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

Lyle Ramshaw, Robert E. Tarjan

HP Laboratories
HPL-2012-72

Keyword(s):
assignment problem; imperfect matching; minimum-cost-matching; unbalanced bipartite graph; weight-scaling algorithm

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 in G 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 that allows us to find such a matching in time $O(m\sqrt{n} \log(nC))$ via weight scaling. Our results also provide insight into the design space of efficient weight-scaling matching algorithms.

These ideas are presented in greater depth in HPL-2012-40 [17].

External Posting Date: June 6, 2012 [Fulltext]
Internal Posting Date: June 6, 2012 [Fulltext]

Approved for External Publication

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

Lyle Ramshaw
HP Labs
lyle.ramshaw@hp.com

Robert E. Tarjan
Princeton and HP Labs
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 in G 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 that allows us to find such a matching in time $O(m\sqrt{s}\log(sC))$ via weight scaling. Our results also provide insight into the design space of efficient weight-scaling matching algorithms.

These ideas are presented in greater depth in HPL-2012-40 [17].

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 these 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 G by $\nu(G)$, we consider two variants of the *assignment problem* [4]:

Perfect Assignments (PerA) 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".

Imperfect Assignments (ImpA) 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 we have $r \leq n \leq m \leq rn$; and we assume throughout that $s \geq 1$ and, once we introduce C, 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. By using Fibonacci heaps, Fredman and Tarjan [9] devised a version that runs in space $O(m)$ and in time $O(mn + n^2 \log n)$. That's the current record among strongly polynomial algorithms. If the weights are integers, then Thorup [18] can reduce 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 it fails to be strongly polynomial. Using weight-scaling, we can solve the assignment problem 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] We here 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 [17], 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 $\log \log r$.

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 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 Appendix B). 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 $O(n)$.

We discuss other related work and some open problems in Appendix A.

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 skew-symmetry that most authors adopt.

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 flow is conserved at all nodes except for the source and the sink. The *value* of a flow f, denoted $|f|$, is the total flow out of the source (and hence into the sink).

Warning: Most 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 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 $p_d(v) = -p_a(v)$.[7] (Our algorithms happen to work by lowering the acquire prices at nodes, which means raising their dispose prices.) 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 $c_p(v,w) := c(v,w) - p_d(v) + p_d(w)$ to define[8] the *net cost*[9] $c_p(v,w)$ from the cost $c(v,w)$. The theory of complementary slackness then gives us the following:

Prop 1. *Consider a flow f of value s 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 $v \to w$ with $c_p(v,w) > 0$ has $f(v,w) = 0$, then f is min-cost among all flows of value s.*

Given a pseudoflow f and prices p on the network N_G, we define an idle arc to be *proper* when its net cost is nonnegative, a saturated arc to be *proper* when its net cost is nonpositive, and an arc with fractional flow to be *proper* only when its net 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 integral 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 $\vdash \to x$ 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 $\vdash \to x$ will be idle, so we must have $p_d(\vdash) \leq p_d(x)$. For all left-dummy arcs to be proper, we need

$$\max_{x \text{ married}} p_d(x) \leq p_d(\vdash) \leq \min_{x \text{ maiden}} p_d(x); \tag{3}$$

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:

$$\max_{y \text{ bachelor}} p_d(y) \leq p_d(\dashv) \leq \min_{y \text{ married}} p_d(y); \tag{4}$$

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

When there are no maidens, choosing $p_d(\vdash)$ large enough makes all left-dummy arcs proper; and, without 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 when our matching is less than perfect, however. In this sense, **ImpA** is a somewhat harder problem than **PerA** (which might explain why **PerA** has been more studied).

The Hungarian Method generalizes easily to compute imperfect matchings that are min-cost [17] because it preserves the maiden and bachelor bounds.[10] 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 net 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 $v \to w$ depends upon its net cost $c_p(v, w)$ only through the quantity $\lceil c_p(v, w)/\varepsilon \rceil$. We are following Gabow-Tarjan by quantizing our net 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 $v \to w$ to be *ε-proper* when $c_p(v, w) > -\varepsilon$ and a saturated arc $v \to w$ 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 5. *Let $v \to w$ be an idle arc whose net cost is known to be a multiple of ε. If the arc $v \to w$ 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, not 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 net costs.

```
FlowAssign(G, t)
    (M, s) := HopcroftKarp(G, t);
    convert M into an integral flow f on N_G with |f| = s;
    set ε := ε̄ and, for all nodes v in N_G, set p_d(v) := 0;
    while ε > ε̲ do
        ε := ε/q;
        Refine(f, p, ε);
    od;
    round prices to integers that make all arcs proper;
```

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

An arc with net cost zero is often called *tight*. So we say that an idle arc $v \to w$ is ε-*tight* when $-\varepsilon < c_p(v,w) \le 0$, while a saturated arc $v \to w$ is ε-*tight* when $0 < c_p(v,w) \le \varepsilon$.[11] 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 $-\varepsilon < c_p(v,w) \le \varepsilon$.

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 $O(m\sqrt{s})$ [17].

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

I1 The flux f on N_G is a flow of value $|f| = s$.

I2 The prices at all nodes in N_G are multiples of ε.

I3 Every arc of N_G, idle or saturated, is ε-proper.

I4 Every saturated bipartite arc is ε-snug.

Note that ε is reduced by a factor of q at the start of each scaling phase. This reduction weakens **I2**, but strengthens **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 $e \ge 0$, so that ε is a positive integer; the *late phases* are those with $e < 0$, so that ε is the reciprocal of an integer.

Refine(f, p, ε)
    $S :=$ {the s women who are matched in f};
    $D :=$ {the s men who are matched in f};
    convert the s bipartite arcs that are saturated in f to idle;
    raise the prices p, as shown in Figure 3, to make all arcs ε-proper;
    int $h := s$; **while** $h > 0$ **do**
        build a shortest-path forest from the current surpluses S;
        raise prices at the forest nodes by multiples of ε, thereby
                generating at least one length-0 augmenting path;
        find a maximal set $\boldsymbol{\mathcal{P}}$ of compatible length-0 augmenting paths;
        augment f along all paths in $\boldsymbol{\mathcal{P}}$, reducing $h = |S| = |D|$ by $|\boldsymbol{\mathcal{P}}|$;
    od;

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

FlowAssign has to do arithmetic on prices, costs, and net costs. But the integer $1/\underline{\varepsilon} = q^{-e}$ is a common denominator for every price that ever arises. For simplicity, *FlowAssign* represents its prices, costs, and net costs as rational numbers with this common denominator, that is, as integer multiples of $\underline{\varepsilon}$. We show, in Corollary 18, 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 shortly. As for the final rounding:

Prop 6. *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. Deferred to Appendix C. □

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.

Refine deals with pseudoflows. A *surplus* of a pseudoflow f is a node other than the sink at which the entering flow exceeds the leaving flow; and a *deficit* is a node other than the source at which the leaving flow exceeds the entering flow.

For a woman x, the *left stub to* x is the pseudoflow that saturates the left-dummy arc $\vdash \to x$, but leaves all other arcs idle. Symmetrically, for a man y, the *right stub from* y saturates only the right-dummy arc $y \to \dashv$. 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 the pseudoflow 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 $w \Rightarrow v$.[12] We will augment along certain 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 $l_p(v \Rightarrow w) := \lceil c_p(v,w)/\varepsilon \rceil$ and the *length* of a backward link $w \Rightarrow v$ to be $l_p(w \Rightarrow v) := 1 - \lceil c_p(v,w)/\varepsilon \rceil$. Note that these definitions are ceiling quantized. **I3** requires that all arcs be maintained ε-proper, and it follows that the lengths of forward and backward links are nonnegative integers. We can also verify that:

Prop 7. *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 8. *An arc is ε-tight just when the link that it generates has length* 0, *and a saturated arc is ε-snug just when its (backward) link has length either* 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 8, is just when all of the underlying arcs are ε-tight.

During *Refine*, we replace the invariant **I1** with **I1′** and we add **I5**:

I1′ The flux f on N_G 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.

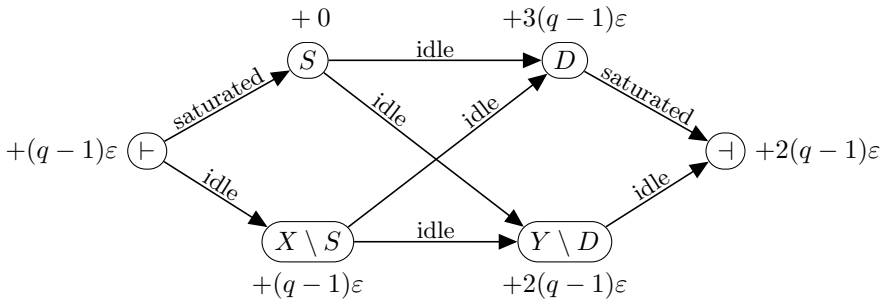

Figure 3: Price increases during the initialization of *Refine*

Prop 9. *In any residual digraph R_f that arises during* Refine*, the in-degree of any woman is at most* 1*, while the in-degree of any surplus is* 0*. Symmetrically, the out-degree of any man is at most* 1*, while the out-degree of any deficit is* 0*.*

Proof. Deferred to Appendix C — we exploit the fact that *Refine* preserves **I1′**. □

Corollary 10. *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

At the start 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 11. *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. Deferred to Appendix C. □

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 17 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[13], we can follow Gabow-Tarjan in using Dial [6] to avoid any logarithmic heap overhead. We

```
for all nodes v, set ℓ(v) := ∞;
for all surpluses σ in S, 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;
```

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

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.

By our assumption about Λ, some deficit δ with $\ell(\delta) \leq \Lambda$ 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

$$p'_d(v) := p_d(v) + (\ell(\delta) - \ell(v))\varepsilon, \tag{12}$$

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 7 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 13. *During the main loop of* Refine*, using equation (12) to raise prices at the nodes in the shortest-path forest preserves **I3** through **I5**.*

Proof. Deferred to Appendix C. □

6.5 Finding compatible augmenting paths

In Hopcroft-Karp, 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, they end at distinct deficits, and they don't share any links. We define augmenting paths to be *node-compatible* when they are node-disjoint, except for the source and sink.

Prop 14. *Augmenting paths in the residual digraph R_f are link-compatible just when they are node-compatible.*

Proof. Deferred to Appendix C. □

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 earlier examined. 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 need special treatment; but it should run faster, since we can cut off the searching sooner.

6.6 Augmenting along those paths

Finally, we augment the pseudoflow 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 links. So all of the underlying arcs are left ε-proper, though no longer ε-tight. The formerly idle bipartite arcs that become saturated during the augmentation, while not left ε-tight, are left ε-snug, by Prop 8; 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.

The inflation bound involves a quantity Δ. 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. At any clean point in the main loop of *Refine*, we define Δ to be the sum of the max increases of all of the rounds of prices increases that have happened so far, during this call to *Refine*.

Prop 15. *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*:*

$$h\Delta \leq (4q+4)s\varepsilon. \tag{16}$$

This 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. Deferred to Appendix C. □

We can now establish that the bound Λ in Section 6.3 can be taken 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 17. *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 18. *The prices in* FlowAssign *remain $O(sC)$.*

Proof. Both deferred to Appendix C. □

Prop 19. *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 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 (16), 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 the following:

Theorem 20. FlowAssign *solves* ***ImpA*** *in space $O(m)$ and time $O(m\sqrt{s}\log(sC))$.*

A Related work and open problems

We now discuss some other related work.

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 the 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 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, on 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].

B 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 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 net costs. But the net 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 net 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 [17]. It isn't clear which subroutine would perform better in practice.

C Deferred proofs

Prop 6. *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. And the argument for saturated arcs is similar.

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 net costs are multiples of $\underline{\varepsilon}$. All idle arcs, which are $\underline{\varepsilon}$-proper by **I3**, are then automatically proper, by Prop 5.

So our goal is to find a k for which the rounding will take all of the saturated arcs that are improper before we round and convert them into proper arcs. Let $v \to w$ be such an arc. It is $\underline{\varepsilon}$-proper, and its net cost is a multiple of $\underline{\varepsilon}$; so, if it fails to be proper, the only possibility is to have $c_p(v,w) = \underline{\varepsilon}$. Of the $1/\underline{\varepsilon}$ possible values for k, there is one that will result in the rounded net cost $c_{\tilde{p}}(v,w)$ jumping all the way up from the fraction $\underline{\varepsilon}$ to the integer 1. But any other value for k will result in $c_{\tilde{p}}(v,w)$ jumping down to the integer 0, meaning that the arc $v \to w$ becomes proper. We avoid the one bad value.

Each saturated arc $v \to w$ that is improper before the rounding determines one bad value for k in this way, and we can determine 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 distinct possibility for k. But note that, of the s saturated left-dummy arcs, all of the ones that are currently improper, however 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 are currently improper must rule out the same possibility for k. As a result, at most $s+2$ possibilities are ruled out overall. Since $1/\underline{\varepsilon} \geq s+3$, 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. □

Prop 9. *In any residual digraph R_f that arises during* Refine*, the in-degree of any woman is at most* 1*, while the in-degree of any surplus is* 0*. Symmetrically, the out-degree of any man is at most* 1*, while the out-degree of any deficit is* 0*.*

Proof. Consider a woman x, and consider a link in R_f that arrives at x. Such a link can arise in one of two ways: 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 f must be saturated also in the 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 $\vdash \to x$ that enters 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.

The argument for the out-degree of a man is symmetric. □

Prop 11. *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 $c_p(\vdash, x) \leq q\varepsilon$. 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 5, that the idle left-dummy arcs are actually proper, with $c_p(\vdash, x) \geq 0$. Symmetrically, the saturated right-dummy arcs $y \to \dashv$ satisfy $c_p(y, \dashv) \leq q\varepsilon$ and the idle right-dummy arcs $y \to \dashv$ are actually proper, with $c_p(y, \dashv) \geq 0$. 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 walk around the hexagon in Figure 3 counterclockwise, 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 $c_p(\vdash, x) = c(\vdash, x) - p_d(\vdash) + p_d(x) \leq q\varepsilon$. 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 $c_{p'}(\vdash, x) \leq \varepsilon$, 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 $c_p(\vdash, x) \geq 0$. And we add $(q-1)\varepsilon$ to the prices at both $\vdash$ and at x; so we end with $c_{p'}(\vdash, x) \geq 0$, 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 $p'_d(x) \leq p_d(x) + (q-1)\varepsilon$. 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 $p'_d(y) \geq p_d(y) + 2(q-1)\varepsilon$. We thus have

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

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

Prop 13. *During the main loop of* Refine, *using* $p'_d(v) := p_d(v) + (\ell(\delta) - \ell(v))\varepsilon$ *to raise prices at the nodes in the shortest-path forest preserves* ***I3*** *through* ***I5****.*

Proof. **Some preliminaries**

For each node v in the network N_G, let's define $i(v)$ to be the multiple of ε by which we raise the price $p_d(v)$. So, for nodes v in the forest, we set $i(v) := \ell(\delta) - \ell(v)$, while, for nodes v not in the forest, we set $i(v) := 0$. We then have the repricing

formula $p'_d(v) = p_d(v) + i(v)\varepsilon$, for all nodes v. Using this formula, we can express the impact of our repricings on an arc $v \to w$ without needing to know whether or not the nodes v and w lie in the forest: $c_{p'}(v, w) = c_p(v, w) + (i(w) - i(v))\varepsilon$.

Lemma 21. *Suppose that, at some point during the construction of the shortest-path forest, the link $v \Rightarrow w$ in the residual digraph is scanned. We can then conclude that, after the forest's construction, we will have*

$$\ell(v) + l_p(v \Rightarrow w) \geq \ell(\delta) - i(w). \tag{22}$$

Proof. Three cases can arise, as we consider the length $L := \ell(v) + l_p(v \Rightarrow w)$ of the new path that we have found from some surplus to w.

First, we might find that $L > \Lambda$, so that we ignore the new path completely. But we are assuming that some path to a deficit is eventually found of length at most Λ, which means that $\Lambda \geq \ell(\delta)$. So $L > \ell(\delta)$ in this case, which implies (22).

If the first case does not pertain, we next consider $\ell(w)$, which might be infinite, finite but larger than L, or at most L. Whichever of these three subcases applies, our processing of the link $v \Rightarrow w$ reduces $\ell(w)$ enough so that $L \geq \ell(w)$ then holds. During the remainder of the forest construction, $\ell(v)$ does not change and $\ell(w)$ can only decrease further. So we still have $L \geq \ell(w)$ after the forest-building has ceased.

When the forest-building has ceased, it might be that w has not entered the forest; that constitutes our second top-level case. We then have $\ell(w) \geq \ell(\delta)$, since δ has entered the forest and nodes enter the forest in nondecreasing order of their ℓ values. So we have $L \geq \ell(w) \geq \ell(\delta)$ in this case, which also implies (22).

Finally, in our third top-level case, w has joined the forest at some point during its construction, along with v. We then have $i(w) = \ell(\delta) - \ell(w)$, so $L \geq \ell(w)$ implies $L \geq \ell(\delta) - i(w)$, and (22) holds for the third and last time. □

Prop. 13A: Idle arcs are left ε-proper

We now establish **I3** for idle arcs. Consider an idle arc $v \to w$ in the network N_G, which gives rise to the forward link $v \Rightarrow w$ in the residual digraph R_f. This arc was ε-proper before we raised our prices, so we had $c_p(v, w) > -\varepsilon$. We must show that $c_{p'}(v, w) = c_p(v, w) + (i(w) - i(v))\varepsilon > -\varepsilon$.

If v does not belong to the forest, we have $i(v) = 0$. The net cost of the arc $v \to w$ couldn't then have decreased, so the arc $v \to w$ ends ε-proper because it started ε-proper. So we henceforth assume that v does belong to the forest.

If v entered the forest, then we must have scanned v, which means that we considered the forward link $v \Rightarrow w$. We can then apply Lemma 21 to deduce that

$$\ell(v) + l_p(v \Rightarrow w) \geq \ell(\delta) - i(w).$$

Since $v \Rightarrow w$ is a forward link and since $u + 1 > \lceil u \rceil$ for all real u, we have

$$\ell(v) + \frac{c_p(v, w)}{\varepsilon} + 1 > \ell(v) + \left\lceil \frac{c_p(v, w)}{\varepsilon} \right\rceil \geq \ell(\delta) - i(w).$$

Multiplying through by ε and rearranging, we find that

$$c_p(v, w) + \big(i(w) - (\ell(\delta) - \ell(v))\big)\varepsilon > -\varepsilon.$$

Since v belongs to the forest, we have $i(v) = \ell(\delta) - \ell(v)$, so $c_{p'}(v, w) > -\varepsilon$. Thus, our price increases do indeed leave all idle arcs ε-proper.

Prop. 13B: Saturated arcs are left ε-proper

To finish verifying **I3**, we next consider a saturated arc $v \to w$ in the network N_G, which gives rise to the backward link $w \Rightarrow v$ in the residual digraph R_f. This arc was ε-proper before we raised prices, so we had $c_p(v,w) \leq \varepsilon$. Our goal is to show that $c_{p'}(v,w) = c_p(v,w) + (i(w) - i(v))\varepsilon \leq \varepsilon$.

If w does not belong to the forest, we have $i(w) = 0$, so the arc $v \to w$ ends ε-proper because it started ε-proper. On the other hand, if w belongs to the forest, we must have scanned w and considered the link $w \Rightarrow v$. Applying Lemma 21 with v and w reversed, we deduce that

$$\ell(w) + l_p(w \Rightarrow v) \geq \ell(\delta) - i(v).$$

Since $w \Rightarrow v$ is a backward link and since $-u \geq -\lceil u \rceil$ for all real u, we have

$$\ell(w) + 1 - \frac{c_p(x,y)}{\varepsilon} \geq \ell(w) + 1 - \left\lceil \frac{c_p(x,y)}{\varepsilon} \right\rceil \geq \ell(\delta) - i(v).$$

Multiplying through by ε and rearranging, we find that

$$\varepsilon \geq c_p(v,w) + \big((\ell(\delta) - \ell(w)) - i(v)\big)\varepsilon.$$

Since w belongs to the forest, we have $i(w) = \ell(\delta) - \ell(w)$, so $c_{p'}(v,w) \leq \varepsilon$ as we hoped. Our price increases thus preserve **I3**.

Prop. 13C: Saturated bipartite arcs are left ε-snug

To show that **I4** is preserved, we consider a saturated, bipartite arc $x \to y$; we must show that $c_{p'}(x,y) = c_p(x,y) + (i(y) - i(x))\varepsilon > -\varepsilon$, which is the lower-bound part of ε-snugness. Since we are now dealing with a lower bound on the net cost of a saturated arc, though, our argument will have to differ significantly from our arguments for the two halves of **I3** above.

If x does not belong to the forest, we have $i(x) = 0$, so the arc $x \to y$ ends ε-snug because it started ε-snug. We henceforth assume that x belongs to the forest.

But how did x get into the forest? Since x is the tail of the saturated bipartite arc $x \to y$, the node x isn't a surplus; so x wasn't put into the forest initially. Since the arc $x \to y$ is saturated, the corresponding link in the residual digraph is the backward link $y \Rightarrow x$, which arrives at x. By Prop 9, the in-degree of a woman in the residual digraph never exceeds 1. So $y \Rightarrow x$ is the only link in the entire residual digraph that arrives at x. And the only way that x could have entered the forest is because y entered the forest first, causing the backward link $y \Rightarrow x$ to be scanned, and, sometime later, x was extracted from the heap with a *delete-min*.

It follows that $\ell(x)$ was determined entirely by the link $y \Rightarrow x$. So we have $\ell(x) = \ell(y) + l_p(y \Rightarrow x) = \ell(y) + 1 - \lceil c_p(x,y)/\varepsilon \rceil$. Since $u + 1 > \lceil u \rceil$ for all real u, we have

$$\frac{c_p(x,y)}{\varepsilon} + 1 > \left\lceil \frac{c_p(x,y)}{\varepsilon} \right\rceil = \ell(y) - \ell(x) + 1,$$

and hence $c_p(x,y) + (\ell(x) - \ell(y))\varepsilon > 0$. Since both x and y belong to the forest, we have $i(x) = \ell(\delta) - \ell(x)$ and $i(y) = \ell(\delta) - \ell(y)$, and thus $\ell(x) - \ell(y) = i(y) - i(x)$. So we find that $c_{p'}(x,y) > 0$ in this case, which is even stronger than the $c_{p'}(x,y) > -\varepsilon$ that we needed. So **I4** is also preserved.

Prop. 13D: R_f remains free of length-0 cycles

To show that **I5** is preserved, we must show that raising the prices at the nodes in the shortest-path forest doesn't cause the residual digraph R_f to acquire any length-0 cycles. But we saw, in Prop 7, that increasing the price at a node v by ε lowers the lengths of all links leaving v by 1 and raises the lengths of all links entering v by 1. So price increases have no effect on the overall length of any cycle. Since **I5** assures us that there were no length-0 cycles before our price increases, there won't be any length-0 cycles after them either. □

Prop 14. *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 10, 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 9, 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.

In a similar way, let y be any man. If 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 then 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. □

Prop 15. *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*:*

$$h\Delta \leq (4q + 4)s\varepsilon. \tag{16}$$

This 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. We prove the inflation bound (16) by calculating $(c_{p'} - c_p)(f' - f)$ in two different ways. But we must first define what we mean by this quantity.

Let f be the flow on the network N_G that was in effect when *Refine* was called. Let p be the prices that were in effect when control entered the main loop of *Refine*. Note that those two times are different. During the initialization code in *Refine*, we convert the input flow into a pseudoflow by zeroing out the flows along all bipartite arcs. We use the symbol f here to denote the input flow, before that zeroing out. We also raise the prices at the various nodes, as described in Figure 3, so as to make all arcs ε-proper with respect to the new pseudoflow. We use the symbol p here to refer to the prices after those increases have happened.

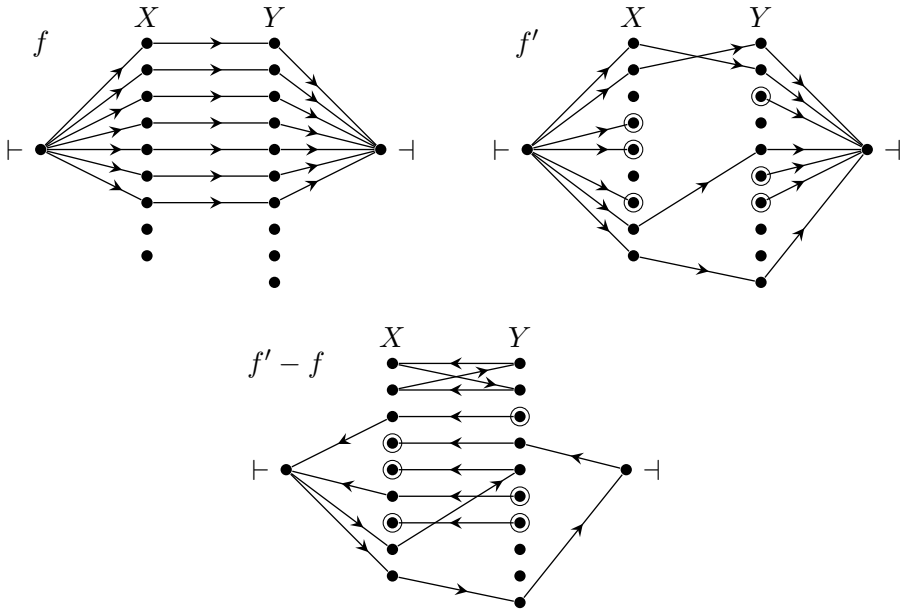

Figure 5: An example of the flux $f' - f$

Let f' be the pseudoflow at the current clean point in the execution of *Refine*, and let p' be the prices at that same clean point. Our proof will compute the quantity $(c_{p'} - c_p)(f' - f) = c_{p'}(f' - f) - c_p(f' - f)$ in two different ways.

Figure 5 shows an example of what might happen — though it is hard for one diagram to show all of the potential patterns. In Figure 5, we have $|X| = 9$, $|Y| = 10$, and $s = 7$. The input flow f of value $|f| = 7$ is laid out very simply in the diagram. We are considering a point in the execution of *Refine* at which there are $h = 3$ remaining surpluses and 3 remaining deficits, which are circled in the diagram of f'. Note that, of the four augmenting paths along which we have augmented so far, at least two of them have visited the source and at least one has visited the sink.

When we subtract f from f', we get the flux $f' - f$. Note that this difference is just a flux, not even a pseudoflow, since it assigns a flow of -1 to various arcs. Since both f and f' have s units of flow leaving the source and s units entering the sink, flow is conserved in the difference flux $f' - f$ at both the source and the sink. Flow is conserved in f at all nodes other than the source and the sink. In f', on the other hand, there are h remaining unit surpluses in X and h remaining unit deficits in Y. Thus, in the difference flux $f' - f$, flow is conserved at all nodes except for those $2h$ nodes, which have the same status in $f' - f$ that they have in f'.

Calculating the value precisely

If a flux conserves flow at some node v, then changing the price at v has no effect on the net cost of the flux. Thus, to calculate the difference $c_{p'}(f' - f) - c_p(f' - f)$, it suffices to consider the changes in price that happen at the $2h$ nodes where $f' - f$ does not conserve flow. Of those $2h$ nodes, h are women with a unit surplus who have had that surplus since we entered the main loop; and the other h are men with a unit deficit who have had that deficit since we entered the main loop.

During the main loop of *Refine*, the only price changes that happen are the rounds of price increases that follow the construction of a shortest-path forest. Note that all h of the remaining surpluses have been roots of trees in every shortest-path forest that we have constructed so far. So all of them have had their prices increased by the max increase in every round of price increases so far. Thus, for each remaining surplus x, we have $c_{p'}(x) = c_p(x) + \Delta$. On the other hand, a round of price increases doesn't change the price of any current deficit. The shortest-path forest includes only one deficit, the deficit δ whose discovery stopped the growth of the forest, and the repricing formula specifies that the price at δ shouldn't change. Thus, for each remaining deficit y, we have $c_{p'}(y) = c_p(y)$.

The flux $f' - f$ has h unit surpluses, at each of which the price increases by precisely Δ between p and p', and it has h unit deficits, at each of which the price doesn't change between p and p'. So we have $(c_{p'} - c_p)(f' - f) = h\Delta$.

Bounding the value

We next derive an upper bound on the quantity $(c_{p'} - c_p)(f' - f)$ by using our bounds on the net costs of individual arcs.

The flow f and the pseudoflow f' each assign a flow of either 0 or 1 to each arc in N_G, so we can think of each of them as a set of arcs. Taking their differences as sets, let $f' \setminus f$ denote those arcs that are saturated in f' but idle in f; and define $f \setminus f'$ symmetrically. We then have $f' - f = (f' \setminus f) - (f \setminus f')$, where the arcs saturated in both f and f' have been omitted from both terms on the right. So we have

$$(c_{p'} - c_p)(f' - f) = c_{p'}(f' \setminus f) - c_{p'}(f \setminus f') - c_p(f' \setminus f) + c_p(f \setminus f'). \tag{23}$$

We now consider each of these four sums in turn.

The first term $c_{p'}(f' \setminus f)$ sums the current net costs of those arcs that are saturated in f', but were idle in f. Since *Refine* maintains all arcs ε-proper by **I3**, any arc that is currently saturated has net cost at most ε. By **I1′**, the current pseudoflow f' saturates precisely s left-dummy arcs, $s - h$ bipartite arcs, and s right-dummy arcs. Some of those arcs may have been saturated also in f, in which case they won't contribute to $c_{p'}(f' \setminus f)$. But we certainly have $c_{p'}(f' \setminus f) \leq 3s\varepsilon$.

We boldly tackle the trickiest case next, which is the fourth and final term $c_p(f \setminus f')$ in equation (23). Recall that f is the flow that was in effect when *Refine* was called, but p is the prices that were in effect somewhat later, when *Refine* entered its main loop. There are precisely s left-dummy arcs in the flow f, so at most s in $f \setminus f'$. Each of those arcs $\vdash \to x$ was saturated when *Refine* entered its main loop, so we have $c_p(\vdash, x) \leq \varepsilon$ by **I3**. Thus, the left-dummy arcs contribute at most $s\varepsilon$; and a similar argument shows the same for the right-dummy arcs.

The bipartite arcs are the tricky ones. Let $x \to y$ be a bipartite arc that was saturated in f. When *Refine* was called, this arc was saturated and $(q\varepsilon)$-proper. During the initialization of *Refine*, we first changed the flow to make the arc $x \to y$ idle and we then changed prices as described in Figure 3. Since x was then a woman with a unit surplus, we left the price at x unchanged. But, since y was a man with a unit deficit, we added $3(q-1)\varepsilon$ to the price at y. It follows that $c_p(x, y) \leq q\varepsilon + 3(q-1)\varepsilon = (4q-3)\varepsilon$. There are at most s arcs of this type, so they

contribute $(4q-3)s\varepsilon$ to our sum. Adding in the dummy arcs, we conclude that $c_p(f \setminus f') \leq (4q-1)s\varepsilon$.

What about the second term $-c_{p'}(f \setminus f')$ in (23)? This term subtracts off the net costs, in current prices, of those arcs $v \to w$ that were saturated in f, but are idle in f'. Since those arcs currently idle, **I3** tells us that $c_{p'}(v,w) > -\varepsilon$, and hence $-c_{p'}(v,w) < \varepsilon$. Furthermore, if the arc $v \to w$ is a dummy, then Prop 5 tells us, in fact, that $c_{p'}(v,w) \geq 0$, and hence $-c_{p'}(v,w) \leq 0$. So the dummy arcs $v \to w$ don't contribute anything. There were s bipartite arcs that were saturated in f, of which at most s could be currently idle, so we have $-c_{p'}(f \setminus f') < s\varepsilon$.

The third term $-c_p(f' \setminus f)$ is similar. It subtracts off the net costs, at the time of entry into the main loop, of those arcs $v \to w$ that are currently saturated, but were idle when *Refine* was called. Such an arc $v \to w$ must have been idle also when *Refine* entered its main loop, since converting f into a pseudoflow, while it does idle some saturated arcs, doesn't saturate any idle arcs. Applying **I3** and Prop 5 once again, now at the time of entry into the main loop, we have $-c_p(f' \setminus f) < s\varepsilon$.

Adding everything together, we have $h\Delta \leq (4q+4)s\varepsilon$ □

Corollary 17. *The building of any shortest-path forest in the procedure* Refine *is always halted by finding a deficit δ with $\ell(\delta) \leq (4q+4)s/h$.*

Proof. Suppose that we are about to build a shortest-path forest. We have $h \geq 1$, so the matching that is encoded by the flow component $\hat{f}$ of the current pseudoflow f has size $s-h < s$. Since G has matchings of size at least s, the standard theory of matchings tells us that there must exist a path P in R_f from some maiden to some bachelor that alternates between forward bipartite links and backward bipartite links. The path P may not be an augmenting path in our sense, however, since the maiden μ at which it starts may not be a surplus and the bachelor β at which it ends may not be a deficit. If μ is not a surplus, however, then the left-dummy arc $\vdash \to \mu$ must be idle; so we can tack two more links onto the beginning of P as follows: We choose any surplus we like, say σ; we follow the backward link $\sigma \Rightarrow \vdash$, then the forward link $\vdash \Rightarrow \mu$, and then continue along P. In a similar way, if β is not a deficit, we can tack two more links onto the end of P; we choose any deficit δ and append the links $\beta \Rightarrow \dashv$ and $\dashv \Rightarrow \delta$. The result will be an augmenting path in our sense, from a surplus to a deficit; so some such path does exist.

If we allocated our Dial array Q large enough, then the building of the forest would halt by discovering such a path. Let δ be the deficit whose discovery would halt the building of the forest. The subsequent round of price increases would raise the price at all remaining surpluses by $\ell(\delta)\varepsilon$. The inflation bound (16) would then apply, telling us that $h\Delta \leq (4q+4)s\varepsilon$. But we surely have $\ell(\delta)\varepsilon \leq \Delta$. So we conclude that $\ell(\delta) \leq (4q+4)s/h$. □

Corollary 18. *The prices in* FlowAssign *remain $O(sC)$.*

Proof. How large could our prices get? To start with, how much can they increase during one invocation of *Refine*? During the initialization of *Refine*, we raise the prices by at most $3(q-1)\varepsilon$. Once we enter the main loop, we raise prices only after building each shortest-path forest. From the inflation bound (16), we deduce that the total impact of those price increases, throughout this entire call to *Refine*, is at most $(4q+4)s\varepsilon$. So the call to *Refine* with parameter ε raises prices only by $O(s)\varepsilon$.

On entry to the first call to *Refine*, the prices are zero. That first call has $\varepsilon = \overline{\varepsilon}/q \leq C$, and the values of ε in subsequent calls to *Refine* decrease in a geometric series with ratio $1/q$. So no price ever rises further than $O(sC)$. □

D Endnotes

[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 for our n and r, think of n as standing for $\underline{n}$umerous, while r stands for $\underline{r}$are. 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, TV shows have often resorted to "bachelorette".

[5]The auction algorithm that Bertsekas and Castañon suggest for unbalanced graphs [3] 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. Their auction step maintains a *profitability threshold* λ, where λ can be thought of as a candidate for the price $p_d(\dashv)$.

[6]Because we name our arcs only in their forward directions, we have no 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]It is serendipitous that the primal options "cost" and "benefit" and the dual options "acquire" and "dispose" begin with the first four letters of the alphabet.

[8]Papers about **PerA** often compute net 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 having the same sign. This means that, of the two parts X and Y of the bipartite graph G, acquire prices are used in one part, 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.

[9]The term "reduced cost" is widely used for our "net cost". But "net cost" seems more apt as well as shorter, given that one price occurs with each sign in our cost adjusting formula $c_p(v,w) := c(v,w) - p_d(v) + p_d(w)$.

[10]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 one shortest-path tree instead, with a single, chosen maiden as its root. That variant solves **PerA** perfectly well, but it does not preserve the maiden bound, and hence the imperfect matchings that it computes might not be 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. (Recall that all bipartite arcs are idle at this point.) Both the maiden and bachelor bounds fail in that variant.

[11]Gabow and Tarjan use the term "eligible" for the arcs that we call *ε-tight*.

[12]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 $v \to w$; all of these arcs are oriented from left to right, and it is these arcs that have net 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.

[13]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.

References

[1] Ravindra K. Ahuga, James B. Orlin, Clifford Stein, and Robert E. Tarjan. Improved algorithms for bipartite network flow. *SIAM Journal on Computing* **23** #5 (1994), pp. 906–933.

[2] Dimitri P. Bertsekas. Auction algorithms for network flow problems: A tutorial introduction. *Computational Optimization and Applications* **1** (1992) pp. 7–66.

[3] Dimitri P. Bertsekas and David A. Castañon. A forward/reverse auction algorithm for asymmetric assignment problems. *Computational Optimization and Applications* **1** (1992) pp. 277–297.

[4] Rainer Burkard, Mauro Dell'Amico, and Silvano Martello. *Assignment Problems*, Society for Industrial and Applied Mathematics (SIAM), 2009.

[5] Boris V. Cherkassky, Andrew V. Goldberg, and Craig Silverstein. Buckets, heaps, lists, and monotone priority queues. *ACM-SIAM Symposium on Discrete Algorithms* (1997) pp. 83–92.

[6] Robert B. Dial. Algorithm 360: Shortest path forest with topological ordering. *Communications of the ACM* **12** (1969) pp. 632–633.

[7] Ran Duan and Hsin-Hao Su. A scaling algorithm for maximum weight matching in bipartite graphs. *ACM-SIAM Symposium on Discrete Algorithms* (Jaunary 2012) pp. 1413–1424.

[8] Tomás Feder and Rajeev Motwani. Clique partitions, graph compression and speeding-up algorithms. *Journal of Computer and System Sciences* **51** #2 (1995) pp. 261–272.

[9] Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM* **34** #3 (1987) pp. 596–615.

[10] Harold N. Gabow and Robert E. Tarjan. Faster scaling algorithms for network problems. *SIAM Journal on Computing* **18** #5 (1989) pp. 1013–1036.

[11] Andrew V. Goldberg and Robert Kennedy. An efficient cost scaling algorithm for the assignment problem. *Mathematical Programming* **71** (1995) pp. 153–177.

[12] Andrew V. Goldberg and Robert Kennedy. Global price updates help. *SIAM Journal on Discrete Mathematics* **10** #4 (1997) pp. 551–572.

[13] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing* **2** #4 (1973) pp. 225–231.

[14] Ming-Yang Kao, Tak-Wah Lam, Wing-Kin Sung, and Hing-Fung Ting. A decomposition theorem for maximum weight bipartite matchings. *SIAM Journal on Computing* **31** #1 (2001) pp. 18–26.

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

[16] James B. Orlin and Ravindra K. Ahuja. New scaling algorithms for the assignment and minimum mean cycle problems. *Mathematical Programming* **54** (1992) pp. 41–56.

[17] Lyle Ramshaw and Robert E. Tarjan. On minimum-cost assignments in unbalanced bipartite graphs. HP Labs Technical Report HPL-2012-40 (June 2012).

[18] Mikkel Thorup. Integer priority queues with decrease key in constant time and the single source shortest paths problem. *Journal of Computer and System Sciences* **69** (2004) pp. 330–353.