The Pseudoflow Algorithm and the Pseudoflow-Based Simplex for the Maximum Flow Problem

Dorit S. Hochbaum

Department of Industrial Engineering and Operations Research, and Walter A. Haas School of Business, University of California, Berkeley

Abstract. We introduce an algorithm that solves the maximum flow problem without generating flows explicitly. The algorithm solves directly a problem we call the maximum s-excess problem. That problem is equivalent to the minimum cut problem, and is a direct extension of the maximum closure problem. The concepts used also lead to a new parametric analysis algorithm generating all breakpoints in the amount of time of a single run.

The insights derived from the analysis of the new algorithm lead to a new simplex algorithm for the maximum flow problem – a pseudoflow-based simplex. We show that this simplex algorithm can perform a parametric analysis in the same amount of time as a single run. This is the first known simplex algorithm for maximum flow that generates all possible breakpoints of parameter values in the same complexity as required to solve a single maximum flow instance and the fastest one.

The complexities of our pseudoflow algorithm, the new simplex algorithm, and the parametric analysis for both algorithms are $O(mn \log n)$ on a graph with n nodes and m arcs.

1 Introduction

This extended abstract describes an efficient new approach to the maximum flow and minimum cut problems. The approach is based on a new certificate of optimality inspired by the algorithm of Lerchs and Grossmann, [LG64]. This certificate, called *normalized tree*, partitions the set of nodes into subsets some of which have excess capacity and some have capacity deficit. The nodes that belong to the subsets with excess form the source set of a candidate minimum cut. The algorithm solves, instead of the maximum flow problem, another problem which we call the maximum *s-excess problem*. That problem is defined on a directed graph with arc capacities and node weights and does *not* contain distinguished source and sink nodes. The objective of the s-excess problem is to find a subset of the nodes that maximizes the sum of node weights, minus the weight of the arcs separating the set from the remainder of the nodes. The new problem

[*] Research supported in part by NEC, by NSF award No. DMI-9713482, and by SUN Microsystems.

is shown to be equivalent to the minimum cut problem that is traditionally solved by deriving a maximum flow first. With the new algorithm these problems can be solved without considering flows explicitly. The steps of the algorithm can be interpreted as manipulating pseudoflow – a flow that does not satisfy flow balance constraints. For this reason we choose to call the algorithm *the pseudoflow algorithm.*

The main feature that distinguishes the pseudoflow algorithm from other known algorithms for the maximum flow problem is that it does not seek to either preserve or progress towards feasibility. Instead the algorithm creates "pockets" of nodes so that at optimum there is no residual arc that can carry additional flow between an "excess pocket" and a "deficit pocket". The set of nodes in all the "excess pockets" form the source set of a minimum cut and also the maximum s-excess set.

The certificate maintained by our algorithm bears a resemblance to the basic arcs tree maintained by simplex. It is demonstrated that this certificate is analogous to the concept of a strong basis introduced by Cunningham [C76] if implemented in a certain "extended network" permitting violations of flow balance constraints. It is further shown that the algorithmic steps taken by our algorithm are substantially different from those of the simplex and lead to a different outcome in the next iteration based on the same strong basis in the given iteration.

The contributions in this paper include:

1. The introduction of the pseudoflow maximum s-excess problem that provides a new perspective on the maximum flow problem.
2. A pseudoflow algorithm for the maximum flow problem of complexity $O(mn \log n)$.
3. Parametric analysis conducted with the pseudoflow algorithm that generates all breakpoints in the same complexity as a single run.
4. A new pseudoflow-based simplex algorithm for maximum flow using the lowest label approach.
5. A parametric analysis simplex method that finds all possible parameter breakpoints in the same time as a single run, $O(mn \log n)$.

The parametric simplex method is the first known parametric implementation of simplex to date that finds all breakpoints in the same running time as a single application.

1.1 Notation

For $P, Q \subset V$, the set of arcs going from P to Q is denoted by, $(P, Q) = \{(u, v) \in A | u \in P \text{ and } v \in Q\}$. Let the capacity of arc (u, v) be denoted by c_{uv} or $c(u, v)$. For $P, Q \subset V$, $P \cap Q = \emptyset$, the capacity of the cut separating P from Q is, $C(P, Q) = \sum_{(u,v) \in (P,Q)} c_{uv}$. For $S \subseteq V$, let $\bar{S} = V \setminus S$.

For a graph $G = (V, A)$ we denote the number of arcs by $m = |A|$ and the number of nodes by $n = |V|$.

An arc (u, v) of an unspecified direction is referred to as *edge* $[u, v]$. $[v_1, v_2, \ldots, v_k]$ denotes an *undirected* path from v_1 to v_k. That is, $[v_1, v_2], \ldots, [v_{k-1}, v_k] \in A$.

We use the convention that the capacity of an arc that is not in the graph is zero. Thus for $(a, b) \in A$ and $(b, a) \notin A$, $c_{b,a} = 0$.

The capacity of an edge e is denoted either by $c(e)$ or c_e. The flow on an edge e is denoted either by $f(e)$ or f_e. We use the convention that $f(a, b) = -f(b, a)$. For a given flow or pseudoflow f, the residual capacity of e is denoted by $c_f(e)$ which is $c_e - f_e$.

Given a rooted tree, T, T_v is the subtree suspended from node v that contains all the descendants of v in T. $T_{[v,p(v)]} = T_v$ is the subtree suspended from the edge $[v, p(v)]$. An immediate descendant of a node v, a *child* of v, is denoted by $ch(v)$, and the unique immediate ancestor of a node v, the *parent* of v, by $p(v)$.

2 The Maximum Flow Problem and the Maximum s-Excess Problem

The pseudoflow algorithm described here ultimately finds a maximum flow in a graph. Rather than solving the problem directly the algorithm solves instead the maximum s-excess problem.

Problem Name: *Maximum s-Excess*
Instance: *Given a directed graph $G = (V, A)$, node weights (positive or negative) w_i for all $i \in V$, and nonnegative arc weights c_{ij} for all $(i, j) \in A$.*
Optimization Problem: *Find a subset of nodes $S \subseteq V$ such that $\sum_{i \in S} w_i - \sum_{i \in S, j \in \bar{S}} c_{ij}$ is maximum.*

We elaborate here further on this problem and its relationship to the maximum flow and minimum cut problems.

The maximum flow problem is defined on a directed graph with distinguished source and sink nodes and the arcs adjacent to source and sink, $A(s)$ and $A(t)$, $G_{st} = (V \cup \{s, t\}, A \cup A(s) \cup A(t))$.

The standard formulation of the maximum flow problem with zero lower bounds and x_{ij} variables indicating the amount of flow on arc (i, j) is,

$$\begin{array}{ll} \text{Max} & x_{ts} \\ \text{subject to} & \sum_i x_{ki} - \sum_j x_{jk} = 0 \quad k \in V \\ & 0 \le x_{ij} \le c_{ij} \quad \forall (i, j) \in A. \end{array}$$

In this formulation the first set of (equality) constraints is called the *flow balance* constraints. The second set of (inequality) constraints is called the *capacity* constraints.

Definition 1. *The s- excess capacity of a set $S \subseteq V$ in the graph $G_{st} = (V \cup \{s, t\}, A \cup A(s) \cup A(t))$ is, $C(\{s\}, S) - C(S, \bar{S} \cup \{t\})$.*

We claim that finding a subset $S \subseteq V$ that maximizes $C(\{s\}, S) - C(S, \bar{S} \cup \{t\})$ is equivalent to the maximum s-excess problem.

Lemma 2. *A subset of nodes $S \subseteq V$ maximizes $C(\{s\}, S) - C(S, \bar{S} \cup \{t\})$ in G_{st} if and only if it is of maximum s-excess in the graph $G = (V, A)$.*

We next prove that the s-excess problem is equivalent to the minimum cut problem,

Lemma 3. *S is the source set of a minimum cut if and only if it is a set of maximum s- excess capacity $C(s, S) - C(S, \bar{S} \cup \{t\})$ in the graph.*

Proof. Given an instance of the s-excess problem. Append to the graph $G = (V, A)$ the nodes s and t; Assign arcs with capacities equal to the weights of nodes from s to the nodes of positive weight; Assign arcs with capacities equal to the absolute value of the weights of nodes of negative weights, from the nodes to t.

The sum of weights of nodes in S is also the sum of capacities $C(\{s\}, S) - C(S, \{t\})$ where the first term corresponds to positive weights in S, and the second to negative weights in S:

$$\begin{aligned}\textstyle\sum_{j\in S} w_j - \sum_{i\in S, j\in\bar{S}} c_{ij} &= C(\{s\}, S) - C(S, \{t\}) - C(S, \bar{S}) \\ &= C(\{s\}, S) - C(S, \bar{S} \cup \{t\}). \\ &= C(\{s\}, V) - C(\{s\}, \bar{S}) - C(S, \bar{S} \cup \{t\}). \\ &= C(\{s\}, V) - C(\{s\} \cup S, \bar{S} \cup \{t\}).\end{aligned}$$

The latter term is the capacity of the cut $(\{s\} \cup S, \bar{S} \cup \{t\})$. Hence maximizing the s-excess capacity is equivalent to minimizing the capacity of the cut separating s from t.

To see that the opposite is true, consider the network $(V \cup \{s, t\}, A)$ with arc capacities. Assign to node $v \in V$ a weight that is c_{sv} if the node is adjacent to s and $-c_{vt}$ if the node is adjacent to t. Note that it is always possible to remove paths of length 2 from s to t thus avoiding the presence of nodes that are adjacent to *both* source and sink. This is done by subtracting from the arcs' capacities c_{sv}, c_{vt} the quantity $\min\{c_{sv}, c_{vt}\}$. The capacities then translate into node weights that serve as input to the s-excess problem and satisfy the equalities above. □

We conclude that the maximum s-excess problem is a complement of minimum cut which in turn is a dual of the maximum flow problem. As such, its solution does not contain more flow information than the solution to the minimum cut problem. As we see later, however, it is possible to derive a feasible flow of value equal to that of the cut from the certificate used in the algorithm, in $O(mn)$ time.

The reader may wonder about the arbitrary nature of the s-excess problem, at least in the sense that it has not been previously addressed in the literature. The explanation is that this problem is a relaxation of the maximum closure problem where the objective is to find, in a node weighted graph, a closed set of nodes of maximum total weight. In the maximum closure problem it is required that all successors of each node in the closure set will belong to the set. In the s-excess problem this requirement is replaced by a penalty assigned to arcs of immediate successors that are not included in the set. In that sense the s-excess problem is a *relaxation* of the maximum closure problem. The proof of Lemma 3 is effectively an extension of Picard's proof [Pic76] demonstrating that the maximum weight closed set in a graph (maximum closure) is the source set of a minimum cut.

We provide a detailed account of the use of the pseudoflow and other algorithms for the maximum closure problem in [Hoc96]. We also explain there how these algorithms have been used in the mining industry and describe the link to the algorithm of Lerchs and Grossmann, [LG64].

3 Preliminaries and Definitions

Pseudoflow is an assignment of values to arcs that satisfy the capacity constraints but not necessarily the flow balance constraints. Unlike preflow, that may violate the flow balance constraints only with inflow exceeding outflow, pseudoflow permits the inflow to be either strictly larger than outflow (excess) or outflow strictly larger than inflow (deficit). Let f be a pseudoflow vector with $0 \leq f_{ij} \leq c_{ij}$ the pseudoflow value assigned to arc (i, j). Let $\mathit{inflow}(D)$, $\mathit{outflow}(D)$ be the total amount of flow incoming and outgoing to and from the set of nodes D. For each subset of nodes $D \subset V$,

$$\begin{aligned} \mathit{excess}(D) &= \mathit{inflow}(D) - \mathit{outflow}(D) \\ &= \textstyle\sum_{(u,v)\in(V\cup\{s\}\setminus D, D)} f_{u,v} - \sum_{(v,u)\in(D, V\cup\{t\}\setminus D)} f_{v,u}. \end{aligned}$$

The absolute value of excess that is less than zero is called *deficit*, i.e. $-\mathit{excess}(D) = \mathit{deficit}(D)$.

Given a rooted tree T. For a subtree $T_v = T_{[v,p(v)]}$, let $M_{[v,p(v)]} = M_v = \mathit{excess}(T_v)$ and be called the *mass* of the node v or the arc $[v, p(v)]$. That is, the mass is the amount of flow on $(v, p(v))$ directed towards the root. A flow directed in the opposite direction – from $p(v)$ to v – is interpreted as negative excess or mass.

We define the *extended network* as follows: The network G_{st} is augmented with a set of arcs – two additional arcs per node. Each node has one arc of infinite capacity directed into it from the sink, and one arc of infinite capacity directed from it to the source. This construction is shown in Figure 1. We refer to the appended arcs from sink t as the *deficit arcs* and the appended arcs to the source s as the *excess arcs*. The source and sink nodes are compressed into a 'root' node r. We refer to the extended network's set of arcs as A^{aug}. These include, in addition to the deficit and excess arcs, also the arcs adjacent to source and sink – $A(s)$ and $A(t)$. The *extended network* is the graph $G^{\mathrm{aug}} = (V \cup \{r\}, A^{\mathrm{aug}})$.

Any pseudoflow on a graph has an equivalent feasible flow on the extended network derived from the graph – a node with excess sends the excess back to

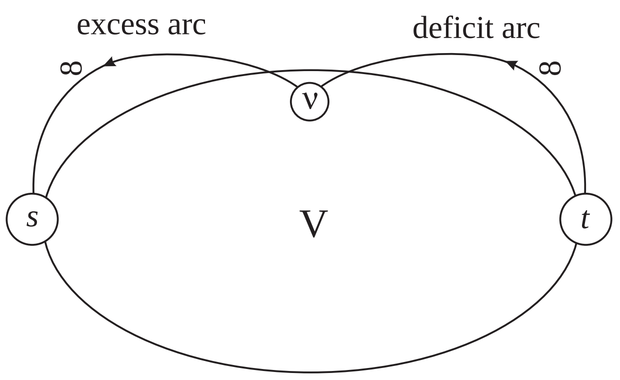

Fig. 1. An extended network with excesses and deficits

the source, and a node with deficit receives a flow that balances this deficit from the sink.

Throughout our discussion of flows on extended networks, all the flows considered saturate the arcs adjacent to source and sink and thus the status of these arcs, as saturated, remains invariant. We thus omit repeated reference to the arcs $A(s)$ and $A(t)$.

4 A Normalized Tree

The algorithm maintains a construction that we call *a normalized tree* after the use of this term by Lerchs and Grossmann in [LG64] for a construction that inspired ours. Let node $r \notin V$ serve as root and represent a contraction of s and t. Let $(V \cup \{r\}, T)$ be a tree where $T \subset \bar{A}$. The children of r are called the *roots* of their respective *branches* or subtrees. The deficit and excess arcs are only used to connect r to the roots of the branches.

A normalized tree is a rooted tree in r that induces a forest in (V, A). We refer to each rooted tree in the forest as *branch*. A branch of the normalized tree rooted at a child of r, r_i, T_{r_i} is called *strong* if $excess(T_{r_i}) > 0$, and *weak* otherwise. All nodes of strong branches are considered strong, and all nodes of weak branches are considered weak.

A normalized tree is depicted in Figure 2.

Consider a rooted forest $T \setminus \{r\}$ in $G = (V, A)$ and a pseudoflow f in G_{st} satisfying the properties:

Property 1 *The pseudoflow f saturates all source-adjacent arcs and all sink-adjacent arcs.*

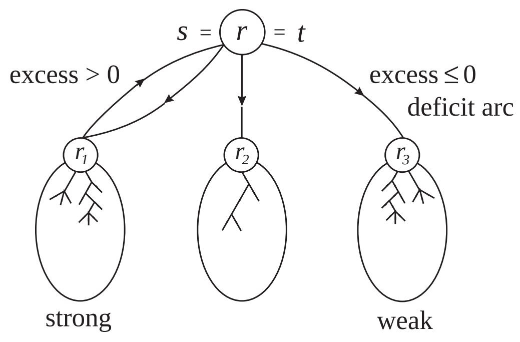

Fig. 2. A normalized tree. Each r_i is a root of a branch

Property 2 *In every branch all downwards residual capacities are strictly positive.*

Property 3 *The only nodes that do not satisfy flow balance constraints are the roots of their respective branches that are adjacent to r in the extended network.*

Definition 4. *A tree T with pseudoflow f is called* normalized *if it satisfies properties 1, 2, and 3.*

Property 3 means, in other words, that in order for T to be a normalized tree, f has to satisfy flow balance constraints in the extended network with only the roots of the branches permitted to send/receive flows along excess or deficit arcs.

We let the excess of a normalized tree T and pseudoflow f be the sum of excesses of its strong branches (or strong nodes). Property 1 implies that the excess of a set of strong nodes in a normalized tree, S, satisfies:

$$\begin{aligned} excess(S) &= inflow(S) - outflow(S) = f(\{s\}, S) + f(\bar{S}, S) - f(S, \bar{S}) \\ -f(S, \{t\}) &= C(\{s\}, S) - C(S, \{t\}) + f(\bar{S}, S) - f(S, \bar{S}). \end{aligned}$$

This equality is used to prove the superoptimality of the set of strong nodes (in the superoptimality Property).

We choose to work with normalized trees that satisfy an optional property stronger than 2:

Property 4 (unsaturated arcs property) *The tree T has all upwards residual capacities strictly positive.*

Another optional implied property is:

Property 5 *All "free" arcs of A with flows strictly between lower and upper bound are included in T.*

With this property *all* arcs that are not adjacent to root are free. T is thus the union of all free arcs including some excess and deficit arcs. The only arcs in T that are not free are deficit arcs with 0 flow – adjacent to 0-deficit branches. All out of tree arcs are thus at their upper or at the lower bounds.

The next property – the superoptimality property – is satisfied by any normalized tree, or equivalently, by any tree-pseudoflow pair satisfying properties 1, 2, and 3. The superoptimality of a normalized tree and the conditions for optimality are stated in the next subsection.

4.1 The Superoptimality of a Normalized Tree

Property 6 (superoptimality) *The set of strong nodes of the normalized tree T is a superoptimal solution to the s-excess problem: The sum of excesses of the strong branches is an upper bound on the maximum s-excess.*

From the proof of the superoptimality property it follows that when all arcs from S to $\bar{S}$ are saturated, then no set of nodes other than S has a larger s-excess. We thus obtain the optimality condition as a corollary to the superoptimality property.

Corollary 5 (Optimality condition). *Given a normalized tree, a pseudoflow f and the collection of strong nodes in the tree S. If f saturates all arcs in $(S, \bar{S})$ then S is a maximum s-excess set in the graph.*

The next Corollary holds only if Property 4 is satisfied. It implies minimality of the optimal solution set S.

Corollary 6 (Minimality). *Any proper subset of the strong nodes is not a maximum s-excess set in (V, A').*

On the other hand, it is possible to append to the set of strong nodes S any collection of 0-deficit branches that have no residual arcs to weak nodes without changing the value of the optimal solution. This leads to a *maximal* maximum s-excess set.

5 The Description of the Pseudoflow Algorithm

The algorithm maintains a superoptimal s-excess solution set in the form of the set of strong nodes of a normalized tree. That is, the sum of excesses of strong branches is only greater than the maximum s-excess. Each strong branch forms an "excess pocket" with the total excess of the branch assigned to its root. Within each branch the pseudoflow is feasible.

Each iteration of the algorithm consists of identifying an infeasibility in the form of a residual arc from a strong node to a weak node. The arc is then added in and the tree is updated. The update consists of pushing the *entire* excess of the strong branch along a path from the root of the strong branch to the merger arc (s', w) and progressing towards the root of the weak branch. The first arc encountered that does not have sufficient residual capacity to accommodate the pushed flow gets *split* and the subtree suspended from that arc becomes a strong branch with excess equal to the amount of flow that could not be pushed through the arc. The process continues along the path till the next bottleneck arc is encountered

Recall that $G = (V, A)$, and A_f is the set of residual arcs with respect to a given pseudoflow f.

5.1 The Pseudoflow Algorithm

```
begin
  Initialize $\forall (s,j) \in A(s)$, $f(s,j) = c(s,j)$.  $\forall (j,t) \in A(t)$, $f(j,t) = c(j,t)$.
  For all arcs $(i,j) \in A$, $f(i,j) = 0$.
  $T = \cup_{j \in V}[r,j]$, the branches of the tree are $\{j\}_{j \in V}$.
  Nodes with positive excess are strong, $S$, and the rest are weak, $W$.
  while $(S,W) \cap A_f \neq \emptyset$ do
      Select $(s', w) \in (S, W)$
      Merge $T \leftarrow T \setminus [r, r_{s'}] \cup (s', w)$.
      Renormalize
      Push $\delta = M_{[r, r_{s'}]}$ units of flow along the path $[r_{s'}, \ldots, s', w, \ldots, r_w, r]$:
      begin
      Let $[v_i, v_{i+1}]$ be the next edge on the path.
      If $c_f(v_i, v_{i+1}) > \delta$ augment flow by $\delta$, $f(v_i, v_{i+1}) \leftarrow f(v_i, v_{i+1}) + \delta$.
      Else, split $\{(v_i, v_{i+1}), \delta - c_f(v_i, v_{i+1})\}$.
      Set $\delta \leftarrow c_f(v_i, v_{i+1})$.
      Set $f(v_i, v_{i+1}) \leftarrow c(v_i, v_{i+1})$; $i \leftarrow i + 1$
      end
    end
end
procedure split $\{(a,b), M\}$
$T \leftarrow T \setminus (a,b) \cup (r,a)$; $M_{(r,a)} = M$.
The branch $T_a$ is strong or 0-deficit with excess $M$.
$A_f \leftarrow A_f \cup \{(b,a)\} \setminus \{(a,b)\}$.
end
```

The push step, in which we augment flow by δ if $c_f(v_i, v_{i+1}) > \delta$, can be replaced by augmentation if $c_f(v_i, v_{i+1}) \geq \delta$. The algorithm remains correct, but the set of strong nodes is no longer minimal among maximum s-excess sets, and will not satisfy Property 4. We prove in the expanded version of this paper that the tree maintained is indeed normalized, which establishes the algorithm's correctness.

5.2 Initialization

We choose an initial normalized tree with each node as a separate branch for which the node serves as root. The corresponding pseudoflow saturates all arcs adjacent to source and to sink. Thus all nodes adjacent to source are strong nodes, and all those adjacent to sink are weak nodes. All the remaining nodes have zero inflow and outflow, and are thus of 0 deficit and set as weak. If a node is adjacent to both source and sink, then the lower capacity arc among the two is removed, and the other has that value subtracted from it. Therefore each node is uniquely identified with being adjacent to either source or sink or to neither.

5.3 Termination

The algorithm terminates when there is no residual arc between any strong and weak nodes.

From Corollary 6 we conclude that at termination the set of strong nodes is a *minimal* source set of a minimum cut. In other words, any proper subset of the set of strong nodes cannot be a source set of a minimum cut. It will be necessary to identify additional optimal solutions, and in particular a minimum cut with *maximal* source set for the parametric analysis.

To that end, we identify the 0-deficit branches among the weak branches. The set of strong branches can be appended with any collection of 0-deficit branches without residual arcs to weak nodes for an alternative optimal solution. The collection of *all* such 0-deficit branches with the strong nodes forms the sink set of a minimum cut that is *maximal.* To see that, consider an analogue of Corollary 6 that demonstrates that no proper subset of a weak branch (of negative deficit) can be in a source set of a minimum cut.

5.4 About Normalized Trees and Feasible Flows

Given any tree in the extended network, and a specification of pseudoflow values on the out of tree arcs, it is possible to determine in linear time $O(n)$ whether the tree is normalized. If the tree is not normalized, then the process is used to derive a normalized tree which consists of a subset of the arcs of the given tree. Given a normalized tree it is possible to derive in $O(n)$ time the values of the pseudoflow on the tree arcs, and in time $O(mn)$ to derive an associated feasible flow. At termination, that feasible flow is a maximum flow.

5.5 Variants of Pseudoflow Algorithm and and their Complexity

Implementing the pseudoflow algorithm in its generic form results in complexity of $O(nM^+)$ iterations, for $M^+ = C(\{s\}, V)$ – the sum of all capacities of arcs adjacent to source. This running time is not polynomial.

A natural variant to apply is capacity scaling. Capacity scaling is implemented in a straightforward way with running time of $O(mn \log M)$, where M is the largest capacity of source-adjacent or sink-adjacent arcs. This running time is polynomial but not strongly polynomial.

Our strongly polynomial variant relies on the lowest label selection rule of a merger arc. With this selection rule, our algorithm runs in strongly polynomial time, $O(mn \log n)$.

The lowest label selection rule is described recursively. Initially all nodes are assigned the label 1, $\ell_v = 1$ for all $v \in V$. The arc (s', w) selected is such that w is a lowest label weak node among all possible active arcs.

Upon a merger using the arc (s', w) the label of the strong node s' becomes the label of w plus 1 and all nodes of the strong branch with labels smaller than that of s' are updated to be equal to the label of s': formally, $\ell_{s'} \leftarrow \ell_w + 1$ and for all nodes v in the same branch with s', $\ell_v \leftarrow \max\{\ell_v, \ell_{s'}\}$.

The lowest label rule guarantees a bound of mn on the total number of iterations. In the *phase implementation* all weak nodes of the same label are processed in one phase. The phase implementation runs in time $O(mn \log n)$.

The lowest label implementation of the algorithm is particulary suitable for parameteric implementation. The algorithm has features that make it especially easy to adjust to changes in capacities. The common type of analysis finding all breakpoints in parametric capacities of arcs adjacent to source and sink that are linear functions of the parameter λ can also be implemented in $O(mn \log n)$.

6 Pseudoflow-Based Simplex

The simplex algorithm adapted to the s-excess problem maintains a pseudoflow with all source and sink adjacent arcs saturated. At termination, the optimal solution delivered by this s-excess version of simplex identifies a minimum cut. The solution is optimal when only source adjacent nodes have positive excess and only sink adjacent nodes have deficits. Additional running time is required to reconstruct the feasible flows on the given tree, that constitute maximum flow.

We show in the full version of the paper that the concept of a strong basis, introduced by Cunningham [C76], is a tree in the extended network satisfying Properties 2, 3 and 5.

6.1 Pseudoflow-Based Simplex Iteration

An entering arc is a merger arc. It completes a cycle in the residual graph. It is thus an arc between two branches. We include an auxiliary arc from sink to source, thus the merger arc completes a cycle. Alternatively we shrink source and sink into a single node r as before.

Nodes that are on the source side of the tree are referred to as *strong*, and those that are on the sink side, as *weak*, with the notation of S and W respectively. Let an *entering arc* with positive residual capacity be (s', w). The cycle created is $[r, r_{s'}, \ldots, s', w, \ldots, r_w, r]$.

The largest amount of the flow that can be augmented along the cycle is the bottleneck residual capacity along the cycle. The first arc attaining this bottleneck capacity is the *leaving arc*.

In the Simpelx the amount of flow pushed is determined by the bottleneck capacity. In the pseudoflow algorithm the entire excess is pushed even though it may be blocked by one or more arcs that have insufficient residual capacity.

The use of the lowest label selection rule in the pseudoflow-based simplex algorithm for the choice of an entering arc leads to precisely the same complexity as that of our pseudoflow algorithm, $O(mn \log n)$.

6.2 Parametric Analysis for Pseudoflow-Based Simplex

Given a series of ℓ parameter values for λ, $\{\lambda_1, \ldots, \lambda_\ell\}$. Let the source adjacent arcs capacities and the sink adjacent arc capacities be a linear function of λ with the source adjacent capacities monotone nondecreasing with λ and the sink adjacent capacities monotone nonincreasing with λ. Recently Goldfarb and Chen [GC96] presented a dual simplex method with running time $O(mn^2)$. This method is adaptable to use for sensitivity analysis for such a sequence of ℓ parameter values, in the same amount of time as a single run, $O(mn^2 + n\ell)$. The algorithm, however, does not generate all parameter breakpoints in a complete parametric analysis. Our algorithm is the first simplex algorithm that does generate all parameter breakpoints.

The parametric analysis process is not described here. We only mention that in order to implement the complete parametric analysis we must recover from the simplex solution the minimal and maximal source sets minimum cuts. To do that, we scan the tree at the end of each computation for one parameter value, and separate 0-deficit branches. This process is equivalent to the normalization of a tree. It adds only linear time to the running time and may be viewd as *basis adjustment*. The running time is linear in the number of nodes in the currently computed graph.

The overall running time of the procedure is identical to that of the pseudoflow algorithm with lowest label, $O(mn \log n + n\ell)$.

6.3 Comparing Simplex to Pseudoflow

Although the simplex implementation in the extended network has the same complexity as that of the pseudoflow algorithm, the two algorithms are not the same. Starting from the same normalized tree a simplex iteration will produce different trees in the following iteration. The pseudoflow algorithm tends to produce trees that are shallower than those produced by simplex thus reducing the average work per iteration (which depends on the length of the path from the root to the merger node). Other details on the similarities and differences between simplex and the pseudoflow algorithm are provided in the full version of the paper.

References

[C76] W. H. Cunningham. A network simplex method. *Mathematical Programming* , 1 (1976), 105-116.

[GGT89] G. Gallo, M. D. Grigoriadis and R. E. Tarjan. A fast parametric maximum flow algorithm and applications. *SIAM Journal of Computing*, Vol. 18, No. 1 (1989) 30-55.

[GT86] A. V. Goldberg and R. E. Tarjan. A New Approach to the Maximum Flow Problem. *J. Assoc. Comput. Mach.*, 35 (1988), 921-940.

[GC96] D. Goldfarb and W. Chen. On strongly polynomial dual algorithms for the Maximum flow problem. To appear, Special issue of *Mathematical Programming B*, (1996).

[GH90] D. Goldfarb and J. Hao. A primal simplex method that solves the Maximum flow problem in at most nm pivots and $O(n^2m)$ time. *Mathematical Programming* , 47 (1990), 353-365.

[Hoc96] D. S. Hochbaum. A new - old algorithm for minimum cut on closure graphs. Manuscript, June (1996).

[LG64] H. Lerchs, I. F. Grossmann. Optimum Design of Open-Pit Mines. *Transactions, C.I.M.*, Vol. LXVIII (1965) 17-24.

[Pic76] J. C. Picard. Maximal Closure of a Graph and Applications to Combinatorial Problems. *Management Science*, 22 (1976), 1268-1272.