The Pseudoflow Algorithm for the Maximum Blocking-Cut and Minimum-Waste Flow Problems

David R. Morrison

1 Introduction

Network flow problems are widely-studied, and many different problems in operations research and theoretical computer science reduce to them. The most well-known version of network flow problem, the maximum-flow problem on directed graphs, was first studied in-depth by Ford, Jr. and Fulkerson (1956), who provided an algorithm to find a maximum flow in a directed network, and also proved the Max-flow/Min-cut theorem.

In this paper, we analyze two closely-related problems to the max-flow/min-cut problem, the minimum-waste flow problem and the maximum blocking-cut problem. The maximum-blocking cut problem was introduced in Radzik (1993); to the best of our knowledge, the minimum-waste flow problem is first formally described in this paper. Both of these problems can in fact be solved simultaneously by the pseudoflow algorithm of Hochbaum (2008); it will be our primary goal in this paper to describe the pseudoflow algorithm and the intuition behind it from the perspective of minimum-waste flows and maximum blocking-cuts.

The minimum-waste flow problem is motivated by an application in the production and delivery setting. Given a set of suppliers and a set of consumers in a transportation network, we seek to minimize the amount of *waste* produced by suppliers while satisfying the as much of the consumers' demands as possible. In particular, each supplier has a fixed quantity of goods that it must either distribute or throw away, and each consumer has some demand for these goods. If the network is over-capacitated (that is, it is impossible to satisfy all of the consumer demands due to capacity constraints of the network), goods must be discarded at some suppliers; our objective is to minimize the amount thrown away.

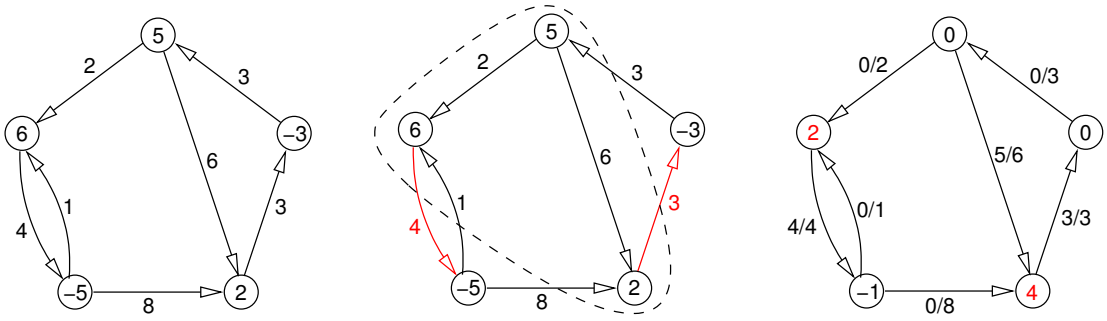
On the other hand, the maximum blocking-cut problem is a generalization of a the maximum closure problem, which arises in open-pit mining and other settings (Lerchs and Grossmann, 1965). In this problem, we are given a set of tasks together with an associated profit or cost and a set of precedence constraints for each task. In the maximum closure problem, we seek a *closed* set of tasks (that is, a set of tasks for which all precedence constraints are satisfied) of maximum profit. The maximum blocking-cut problem generalizes this by relaxing the precedence constraints but charging a penalty to the objective function for each violated constraint (Hochbaum, 2001).

As it turns out, the maximum blocking-cut problem and the minimum-waste flow problem are in fact equivalent; after providing some definitions and examples in Section 2, we prove this equivalence in two ways, first via linear programming in Section 3, and then by combinatorial methods in Section 4. In Section 5, we give two variants of the pseudoflow algorithm of Hochbaum (2008), a generic algorithm and a strongly-polynomial labeling algorithm that mimics the push-relabel algorithm for maximum flows. In Section 6, we show that the maximum blocking-cut problem reduces trivially to the minimum-cut problem, and show how the pseudoflow algorithm can be used to retrieve a maximum flow in a network. Section 7 presents a computational study of the pseudoflow algorithm done by Chandran and Hochbaum (2009). In Section 8, we briefly describe how the pseudoflow algorithm can be used to solve the parametric version of the maximum blocking-cut problem, and we offer some concluding remarks in Section 9.

2 Definitions and Examples

All problems and algorithms take as input a directed graph $G = (V, A)$, where $n = |V|, m = |A|$, and with real-valued arc capacities c_{ij} for all $(i, j) \in A$. A supply vector is given with entries w_i for each node $i \in V$. If $w_i < 0$, we say that node i has demand $-w_i$. For a set S, the induced subgraph of G on S is $G[S] = (S, A_S)$, where $A_S = \{(i, j) \mid (i, j) \in G, i, j \in S\}$. Further, we define the cut $[S, \bar{S}] = \{(i, j) \mid i \in S, j \in \bar{S}\}$, and the capacity of the cut as $c(S, \bar{S}) = \sum_{(i,j)\in[S,\bar{S}]} c_{ij}$. Finally, for a node i, the set of incident arcs to i is $\delta(i)$, the set of outgoing arcs is $\delta^+(i)$, and the set of incoming arcs is $\delta^-(i)$.

The maximum blocking-cut problem is now defined:



(a) A directed graph with node supplies and arc capacities.

(b) A maximum blocking-cut. The value of this set is 6; arcs in red cross the cut and are subtracted from the supply sum.

(c) A minimum-waste pseudoflow. The excess/deficit at every node is given. The waste (in red) is 6.

Figure 1: An example graph with the maximum blocking-cut and minimum-waste flow.

Definition 2.1 (Hochbaum 2008, Definition 2.1)**.** *The* **surplus** *of a set $S \subseteq V$ is*

$$surplus(S) = \sum_{i \in S} w_i - \sum_{(i,j) \in [S,\bar{S}]} c_{ij}$$

Definition 2.2 (Hochbaum 2008)**.** *The* **Maximum Blocking-Cut** *(MBC) problem is to find $S \subseteq V$ that maximizes $surplus(S)$.*

An example problem instance is given in Figure 1(a), and the maximum blocking-cut is shown in Figure 1(b).

A **pseudoflow** is a function $f : A \to \mathbb{R}$ such that $0 \leq f_{ij} \leq c_{ij}$ for each $(i,j) \in A$ (with the exception of Sections 6 and 7, we use the term "pseudoflow" and "flow" interchangeably). We define the **excess** of a node $i \in V$ as

$$ex(i) = \sum_{j \in \delta^-(i)} f_{ji} - \sum_{j \in \delta^+(i)} f_{ij}$$

and the excess of $S \subseteq V$ as $\sum_{i \in S} ex(i)$. If $ex(i) < 0$, we say that node i has a **deficit** of $-ex(i)$. Furthermore, we define the **waste** at a node as $\max\{0, ex(i) + w_i\}$, and the waste of a set $S \subseteq V$ as $\sum_{i \in S} waste(i)$. For a pseudoflow f, we say that $waste(f) = \sum_{i \in V} waste(i)$, and $ex(f) = \sum_{i \in V} ex(i)$. With these definitions, we can formulate the minimum-waste flow problem:

Definition 2.3. *The* **Minimum-Waste Flow** *(MWF) problem is to find a pseudoflow f that minimizes $waste(f)$.*

A minimum-waste pseudoflow for the problem in Figure 1(a) is shown in Figure 1(c).

Some further definitions will be helpful in discussing the problems and proving later results. Let f be a pseudoflow. We define the **residual network of** G on the same vertex set as G; for each $(i,j) \in A$, we create two **residual arcs**, a **forward residual arc** $\langle i,j \rangle$ and a **backwards residual arc** $\langle j,i \rangle$. The **residual capacity** c^f_{ij} for all forward residual arcs is $c_{ij} - f_{ij}$, and is f_{ij} for all backwards residual arcs. We say an arc is **saturated** if $f_{ij} = c_{ij}$, or equivalently if the residual capacity of the forward arc is 0. Finally, we define the residual capacity of a cut $[S, \bar{S}]$ as

$$\sum_{\langle i,j \rangle \in [S,\bar{S}]} c^f_{ij} + \sum_{\langle j,i \rangle \in [\bar{S},S]} c^f_{ij} = \sum_{(i,j) \in [S,\bar{S}]} (c_{ij} - f_{ij}) + \sum_{(i,j) \in [\bar{S},S]} f_{ij} \tag{2.1}$$

and the flow crossing the cut $f(S, \bar{S})$ as $\sum_{(i,j) \in [S,\bar{S}]} f_{ij}$.

3 Linear Programming Methods

As is standard in combinatorial optimization, we turn to linear programming to provide some insight into the two problems defined in Section 2. First, we present an integer program for the maximum blocking-cut problem:

$$MBC_{IP}: \quad \text{Maximize} \sum_{i \in V} x_i w_i - \sum_{(i,j) \in A} y_{ij} c_{ij} \tag{3.1a}$$

$$\text{Subject to } x_i - x_j \leq y_{ij} \quad \forall\ (i,j) \in A \tag{3.1b}$$

$$x_i, y_{ij} \in \{0,1\} \quad \forall\ i \in V, (i,j) \in A \tag{3.1c}$$

We create a binary variable x_i for each vertex i representing whether $i \in S$ or not; then, for each arc (i,j) we create a variable y_{ij} that represents whether or not $(i,j) \in [S, \bar{S}]$. Equation (3.1a) is the surplus of the chosen set; constraint (3.1b) indicates that if two endpoints of an arc are in different sets such that the arc leaves S, the arc must be in $[S, \bar{S}]$. In all other cases, the objective function will be maximized if $y_{ij} = 0$.

The linear programming relaxation of (3.1) is the following:

$$MBC_{LP}: \quad \text{Maximize} \sum_{i \in V} x_i w_i - \sum_{(i,j) \in A} y_{ij} c_{ij} \tag{3.2a}$$

$$\text{Subject to } x_i - x_j \leq y_{ij} \quad \forall\ (i,j) \in A \tag{3.2b}$$

$$0 \leq x_i \leq 1 \quad \forall\ i \in V \tag{3.2c}$$

$$y_{ij} \geq 0 \quad \forall\ (i,j) \in A \tag{3.2d}$$

Notice first that we can discard the upper bounds on the y_{ij} variables, since $x_i - x_j \leq 1$, and (3.2a) is maximized when the y_{ij}'s are minimized. Furthermore, notice that the polytope formed by the set of constraints (3.2b) is precisely the minimum-cut polytope. It is well-known that the matrix defining the minimum-cut polytope is totally unimodular. Furthermore, Theorem 13 from Qi (accessed June 2011) states that any TUM matrix taken with variable bound constraints such as (3.2c) is also TUM; thus, the polytope associated with (3.2) is TUM, which in particular implies that for integral w_i and c_{ij}, an integral optimal solution exists and can be found in polynomial time via the ellipsoid method.

The dual linear program of (3.2) is the following:

$$MWF_{LP}: \quad \text{Minimize} \sum_{i \in V} u_i \tag{3.3a}$$

$$\text{Subject to } u_i + \sum_{j \in \delta^+(i)} f_{ij} - \sum_{j \in \delta^-(i)} f_{ji} \geq w_i \tag{3.3b}$$

$$0 \leq f_{ij} \leq c_{ij} \tag{3.3c}$$

$$u_i \geq 0 \tag{3.3d}$$

Since (3.2) essentially defines a minimum-cut polytope, it is unsurprising that (3.3) looks remarkably similar to a linear program for maximum-flow. In fact, we claim that the above linear program actually defines the minimum-waste flow problem. This is formalized in the following theorem:

Theorem 3.1. *The dual of the maximum blocking-cut problem is the minimum-waste flow problem.*

Proof. An optimal solution f^*, u^* to (3.3) induces a valid pseudoflow on G, since no arc capacities are violated, by constraint (3.3c). Furthermore, rewriting contraint (3.3b) shows that $u^* \geq w_i + ex(i)$ for all i. If $w_i + ex(i) > 0$, the objective is minimized when $u^* = w_i + ex(i)$, and if $w_i + ex(i) \leq 0$, $u^* = 0$. Therefore, $u_i = waste(i)$. ∎

By the strong duality theorem, we know that the value of the maximum blocking-cut problem is exactly equal to the value of the minimum-waste flow problem. In the next section, we seek a proof of this fact that does not require an appeal to linear programming methods.

4 The Maximum Blocking-Cut/Min-Waste Flow Theorem

From our linear programming discussion in Section 3, it should be clear that the maximum blocking-cut problem and the minimum-waste flow problem are intimately related. In this section, we derive similar results without using results from linear programming theory. This will provide several insights that will be helpful when developing an algorithm to solve the minimum-waste flow problem.

Let $S \subseteq V$ be a candidate blocking set. Intuitively, supply nodes in S can eliminate waste by sending flow to nodes in S with positive demand, as well as to nodes outside of S. The remaining supply must be discarded. This implies that for any set S, the surplus of S is a lower bound on the value of the minimum-waste flow. We prove this formally in the following lemma:

Lemma 4.1 (Hochbaum 2008, Lemma 4.2)**.** *Let f be a minimum-waste flow on G. For any set $S \subseteq V$, $surplus(S) \leq waste(f)$. In particular, the value of the maximum blocking-cut is a lower bound on the minimum-waste flow.*

Proof. Fix f as a minimum-waste pseudoflow, and let $S \subseteq V$. Since waste is non-negative, and $waste(i) \geq w_i + ex(i)$, we have

$$waste(f) \geq waste(S) \geq \sum_{i \in S} w_i + ex(S) = \sum_{i \in S} w_i + \sum_{i \in S} \left(\sum_{j \in \delta^-(i)} f_{ji} - \sum_{j \in \delta^+(i)} f_{ij} \right)$$

In the last sum, notice that for each $(i, j) \in G[S]$, the term f_{ij} appears exactly once in $ex(j)$, and the term $-f_{ij}$ appears exactly once in $ex(i)$. Therefore, the sum telescopes, and the only terms that remain are those that appear in $[S, \bar{S}]$ and $[\bar{S}, S]$.

Applying the definition of residual capacity to $f(\bar{S}, S)$ and equation (2.1) to $f(S, \bar{S})$, we get

$$\begin{aligned} waste(f) \geq \sum_{i \in S} w_i + f(\bar{S}, S) - f(S, \bar{S}) &= \sum_{i \in S} w_i + f(\bar{S}, S) - \left(c(S, \bar{S}) - \sum_{(i,j) \in A \cap [S, \bar{S}]} c_{ij}^f \right) \\ &= \sum_{i \in S} w_i - c(S, \bar{S}) + c^f(S, \bar{S}) \geq surplus(S) \end{aligned}$$

Since S was arbitrary, the value of the maximum blocking-cut is a lower bound on the minimum-waste flow. ∎

In fact, we can extend the above lemma to achieve equality when the blocking-cut and flow are optimal. To do this, we need to argue about the structure of a maximum blocking-cut. If $w_i + ex(i) > 0$, we say that node i is **strictly strong**, and if $w_i + ex(i) < 0$, it is **strictly weak**.

Lemma 4.2. *Let S be a maximum blocking-cut in G. Then there exists a minimum-waste pseudoflow f with no strictly strong nodes in $\bar{S}$.*

Proof. Let f be a minimum-waste pseudoflow that maximizes the waste in $\bar{S}$, and let W^- and W^+ be the set of strictly weak (strictly strong) nodes in G, respectively. Then, define $K = \bar{S} \cap W^+$ to be the set of strictly strong nodes in $\bar{S}$. Let $\mathcal{R}$ be the set of all nodes in $\bar{S}$ reachable from K in the residual graph of f. Notice that $\mathcal{R} \cap W^- = \varnothing$, since otherwise, we could augment flow to reduce the waste of f. Further, notice that the capacity of any residual arc $\langle i, j \rangle$ from S into $K \cup \mathcal{R}$ is 0, since otherwise, we could increase the waste of f in $\bar{S}$ by augmenting flow from S.

Let $S^+ = K \cup \mathcal{R}$. Since the excess at all nodes in $\mathcal{R} \setminus K$ is 0, we have the following:

$$0 < waste(K) = waste(S^+) = \sum_{i \in S^+} waste(i) = \sum_{i \in S^+} w_i + ex(i)$$

Notice that for each arc $(i,j) \in G[S^+]$, f_{ij} appears in $ex(j)$ and $-f_{ij}$ appears in $ex(i)$, and so cancel out. By choice of $\mathcal{R}$ and f, therefore, we have the following:

$$0 < waste(S^+) = \sum_{i \in S^+} w_i - \sum_{(i,j) \in [S^+, \bar{S} \setminus S]} c_{ij} + \sum_{(i,j) \in [S, S^+]} c_{ij}$$

Rearranging shows that $-c(S, S^+) < \sum_i w_i - c(S^+, \bar{S} \setminus S)$. In particular, notice that $surplus(S \cup S^+) > surplus(S)$, contradicting the fact that $surplus(S)$ is maximum. ∎

By interchanging the roles of W^- and W^+ in the above proof, as well as swapping "strictly strong" with "strictly weak", we can prove the following similar result:

Lemma 4.3. *Let S be a maximum blocking-cut, and let f be a minimum-waste pseudoflow. Then there are no strictly weak nodes in S.*

We need one final lemma before proving the main result in this section; this lemma follows a similar line of argument to the above:

Lemma 4.4. *Let S be a maximum blocking-cut and f be a minimum-waste pseudoflow. There is no path from S to $\bar{S}$ in the residual graph.*

Proof. Define W^+ and W^- as in Lemma 4.2. Suppose residual arc $\langle i,j \rangle \in [S, \bar{S}]$ is not saturated by a minimum-waste pseudoflow f. Then, if there exists a residual path from $u \in S \cap W^+$ to i and a residual path from j to $v \in \bar{S} \cap W^-$, then we can augment flow along this path to reduce the waste. So it must be the case that at least one of these paths does not exist.

Without loss of generality, suppose there is no residual path from j to $v \in \bar{S} \cap W^-$; then let $\mathcal{R}$ be the set of all nodes in $\bar{S}$ reachable from j in the residual graph. Each node in $\mathcal{R}$ has non-negative excess, so $waste(\mathcal{R}) \geq 0$.

Since for each arc $(u,v) \in G[\mathcal{R}]$, f_{uv} appears once in $ex(v)$ and $-f_{vu}$ appears once in $ex(u)$, the only remaining arcs are those leaving or entering $\mathcal{R}$; all arcs leaving $\mathcal{R}$ must be saturated, and any entering arc that is not (i,j) must have 0 flow, by choice of $\mathcal{R}$. Thus, we have shown that

$$0 \leq waste(\mathcal{R}) = \sum_{i \in \mathcal{R}} w_i + f_{ij} - \sum_{(u,v) \in [\mathcal{R}, \bar{S} \setminus \mathcal{R}]} c_{uv}$$

In particular, we have $-c_{ij} \leq -f_{ij} \leq surplus(\mathcal{R})$, so taking $S \cup \mathcal{R}$ as our blocking-cut does not decrease the value of the surplus. A similar proof holds if instead there is no residual path from $S \cap W^+$ to i. ∎

Taken together, the above three lemmas prove the maximum blocking-cut/minimum-waste flow theorem, the analogous statement to the max-flow/min-cut theorem in this setting:

Theorem 4.1 (Maximum blocking-cut/Minimum-waste flow)**.** *Let S be a maximum blocking-cut and f be a minimum-waste pseudoflow. Then $surplus(S) = waste(f)$.*

Proof. Let W^+ and W^- be defined as in Lemma 4.2. Then, by Lemma 4.2, all strictly strong nodes are in S, and by Lemma 4.3, all strictly weak nodes are in $\bar{S}$. Furthermore, if i is strictly strong, $waste(i) = w_i + ex(i)$; otherwise, $waste(i) = 0$. Therefore, we see that $waste(f) = waste(S)$.

Notice that for each $(i,j) \in G[S]$, f_{ij} appears once in $ex(j)$ and $-f_{ij}$ appears once in $ex(i)$. Therefore, we have

$$waste(S) = \sum_{i \in S} w_i - \sum_{(i,j) \in [S, \bar{S}]} f_{ij}$$

but by Lemma 4.4, $f_{ij} = c_{ij}$ for all arcs in $[S, \bar{S}]$ and $f_{ij} = 0$ for all arcs in $[\bar{S}, S]$, proving the result. ∎

Notice that the proofs in this section are non-constructive; Hochbaum (2008) provides a constructive proof of the same result by giving an algorithm that simultaneously finds a minimum-waste pseudoflow and a maximum blocking-cut. The proof of correctness of this algorithm mimics many of the ideas in this section, but utilizes more algorithmic ideas and specialized data structures. We believe that the results in this section provide some novel insights into the problem by distilling out the algorithmic aspects of the problem and focusing entirely on its mathematical structure. In the following section, we present the algorithm of Hochbaum (2008).

5 The Pseudoflow Algorithm

In order to describe the pseudoflow algorithm of Hochbaum (2008), we perform the following graph transformation; given a graph G, let the **extended graph** $G^{ext} = (V + r, A \cup A_r)$, where r is a designated "root" node, and $A_r = \{(r, i) \mid i \in V \text{ and } w_i > 0\} \cup \{(i, r) \mid i \in V \text{ and } w_i \leq 0\}$. For each node $i \in V$ with positive supply, let $c_{ri} = w_i$, and for each node $j \in V$ with non-negative demand, let $c_{jr} = w_j$. Finally, we delete the supply values at each node in V, leaving a directed graph with only arc capacities (see Figure 2).

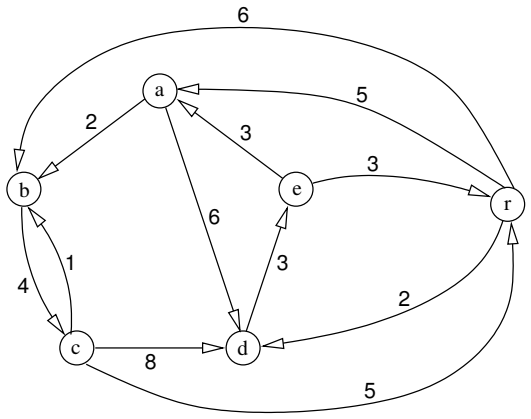


Figure 2: The transformation of the graph in Figure 1(a) into G^{ext}.

In G^{ext}, we consider a blocking-cut $S \subseteq V$ as before; in this setting, the value of the blocking cut S is given by

$$surplus^{ext}(S) = \sum_{i \in \delta^+(r)} c_{ri} - \sum_{i \in \delta^-(r)} c_{ir} - \sum_{(i,j) \in [S, \bar{S}]} c_{ij}$$

The first two terms in the above expression are simply the weights of the nodes in G, and the last term is the capacity of the cut, as before.

We seek a pseudoflow f that saturates all arcs in $\delta(r)$. The excess of a node is defined as before, and $waste^{ext}(i) = \max\{0, ex(i)\}$. As before, we seek a pseudoflow minimizing the waste in the G^{ext}. Notice that in particular, given a pseudoflow f on G^{ext}, $waste^{ext}(f) = waste(f)$ when f is restricted to arcs in G. Furthermore, since no capacity constraints are violated, f restricted to G is still a valid pseudoflow, which implies that finding a minimum-waste pseudoflow on G^{ext} saturating all arcs in $\delta(r)$ is equivalent to finding the minimum-waste pseudoflow in G.

5.1 Normalized Trees

The ideas in Section 4 give a flavor of the pseudoflow algorithm: start with an arbitrary flow, and augment flow from strictly strong nodes to strictly weak nodes until the cut between strong and weak nodes is saturated. Once this occurs, Lemma 4.4 implies that we have found an optimal solution.

To this end, we develop a data structure that will enable us to keep the strictly strong and strictly weak nodes of the graph separate; this data structure is called a **normalized tree**, and was first described by Lerchs

and Grossmann (1965). A normalized tree T^f is a spanning tree in the extended graph G^{ext} rooted at r, defined with respect to a pseudoflow f, and satisfying some additional properties.

Before stating the properties that define a normalized tree, we introduce a few further definitions; we say that an arc is **upward** (**downward**) if it points towards (away from) the root. In general, we are not concerned with the direction of an arc; if it does not matter which way arc (i, j) goes, we will refer to it as an **edge**, and denote it $[i, j]$. The parent of a node i is the unique node adjacent to i on the (undirected) path from i to r (note that r has no parent). A child of a node i is any adjacent node that is not i's parent.

Finally, we define a **branch** of T^f as the subtree of T^f rooted at one of r's children. If r has k children, then there are k branches, denoted $T_1^f, ..., T_k^f$; the root of the i^{th} branch is denoted as r_i. With respect to these definitions and a pseudoflow f, a normalized tree T^f must satisfy the following four properties (Hochbaum, 2008):

1. All edges in $\delta(r)$ must be saturated by f.
2. Any edge not in T^f must be at its lower or upper bound.
3. The downwards residual arcs in every branch of T^f must have strictly positive capacity.
4. The only nodes with non-zero excess are the branch roots.

Notice that Property 4 is a very strong property: it means that the only strictly strong and strictly weak nodes in the graph are branch roots. We say that a node is **strong** if it is contained in a branch with a strictly strong root, and **weak** otherwise (in particular, branch roots with zero excess are weak).

It is trivial to see that the set of all strong nodes in T^f provides an upper bound on the maximum blocking-cut set, by Theorem 3.1, and since f is a valid pseudoflow on G. This provides an alternate proof of the following theorem:

Theorem 5.1 (Hochbaum 2008, Property 5)**.** *For a normalized tree T^f, the surplus of the set of strong nodes provides an upper bound on the value of the maximum blocking-cut.*

Furthermore, Lemma 4.4 gives an optimality criterion for a normalized tree T^f:

Corollary 5.1 (Hochbaum 2008, Corollary 4.1)**.** *Let T^f be a normalized tree. If S is the set of strong nodes of T^f and $c^f(S, \bar{S}) = 0$, then S is an optimal blocking-cut.*

This gives us the core for the pseudoflow algorithm: while the residual capacity between strong and weak nodes is non-zero, augment flow from a strong node to a weak node, making sure to maintain the normalized tree properties.

5.2 The Pseudoflow Algorithm

We first present a generic version of the pseudoflow algorithm, which is easier to understand, but is not guaranteed to terminate in polynomial time. As described above, the algorithm picks an arbitrary residual arc from a strong node s to a weak node w in T^f (we say the chosen arc $\langle s, w\rangle$ is a **merger** arc). Let $s \in T_k^f$ and $w \in T_\ell^f$. Then, all of the excess at the branch root r_k is pushed to s, across the arc $\langle s, w\rangle$, and to the branch root r_ℓ. However, in this process, some edges may become saturated. Let $[i, j]$ be a saturated edge along this path; then, we send $\delta = c_{ij}^f$ units of flow along the edge; the remaining flow is shipped directly from i to r, and i becomes the root of a new strong branch. This process is called **splitting**. Pseudocode for the generic pseudoflow algorithm is given in Figures 3 and 4.

The `Initialize` procedure in line 1 of `GenericPseudoflow` takes an input graph G and converts it to G^{ext}, and computes an initial normalized tree and pseudoflow on G^{ext}. One feature of the `GenericPseudoflow` algorithm is that any initial pseudoflow can be used in procedure `Initialize`. Two methods compared in Chandran and Hochbaum (2009) are the simple normalized tree method and the saturate-all method. The simple normalized tree method saturates all arcs in $\delta(r)$, and sets the flow on all other arcs to 0. In this

Function `GenericPseudoflow`(G)

1 $(G^{ext}, f, T^f) =$ `Initialize`(G)
2 $S = \text{strong}(T^f)$
3 **while** $c^f(S, \bar{S}) > 0$**:**
4 Choose $\langle s, w \rangle$ such that $s \in S, w \in \bar{S}$, and $c^f_{sw} > 0$
5 $T^f =$ `Merge`$(T^f, [s, w])$
6 $S = \text{strong}(T^f)$
7 return S

Figure 3: The generic pseudoflow algorithm for the maximum blocking-cut problem

Function `Merge`$(T^f, [s, w])$

1 $r_k = \text{branch\_root}(s); r_\ell = \text{branch\_root}(w)$
2 $\delta = ex(r_k)$
3 $T^f = T^f - [r, r_k] + [s, w]$
4 $P = [r_k, ..., s, w, ..., r_\ell] \in T^f$
5 **for each** edge $[i, j] \in P$, in order from r_k**:**
6 **if** $\delta \leq c^f_{ij}$**:** augment δ flow along $[i, j]$
7 **else:** $(T^f, \delta) =$ `Split`$(T^f, [i, j])$
8 return T^f

Function `Split`$(T^f, [i, j], \delta)$

1 augment c^f_{ij} units of flow along $[i, j]$
2 $T^f = T^f - [i, j] + [i, r]$
3 return (T^f, c^f_{ij})

Figure 4: Helper functions for the pseudoflow algorithm

method, each node is a branch root; strong nodes are ones with positive supply, and weak nodes are ones with non-negative demand. On the other hand, the saturate-all method sets all arcs in G^{ext} to their upper bounds; strong nodes are determined by the net flow into a node.

However, an arbitrary pseudoflow may be used as an initial pseudoflow, as long as a process of renormalization is undertaken to construct a valid normalized tree (see the electronic companion to Hochbaum (2008) for more details). One use for this property is to drastically reduce the number of iterations needed to solve a problem instance, if a "near-optimal" starting solution is used. In particular, it allows for *warm starts*, where the initial pseudoflow is the solution to a closely-related problem instance. We do not provide pseudocode for `Initialize`, as it is a relatively straightforward procedure.

To prove the correctness of the pseudoflow algorithm, we need to argue two things: first, that after each iteration of the while loop at line 3 of `GenericPseudoflow` the new tree T^f is still normalized, and secondly that the algorithm terminates. The first is shown in the next lemma:

Lemma 5.1. *If T^f is a valid normalized tree at the beginning of an iteration in* `GenericPseudoflow`*, the resulting tree at the end of the iteration is still a valid normalized tree.*

Proof. We must show that the four properties of a normalized tree are satisfied. Since no flow is ever augmented along arcs in $\delta(r)$, they all remain at capacity. Further, the only edges removed from T^f in an iteration are $[r, r_k]$ in line 3 of `Merge` and any saturated arc that we provide as an argument to `Split` (line 2 of `Split`); these arcs are all at their upper or lower bounds, so Property 2 is satisfied.

Notice that the path $[r_k, ..., s, w]$ reverses direction once it is merged into w's branch, and since we push a positive amount of flow along it, all (new) downward residual arcs have positive capacity. Further, if any downward residual arc in the path $[w, ..., r_\ell]$ reaches 0, that arc is split and leaves the tree, so Property 3 is satisfied. Finally, since we are augmenting flow along P, the balance at every non-branch-root remains satisfied. Thus, after the completion of an iteration of `GenericPseudoflow`, T^f is still normalized. ∎

To establish the finite termination of the `GenericPseudoflow` algorithm, we prove the following two results:

Lemma 5.2 (Hochbaum 2008, Lemma 6.1)**.** *At each iteration of* `GenericPseudoflow`*, either the total waste is decreased or at least one weak node becomes strong.*

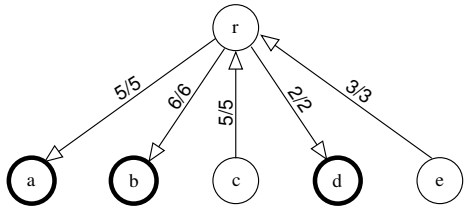
Proof. At each iteration, a positive amount of flow from r_k reaches node s, since downward residual arcs in T^f have non-zero capacity. Then, either a positive amount of flow reaches r_ℓ (which decreases total waste), or there exists $[u, v]$ in the weak branch such that $c^f_{uv} = 0$ and $[u, v]$ is upwards. In this case, u becomes strong. ■

Furthermore, assuming input data is integral, we can prove the following bound on the number of executions of the while loop in line 3:

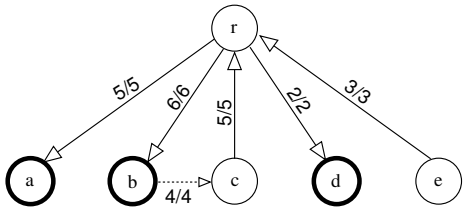
Lemma 5.3 (Hochbaum 2008, Corollary 6.3)**.** *Let $[S, \bar{S}]$ be an optimal blocking-cut for G. Then, the* `GenericPseudoflow` *algorithm terminates in $O(nC^*)$ iterations, where $C^* = c(S, \bar{S})$.*

Proof. Let M be the sum of arc capacities in $\delta^+(r)$; this is the amount of generated waste at the beginning of the algorithm, using the simple flow initialization. Then, since all arcs across the optimal blocking cut are saturated (Lemma 4.4), the waste can decrease to at most $M - C^*$. Since there can be at most n iterations of the while loop in `GenericPseudoflow` before the total waste is decreased (Lemma 5.2), there can be at most $O(nC^*)$ iterations of the while loop before the optimality conditions are satisfied. ■

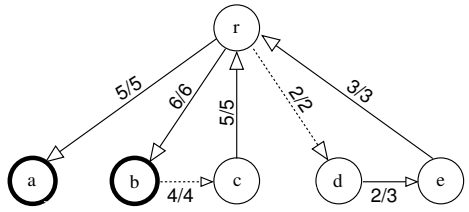
An example of the pseudoflow algorithm run on the graph in Figure 1(a) is shown in Figure 5.



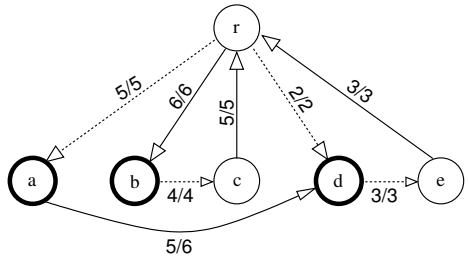
(a) The initial normalized tree



(b) Arc (b, c) enters the normalized tree; the excess at b is pushed to c and reaches capacity, so (b, c) is split.



(c) Arc (d, e) enters the tree; all excess at d is pushed to e. Node e remains weak, and node d becomes weak.



(d) Arc (a, d) enters the tree; all excess at a is pushed to d; $c^f_{de} = 1$, so the arc is split. Node d becomes strong.

Figure 5: The pseudoflow algorithm run on the graph in Figure 1(a), transformed into G^{ext} (Figure 2). We use the simple normalized tree initialization procedure. Strong nodes are bold, all other nodes are weak. Arcs not shown are at their lower bounds, and dashed arcs are non-tree arcs at their upper bounds. After the third step, no residual arcs exist between S and $\bar{S}$, so the algorithm terminates with the maximum blocking-cut shown in Figure 1(b) and the minimum-waste flow shown in Figure 1(c).

Clearly, we have a lot of flexibility in the choice of merger arc in `GenericPseudoflow`. If we choose the merger intelligently, we can get a substantial improvement in the running time of the algorithm; this is the topic of the next section.

5.3 The Pseudoflow-Labeling Algorithm

The pseudoflow algorithm shares many characteristics with the push-relabel algorithm of Goldberg and Tarjan (1988) for maximum flows in directed graphs. Recall that in the push-relabel algorithm, each node has an associated label which is an lower bound on the shortest path distance from that node to the sink. An arc is chosen to push flow across if the label of the tail is greater than the label of the head, and the arc has positive residual capacity. If no such arc exists and the solution is non-optimal, the label of some node is increased.

We employ a similar idea for the pseudoflow-labeling algorithm, following the description in Chandran and Hochbaum (2009), which uses slightly more standard terminology than Hochbaum (2008). Each node i in the graph has an associated label $l(i)$; we say that the labeling function is **valid** if for every residual arc (i, j), we have $l(i) \leq l(j) + 1$, and furthermore every strictly weak node j has $l(j) = 0$. We say than a residual arc $\langle i, j \rangle$ is **admissible** if $l(i) = l(j) + 1$.

Let $P = \langle i_1, i_2, ..., w \rangle$ be a path from node i_1 to a strictly weak node w in the residual graph, and let l be a valid labeling. Since $l(w) = 0$, and $l(i_1) \leq l(i_2) + 1 \leq l(i_3) + 2 \leq ... \leq l(w) + len(P)$, we have shown that in a valid labeling, $l(i)$ is a lower bound on the distance from i to a strictly weak node in the residual graph, and thus the label of any node is upper-bounded by n.

Function `PseudoflowLabel`(G)

1 $(G^{ext}, f, T^f) =$ `Initialize`(G)
2 $l(i) = 0 \forall\ i \in V$
3 $S = \text{strong}(T^f)$
4 **while** $c^f(S, \bar{S}) > 0$**:**
5 $r_k = \text{argmin}\{l(r_i) \mid r_i \in S \text{ and } r_i \text{ is a branch root}\}$
6 $stack = [r_k]$
7 $\langle\!\langle$ *Depth-first search for an admissible arc* $\rangle\!\rangle$
8 **while** $stack \neq \varnothing$**:**
9 $s = stack.\text{pop}()$
10 **if** $\exists\ w \in \delta^+(s)$ such that $l(s) = l(w) + 1$ and $c^f_{sw} > 0$ **:**
11 `Merge`$(T^f, (s, w))$
12 $stack = \varnothing$
13 **else if** s has no children with label $l(s)$**:** $l(s)\ +\!= 1$
14 **else:**
15 **for each** child u of s with label $l(s)$**:** $stack.\text{push}(u)$
16 $S = \text{strong}(T^f)$
17 return S

Figure 6: The pseudoflow-labeling algorithm.

If every node in a branch has label k, we say that the branch is a **label-k branch**. A branch is an **active** branch if its root is strictly strong and it is not a label-n branch. At each iteration of the pseudoflow-labeling algorithm, we select an admissible merger arc $\langle i, j \rangle$ such that i is in an active branch, and $l(i)$ has the lowest label among all nodes in active branches. This choice of label ensures that j is weak, and thus $\langle i, j \rangle$ is a valid merger arc in line 4 of `GenericPseudoflow`.

To ensure that we can quickly find a node of lowest label in an active branch, we further enforce the monotonicity property on the labeling function l; specifically, for each downwards branch edge $[i, j]$, we must

have either $l(i) = l(j)$ or $l(i) = l(j) - 1$. This ensures that the branch root is the lowest-labeled node, and along any path from the branch root, node labels are non-decreasing. We can then find the set of lowest-labeled nodes in the branch by performing depth-first search from the branch root.

Let T_k^f be the branch with lowest-labeled root; if no admissible arc is found in the T_k^f, then we find a node i such that $l(i) = l(r_k)$ and every child j of i satisfies $l(j) = l(i)+1$. We then increment i's label by 1. This new labeling clearly satisfies the monotonicity property, by construction. Pseudocode for the pseudoflow-labeling algorithm is given in Figure 6; the following lemma shows that the new labeling produced by an iteration of `PseudoflowLabel` is valid.

Lemma 5.4. *If l is a valid labeling at some iteration of* `PseudoflowLabel`*, and the algorithm relabels node i in that iteration, the new labeling l' is valid.*

Proof. By Lemma 5.2, notice that once a strictly-weak node satisfies its demand, it never becomes strictly weak again. Furthermore, since only labels of strong nodes are increased, all strictly weak nodes w have $l'(w) = 0$.

Thus, it remains to show that for all residual arcs (u, v), we have $l'(u) \leq l'(v)+1$. Notice that no other labels except i changed, so we only need to consider edges $[i, j]$ incident to i. There are three cases to consider:

Edge $[i, j]$ enters node i: In this case, $l'(j) = l(j) \leq l(i) + 1 \leq l'(i) + 1$.

Edge $[i, j]$ leaves node i and node j is strong: By choice of i, we have that $l(i) \leq l(j)$, so $l'(i) = l(i) + 1 \leq l(j) + 1 = l'(j) + 1$.

Edge $[i, j]$ leaves node i and node j is weak: Since no arcs from i were admissible, we see that $l(i) < l(j) + 1$; therefore, $l'(i) = l(i) + 1 \leq l(j) + 1 = l'(j) + 1$.

Thus, l' is a valid labeling. ∎

Notice that the labeling procedure is only a method of choosing between the possible merger arcs; it does not affect any other portion of the algorithm, and thus `PseudoflowLabel` is still correct. However, the running time is much improved. To show this, we first need the following lemma:

Lemma 5.5. *Let (i, j) be the admissible arc chosen at some iteration of* `PseudoflowLabel`*; let r_k be the root of the branch containing i and r_ℓ the root of the branch containing j. Then, for each vertex $u \in [r_k, ..., i]$, we have $l(u) = l(i)$, and for each edge $[u, v] \in [j, ..., r_\ell]$, we have $l(u) \geq l(v)$. In particular, the path $[r_k, ..., i, j, ..., r_\ell]$ has non-increasing labels.*

Proof. The first condition follows by choice of i; the second condition follows from the monotonicity of l. ∎

To bound the running time, we first analyze the number of merge steps that can be performed over the course of the algorithm.

Lemma 5.6. *The total number of merge steps performed by* `PseudoflowLabel` *is $O(mn)$*

Proof. Let $\langle i, j\rangle$ be a merger arc at some iteration t_1; this arc then enters the normalized tree T^f, or becomes saturated. If $\langle i, j\rangle$ is to be a merger arc again at some later iteration $t_3 > t_1$, it must have previously left T^f at iteration t_2, where $t_1 < t_2 < t_3$. To leave the normalized tree, the residual arc $\langle j, i\rangle$ must become saturated, and then $\langle i, j\rangle$ must become admissible again. If $\langle j, i\rangle$ becomes saturated, by Lemma 5.5, we must have $l(j) \geq l(i)$. For $\langle i, j\rangle$ to become admissible again, node i has to be relabeled at least once; since node labels are bounded by n, arc $\langle i, j\rangle$ can be a merger arc at most n times. ∎

This in turn yields the following result:

Theorem 5.2 (Chandran and Hochbaum 2009, Lemma 4.1)**.** *The total running time of* `PseudoflowLabel` *is $O(mn^2)$.*

Proof. The `Merge` procedure iterates through the for loop in line 5 at most n times, and all other steps (including the `Split` procedure) take a constant amount of work, meaning that `Merge` does $O(mn^2)$ work over the course of the algorithm, by Lemma 5.6. For each node s in line 10 of `PseudoflowLabel`, we only need to check for admissible arcs once per label, so the total number of checks over the course of the algorithm is $O(mn)$.

Since node labels are bounded by n, the relabeling step in line 13 executes at most $O(n^2)$ times. Finally, for each value k of the lowest-labeled branch root, we scan every arc in T^f at most twice, and every arc outside of T^f at most once, so the DFS procedure takes $O(m)$ work for this phase, or $O(mn)$ total work over the course of the algorithm, yielding a worst-case running time of $O(mn^2)$. ∎

We can further improve the running time of the `PseudoflowLabel` algorithm by using the dynamic trees data structure of Sleator and Tarjan (1983). This data structure enables `Merge` to be implemented in $O(\log n)$ time instead of $O(n)$ time as presented above. This, together with Lemma 5.6, gives an $O(mn \log n)$ running time bound on the `PseudoflowLabel` algorithm. In fact, using dynamic trees and a variant of `PseudoflowLabel` that selects an admissible arc from the highest-labeled branch root, Hochbaum and Orlin (2007) show an $O(mn \log(n^2/m))$ worst-case running time bound on the pseudoflow-labeling algorithm, the same worst-case time as push-relabel for the max-flow problem. In the next section, we show how to use the pseudoflow-labeling algorithm to solve the max-flow problem in the same complexity.

6 Maximum Flows from Minimum-Waste Pseudoflows

As might be guessed from the LP formulation of the maximum blocking-cut problem in Section 3, the maximum blocking-cut problem and the min-cut problem are equivalent. Thus, we can use the pseudoflow algorithm to solve for maximum flows in a graph. In this section, we first prove the equivalence of the min-cut and maximum blocking-cut problem, and then briefly describe how to recover maximum flows from a minimum-waste pseudoflow, following a description of the same procedure in Hochbaum (2008).

For this problem, we are given a directed graph $G_{st} = (V + s + t, A)$, with two distinguished nodes, a **source** s and a **sink** t. In this section, we use the term "flow" in the traditional sense, that is, a function $f : A \to \mathbb{R}$ that has 0 excess at every vertex except the source and sink. We transform G_{st} into G^{ext} by collapsing s and t into a single vertex r; we can now apply the pseudoflow algorithm to this problem.

We can also transform G_{st} into a node-weighted graph by removing s and t, and setting $w_i = c_{si}$ for each node adjacent to s, and $w_j = -c_{jt}$ for each node incident to t. Nodes incident to neither s nor t have weight 0; if a node is incident to both s and t, we send $\min\{c_{si}, c_{it}\}$ units of flow along it and delete the saturated arc, setting the node weight of i to the remainder. Clearly this procedure can be reversed without loss of information.

Theorem 6.1 (Hochbaum 2008, Lemma 3.1)**.** *Given a graph G_{st} and the reduced graph G, a minimum cut set S in G_{st} is a maximum blocking-cut in G, and vice versa.*

Proof. In G_{st}, for a set $S \subseteq V$, the surplus of S is defined as $c(s, S) - c(S, t) - c(S, V \setminus S)$. Thus, the maximum-surplus set is given by

$$\begin{aligned}
\max_{S \subseteq V} \{c(s, S) - c(S, t) - c(S, V \setminus S)\} &= \max_{S \subseteq V} \{c(s, V) - c(s, V \setminus S) - c(S, t) - c(S, V \setminus S)\} \\
&= c(s, V) - \min_{S \subseteq V} \{c(s, V \setminus S) + c(S, V \setminus S) + c(S, t)\} \\
&= c(s, V) - \min_{S \subseteq V + s, s \in S} c(S, \bar{S})
\end{aligned}$$

Since $c(s, V)$ is constant, this completes the proof. ∎

The above theorem states that to find a minimum cut in G_{st}, we simply need to find a maximum blocking-cut. However, more often we are interested in finding a maximum flow, not just the associated bottleneck. Clearly, the minimum-waste pseudoflow is not a maximum flow! To recover the maximum flow, we need to do a bit more work.

Given a minimum-waste pseudoflow f on G_{st}, we construct a maximum flow via the following procedure. Create two additional nodes $\bar{s}$ and $\bar{t}$; add an arc from t and every strictly strong node in V to $\bar{t}$, and add an arc from $\bar{s}$ to s and every strictly weak node in V. All arcs in $\delta(\bar{s})$ and $\delta(\bar{t})$ have infinite capacity.

Using the flow decomposition procedure of Ahuja et al. (1993), we can find a set of paths in the residual graph from every strictly strong node in V to $\bar{s}$; augmenting flow along these paths will eliminate the excess at every strictly strong node in V. Furthermore, since all strictly strong nodes are in the same partition as s, no flow is augmented across the cut, leaving those arcs saturated by Lemma 4.4. We can then repeat the same procedure by augmenting flow from $\bar{t}$ to every strictly weak node, thus eliminating their deficit (Hochbaum, 2008).

The flow decomposition procedure can be run in $O(mn)$ time using depth-first search, or in time $O(m \log n)$ using the dynamic trees algorithm of Sleator and Tarjan (1983). In the latter case, this does not decrease the worst-case complexity of the pseudoflow-labeling algorithm by more than a constant factor. In the next section we show that in practice the pseudoflow-labeling algorithm with the above flow computation procedure consistently outperforms the current best solver for maximum flow problems.

7 Computational Results

In Chandran and Hochbaum (2009), the pseudoflow algorithm is compared computationally to the push-relabel algorithm of Goldberg and Tarjan (1988). In particular, they use the transformation described at the beginning of Section 6 to transform a graph G_{st} with distinguished source and sink nodes into a graph G^{ext}; then, they use the procedure in Section 6 to recover the actual flow value from the graph, and compare the running times and number of iterations taken to an implementation of the push-relabel algorithm run on G_{st}.

The implementation of the pseudoflow algorithm uses the strongly-polynomial variant described in Section 5.3, with the following modifications. First, they do not employ a dynamic trees data structure to merge the tree structures, instead using the $O(mn^2)$ variant of the algorithm. Secondly, they use the highest-label variant of `PseudoflowLabel` presented at the end of Section 5.3. Finally, they employ the following two heuristics, which do not affect the worst-case running time of the algorithm, but substantially improve the performance in practice:

Heuristic 1. *For each vertex in the graph, the index of the last arc scanned in search of a merger is maintained. If a vertex is visited more than once in successive iterations, the search for a merger arc is resumed from this index, instead of from the beginning of the list of incident arcs. When the vertex label is increased, this pointer is reset to* 0.

Heuristic 1 ensures that for each node, every arc incident to that node is scanned at most once while the node's label remains constant. If arc (i, j) is not admissible at some point in the algorithm, then $l(i) \leq l(j)$. Since node labels are strictly increasing, this means that until i's label increases, $l(i) < l(j) + 1$ at all future iterations. Thus, this heuristic does not invalidate the search for admissible arcs.

Heuristic 2. *If a branch root has label $l(r_k)$, and no vertices in the graph have label $l(r_k) - 1$, the labels of all nodes in r's branch are set to n.*

In particular, since within a branch labels are non-decreasing, this implies that for all $i \in T_k^f$, $l(i) \geq l(r_k)$. Furthermore, since no node in the graph has label $l(r_k) - 1$, and the node labels are lower bounds on the distance from the node to a strictly weak node, this means that no residual path from any node in T_k^f exists to a strictly weak node. Thus, it is part of the minimum blocking-cut, and Heuristic 2 discards it for the remainder of the algorithm.

The pseudoflow algorithm with these heuristics is compared to the most efficient implementation of the push-relabel algorithm for maximum flows by Goldberg (accessed June 2007). This implementation does not make use of dynamic trees, and so has a worst-case running time of $O(mn^2)$ as well. In addition, it employs similar heuristics to Heuristics 1 and 2.

Chandran and Hochbaum (2009) compare the two algorithms across ten different classes of randomly generated graphs using the DIMACS graph generator. Across all ten classes, problem sizes range from 128

vertices and 7920 arcs to 1 048 578 vertices and 3 145 664 arcs. It was found that the pseudoflow-labeling algorithm outperformed the push-relabel algorithm by factor of 2 or 3 in most cases. The pseudoflow algorithm never performed worse than the push-relabel algorithm, and in only one class of generated problems did push-relabel perform evenly with the pseudoflow algorithm, and even then only on the largest problem sizes generated for that instance.

8 Sensitivity Analysis

In many applications, we are concerned with not only the value of the minimum-waste flow for a specific set of supply values, but we are also interested in solving the problem over a range of supply values. This may represent, for example, the changing market value over time of the ore being mined from an open-pit mining project (Hochbaum, 2001). Or, in the context of the maximum-flow problem, this represents the situation where the capacities of arcs incident to source and sink nodes are given parametrically.

In particular, we study the case where the supply at nodes is given as a monotone increasing function of a parameter λ, and the demand at nodes is a monotone decreasing function of λ. The goal is to identify multiple maximum blocking-cuts for various values of λ.

There are two cases we consider; first, the case when we are given a series of **breakpoints** $\lambda_1 < \lambda_2 < ... < \lambda_k$, that is, k parameter values for which we are interested in computing a maximum blocking-cut. After solving the complete problem for λ_1, we can leave the distance labels alone, and renormalize the tree, to then solve the problem for λ_2 (this is precisely the "warm start" procedure described in Section 5.2). The tree renormalization process takes $O(n)$ time, which means that the total running time for the algorithm is $O(mn \log n + kn)$; when $k = O(m \log n)$, this analysis runs within a constant factor of the time needed to solve a single problem instance (Hochbaum, 2008).

On the other hand, we may be interested in finding *all* of the breakpoint values – that is, every value of λ at which the set of vertices forming the maximum blocking-cut changes. As it turns out, this problem can be solved in the same computational complexity as solving a single problem instance! Notice that as λ increases, the size of the maximum blocking-cut can only increase; strong nodes in the normalized tree can only become stronger, and some weak nodes may become strong. We can exploit this structure using a method of iterated contraction to narrow down on the values of each of the points at which the maximum blocking-cut set changes. This procedure is quite similar to the method of Gallo et al. (1988) for performing sensitivity analysis on a max-flow/min-cut instance. This algorithm is somewhat complex and beyond the scope of this paper; for more details, we refer the reader to Hochbaum (2008) and Gallo et al. (1988).

9 Conclusion

In this paper, we present the work of Hochbaum (2008) from the perspective of the maximum blocking-cut/minimum-waste flow problem, instead of the more traditional max-flow/min-cut viewpoint. We believe that the minimum-waste flow problem is a relevant problem in and of itself, and demonstrate how it can be solved using the pseudoflow algorithm.

In addition, we perform analysis of the problems using linear programming techniques, as well as providing an independent proof of the equality of the maximum blocking-cut problem and the minimum-waste flow problem, and we give a proof from Hochbaum (2008) showing that these problems are equivalent to the max-flow/min-cut problems. Finally, we present some computational results from Chandran and Hochbaum (2009) comparing the pseudoflow algorithm to the best-known implementation of the push-relabel algorithm, which show that the pseudoflow algorithm performs better in practice than push-relabel on most problems.

Future research on this problem could be directed in a number of different directions. First, many problems of interest deal with **generalized networks**; a generalized network is a directed graph with arc capacities, together with a set of **arc multiplicities** μ_{ij}, where if f_{ij} units of flow leave vertex i, $\mu_{ij} f_{ij}$ units of flow arrive at vertex j. A modified version of the push-relabel algorithm exists for the generalized network setting (Tardos and Wayne, 1998), and it would be interesting to adapt the pseudoflow algorithm to generalized networks to see what improvements can be made.

Finally, a number of network flow problems come with additional **side constraints** on the flow values that travel across arcs. One example of such a constrained network problem is the **equal flow problem**, in which some subsets of arcs in the graph are required to carry identical amounts of flow Ahuja et al. (1999). It would be interesting to extend the pseudoflow algorithm to settings where the networks have additional side constraints, in an attempt to find better algorithms for these problems.

The pseudoflow algorithm is an important new development in the field of computer science theory and operations research, and should be useful in a large number of applications – both in existing algorithms that employ max-flow solvers as a black box, and in the study of new and related problems, such as the maximum blocking-cut/minimum-waste flow problem.

References

Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows*. Prentice Hall, 1993.

Ravindra K. Ahuja, James B. Orlin, Giovanni M. Sechi, and Paola Zuddas. Algorithms for the simple equal flow problem. *Management Science*, 45(10):1440–1455, October 1999.

Bala G. Chandran and Dorit S. Hochbaum. A computational study of the pseudoflow and push-relabel algorithms for the maximum-flow problem. *Operations Research*, 57(2):358–376, March-April 2009.

Lester R. Ford, Jr. and Delbert R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, February 1956.

Giorgio Gallo, Michael D. Grigoriadis, and Robert E. Tarjan. A fast parametric maximum flow algorithm and applications. *SIAM Journal of Computing*, 18(1):30–55, February 1988.

Andrew V. Goldberg. Andrew Goldberg's network optimization library, accessed June 2007. `http://avglab.com/andrew/soft.html`.

Andrew V. Goldberg and Robert E. Tarjan. A new approach to the maximum-flow problem. *Journal of the Association for Computing Machinery*, 35(4):921–940, October 1988.

Dorit S. Hochbaum. A new-old algorithm for minimum-cut and maximum-flow in closure graphs. *Networks*, 37(4):171–193, February 2001.

Dorit S. Hochbaum. The pseudoflow algorithm: A new algorithm for the maximum-flow problem. *Operations Research*, 56(4):992–1009, July-August 2008.

Dorit S. Hochbaum and James B. Orlin. The pseudoflow algorithm in $O(mn \log(n^2/m))$ and $O(n^3)$. Manuscript, University of California, Berkeley, 2007.

Helmut Lerchs and Ingo F. Grossmann. Optimum design of open-pit mines. *Transactions of the Canadian Institute of Mining and Metallurgy and the Mining Society of Nova Scotia*, 68:17–24, 1965.

GuoJun Qi. Integer programming, integer polyhedra, and totally unimodular matrices. Class notes, Combinatorial Optimization (CS598CSC) at UIUC, accessed June 2011. `http://www.cs.illinois.edu/class/sp10/cs598csc/Lectures/Lecture4.pdf`.

Tomasz Radzik. Parametric flows, weighted means of cuts, and fractional combinatorial optimization. In Panos M. Pardalos, editor, *Complexity in Numerical Optimization*, pages 351–386. World Scientific Publishing Co., 1993.

Daniel D. Sleator and Robert E. Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26:362–391, 1983.

Éva Tardos and Kevin D. Wayne. Simple generalized maximum flow algorithms. In *7th International Integer Programming and Combinatorial Optimization Conference*, 1998.