9.6. *While $\Gamma_{\max} \leq k$, the amount of work done in relabelings at nodes v with $\delta(v) > k$ and pushes from those nodes is $O(n^2/k)$.*

Proof. For convenience, we say a node that gets relabeled under the conditions of the lemma is a *bad* node. We process a given node v either because we selected a unit of excess at v, or because the most recent operation was a *push* from one of v's neighbors to v. If a unit of v's excess is selected, we have $\delta(v) \leq \Gamma_{\max}$ (indeed without global updates, $\delta(v) = \Gamma_{\max}$) which implies $\delta(v) \leq k$, so v cannot be a bad node. In the second case, the unit of excess just pushed to v will remain at v until $\Gamma_{\max} \geq \delta(v)$ because the condition $\delta(v) > \mu$ will cause excess at a different node to be selected immediately after v is relabeled. We cannot select v's excess until $\Gamma_{\max} \geq \delta(v)$, and at such a time, Corollary 9.5 shows that the total excess remaining is $O(n/k)$. Since each relabeling of a bad node leaves a unit of excess that must remain at that node until $\Gamma_{\max} \geq k$, the number of relabelings of bad nodes is $O(n/k)$. Because every node has degree at most n, the work done in pushes and relabelings at bad nodes is $O(n^2/k)$. ☐

Recall that the algorithm initiates global update only after a unit of excess has canceled some deficit since the last global update. The next lemma, analogous to Lemma 4.6, shows that this rule cannot introduce too great a delay.

LEMMA 9.7. *Between any two consecutive global update operations, the algorithm does $\Theta(m)$ work.*

Proof. As in the unweighted case, it suffices to show that the algorithm does $O(m)$ work in canceling a deficit immediately after a global update operation, and $O(m)$ work in selecting nodes to process. The definition of a global update operation suffices to ensure that a unit of excess reaches some deficit immediately after a global update and before any relabeling occurs, and Lemma 9.1 shows that the extra work done between global updates in selecting nodes to process is $O(n)$. $\square$

Lemma 9.2 and Lemma 9.6 show that the algorithm takes $O(km + n^2/k)$ time when $\Gamma_{\max} \le k$. Corollary 9.5 says that when $\Gamma_{\max} \ge k$, the total excess remaining is $O(n/k)$, and Lemma 9.7 shows that $O(m)$ work suffices to cancel each unit of excess remaining. Therefore the total work in an execution of *refine* is $O\big(km+n^2/k+nm/k\big)$, 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, giving the following theorem:

THEOREM 9.8. *The Minimum Change Discharge algorithm with global updates computes a minimum cost circulation in a matching network in $O\big(\sqrt{n}m\log(nC)\big)$ time.*

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 cannot improve performance in the assignment problem context.

10. 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 change 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. These two nodes and the edge between them are present only to establish the initial value of ϵ and the costs of arcs introduced in the reduction, and are ignored in our description of the execution.

At the beginning of the first scaling iteration, $\epsilon = \alpha$. The iteration starts by setting $\epsilon = 1$. From this point on the execution is similar to the execution of the Minimum Distance Discharge algorithm given in Section 6, but the details differ because of the asymmetric definitions of ϵ-optimality and admissibility we use in the weighted case.

Figure 9.2 details an execution of the Minimum Change Discharge algorithm without global updates. As in the unweighted case, every relabeling changes a node price by at most two, and the algorithm does $\Omega(n^2)$ relabelings. Consequently, the relabelings require $\Omega(nm)$ work, and we have the following theorem:

THEOREM 10.1. *For any function $m(n)$ in the range $n \le m(n) < n^2/4$, there exists an infinite family of instances of the assignment problem having $\Theta(n)$ nodes and $\Theta\big(m(n)\big)$ edges on which the Minimum Change Discharge implementation of* refine *without global updates runs in $\Omega\big(nm(n)\big)$ time.*

11. Conclusions and Open Questions. We have presented algorithms that achieve the best time bounds known for bipartite matching, *i.e.*, $O\left(\sqrt{n}m\frac{\log(n^2/m)}{\log n}\right)$, and for the assignment problem in the cost-scaling context, *i.e.*, $O\left(\sqrt{n}m\log(nC)\right)$. 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

1. Initialization establishes $|X|$ units of excess, one at each node of X.

2. While some node $w \in Y$ has no excess,

  2.1. a unit of excess moves from a node of X to w;

  2.2. w is relabeled so that $p(w) = -2$.

  Remark: Now every node of Y has one unit of excess.

3. Active nodes in X are relabeled one-by-one so that each has price -2.

4. A unit of excess moves from the most recently relabeled node of X to a node of Y, then to t, and on to cancel a unit of deficit at s.

5. While more than one node of Y has excess,

  5.1. A unit of excess moves to t and thence to s from a node of Y;

6. The remaining unit of excess at a node of Y moves to a node $v \in X$ with $p(v) = 0$, and v is relabeled so that $p(v) = -2$.

7. $\ell \leftarrow 1$.

8. While $\ell \leq \alpha n/2 - 1$,

  Remark: All excesses are at nodes of X, and these nodes have price -2ℓ; all other nodes in X have price $-2\ell + 2$; all nodes in Y have price -2ℓ.

  8.1. A unit of excess is selected, and while some node $x \in X$ has $p(x) = -2\ell + 2$,
  - the selected unit moves from some active node v to w, a neighbor of x in G_f (for a given x there is a unique such w);
  - the unit of excess moves from w to x;
  - x is relabeled so $p(x) = -2\ell$.

  Remark: Now all nodes in $X \cup Y$ have price -2ℓ; all excesses are at nodes of X.

  8.2. While some node $w \in Y$ has $p(w) = -2\ell$ and some node $v \in X$ has $e_f(v) = 1$,
  - a unit of excess moves from v to w;
  - w is relabeled so $p(w) = -2\ell - 2$.

  Remark: The following loop is executed only if $|X| < 2|Y|$. All active nodes in Y have price $-2\ell - 2$, and all other nodes in Y have price -2ℓ.

  8.3. If a node in Y has price -2ℓ, a unit of excess is selected, and while some node $y \in Y$ has $p(y) = -2\ell$,
  - the selected unit moves from some $w \in Y$ with $e_f(w) = 1$ to $v \in X$ with $p(v) = -2\ell$, and then to y;
  - y is relabeled so $p(y) = -2\ell - 2$.

  Remark: The following loop is executed only if $|X| > 2|Y|$.

  8.4. For each node $v \in X$ with $e_f(v) = 1$,
  - v is relabeled so $p(v) = \max\{-2\ell - 2, -\alpha n\}$.

  8.5. For each node $w \in Y$ with $e_f(w) = 1$,
  - a unit of excess moves from w to $v \in X$ with $p(v) = -2\ell$;
  - v is relabeled so $p(v) = \max\{-2\ell - 2, -\alpha n\}$.

  8.6. $\ell \leftarrow \ell + 1$.

9. Excesses move one-by-one from active nodes in X (which have price $-\alpha n$) to s.

FIG. 9.2. *The Minimum Change Discharge execution on bad examples.*

development of algorithms.

We have shown a family of assignment instances on which *refine* without global updates performs poorly, but the poor performance seems to hinge on details of the reduction so it happens 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.

12. Acknowledgment. The authors would like to thank an anonymous referee whose careful reading led us to several corrections and improvements.

REFERENCES

[1] R. J. ANDERSON AND J. C. SETUBAL, *Goldberg's Algorithm for the Maximum Flow in Perspective: a Computational Study*, in Network Flows and Matching: First DIMACS Implementation Challenge, D. S. Johnson and C. C. McGeoch, eds., AMS, 1993, pp. 1–18.

[2] U. DERIGS AND W. MEIER, *Implementing Goldberg's Max-Flow Algorithm — A Computational Investigation*, ZOR — Methods and Models of Operations Research, 33 (1989), pp. 383–403.

[3] R. B. DIAL, *Algorithm 360: Shortest Path Forest with Topological Ordering*, Comm. ACM, 12 (1969), pp. 632–633.

[4] E. A. DINIC, *Algorithm for Solution of a Problem of Maximum Flow in Networks with Power Estimation*, Soviet Math. Dokl., 11 (1970), pp. 1277–1280.

[5] S. EVEN AND R. E. TARJAN, *Network Flow and Testing Graph Connectivity*, SIAM J. Comput., 4 (1975), pp. 507–518.

[6] T. FEDER AND R. MOTWANI, *Clique Partitions, Graph Compression and Speeding-up Algorithms*, in Proc. 23st Annual ACM Symposium on Theory of Computing, 1991, pp. 123–133.

[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, 1988, pp. 514–527.

[9] ———, *Faster Scaling Algorithms for Network Problems*, SIAM J. Comput., (1989), pp. 1013–1036.

[10] A. V. GOLDBERG, *Efficient Graph Algorithms for Sequential and Parallel Computers*, PhD thesis, M.I.T., Jan. 1987. (Also available as Technical Report TR-374, Lab. for Computer Science, M.I.T., 1987).

[11] ———, *An Efficient Implementation of a Scaling Minimum-Cost Flow Algorithm*, in Proc. 3rd Integer Prog. and Combinatorial Opt. Conf., 1993, pp. 251–266.

[12] A. V. GOLDBERG, S. A. PLOTKIN, AND P. M. VAIDYA, *Sublinear-Time Parallel Algorithms for Matching and Related Problems*, J. Algorithms, 14 (1993), pp. 180–213.

[13] A. V. GOLDBERG AND R. E. TARJAN, *A New Approach to the Maximum Flow Problem*, J. Assoc. Comput. Mach., 35 (1988), pp. 921–940.

[14] ———, *Finding Minimum-Cost Circulations by Successive Approximation*, Math. of Oper. Res., 15 (1990), pp. 430–466.

[15] J. E. HOPCROFT AND R. M. KARP, *An $n^{5/2}$ Algorithm for Maximum Matching in Bipartite Graphs*, SIAM J. Comput., 2 (1973), pp. 225–231.

[16] A. V. KARZANOV, *O nakhozhdenii maksimal'nogo potoka v setyakh spetsial'nogo vida i nekotorykh prilozheniyakh*, in Matematicheskie Voprosy Upravleniya Proizvodstvom, vol. 5, Moscow State University Press, Moscow, 1973. In Russian; title translation: On Finding Maximum Flows in Network with Special Structure and Some Applications.

[17] ———, *Tochnaya otzenka algoritma nakhojdeniya maksimalnogo potoka, primenennogo k zadache "o predstavitelyakh"*, in Problems in Cibernetics, vol. 5, Nauka, Moscow, 1973, pp. 66–70. In Russian; title translation: The exact time bound for a maximum flow algorithm applied to the set representatives problem.

[18] H. W. KUHN, *The Hungarian Method for the Assignment Problem*, Naval Res. Logist. Quart., 2 (1955), pp. 83–97.

[19] Q. C. NGUYEN AND V. VENKATESWARAN, *Implementations of Goldberg-Tarjan Maximum Flow Algorithm*, in Network Flows and Matching: First DIMACS Implementation Challenge, D. S. Johnson and C. C. McGeoch, eds., AMS, 1993, pp. 19–42.

[20] J. B. ORLIN AND R. K. AHUJA, *New Scaling Algorithms for the Assignment and Minimum Cycle Mean Problems*, Math. Prog., 54 (1992), pp. 41–56.

Appendix A. The First Scaling Iteration. Let G be the network produced by reducing an assignment problem instance to the minimum cost circulation problem as in Section 8. When *refine* initializes by saturating all negative arcs in this network, the only deficit created will be at s by our assumption that the input costs are nonnegative.

For a pseudoflow f in G, define $\mathcal{E}_t(f)$ to be the amount of f's excess that can reach s by passing through t. $\mathcal{E}_t(f)$ corresponds to the residual flow value in the unweighted case (see Section 4).

The $(\alpha\epsilon)$-optimality of the initial flow and price function played an important rôle in the proof of Lemma 9.4, specifically by lower-bounding the initial cost of any arc that currently carries a unit of flow. In contrast, the first scaling iteration may have many arcs that carry flow and have extremely negative costs relative to ϵ, specifically those arcs of the form (s, v) introduced by the reduction. But to counter this difficulty, the first iteration has an advantage that later iterations lack: an *upper* bound (in terms of ϵ) on the initial cost of *every* residual arc in the network. Specifically, recall that the value of ϵ in the first iteration is C/α, where C is the largest cost of an edge in the given assignment instance. So for any arc a other than the (v, s) arcs introduced by the reduction, $c(a) \leq \alpha\epsilon$ in the first scaling iteration.

Lemma A.1. *At any point during the first execution of* refine, $\mathcal{E}_t \times \Gamma_{\max} \leq n(2 + \alpha)$.

Proof. Let f', p' be the current pseudoflow and price function, and as in the proof of Lemma 9.4, let f, p be the pseudoflow and price function at the most recent point when $\Gamma(f, p) = \Gamma_{\max}$. As before, it is enough to prove the lemma for f, p; this will imply the claim holds for f', p'.

Let f^* be a minimum cost circulation in G, and let $f' = f^* - f$. Recall that the costs on the (s, v) arcs are negative enough that f^* must correspond to a matching of maximum cardinality. Therefore, f' moves $\mathcal{E}_t(f)$ units of f's excess to s through t, and returns the remainder to s without its passing through t. Now $-f'$ is a pseudoflow in G_{f^*}, and can be decomposed into cycles and paths according to Lemma 9.3; as in the proof of Lemma 4.4, let $\mathcal{P}$ denote the induced decomposition of f'. Let $\mathcal{Q} \subseteq \mathcal{P}$ be the set of paths that pass through t, and note that $\mathcal{E}_t(f) = |\mathcal{Q}|$. Let $e_f^t(v)$ denote the number of paths of $\mathcal{Q}$ beginning at node v. The only deficit in f is at s, so $e_f^t(v)$ is precisely the amount of v's excess that reaches s by passing through t if we imagine augmenting f along the paths of $\mathcal{P}$. Of particular importance is that no path in $\mathcal{Q}$ uses an arc of the form (s, v) or (v, s) for $v \neq t$.

Observe that

$$\mathcal{E}_t(f) \times \Gamma_{\max} \leq \sum_{e_f^t(v)>0} e_f^t(v)\delta(v),$$

so by the definition of δ,

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

Now note that for any path P from v to s, we have $p(v) = c_p(P) - c(P)$ because $p(s) = 0$. Every arc used in the decomposition $\mathcal{P}$ appears in G_f. By ϵ-optimality of f, each of the n or fewer arcs a in G_f with negative reduced cost has $c_p(a) \geq -2\epsilon$, so we have $\sum_{P\in\mathcal{Q}} c_p(P) \geq -2n\epsilon$. Next, we use the upper bound on the initial costs to note that $\sum_{P\in\mathcal{Q}} c(P) \leq \alpha n\epsilon$, so

$$\epsilon \times \mathcal{E}_t(f) \times \Gamma_{\max} \leq -\sum_{e_f^t(v)>0} e_f^t(v)p(v) \leq 2n\epsilon + \alpha n\epsilon = n(2+\alpha)\epsilon,$$

and the lemma follows. ∎

LEMMA A.2. *At any point during the first execution of* refine*,* $\mathcal{E} \times (\Gamma_{\max} - \alpha n) \leq n(2+\alpha)$.

Proof. Essentially the same as the proof of Lemma A.1, except that if $\Gamma_{\max} > \alpha n$, each path from an excess to the deficit at s will include one arc of the form (v, s), and each such arc has original cost $-nC = -\alpha n\epsilon$. ∎

Lemmas A.1 and A.2 allow us to split the analysis of the first scaling iteration into four stages, much as we did with the Minimum Distance Discharge algorithm for matching. Specifically, the analysis of Section 9 holds up until the point where $\Gamma_{\max} \geq \alpha n$, with Lemma A.1 taking the place of Lemma 9.4. Straightforward extensions of the relevant lemmas show that the algorithm does $O(km + n^2/k)$ work when $\Gamma_{\max} \in [\alpha n, \alpha n + k]$, and when $\Gamma_{\max} > \alpha n + k$, Lemma A.2 bounds the algorithm's work by $O(nm/k)$. The balancing works as before: choosing $k = \Theta(\sqrt{n})$ gives a bound of $O(\sqrt{n}m)$ time for the first scaling iteration.