2012 IEEE 53rd Annual Symposium on Foundations of Computer Science

A Weight-Scaling Algorithm for Min-Cost Imperfect Matchings in Bipartite Graphs

Lyle Ramshaw
HP Labs, 1501 Page Mill Rd
Palo Alto, CA 94304, USA
lyle.ramshaw@hp.com

Robert E. Tarjan
Princeton University and HP Labs
ret@cs.princeton.edu *and*
robert.tarjan@hp.com

***Abstract*— Call a bipartite graph $G = (X, Y; E)$ *balanced* when $|X| = |Y|$. Given a balanced bipartite graph G with edge costs, the *assignment problem* asks for a perfect matching in G of minimum total cost. The Hungarian Method can solve assignment problems in time $O(mn + n^2 \log n)$, where $n := |X| = |Y|$ and $m := |E|$. If the edge weights are integers bounded in magnitude by $C > 1$, then algorithms using *weight scaling*, such as that of Gabow and Tarjan, can lower the time to $O(m\sqrt{n}\log(nC))$.**

There are important applications in which G is *unbalanced*, with $|X| \neq |Y|$, and we require a min-cost matching of size $r := \min(|X|, |Y|)$ or, more generally, of some specified size $s \leq r$. The Hungarian Method extends easily to find such a matching in time $O(ms + s^2 \log r)$, but weight-scaling algorithms do not extend so easily. We introduce new machinery to find such a matching in time $O(m\sqrt{s}\log(sC))$ via weight scaling. Our results provide some insight into the design space of efficient weight-scaling matching algorithms.

1. Introduction

Consider a bipartite graph $G = (X, Y; E)$. We refer to the vertices in X as *women* and the vertices in Y as *men*.[1] We call G *balanced* when $|X| = |Y|$, so there are the same number of women as men; otherwise, G is *unbalanced*.[2] We denote the size of the larger part by $n := \max(|X|, |Y|)$, while we introduce the symbol $r := \min(|X|, |Y|)$ to denote the size of the smaller part.[3] And we call a graph *asymptotically unbalanced* when $r = o(n)$.

Each edge (x, y) in G has a weight $c(x, y)$, which is a real number with no sign restriction. We interpret those weights as costs, whose sums we try to minimize. Negating all of the weights, defining $b(x, y) := -c(x, y)$ for each edge (x, y), would convert minimizing cost into maximizing benefit.

A *matching* is a set M of edges that don't share any vertices; its *size* is $s := |M|$ and its *cost* is $c(M) := \sum_{(x,y)\in M} c(x, y)$. We refer to matched vertices as *married*. A matching of size n in a balanced graph is called *perfect*. We call a matching of size r in an unbalanced graph *one-sided perfect*; such a matching leaves $n - r$ vertices in the larger part as either *maidens*[4] or *bachelors*. A matching of size $s < r$ is *imperfect*; such a matching leaves both some maidens and some bachelors.

Denoting the maximum size of any matching in the graph G by $\nu(G)$, we consider two variants of the *assignment problem* [4]:

PerA (Perfect Assignments) Let G be a balanced bipartite graph with edge weights. If $\nu(G) = n$, compute a min-cost perfect matching in G; otherwise, return the error code "infeasible".

ImpA (Imperfect Assignments) Let G be a bipartite graph with edge weights, either balanced or unbalanced, and let $t \geq 1$ be a target size. Compute a min-cost matching in G of size $s := \min(t, \nu(G))$.

For **ImpA**, our time bounds are functions of s, the size of the output matching, and are hence output sensitive. For simplicity in our time bounds, we assume that our bipartite graphs have no isolated vertices, so $r \leq n \leq m \leq rn$; we also assume throughout that $s \geq 1$ and, once we introduce the bound C on the magnitudes of the costs, that $C > 1$.

1.1. Known algorithms for *PerA*

Most published assignment algorithms solve **PerA**. The ones that perform the best in practice are local; their updates change the matching status of just a few edges and the prices just at the vertices that those edges touch. Algorithms of this type include the *push-relabel* algorithms of Goldberg [11] and the *auction* algorithms of Bertsekas [2]. But local algorithms don't get the best time bounds.

The granddaddy of algorithms for **PerA** is the Hungarian Method [15], which is purely global; it builds up its min-cost matchings by augmenting along entire augmenting paths, from a maiden to a bachelor. The bounds on the Hungarian Method have been improved repeatedly. Fredman and Tarjan [9] used Fibonacci heaps to devise a version that

[1] To remember which of X and Y consists of women and which of men, think of how sex chromosomes work in mammals.

[2] Some authors use "symmetric" and "asymmetric" for the properties that we call *balanced* and *unbalanced*.

[3] As a mnemonic aid, think of n as standing for numerous, while r stands for rare. Some authors use n_2 and n_1 for our n and r.

[4] English doesn't have a single word that means simply an unmarried woman; "maiden" and "spinster" both have irrelevant overtones, which are politically incorrect to boot. Indeed, when faced with this challenge, television shows have typically resorted to "bachelorette".

0272-5428/12 $26.00 © 2012 IEEE
DOI 10.1109/FOCS.2012.9

runs in space $O(m)$ and in time $O(mn + n^2 \log n)$. This is the current record among strongly polynomial algorithms. For integer edge weights, Thorup [19] reduced the time to $O(mn + n^2 \log\log n)$.

Weight-scaling is another way to reduce the time; it also requires that the edge weights be integers and fails to be strongly polynomial. Using weight-scaling, the assignment problem can be solved in space $O(m)$ and in time $O(m\sqrt{n}\log(nC))$, where $C > 1$ is a bound on the magnitudes of the edge weights. This time bound is achieved by algorithms due to Gabow and Tarjan [10], to Orlin and Ahuja [16], and to Goldberg and Kennedy [12]. Of these, Gabow-Tarjan is purely global, while the other two are hybrids of local and global techniques.

1.2. Computing min-cost imperfect matchings

When we tackle an assignment problem in practice, our goal is often a min-cost matching that is less than perfect, frequently because our graph is unbalanced. One way to solve such a problem is to reduce it to **PerA**, perhaps by making two copies of our unbalanced graph G with one copy flipped over, thus building a balanced graph G' with $n' = n + r$. Such doubling reductions handle only those instances of **ImpA** in which $t \geq \nu(G)$, since there is no obvious way to impose a bound $t < \nu(G)$ on the size of the matching in G that we extract from G'. But the bigger problem with these doubling reductions is that we gain no speed advantage when $r \ll n$.

Instead of reducing, we can develop algorithms that solve **ImpA** directly. On the practical side, Bertsekas and Castañon [3] generalized an auction algorithm to work directly on unbalanced graphs.[5] Here, we explore that direct approach theoretically, replacing n's in the bounds of **PerA** algorithms with r's or s's. Ahuja, Orlin, Stein, and Tarjan [1] replaced lots of n's with r's in the bounds of bipartite network-flow algorithms; but the corresponding challenge for **ImpA** seems to be new.

The Hungarian Method is an easy success. In an accompanying report [18], we show that the Hungarian Method solves **ImpA** in time $O(ms + s^2 \log r)$. If the costs are integers, Thorup's technique can reduce the $\log r$ to $\log\log r$.

But weight-scaling algorithms are harder to generalize. Goldberg-Kennedy [12] can compute imperfect matchings that are min-cost; but it isn't clear whether the n's in their time bound can be replaced with r's or s's. Worse yet, a straightforward attempt to compute an imperfect matching with either Gabow-Tarjan [10] or Orlin-Ahuja [16] may result in a matching that fails to be min-cost. In Section 3, we derive the *maiden* and *bachelor bounds*, inequalities that help to prove that an imperfect matching is min-cost. The Hungarian Method preserves these bounds naturally, as does Goldberg-Kennedy; but neither Gabow-Tarjan nor Orlin-Ahuja does so.

Our central result is *FlowAssign*, a purely global, weight-scaling algorithm that solves **ImpA** in space $O(m)$ and time $O(m\sqrt{s}\log(sC))$. Roughly speaking, *FlowAssign* is Gabow-Tarjan with dummy edges to a new source and sink added, to enforce the maiden and bachelor bounds. *FlowAssign* also simplifies Gabow-Tarjan in two respects. First, Gabow-Tarjan adjusts some prices as part of augmenting along an augmenting path. Those price adjustments turn out to be unnecessary, and we don't do them in *FlowAssign* (though we could, as we discuss in Section 8). Second, we sometimes want prices that, through complementary slackness, prove that our output matching is indeed min-cost. Gabow and Tarjan compute such prices in a $O(m)$ postprocessing step. In *FlowAssign*, we compute such prices simply by rounding, to integers, the prices that we have already computed, that rounding taking time $O(n)$.

1.3. Related work and open problems

Given a balanced bipartite graph G without edge weights, the algorithm of Hopcroft and Karp [13] computes a matching in G of maximum size in time $O(m\sqrt{n})$. If the graph G is sufficiently dense, then Feder and Motwani [8] show how to improve on Hopcroft-Karp by a factor of as much as $\log n$: They compute a max-size matching in time $O(m\sqrt{n}\log(n^2/m)/\log n)$. If m is nearly n^2, then the log factor in the numerator is small and we are essentially dividing by $\log n$. But the improvement drops to a constant as soon as m is $O(n^{2-\epsilon})$, for any positive ϵ. It would be interesting to generalize the Feder-Motwani technique to unbalanced graphs. By using a doubling reduction, their algorithm can handle an unbalanced graph in the time bound given above. But perhaps that time could be improved by tackling the unbalanced case directly — perhaps improved to $O(m\sqrt{r}\log(rn/m)/\log n)$.

Given a balanced bipartite graph G with edge weights, we might want to compute a matching in G that has the maximum possible benefit, among all matchings of any size whatever. Duan and Su [7] recently found a weight-scaling algorithm for this *max-weight matching problem* that runs in time $O(m\sqrt{n}\log C)$. Thus, they reduced the logarithmic factor from $\log(nC)$ to $\log C$. They pose the intriguing open question of whether that same reduction can be achieved for the assignment problem, where the optimization is over matchings of some fixed size. Duan and Su did not consider the asymptotically unbalanced case, however, so they made no attempt to replace n's with r's. Their algorithm might generalize to unbalanced graphs straightforwardly, in time $O(m\sqrt{r}\log C)$; we leave that as an open question. Given an unbalanced graph G, we can reduce the max-weight

[5] The auction algorithm that Bertsekas and Castañon [3] suggest for unbalanced graphs computes one-sided-perfect matchings that leave bachelors. These matchings are nevertheless min-cost because Bertsekas and Castañon introduce an auction step that preserves the bachelor bound of Section 3. Their auction step maintains a *profitability threshold* λ, where λ can be thought of as a candidate for the price $p_d(\dashv)$.

matching problem for G to the assignment problem for a graph G' that is even more unbalanced than G. We construct G' by adding r new vertices to the larger part of G and connecting each of the r vertices in the smaller part of G to one of these new vertices with a zero-weight edge. By using *FlowAssign* to find a one-sided-perfect matching of maximum weight in G', we can find a max-weight matching in G in time $O(m\sqrt{r}\log(rC))$. Thus, we can reduce the $\sqrt{n}$ factor to $\sqrt{r}$, but only at the price of bumping the logarithmic factor back up from $\log C$ to $\log(rC)$.

Recall that Feder and Motwani showed how to speed up Hopcroft-Karp a bit, for quite dense graphs. Suppose we have a fairly dense, balanced bipartite graph G with positive edge weights, but most of those weights are quite small; and we want to compute a max-weight matching in G. If all of the weights were precisely 1, then a max-weight matching would be the same as a max-size matching, so we could use Feder-Motwani. Kao, Lam, Sung, and Ting [14] showed that a similar improvement is possible as long as most of the edge weights are quite small. Assuming that the edge weights are positive integers and letting W denote the total weight of all of the edges in G, they compute a max-weight matching in time $O(\sqrt{n}\,W\log(n^2C/W)/\log n)$. When $C = O(1)$ and hence $W = \Theta(m)$, their bound matches that of Feder and Motwani. But they continue to achieve improved performance until W gets up around $m\log(nC)$, at which point we are better off reducing to an assignment problem. If someone generalizes Feder-Motwani to the asymptotically unbalanced case, it might then be worth generalizing the Kao-Lam-Sung-Ting result. The main issue would be replacing their initial $\sqrt{n}$ with $\sqrt{r}$.

Finally, a more practical note: *FlowAssign*, like Gabow-Tarjan [10], is a purely global algorithm, so it might not perform all that well in practice. To find an algorithm for **ImpA** that performs well in both theory and practice, one might do better by aiming for a hybrid of local and global techniques, perhaps by starting with Goldberg-Kennedy [12] or with Orlin-Ahuja [16].

2. FROM MATCHINGS TO FLOWS

We begin by constructing a flow network N_G from the graph G, thereby converting min-cost matchings in G into min-cost integral flows on N_G. This conversion is quite standard; but we renounce a popular skew-symmetry.

Each vertex in G becomes a node in N_G, and each edge (x, y) in G becomes an arc $x \to y$, directed from x to y; we refer to these arcs as *bipartite*. The network N_G also includes a source node $\vdash$ and a sink node $\dashv$. For each woman x in X, we add a *left-dummy* arc $\vdash \to x$ and, for each man y, a *right-dummy* arc $y \to \dashv$. Each arc has a per-unit-of-flow cost, that cost being $c(x, y)$ for the bipartite arc $x \to y$ and zero for all dummy arcs. And all arcs in the flow network N_G have unit capacity.

Let's define a *flux* on the network N_G to be a function f that assigns a flow $f(v, w) \in \mathbb{R}$ to each arc $v \to w$ in N_G, with no restrictions whatsoever. We define the *cost* of a flux f by $c(f) := \sum_{v\to w} f(v, w)c(v, w)$. A *pseudoflow* is a flux in which the flow along each arc satisfies $0 \le f(v, w) \le 1$. A *flow* is a pseudoflow in which, at all nodes v except for the source and the sink, the total flow into v equals the total flow out of v, so flow is conserved at v. The *value* of a flow f, denoted $|f|$, is the total flow out of the source (and hence also into the sink).

Warning: Many authors set things up so that the functions f and c that measure flow quantity and per-unit-of-flow cost are skew-symmetric, with $f(w, v) = -f(v, w)$ and $c(w, v) = -c(v, w)$. We instead name the arcs in our flow network N_G only in their forward direction, from the source toward the sink.[6] We explain why in Section 4.

If f is an integral pseudoflow on the network N_G, then each arc $v \to w$ is either *idle*, with $f(v, w) = 0$, or *saturated*, with $f(v, w) = 1$. Matchings M in the graph G correspond to integral flows f on the network N_G; in this correspondence, the edges in the matching become the saturated bipartite arcs of the flow, and we have $|M| = |f|$ and $c(M) = c(f)$. Thus, a min-cost matching of some size s corresponds to a min-cost integral flow of value s. Without the constraint of integrality, minimizing the cost of a flow of value s is a linear program, a fact that we next exploit.

For each node in N_G, we invent a dual variable whose value is the per-unit price of our commodity at that node. Rather than working with the price $p_a(v)$ to *acquire* a unit of the commodity at the node v, we instead work with the price $p_d(v)$ to *dispose* of a unit of the commodity at v, where $p_d(v) = -p_a(v)$. (Our algorithms happen to work by lowering the acquire prices at nodes, which means raising their dispose prices — and prices in real life typically rise.) Given prices at the nodes of N_G, we adjust the cost of each arc $v \to w$ to account for the difference in prices between v and w by using[7] $c_p(v, w) := c(v, w) - p_d(v) + p_d(w)$ to define the *reduced cost* $c_p(v, w)$. The theory of complementary slackness then gives us:

Prop 1: Consider a flow f on the network N_G and prices p at its nodes. If every arc $v \to w$ with $c_p(v, w) < 0$ has $f(v, w) = 1$ and every arc with $c_p(v, w) > 0$ has $f(v, w) = 0$, then f is min-cost among flows of its value.

Given a pseudoflow f and prices p on the network N_G, we define an idle arc to be *proper* when its reduced cost is nonnegative, a saturated arc to be *proper* when its reduced

[6] Since we have only forward arcs, we don't need to divide by 2 in the equation $c(f) := \sum_{v\to w} f(v, w)c(v, w)$ defining the cost of a flux.

[7] Papers about **PerA** often compute reduced costs via $c_p(x, y) := c(x, y) + p(x) + p(y)$ or $c_p(x, y) := c(x, y) - p(x) - p(y)$, with the prices at x and at y entering with the same sign. This means that, of the two parts X and Y of the bipartite graph G, acquire prices are used in one, while dispose prices are used in the other. Since our flow network has source and sink nodes, however, it seems simpler to use either acquire prices throughout or dispose prices throughout; we use dispose prices throughout.

cost is nonpositive, and an arc with fractional flow to be *proper* only when its reduced cost is zero. We call the pair (f, p) *proper* when all of the arcs in N_G are proper.

Corollary 2: Let f be a flow on the network N_G. If prices p exist that make the pair (f, p) proper, then f is min-cost among flows of its value.

3. THE MAIDEN AND BACHELOR BOUNDS

Consider using some prices p at the nodes of the network N_G to show that some flow f on N_G is min-cost, via Corollary 2. In particular, consider the left-dummy arcs in N_G. If x is a married woman in the matching corresponding to f, then the left-dummy arc $\vdash \rightarrow x$ will be saturated. For this arc to be proper, we must have $c_p(\vdash, x) = c(\vdash, x) - p_d(\vdash) + p_d(x) \leq 0$, which, since $c(\vdash, x) = 0$, means $p_d(x) \leq p_d(\vdash)$. On the other hand, if x is a maiden, then the arc $\vdash \rightarrow x$ will be idle, so we must have $p_d(\vdash) \leq p_d(x)$. For all left-dummy arcs to be proper, we need

$$\max_{x \text{ married}} p_d(x) \leq p_d(\vdash) \leq \min_{x \text{ maiden}} p_d(x); \tag{1}$$

so the maidens have to be the most expensive women. Similarly, for all right-dummy arcs to be proper, the bachelors have to be the cheapest men:

$$\max_{y \text{ bachelor}} p_d(y) \leq p_d(\dashv) \leq \min_{y \text{ married}} p_d(y); \tag{2}$$

We refer to these as the *maiden* and *bachelor bounds*.

When there are no maidens, choosing $p_d(\vdash)$ large enough makes all left-dummy arcs proper; and, with no bachelors, choosing $p_d(\dashv)$ small enough makes all right-dummy arcs proper. So we can prove that a perfect matching is min-cost without worrying about the maiden and bachelor bounds. We do need to worry, however, when our matching is less than perfect. In this sense, **ImpA** is a little harder than **PerA** (which might explain why **PerA** has been more studied).

The Hungarian Method generalizes easily [18] to compute imperfect matchings that are min-cost because it preserves the maiden and bachelor bounds.[8] Each round of price increases raises the prices at all remaining maidens by at least as much as it raises any prices. Since all women start out at a common price, the remaining maidens are always the most expensive women. And the price at a man doesn't rise at all until after that man gets married. Since all men start out at a common price, the remaining bachelors are always the cheapest men.

Gabow-Tarjan [10] is a weight-scaling algorithm, so it carries out a sequence of *scaling phases*. During each phase, the prices at the nodes are raised much as in the Hungarian Method. But the women start each phase, not at some common price, but at whatever prices were computed during the previous phase. If all phases chose the same women to be their final maidens, we'd still be okay: The prices at those perpetual maidens would both start and end each phase at least as high as the prices at the other women. But the scaling phases decide independently which women will be their final maidens. So Gabow-Tarjan does not preserve the maiden bound, and we can't trust it to compute imperfect matchings that are min-cost. Gabow-Tarjan doesn't preserve the bachelor bound either, for similar reasons.

FlowAssign operates on the network N_G, rather than on G itself. We maintain prices at the source and sink, and we strive to make the dummy arcs proper, as well as the bipartite arcs, thereby ensuring the maiden and bachelor bounds. *FlowAssign* also differs from Gabow-Tarjan in that each scaling phase remembers which vertices ended up matched versus unmatched, at the end of the preceding phase.

4. ARCS BEING ε-PROPER, ε-TIGHT, OR ε-SNUG

Weight-scaling algorithms are built on some notion of arcs being approximately proper. Roughly speaking, for $\varepsilon > 0$, an arc is "ε-proper" when its reduced cost is within ε of making the arc proper. In *FlowAssign*, however, we treat the boundary cases in a one-sided manner. In fact, throughout *FlowAssign*, how we treat an arc $v \rightarrow w$ depends upon its reduced cost $c_p(v, w)$ only through the quantity $\lceil c_p(v, w)/\varepsilon \rceil$. We are following Gabow-Tarjan by quantizing our reduced costs to multiples of ε; but we are adding a new wrinkle by adopting this *ceiling quantization*.

Given an integral pseudoflow f and prices p, we define an idle arc $v \rightarrow w$ to be *ε-proper* when $c_p(v, w) > -\varepsilon$ and a saturated arc $v \rightarrow w$ to be *ε-proper* when $c_p(v, w) \leq \varepsilon$. Note that we allow equality in the bound for the saturated case, but not in the bound for the idle case, as dictated by our ceiling quantization.

Prop 3: Let $v \rightarrow w$ be an idle arc whose reduced cost is known to be a multiple of ε. If the arc $v \rightarrow w$ is ε-proper, then it is automatically proper.

Proof: We must have $c_p(v, w) > -\varepsilon$, so we actually have $c_p(v, w) \geq 0$. ■

We don't get the analogous automatic properness for saturated arcs. But idle arcs are typically in the majority; that's why we quantize with ceilings, rather than floors.

And about skew-symmetry: We avoid backward arcs in our network N_G since, if we allowed them, we would have to use floor quantization on their reduced costs.

An arc with reduced cost precisely zero is often called *tight*. So we say that an idle arc $v \rightarrow w$ is *ε-tight* when

[8] More precisely, there is a variant of the Hungarian Method that preserves the maiden and bachelor bounds: a variant that, in each iteration, computes a shortest-path forest with all of the remaining maidens as tree roots. A different variant builds just a single shortest-path tree, with some chosen maiden as its root. That variant solves **PerA** perfectly well, but it doesn't preserve the maiden bound, and hence it may compute imperfect matchings that are not min-cost. Yet another variant adds a preprocessing step that uses a local criterion to optimize the initial prices. Taking the vertices in turn, the price at each woman is raised as far as possible while the price at each man is lowered as far as possible, while keeping all bipartite arcs proper. Both the maiden and bachelor bounds fail in that variant.

FlowAssign(G, t)
  $(M, s) :=$ *HopcroftKarp*(G, t);
  convert M into a flow f on N_G with $|f| = s$;
  for all nodes v in N_G, set $p_d(v) := 0$;
  $\varepsilon := \overline{\varepsilon}$; **while** $\varepsilon > \underline{\varepsilon}$ **do**
    $\varepsilon := \varepsilon/q$;
    Refine(f, p, ε);
  od;
  round prices to integers that make all arcs proper;

Figure 1. The high-level structure of *FlowAssign*

$-\varepsilon < c_p(v, w) \leq 0$, while a saturated arc $v \to w$ is *ε-tight* when $0 < c_p(v, w) \leq \varepsilon$.[9] Note that ε-tight arcs are ε-proper, but just barely so, in the sense of "just barely" that our ceiling quantization allows. For saturated arcs, we also define a weaker notion: A saturated arc $v \to w$ is *ε-snug* when $-\varepsilon < c_p(v, w) \leq \varepsilon$.

5. Starting and stopping *FlowAssign*

Figure 1 shows *FlowAssign*. Given a bipartite graph G with integral costs and a target size t for the output matching, *FlowAssign* computes a matching in G of size $s := \min(t, \nu(G))$ and also computes integral prices that demonstrate that its output matching is min-cost. *FlowAssign* runs in space $O(m)$ and in time $O(m\sqrt{s}\log(sC))$. The parameter $q > 1$ is an integer constant; $q = 8$ or $q = 16$ might be good choices.

We begin by ignoring the costs and invoking the max-size matching algorithm of Hopcroft and Karp [13] to compute both $s := \min(t, \nu(G))$ and some matching M in G of size s. As published, Hopcroft-Karp computes a matching of size $\nu(G)$ in time $O(m\sqrt{\nu(G)})$, which we couldn't afford. But our goal is a matching of size only s, and Hopcroft-Karp can compute a matching of that size in time $O(m\sqrt{s})$ [18].

The primary state of *FlowAssign* consists of the flux f, the prices p, and the real number ε. Here are four invariant properties of that state:

- **I1** The flux f on N_G is a flow of value $|f| = s$.
- **I2** For all nodes v, the price $p_d(v)$ is a multiple of ε.
- **I3** Every arc, whether idle or saturated, is ε-proper.
- **I4** Every saturated bipartite arc is ε-snug.

We reduce ε by a factor of q at the start of each scaling phase. This reduction makes it easier to satisfy **I2**, but harder to satisfy **I3** and **I4**. The routine *Refine* begins with special actions that reestablish **I3** and **I4**, given the new, smaller ε.

In each call to *Refine*, we have $\varepsilon = q^e$ for some integer e. The initial value $\overline{\varepsilon} = q^{\overline{e}}$ is the smallest power of q that strictly exceeds C; so we set $\overline{e} := 1 + \lfloor \log_q C \rfloor$. The final $\underline{\varepsilon} = q^{\underline{e}}$ is the largest power of q that is strictly less than $1/(s+2)$; so we set $\underline{e} := -\big(1 + \lfloor \log_q(s+2) \rfloor\big)$. The number of calls to *Refine* is $\overline{e} - \underline{e} = O(\log(sC))$. The *early scaling phases* are those with $e \geq 0$, so that ε is a positive integer; the *late phases* are those with $e < 0$, so that ε is the reciprocal of an integer.

FlowAssign has to do arithmetic on prices, costs, and reduced costs. But the integer $1/\underline{\varepsilon} = q^{-\underline{e}}$ is a common denominator for every price that ever arises. For simplicity, *FlowAssign* represents its prices, costs, and reduced costs as rational numbers with this common denominator, that is, as integer multiples of $\underline{\varepsilon}$. We show, in Corollary 14, that the prices remain $O(sC)$. Since $1/\underline{\varepsilon} = O(s)$, the numerators that we manipulate are $O(s^2C)$, so triple precision suffices.

Returning to Figure 1, we claim that our invariants hold at entry to the main loop, with $\varepsilon = \overline{\varepsilon}$. The magnitude of any cost is at most C, all prices are then zero, and $\overline{\varepsilon} > C$, so we have $-\overline{\varepsilon} < c_p(v, w) < \overline{\varepsilon}$, for every arc $v \to w$ in N_G. Both **I3** and **I4** follow from this, and **I1** and **I2** are clear. So the scaling phases can commence, as we discuss in Section 6. As for the final rounding:

Prop 4: The final call to *Refine* has $\varepsilon = \underline{\varepsilon} < 1/(s+2)$. After that call, suppose that we round our prices to integers by computing $\widetilde{p_d}(v) := \lfloor p_d(v) + k\underline{\varepsilon} \rfloor$ for some integer k in the range $[0 \,..\, 1/\underline{\varepsilon})$, the same k for all nodes v. We will always be able to find a k in this range for which the resulting rounded prices make all arcs proper.

Proof: This rounding operation is monotonic and commutes with integer shifts, so an arc that is proper before we round will remain proper afterward. For example, an idle arc $v \to w$ that is proper before we round has $c(v, w) + p_d(w) \geq p_d(v)$; this implies $c(v, w) + \widetilde{p_d}(w) \geq \widetilde{p_d}(v)$, so the arc will be proper afterward as well.

We claim next that all of the idle arcs are proper before we round. Since all costs are integers and $\underline{\varepsilon}$ is the reciprocal of an integer, all costs are multiples of $\underline{\varepsilon}$. All prices are multiples of $\underline{\varepsilon}$ as well, by **I2**, so all reduced costs are multiples of $\underline{\varepsilon}$. All idle arcs, which are $\underline{\varepsilon}$-proper by **I3**, are then automatically proper, by Prop 3.

So all of the arcs that start out improper must be saturated, and our goal is to find a k that will convert all such arcs into being proper. Any such arc $v \to w$ is $\underline{\varepsilon}$-proper, and its reduced cost is a multiple of $\underline{\varepsilon}$; so we must have $c_p(v, w) = \underline{\varepsilon}$, which means that $p_d(w) \equiv p_d(v) + \underline{\varepsilon} \pmod 1$. Of the $1/\underline{\varepsilon}$ possible values for k, one will make $p_d(w)$ round up while making $p_d(v)$ round down, thus sending the rounded reduced cost $c_{\tilde{p}}(v, w)$ all the way up from $\underline{\varepsilon}$ to 1; but all others will cause $p_d(w)$ and $p_d(v)$ to round in the same direction, resulting in $c_{\tilde{p}}(v, w) = 0$ and the arc $v \to w$ becoming proper. We avoid the one bad value.

Each saturated arc $v \to w$ that starts out improper determines one bad value for k in this way, and we can compute that bad value from the fractional part of either $p_d(v)$ or $p_d(w)$. The flow f has precisely $3s$ saturated arcs. Each of the s saturated bipartite arcs might rule out a different possibility for k. Of the s saturated left-dummy arcs, however, all of the ones that start out improper, however

[9] Gabow and Tarjan [10] call "eligible" the arcs that we call *ε-tight*.

Refine(f, p, ε)
  convert the s bipartite arcs saturated in f to idle;
  raise prices, as in Fig. 3, to make all arcs ε-proper;
  $S = \{\text{surpluses}\} := \{\text{the } s \text{ women matched on entry}\}$;
  $D = \{\text{deficits}\} := \{\text{the } s \text{ men matched on entry}\}$;
  int $h := s$; **while** $h > 0$ **do**
    build a shortest-path forest with current surpluses as tree roots, until reaching a current deficit;
    raise prices at the forest nodes by multiples of ε, creating at least one length-0 augmenting path;
    find a maximal set $\boldsymbol{\mathcal{P}}$ of compatible length-0 augmenting paths;
    augment f along each path in $\boldsymbol{\mathcal{P}}$, shrinking S and D so as to reduce $|S| = |D| = h$ by $|\boldsymbol{\mathcal{P}}|$;
  od;

Figure 2. The high-level structure of *Refine*

many of them there are, must rule out the same possibility for k, since all left-dummy arcs leave the same node: the source. In a similar way, of the s saturated right-dummy arcs, all of the ones that start out improper must rule out the same possibility for k. As a result, at most $s + 2$ possibilities are ruled out overall. Since $1/\varepsilon \geq s + 3$, we will be able to find a k that is not bad for any arc. Rounding all prices to integers using this value for k makes all arcs proper, thus demonstrating that the output flow f is min-cost. ■

6. THE SCALING PHASE *Refine*

The routine *Refine*, sketched in Figure 2, carries out a scaling phase, much as in Gabow-Tarjan. As in the Hungarian Method, we do a Dijkstra-like search to build a shortest-path forest and do a round of price increases. But then, as in Hopcroft-Karp, we augment, not just along the single length-0 augmenting path that our price increases have ensured, but along a maximal set of compatible such paths.

During *Refine*, the flux f temporarily degenerates from a flow into a pseudoflow. A *surplus* of a pseudoflow is a node other than the sink at which more flow enters than leaves; and a *deficit* of a pseudoflow is a node other than the source at which more flow leaves than enters.

For a woman x, the *left stub to* x is the pseudoflow on N_G that saturates the left-dummy arc $\vdash \rightarrow x$, but leaves all other arcs idle. Symmetrically, for a man y, the *right stub from* y saturates only the right-dummy arc $y \rightarrow \dashv$. Any pseudoflow f that arises in *Refine* is the sum of some flow, some left-stubs, and some right-stubs. The flow component, which we denote $\hat{f}$, encodes the matching that *Refine* has constructed so far, during this scaling phase. The left-stubs remember those women who were matched at the end of the previous phase and who have not yet been either matched or replaced in this phase. Those women are the surpluses of f, and they constitute the set S. The right-stubs remember the previously matched men in a similar way. Those men are the deficits of f, and they constitute the set D.

When *Refine* is called, f is an integral flow of value $|f| = s$. But *Refine* starts by altering f so as to zero the flow along the s bipartite arcs that were saturated. The initialization of *Refine* then raises prices so that every arc in N_G becomes ε-proper, for the resulting pseudoflow f and for the new, smaller value of ε.

The rest of *Refine*, its *main loop*, finds augmenting paths and augments along them, each such path joining a surplus in S to a deficit in D. By augmenting along s such paths, we return f to being a flow once again, but now with all arcs ε-proper, rather than just $(q\varepsilon)$-proper. Unlike in Gabow-Tarjan, however, our augmenting paths are allowed to visit the source and sink. For example, such a path could first back up from a surplus x along the saturated left-dummy arc $\vdash \rightarrow x$ and then move forward along some idle left-dummy arc $\vdash \rightarrow x'$. When we augment along that path, the arcs reverse their idle-versus-saturated status, thus recording the fact that x' has replaced x in the set of women who are going to end up married.

6.1. The residual digraph R_f

Given an integral pseudoflow f, we define the *residual digraph* R_f as follows. Every node in N_G becomes a node in R_f. Every idle arc $v \rightarrow w$ in N_G becomes a forward link $v \Rightarrow w$ in R_f; and every saturated arc $v \rightarrow w$ becomes a backward link $w \Rightarrow v$.[10] We will augment along paths in R_f. Note that, because of the source and sink, paths in R_f need not alternate between forward and backward links.

We define the *length* of a forward link $v \Rightarrow w$ in the residual digraph R_f to be $l_p(v \Rightarrow w) := \lceil c_p(v, w)/\varepsilon \rceil$, while the *length* of a backward link $w \Rightarrow v$ in R_f is defined by $l_p(w \Rightarrow v) := 1 - \lceil c_p(v, w)/\varepsilon \rceil$. Note that these definitions are ceiling quantized. It follows from **I3** that the lengths of forward and backward links are nonnegative integers. We can also verify that:

Prop 5: Raising the price $p_d(v)$ at some node v in N_G by ε lowers the length of any link in R_f leaving v by 1 and raises the length of any link entering v by 1.

Prop 6: An arc is ε-tight just when the link that it generates has length 0, and a saturated arc is ε-snug just when the backward link that it generates has length 0 or 1.

An *augmenting path* is a simple path in R_f that starts at a surplus and ends at a deficit; it is allowed to visit either the source or the sink or both (in either order). An augmenting path has length zero just when all of its links are length zero, which, by Prop 6, happens just when all of the arcs underlying its links are ε-tight.

[10]Note that we use different terms on our three different levels of graphs: The original bipartite graph G has vertices and edges (x, y). The flow network N_G has nodes and arcs $v \rightarrow w$; all of these arcs are oriented from left to right, and it is these arcs that have reduced costs. The residual digraph R_f has nodes and links $v \Rightarrow w$; some of the links go forward while others go backward, and each link has a quantized length, which is a nonnegative integer.

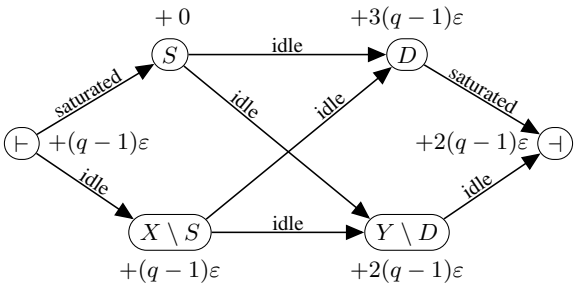

Figure 3. Price increases during the initialization of *Refine*

During *Refine*, we generalize the invariant **I1** into **I1′** (of which **I1** is the special case $h = 0$) and we add **I5**:

I1′ The flux f is a pseudoflow consisting of an integral flow $\hat{f}$ of value $|\hat{f}| = s - h$ supplemented by left stubs to each of the women in S and by right stubs from each of the men in D, where $|S| = |D| = h$.

I5 The residual digraph R_f has no cycles of length zero.

Prop 7: In any residual digraph R_f that arises during *Refine*, the in-degree of a woman is at most 1, while the in-degree of a surplus is 0. Symmetrically, the out-degree of a man is at most 1, while the out-degree of a deficit is 0.

Proof: Consider a woman x, and consider a link in R_f that arrives at x. Such a link can arise either because the left-dummy arc $\vdash \to x$ is idle, leading to the forward link $\vdash \Rightarrow x$, or because some bipartite arc leaving x, say $x \to y$, is saturated, leading to the backward link $y \Rightarrow x$. Any bipartite arc that is saturated in the pseudoflow f must be saturated also in its flow component $\hat{f}$, since no stub saturates any bipartite arcs. Since flow is conserved at x in the flow $\hat{f}$, there can't be more than one bipartite arc leaving x that is saturated, and there can't be even one such saturated arc unless the left-dummy arc $\vdash \to x$ is also saturated. So the in-degree of x in R_f is at most 1. Furthermore, for x to be a surplus, the left-dummy arc $\vdash \to x$ must be saturated and no bipartite arc leaving x can be saturated; so the in-degree of x is then 0.

We analyze the out-degree of a man symmetrically. ■

Corollary 8: On any augmenting path in *Refine*, the only surplus is the surplus at which it starts and the only deficit is the deficit at which it ends.

6.2. Before the main loop starts

During the initialization of *Refine*, zeroing the flow along all bipartite arcs establishes **I1′** with $h = s$ and $\hat{f} = 0$. It also establishes **I4** trivially, since all bipartite arcs are now idle. There are now also no links $y \Rightarrow x$ in the residual digraph that go backward from the men's side to the women's side; hence, there can't be any cycles at all in the residual digraph, so **I5** holds. As for **I2**, all prices are currently multiples of ε and will remain so. We then establish **I3** by raising prices as indicated in Figure 3.

Prop 9: While initializing *Refine*, raising our prices as indicated in Figure 3 makes all arcs ε-proper for the new, smaller ε, thereby establishing **I3**.

Proof: Before we raise any prices, all of the saturated left-dummy arcs $\vdash \to x$ are $(q\varepsilon)$-proper, so they satisfy $c_p(\vdash, x) \le q\varepsilon$. All of the idle left-dummy arcs $\vdash \to x$ are also $(q\varepsilon)$-proper, all of the prices are multiples of $q\varepsilon$, and the costs of all dummy arcs are zero. It follows from Prop 3 that the idle left-dummy arcs are actually proper, with $c_p(\vdash, x) \ge 0$. Symmetrically, the saturated right-dummy arcs $y \to \dashv$ satisfy $c_p(y, \dashv) \le q\varepsilon$ and the idle right-dummy arcs are actually proper, with $c_p(y, \dashv) \ge 0$. The bipartite arcs $x \to y$ come in two flavors. Some of them were idle also in the flow that was in effect when *Refine* was called. Those arcs were idle and $(q\varepsilon)$-proper, so they satisfy $c_p(x, y) > -q\varepsilon$. The others are idle now, but they were saturated when *Refine* was called. By **I4**, we conclude that $c_p(x, y) > -q\varepsilon$ also for those arcs.

We now analyze Figure 3 from left to right, verifying that the indicated price increases leave all arcs ε-proper. Let's use p' to denote the prices after we have raised them.

Consider first a saturated left-dummy arc $\vdash \to x$. We start with $c_p(\vdash, x) = c(\vdash, x) - p_d(\vdash) + p_d(x) \le q\varepsilon$. The woman x is now definitely a surplus, so we leave the price at x unchanged: $p'_d(x) := p_d(x)$. But we raise the price at the source: $p'_d(\vdash) := p_d(\vdash) + (q-1)\varepsilon$. So we have

$$c_{p'}(\vdash, x) = c(\vdash, x) - p'_d(\vdash) + p'_d(x) = c_p(\vdash, x) - (q-1)\varepsilon.$$

So $c_{p'}(\vdash, x) \le \varepsilon$, and the saturated left-dummy arc $\vdash \to x$ is left ε-proper.

What about an idle left-dummy arc $\vdash \to x$? We start with $c_p(\vdash, x) \ge 0$. And we add $(q-1)\varepsilon$ to the prices at both $\vdash$ and at x; so we end with $c_{p'}(\vdash, x) \ge 0$, and the idle arc $\vdash \to x$ is also left ε-proper.

We consider the bipartite arcs $x \to y$ next. They are now all idle, and we have seen that $c_p(x, y) > -q\varepsilon$, whether the arc $x \to y$ was idle or saturated at the call to *Refine*. The price at x either stays the same or goes up by $(q-1)\varepsilon$, according as x does or does lie in S. So $p'_d(x) \le p_d(x) + (q-1)\varepsilon$. The price at y goes up either by $3(q-1)\varepsilon$ or by $2(q-1)\varepsilon$, according as y does or does not lie in D. So $p'_d(y) \ge p_d(y) + 2(q-1)\varepsilon$. We thus have

$$\begin{aligned} c_{p'}(x, y) &= c(x, y) - p'_d(x) + p'_d(y) \\ &\ge c(x, y) - p_d(x) - (q-1)\varepsilon + p_d(y) + 2(q-1)\varepsilon \\ &\ge c_p(x, y) + (q-1)\varepsilon, \end{aligned}$$

which implies $c_{p'}(x, y) > -\varepsilon$. Thus, all bipartite arcs are left ε-proper.

The right-dummy arcs are similar to the left-dummy arcs. The idle ones start out proper and remain proper, since we raise the prices at both ends by the same amount. For the saturated ones, we raise the price at the left end by $(q-1)\varepsilon$ more than we raise the price at the right end. This ensures

```
for all nodes v, set ℓ(v) := ∞;
for all surpluses σ, set ℓ(σ) := 0 and insert(σ, 0);
do v := delete-min();
  for all links v ⇒ w leaving v in R_f do
    L := ℓ(v) + l_p(v ⇒ w); L_old := ℓ(w);
    if L ≤ Λ and L < L_old then
      set ℓ(w) := L;
      if L_old = ∞ then insert(w, L)
          else decrease-key(w, L) fi;
    fi;
  od;
  add v to the forest;
until v is a deficit od;
```

Figure 4. Building a shortest-path forest in *Refine* via Dijkstra

that the saturated right-dummy arcs are left ε-proper, so **I3** is established. ■

6.3. Building the shortest-path forest

The main loop of *Refine* starts by building a shortest-path forest with the h surpluses remaining in S as the roots of the trees, stopping when a deficit first joins the forest. We will prove a bound in Corollary 13 on how long a path we might need, to first reach a deficit. For now, we just assume that Λ is some such bound: Whenever we start building a forest, we assume that some path in R_f from some surplus to some deficit will be found whose length is at most Λ.

We build the forest using a Dijkstra-like search, as shown in Figure 4. The value $\ell(v)$, when finite, stores the minimum length of any path in R_f that we've found so far from some surplus to v. We use a heap to store those nodes v with $\ell(v) < \infty$ until they join the forest. The key of a node v in the heap is $\ell(v)$. The commands *insert*, *delete-min*, and *decrease-key* operate on that heap.

Because our keys are small integers and our heap usage is monotone[11], we can follow Gabow-Tarjan in using Dial [6] to avoid any logarithmic heap overhead. We maintain an array Q, where $Q[j]$ points to a doubly-linked list of those nodes in the heap that have key j, the double-linking enabling the deletion that is part of a *decrease-key*. Exploiting our assumed bound Λ, we allocate the array Q as $Q[0\,..\,\Lambda]$. We ignore any paths we find whose lengths exceed Λ. We also maintain an integer B, which stores the value $\ell(v)$ for the node v that was most recently added to the forest. We add nodes v to the forest in nondecreasing order of $\ell(v)$, so B never decreases. To implement *delete-min*, we look at the lists $Q[B]$, $Q[B+1]$, and so on, removing and returning the first element of the first nonempty list we find.

[11]When building a shortest-path forest, our heap usage is *monotone* in the sense of Cherkassky, Goldberg, and Silverstein [5]: If k is the key of a node that was just returned by a *delete-min*, then the key parameter in any future call to *insert* or *decrease-key* will be at least k. The Dial technique depends upon this monotonicity.

By our assumption about Λ, some deficit δ with $\ell(\delta) \le \Lambda$ eventually enters the forest, at which point we stop building it. The space and time that we spend are both $O(m + \Lambda)$, where the Λ term accounts for the space taken by the array Q and for the time taken to scan that array once while doing *delete-min* operations.

6.4. Raising the prices

The next step in the main loop of *Refine* is to raise prices. For each node v in the shortest-path forest, we set the new price $p'_d(v)$ by

$$p'_d(v) := p_d(v) + (\ell(\delta) - \ell(v))\varepsilon, \tag{3}$$

where δ in D is the deficit whose discovery halted the growth of the forest. Let σ be the surplus at the root of the tree that δ joins. We have $p'_d(\sigma) = p_d(\sigma) + \ell(\delta)\varepsilon$, but $p'_d(\delta) = p_d(\delta)$. So Prop 5 tells us that our price increases shorten the path from σ to δ by $\ell(\delta)$ length units. If our invariants are preserved, the price increases must leave that path of length zero; and it's clear that **I1′** and **I2** continue to hold.

Prop 10: During the main loop of *Refine*, using (3) to raise prices at the nodes in the shortest-path forest preserves invariants **I3** through **I5**.

Proof: See the unabridged version [17]. ■

6.5. Finding compatible augmenting paths

In Hopcroft-Karp [13], augmenting paths are compatible when they are vertex-disjoint. But we need a more liberal notion of compatibility in *Refine*, which we can define in two different, equivalent ways: We define augmenting paths to be *link-compatible* when they start at distinct surpluses, end at distinct deficits, and don't share any links. We define augmenting paths to be *node-compatible* when they are node-disjoint, except perhaps for the source and sink.

Prop 11: Augmenting paths in the residual digraph R_f are link-compatible just when they are node-compatible.

Proof: It's easy to see that node-compatible augmenting paths must also be link-compatible. By node-compatibility, they must start at distinct surpluses and end at distinct deficits. And they can't share any links, since every link has at least one end node that isn't the source or the sink.

Conversely, consider some augmenting paths that are link-compatible, and let x be any woman. If x is a surplus, then, by Corollary 8, an augmenting path can visit x only by starting at x, which only one of our link-compatible paths can do. If x is not a surplus, then an augmenting path can visit x only by arriving at x over a link. By Prop 7, the in-degree of x in R_f is at most 1, and at most one of our link-compatible paths can travel over any single link. So x is visited by at most one of our paths.

Similarly, if a man y is a deficit, then an augmenting path can visit y only by ending at y, which only one of our paths can do. If y is not a deficit, an augmenting path can visit y only if it leaves y along a link. But there is at most one

link leaving y, which at most one of our paths can traverse. So link-compatible paths are also node-compatible. ■

Let R_f^0 denote the subgraph of R_f whose links are of length 0. To find a maximal set $\boldsymbol{\mathcal{P}}$ of compatible augmenting paths in the subgraph R_f^0, we can search for paths that are either link-compatible or node-compatible. To find a maximal set of link-compatible paths, we do a depth-first search of the subgraph R_f^0, starting at each surplus in turn and trying for a path to a deficit. This search is allowed to revisit a node that it already visited, resuming the processing of that node by examining outgoing links that were not examined earlier. Despite revisiting nodes in this way, we won't output any paths that aren't simple, because **I5** assures us that the subgraph R_f^0 is acyclic. Alternatively, we could search for node-compatible paths. That search is a bit more complicated, since the source and sink are then the only nodes that the search is allowed to revisit [18]; but it should run faster, since we can cut off the searching sooner.

6.6. Augmenting along those paths

Finally, we augment f along each of the paths in $\boldsymbol{\mathcal{P}}$. These augmentations reverse the forward-versus-backward orientation of each link along the path and the idle-versus-saturated status of each underlying arc. This restores the flow balance of the surplus at which the path starts and of the deficit at which it ends, while no other flow-balances are affected. So **I1′** is preserved, but with $h = |S| = |D|$ reduced by $|\boldsymbol{\mathcal{P}}|$. Augmenting doesn't change any prices, so **I2** is preserved. As for **I3**, the length-0 forward links along an augmenting path become length-1 backward links, while the length-0 backward links become length-1 forward. So the underlying arcs are all left ε-proper, although no longer ε-tight. The idle bipartite arcs that become saturated during the augmentation, while not left ε-tight, are left ε-snug, by Prop 6; so **I4** is also preserved. Finally, for **I5**: Augmentation reverses the directions of some links, and this may well produce cycles in R_f. But every link that changes state during an augmentation ends up being of length 1; so no cycle that exploits any such link can be of length 0.

7. Analyzing the performance

To finish analyzing *FlowAssign*, we need three things. First, we must choose Λ and show that the building of every shortest-path forest finds a path from a surplus to a deficit of length at most Λ. Second, we must show that our prices remain $O(sC)$. Third, to achieve the weight-scaling time bound, we must show that the main loop of *Refine* executes $O(\sqrt{s})$ times. The key to all three is the *inflation bound*, which limits the total amount by which prices can increase during a call to *Refine*. This bound applies at any clean point in *Refine*'s main loop, where a *clean point* is just before or just after the execution of one of the four statements of the main loop. By the way, the analyses of Hopcroft-Karp [13] and of Gabow-Tarjan [10] involve analogous bounds.

In the round of price increases that follows the building of a shortest-path forest, the surpluses at the roots of the trees in that forest have their prices increased by $\ell(\delta)\varepsilon$, where δ is the deficit whose discovery stopped the building of the forest. Note that this is the largest increase that happens, during that round, to the price at any node. Let's refer to that quantity as the *max increase* of that round.

Prop 12: Consider any clean point during an execution of the main loop of *Refine*. Let Δ denote the sum of the max increases of all of the rounds of price increases so far, during this call to *Refine*. We then have the *inflation bound*:

$$h\Delta \leq (4q+4)s\varepsilon. \tag{4}$$

This bound holds even after a round of price increases, during which Δ increased, and before the subsequent batch of augmentations, which will cause h to decrease.

Proof: Let f be the flow when *Refine* was called, while f' is the current pseudoflow. Let p be the prices when *Refine* entered its main loop, while p' is the current prices. We show that $(c_{p'} - c_p)(f' - f) = h\Delta$ by considering the changes in price at the $2h$ nodes where the flux $f' - f$ does not conserve flow. We then show (4) by using our bounds on the reduced costs of individual arcs. For details, see [17]. ■

We can hence take the bound Λ of Section 6.3 to be $\Lambda := (4q+4)s/h = O(s)$; so each iteration of the main loop in *Refine* takes space and time $O(m + \Lambda) = O(m)$. And we can show that our prices remain $O(sC)$.

Corollary 13: The building of any shortest-path forest in the procedure *Refine* is always halted by finding a deficit δ with $\ell(\delta) \leq (4q+4)s/h$.

Corollary 14: The prices in *FlowAssign* remain $O(sC)$.

Proof: For both, see the unabridged version [17]. ■

Prop 15: The main loop of *Refine* executes $O(\sqrt{s})$ times.

Proof: Note first that every iteration of the main loop reduces h: The repricing ensures that at least one length-0 augmenting path exists in R_f, so we have $|\boldsymbol{\mathcal{P}}| \geq 1$.

We claim next that every iteration of the main loop, except perhaps the first, increases Δ by at least ε. Note that Δ could fail to increase in some iteration only if we found a path in R_f from some surplus to some deficit all of whose links were already of length 0, without any need for any price increases. If such a path A existed in any iteration after the first, however, consider the maximal set $\boldsymbol{\mathcal{P}}$ that was computed near the end of the preceding iteration. The only changes to the state (f, p) that happen after $\boldsymbol{\mathcal{P}}$ is computed and before A is discovered are the augmentations along the paths in $\boldsymbol{\mathcal{P}}$. But those augmentations affect only the links along those paths, and none of those links can appear in A, since the augmentations leave those links with length 1. So the length-0 augmenting path A must be link-compatible with all of the paths in $\boldsymbol{\mathcal{P}}$, and is hence compatible with them. But $\boldsymbol{\mathcal{P}}$ was maximal; so no such A can exist, and Δ increases in all iterations of the main loop after the first.

Consider the state after $\sqrt{(4q+4)s}$ iterations of the main loop. We must have $\Delta \geq \sqrt{(4q+4)s}\,\varepsilon$, since Δ increases in every iteration. Applying the inflation bound (4), we deduce that $h \leq \sqrt{(4q+4)s}$. Since h decreases in every iteration, we see that the total number of iterations is at most $2\sqrt{(4q+4)s} = O(\sqrt{s})$. ■

So we have finally established our main result:

Theorem 16: FlowAssign solves the problem **ImpA** in space $O(m)$ and in time $O(m\sqrt{s}\log(sC))$.

8. The variant subroutine *TightRefine*

One way in which *FlowAssign* differs from Gabow-Tarjan involves invariant **I4**, which, you recall, requires that all saturated bipartite arcs be kept ε-snug.

Since Gabow-Tarjan works directly on the graph G, all arcs in Gabow-Tarjan are bipartite. And Gabow-Tarjan keeps all of its saturated arcs, not only ε-snug, but actually ε-tight. Perhaps Gabow and Tarjan did this because they were following the Hungarian Method, which keeps its saturated arcs precisely tight.

Once we move from the graph G to the flow network N_G, it is hopeless to keep all saturated arcs ε-tight. We can and do keep all saturated arcs ε-proper, which puts an upper bound on their reduced costs. But the reduced costs of the dummy saturated arcs may get large negative — that seems unavoidable. For the bipartite saturated arcs, however, we can and do impose a lower bound on their reduced costs. In **I4**, we insist that they be ε-snug. Following Gabow-Tarjan, we could go further and insist that they actually be ε-tight. Let's refer to that stronger invariant as **I4′**.

The main difficulty with **I4′** crops up when augmenting along an augmenting path. With the executable code of *Refine* as it now stands, the arcs along an augmenting path that change status from idle to saturated end up being ε-snug, but not ε-tight. The bipartite arcs of this type would blatantly violate **I4′**.

Gabow and Tarjan deal with this difficulty by changing the code. In our language, they increase the price of every man along an augmenting path by ε, as part of doing the augmentation. With those price increases, the bipartite arcs that change status to saturated end up ε-tight. The good news is that this change to the code succeeds even in our more complicated context of *FlowAssign*, where there are dummy arcs, augmenting paths can visit the source and the sink, and so forth. So we end up with two variants of the algorithm *FlowAssign*, one using the subroutine *SnugRefine* that we've analyzed in this paper and the other using *TightRefine*, a variant in which augmenting along an augmenting path includes raising the prices of the men on that path by ε. *TightRefine* is more delicate to analyze than *SnugRefine*, but both routines do the same job in the same space and time bounds [18]. It isn't clear which subroutine would perform better in practice.

References

[1] R. K. Ahuga, J. B. Orlin, C. Stein, and R. E. Tarjan, "Improved algorithms for bipartite network flow," *SIAM J. on Computing*, vol. 23, no. 5, pp. 906–933, 1994.

[2] D. P. Bertsekas, "Auction algorithms for network flow problems: A tutorial introduction," *Computational Optimization and Applications*, vol. 1, pp. 7–66, 1992.

[3] D. P. Bertsekas and D. A. Castañon, "A forward/reverse auction algorithm for asymmetric assignment problems," *Computational Optimization and Applications*, vol. 1, pp. 277–297, 1992.

[4] R. Burkard, M. Dell'Amico, and S. Martello, *Assignment Problems*, Society for Industrial and Applied Mathematics (SIAM), 2009.

[5] B. V. Cherkassky, A. V. Goldberg, and C. Silverstein, "Buckets, heaps, lists, and monotone priority queues," *ACM-SIAM Symposium on Discrete Algorithms (SODA'97)*, SIAM, pp. 83–92, 1997.

[6] R. B. Dial, "Algorithm 360: Shortest path forest with topological ordering," *Commun. ACM*, vol. 12, no. 11, pp. 632–633, 1969.

[7] R. Duan and H.-H. Su, "A scaling algorithm for maximum weight matching in bipartite graphs," *ACM-SIAM Symposium on Discrete Algorithms (SODA'12)*, SIAM, pp. 1413–1424, 2012.

[8] T. Feder and R. Motwani, "Clique partitions, graph compression and speeding-up algorithms," *J. of Computer and System Sciences*, vol. 51, no. 2, pp. 261–272, 1995.

[9] M. L. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *J. of the ACM*, vol. 34, no. 3, pp. 596–615, 1987.

[10] H. N. Gabow and R. E. Tarjan, "Faster scaling algorithms for network problems," *SIAM J. on Computing*, vol. 18, no. 5, pp. 1013–1036, 1989.

[11] A. V. Goldberg and R. Kennedy, "An efficient cost scaling algorithm for the assignment problem," *Mathematical Programming*, vol. 71, pp. 153–177, 1995.

[12] ——, "Global price updates help," *SIAM J. on Discrete Mathematics*, vol. 10, no. 4, pp. 551–572, 1997.

[13] J. E. Hopcroft and R. M. Karp, "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs," *SIAM J. on Computing*, vol. 2, no. 4, pp. 225–231, 1973.

[14] M.-Y. Kao, T.-W. Lam, W.-K. Sung, and H.-F. Ting, "A decomposition theorem for maximum weight bipartite matchings," *SIAM J. on Computing*, vol. 31, no. 1, pp. 18–26, 2001.

[15] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Research Logistics*, vol. 2, pp. 83–97, 1955; republished with historical remarks in *Naval Research Logistics*, vol. 52, pp. 6–21, 2005.

[16] J. B. Orlin and R. K. Ahuja, "New scaling algorithms for the assignment and minimum mean cycle problems," *Mathematical Programming*, vol. 54, pp. 41–56, 1992.

[17] L. Ramshaw and R. E. Tarjan, "A weight-scaling algorithm for min-cost imperfect matchings in bipartite graphs," HP Labs technical report HPL-2012-72R1, www.hpl.hp.com/techreports/HPL-2012-72R1.html, 2012.

[18] ——, "On minimum-cost assignments in unbalanced bipartite graphs," HP Labs technical report HPL-2012-40R1, www.hpl.hp.com/techreports/HPL-2012-40R1.html, 2012.

[19] M. Thorup, "Integer priority queues with decrease key in constant time and the single source shortest paths problem," *J. of Computer and System Sciences*, vol. 69, no. 3, pp. 330–353, 2004.