Strongly polynomial primal monotonic build-up simplex algorithm for maximal flow problems

Tibor Illés[a], Richárd Molnár-Szipai[b]

[a]Budapest University of Technology and Economics, illes@math.bme.hu
[b]Budapest University of Technology and Economics, mricsi@math.bme.hu, *corresponding author*

Abstract: The maximum flow problem (MFP) is a fundamental model in operations research. The network simplex algorithm is one of the most efficient solution methods for MFP in practice. The theoretical properties of established pivot algorithms for MFP are less understood. Variants of the primal simplex and dual simplex methods for MFP have been proven strongly polynomial, but no similar result exists for other pivot algorithms like the monotonic build-up or the criss-cross simplex algorithm.

The monotonic build-up simplex algorithm (MBU SA) starts with a feasible solution, and fixes the dual feasibility one variable at a time, temporarily losing primal feasibility. In the case of maximum flow problems, pivots in one such iteration are all dual degenerate, bar the last one. Using a labelling technique to break these ties we show a variant that solves the maximum flow problem in $2|V||E|^2$ pivots.

Keywords: maximum flow problem; pivot algorithm; MBU algorithm

1. Introduction

The maximal flow problem is one of the basic models in network optimization that has a wide range of applications (see e.g. [1]). It is not surprising then, that it has been studied extensively, with numerous solution methods developed.

The first substantial results are due to Ford and Fulkerson [8], including the maximum flow – minimum cut theorem, and the idea of augmenting path algorithms. Later Edmonds and Karp [7], and independently Dinic [6] proved that the shortest augmenting path algorithm is strongly polynomial.

Another family of algorithms use so-called preflows. A preflow is a flow except that each intermediate node are allowed to have more inflow than outflow (but not the other way around). The first such algorithm by Karzanov [14] used preflows only to solve the subproblems in Dinic's algorithm, so the conservation equations are restored at the end of each phase. Later preflow algorithms took a more holistic approach, where the preflow becomes a flow only at the final step (see e.g. Goldberg and Tarjan [9]). This phenomenon is similar to how primal feasibility is restored only at the last pivot of a dual simplex algorithm, while the augmenting path algorithms are more like the primal simplex algorithm in that they proceed through feasible solutions.

In fact, the maximum flow problem is a special linear programming problem, therefore it can be solved using pivot algorithms. Indeed, the description of the network simplex algorithm appears as far back as Dantzig's book on linear programming [5]. The first published strongly polynomial pivot algorithm for a network optimization problem is a dual simplex algorithm for the minimum cost flow problem due to Orlin [15], while the first primal simplex variant, specifically for the maximum flow problem, is by Goldfarb and Hao [10, 11]. Later Orlin showed that a strongly polynomial variant of the primal simplex algorithm for minimum cost flow problems also exists [16]. Pivot algorithms that traverse bases that are neither primal nor dual feasible, however, received less attention, no such algorithm has been proven strongly polynomial so far. We show that the primal monotonic build-up simplex algorithm [2] has a strongly polynomial variant for the maximum flow problem.

A further generalisation of the preflow concept was given by Hochbaum [12], where the so-called pseudoflow can have unbalanced intermediate nodes even with the outflow being greater than the inflow. The algorithm solves the maximum blocking cut problem first, which provides a minimum cut, and then recovers a maximum flow. After each iteration a feasible flow could be recovered using a process analogous to flow decomposition, giving the algorithm a primal character. In fact, Section 10 describes a simplex variant of the algorithm.

In order to use the primal MBU SA (or a primal simplex variant), one needs a primal feasible basic solution to start with. If the lower bounds are nonzero, finding such a starting solution becomes nontrivial. A classic technique consists of slightly expanding the network, solving a (zero lower bound) maximum flow problem, and then converting the solution back to a feasible flow of the original problem (if such a flow exists). This is similar to solving the first phase problem in linear programming. Another approach is to start with an arbitrary basic solution, and use a suitable pivot algorithm to reach a feasible basic solution. One such pivot algorithm is the feasibility MBU SA, which can be thought of as a dual MBU SA with the objective function of constant zero. The authors showed that the feasibility MBU SA has a variant that finds a feasible basic solution in strongly polynomial time in [13].

The structure of the article is as follows: in the next section we describe the MBU SA for linear programming in general and how it works on maximum flow problems in particular, then we describe our variant in detail. In the third section we describe the generic pseudoflow algorithm [12], place it in a pivot framework and compare it with the MBU SA. In the fourth section we prove the polynomiality of our MBU SA variant, and then conclude the article with some remarks and possible future research directions.

2. Preliminaries

The reader is expected to have a basic understanding of the simplex method of linear programming (e.g. basic solutions, primal and dual feasibility), and of the maximum flow problem. For reference, see [1, 5, 8].

Consider a linear programming problem in the following form:

$$\begin{aligned} \max \mathbf{c}^T\mathbf{x} \\ A\mathbf{x} = \mathbf{b} \\ \mathbf{x} \geq \mathbf{0} \end{aligned}$$

Where $A \in \mathbb{R}^{n\times m}$, $\mathbf{x}, \mathbf{c} \in \mathbb{R}^m$, $\mathbf{b} \in \mathbb{R}^n$. Without loss of generality we may assume that A has full row rank. Let B be an $n \times n$ invertible submatrix of A, I_B the set of column indices of B, and I_N the remaining column indices. Then the solution $\mathbf{x}_B = B^{-1}\mathbf{b}$, $\mathbf{x}_N = \mathbf{0}$ is a so-called basic solution, with the corresponding simplex tableau:

$$\begin{array}{|c|c|} \hline \overline{A} & \overline{\mathbf{b}} \\ \hline \overline{\mathbf{c}}^T & -z_0 \\ \hline \end{array} \quad := \quad \begin{array}{|c|c|} \hline B^{-1}A & B^{-1}\mathbf{b} \\ \hline \mathbf{c}^T - \mathbf{c}_B^T B^{-1}A & -\mathbf{c}_B^T B^{-1}\mathbf{b} \\ \hline \end{array}$$

Using the notations $\overline{a}_{i,j}$, $\overline{b}_i$ and $\overline{c}_j$ for the elements of the current tableau, the *primal MBU SA* is as follows:

1. Start with a primal feasible basic solution
2. Choose a dual infeasible variable x_{p^*}. We refer to x_{p^*} as the driving variable [2]. If there are none, stop, we have an optimal solution.
3. Select the leaving variable using a minimum ratio test on feasible basic variables:
$$q = \arg\min \left\{ \frac{\overline{b}_q}{\overline{a}_{q,p^*}} \ : \ \overline{a}_{q,p^*} > 0, \overline{b}_q \geq 0 \right\}$$
Let $\vartheta_1 = |\overline{c}_{p^*}|/\overline{a}_{q,p^*}$.
4. Choose the entering variable using a minimum ratio test on dual feasible nonbasic variables:
$$p = \arg\min \left\{ \frac{\overline{c}_p}{|\overline{a}_{q,p}|} \ : \ \overline{c}_p \geq 0, \overline{a}_{q,p} < 0 \right\}$$
Let $\vartheta_2 = \overline{c}_p/|\overline{a}_{q,p}|$.
5. Variable x_q leaves the basis. If $\vartheta_2 < \vartheta_1$, then x_p enters the basis and go to step 3, otherwise x_{p^*} enters the basis, and go to step 2.

The name of the algorithm comes from the property that dual feasible variables do not lose their feasibility (due to step 4), and thus the set of these variables monotonically build up. The fact that we might select the variable x_p instead of x_{p^*} on whose column we took the minimum ratio test means that we can lose primal feasibility. However, when x_{p^*} finally enters the basis, primal feasibility is restored (see [2] for details).

Let $G = (V, E)$ be a connected directed graph with two distinguished nodes s and t, the source and the sink, respectively. Given lower and upper bounds on the arcs and obeying conservation of flow at intermediate nodes, we wish to maximize the amount of flow from s to t. Introducing an arc from t to s, the maximum flow problem can be stated as the following linear program:

$$
\begin{aligned}
& \max x_{t,s} \\
\forall v \in V : \quad & \sum_{(w,v)\in E} x_{w,v} - \sum_{(v,w)\in E} x_{v,w} = 0 \\
\forall e \in E : \quad & l_e \le x_e \le u_e
\end{aligned}
$$

Throughout the article we are using the following notations:

- v, w and z for nodes of a graph
- e for an arc of a graph
- p and q for the entering and leaving arc of a graph
- $p^* = (g, h)$ as the arc corresponding to the driving variable

Now let us break down how the primal monotonic build-up simplex algorithm works on maximum flow problems step by step.

Step 1: Start from a primal feasible basic solution. The basic variables correspond to the arcs of a spanning tree T containing (t, s), with the nonbasic arcs having flow values of either the lower or the upper bound. The basic variables are then uniquely determined by the conservation equations. If the lower bounds are zero, then $\mathbf{x} = \mathbf{0}$ is such a feasible basic solution with an arbitrary spanning tree. Otherwise finding such a starting solution is not trivial, one can do so by transforming the network and solving another (zero lower bounds) maximum flow problem (see e.g. [1] section 6.2), which corresponds to solving the first phase of a two phase linear programming problem. Another way is to use the feasibility MBU SA [13] (which is a specialization of [4]). Having zero or nonzero lower bounds do not affect the algorithm otherwise.

Step 2: Choose a dual infeasible variable x_e. Let $T^i \subset E$ be the spanning tree before the ith pivot. Dropping (t, s) from T^i disconnects the spanning tree, with one subset containing s, and the other one containing t. These vertex sets are denoted by S^i and Z^i respectively. The reduced cost $\overline{\mathbf{c}}_e$ of a nonbasic arc e is:

$$
\overline{\mathbf{c}}_e = \begin{cases} 1 & \text{if } e : S^i \to Z^i \text{ and } x_e = u_e \text{ or } e : Z^i \to S^i \text{ and } x_e = l_e \\ -1 & \text{if } e : S^i \to Z^i \text{ and } x_e = l_e \text{ or } e : Z^i \to S^i \text{ and } x_e = u_e \\ 0 & \text{otherwise: } e : S^i \to S^i \text{ or } e : Z^i \to Z^i \end{cases}
$$

Without loss of generality we can assume that the *driving variable* $p^* = (g, h) \in E$ is on its lower bound, i.e. $g \in S^i$ and $h \in Z^i$.

Step 3: Choose the leaving variable with a primal ratio test. $T^i \cup (g, h)$ contains a unique cycle C^i, consider it directed according to (g, h). Then the leaving arc $q \in C^i$ is an arc where

$$
\delta = \min \left\{ \begin{array}{ll} u_q - x_q : & \text{q is forward, } x_q \le u_q \text{; or} \\ x_q - l_q : & \text{q is backward, } x_q \ge l_q \end{array} \right\}
$$

takes its value. (We are examining how much we could augment along the cycle C_i, not counting arcs that are already infeasible in the appropriate direction.) Note that such an arc q is uniquely determined if the bounds are sufficiently diverse, so we will choose arbitrarily, should a tie occur.

Step 4: Choose the entering variable with a dual minimum ratio test. The simplex tableau, along with the reduced costs is totally unimodular, so this ratio test can result in either a 0, or a 1 quotient. As $\vartheta_1 = 1$ the only instance when we do not let p^* enter the basis is if we find an arc p with ratio 0, i.e. with reduced cost 0, and $a_{q,p} = -1$. As we've seen, $\overline{\mathbf{c}}_p = 0$ means that p is either $S^i \to S^i$ or $Z^i \to Z^i$, and $\vartheta_2 < \vartheta_1$ means that performing the pivot (g, h) for q would make p dual infeasible. In the case of $q \in S^i$, dropping q from the tree would disconnect s and g, suitable entering variables would either be arcs from the subtree of s to the subtree of g on their lower bounds, or arcs the other way around on their upper bounds.

The general properties of the MBU SA carry over to this more special problem, that is no dual feasible variable ever becomes dual infeasible, and while primal feasibility can be temporarily lost during the phase of a driving variable, it is restored at the end of such a phase when the driving variable enters the basis.

These properties are easier to see on maximum flow problems, however. The fact that no dual infeasible variables are created are clear from Step 4. The restoration of primal feasibility is a consequence of the following lemma, which follows the structure of Theorem 1. in [2].

Lemma 1. *Assume that a MFP is solved by the MBU SA, $p^* = (g,h)$ is the current driving variable with $g \in S$ and $h \in Z$. Then the following statements hold for all pivot index $i \geq 0$, while p^* does not enter the basis (δ^0 is interpreted as 0):*

a) all primal infeasible arcs after the ith pivot are on the $s \to g$ and the $h \to t$ paths,

b) the arcs in the direction of the paths can only violate their lower bound, the backward arcs only their upper bounds, both by at most δ^i,

c) the next pivot will have $\delta^{i+1} \geq \delta^i$.

When p^* enters the basis, primal feasibility is restored.

Proof: For $i = 0$ the statements hold, as there are no primal infeasible arcs before the first pivot, $\delta^0 = 0$, and $\delta^1 \geq 0$ will be true by definition.

For $i \geq 1$ we use induction. Assume that the lemma holds for the pivots up to $i-1$. Without loss of generality we might assume that the leaving arc q, and therefore the entering arc p too, is in S. The cycle resulting from adding p has a common segment with the $s \to g$ path (containing q), let us denote its endpoints by a and b, see Figure 1. After the pivot this $a \to b$ segment will be replaced by the other half of the cycle (containing p) in the $s \to g$ path. The flow value on the old $a \to b$ segment (containing q) is increased by δ^i, the flow value on the new $a \to b$ segment (containing p) is decreased by δ^i.

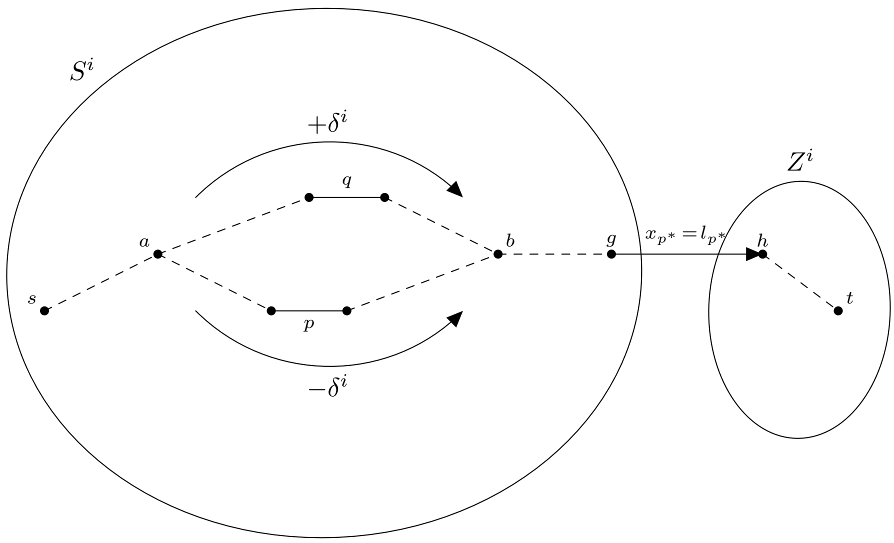


Figure 1: The structure of pivot i.

According to the induction hypothesis, the primal infeasible arcs before pivot i are on the $s \to g$ path, with infeasibility at most δ^{i-1} and $\delta^i \geq \delta^{i-1}$. Therefore increasing the flow value on the old $a \to b$ segment restores the primal feasibility of those arcs, and so statement a) holds.

The arcs on the new $a \to b$ segment were feasible before pivot i by the induction hypothesis, and the flow value was decreased by δ^i on the segment. This way the arcs in the $a \to b$ direction (which is the same as the direction of the new $s \to g$ path) could go below their lower bounds by at most δ^i, and the arcs in the opposite direction can surpass their upper bounds by at most δ^i. Thus statement b) holds.

Finally, as the arcs on the new $a \to b$ segment were decreased by δ^i from a feasible position, they can be increased by at least that much in the next pivot. The flow values of the arcs on the $s \to a$, $b \to g$ and $h \to t$ paths were not changed, but they could have been increased by at least δ^i by the calculation of δ before pivot i. Therefore after the pivot all arcs on the new $s \to g$ and $h \to t$ paths can be increased by at least δ^i, and so $\delta^{i+1} \geq \delta^i$ will hold too.

If p^* enters the basis in the $k+1$st pivot, then the previous statements hold for pivot k: the infeasible arcs are all on the $s \to g$ and $h \to t$ paths, their infeasibility is at most δ^k, their direction is such that increasing the flow along the paths decrease their infeasibility, and due to $\delta^{k+1} \geq \delta^k$ the $k+1$st pivot will make them feasible again. ■

Note that by the above lemma primal feasibility can be restored at any pivot by letting the driving variable enter the basis, however, new dual infeasible arcs appear if there were candidates for p. This is consistent with the behaviour of the primal MBU SA on linear programming problems.

To describe our variant we will consider the subtrees S^i and Z^i to be rooted at g and h respectively. Picturing the tree with the root at the top, and using the usual notions of "parent" and "child" node, T^i_v will denote the subtree in T^i spanning the node v and its descendants. Using this image, we will also refer to the nodes of a basic arc as the "upper" and "lower" nodes. For a basic arc $q \in T^i$ we will use the notation T^i_q for the subtree "below" q, that is, T^i_v, where v is the lower node of q (we will use this notation only in the context of the leaving variable q).

We will use the concept of a *pseudo-augmenting path* (PAP) as defined in [11]: a pseudo-augmenting path from v to w with respect to a basic solution $\mathbf{x}$, and spanning tree T^i is a directed path from v to w that can use (v_1, v_2) if

- $(v_1, v_2) \in E \backslash T^i$ and $x_{v_1,v_2} = l_{v_1,v_2}$,
- or $(v_2, v_1) \in E \backslash T^i$ and $x_{v_2,v_1} = u_{v_2,v_1}$,
- or if either $(v_1, v_2) \in T^i$ or $(v_2, v_1) \in T^i$.

Note that nonbasic arcs are used the same way as with classical augmenting path algorithms, the addition of using basic arcs in any direction makes the notion compatible with the basis structure.

The *label* $d^i(v)$ of vertex v before pivot i with respect to the current driving variable (g, h) and basis structure T^i is the length of the shortest PAP from h to v within T^i_h if $v \in T^i_h$, and the length of the shortest PAP from v to g within T^i_g if $v \in T^i_g$.

Using these labels to choose the entering variable results in the following variant of the *primal monotonic build-up simplex algorithm*:

Algorithm 1 Primal monotonic build-up simplex algorithm with labelling

Start with a primal feasible basic solution $\mathbf{x}$.
while $\mathbf{x}$ is not optimal **do**
  Let $p^* = (g, h)$ be an arbitrary dual infeasible arc.
  while p^* is dual infeasible **do**
    Let q be an arc limiting further augmentation along the cycle in $T^i \cup (g, h)$.
    if there is a possible entering arc between T^i_q and the rest of T^i_g or T^i_h (whichever q is in) **then**
      Let p be such an arc with minimal label in $E \backslash T^i_q$.
      Perform a pivot with p entering and q leaving the basis.
    else
      Perform a pivot with p^* entering the basis and q leaving.
    end if
  end while
end while

3. Comparison with the pseudoflow algorithm of Hochbaum

Let us first extend our network. Recall that $G = (V, E)$ is a directed graph with source s and sink t, to which we added an arc (t, s). For each node v, two arcs of infinite capacity (and 0 lower bound) are added: (t, v) and (v, s). These arcs are referred to as *deficit arcs* and *excess arcs* respectively. Finally, the nodes s and t are shrunk into a single node r, referred to as the *root*. This extended network will be denoted by G^{EXT}.

The pseudoflow algorithm aims to maximize the flow value, while maintaining a pseudoflow at every step, that is flow values that satisfy the capacity constraints, but might violate the flow balance constraints. Such a pseudoflow has a corresponding feasible flow on the graph G^{EXT}, as a node v with deficit can be balanced by letting flow in via the (r, v) arc, and excess can be drained using the excess arc (v, r).

The pseudoflow algorithm maintains a construction called a *normalized tree*, which is a spanning tree T in G^{EXT} with root r. The children of r are denoted by r_i and called the *roots* of their respective *branches*. Then a normalized tree must satisfy the following properties with respect to the current pseudoflow x:

1. all arcs (s, v) and (v, t) in the original network are saturated ($x_e = u_e$ for such e arcs),
2. x is equal to the lower or upper bounds on out-of-tree arcs.
3. In every branch, all downwards residual capacities are strictly positive.
4. Only the roots r_i might not satisfy flow balance constraints in the original network.

Note that property 4 is a consequence of property 2, as an unbalanced node v must have positive flow value on either its deficit or excess arc, which arc (due to property 2) must be in T, and thus v is a neighbor of r in the tree.

To describe the algorithm we introduce further notions. A branch of the current normalized tree T is called *strong*, if its root r_i has positive excess, and is called *weak* otherwise (including the case that r_i is balanced). The collection of strong branches will be denoted by S, the weak branches by W. Also, we denote the excess of a node v in the original network by $e(v)$ (the sum of incoming flow minus the sum of outgoing flow).

The algorithm starts with an initial normalized tree and pseudoflow, and looks for arcs between S and W with residual capacity towards W. After pushing flow from the root of the strong branch towards the root of the weak branch, a restructuring is performed to maintain the normalized tree structure.

Algorithm 2 Generic pseudoflow algorithm

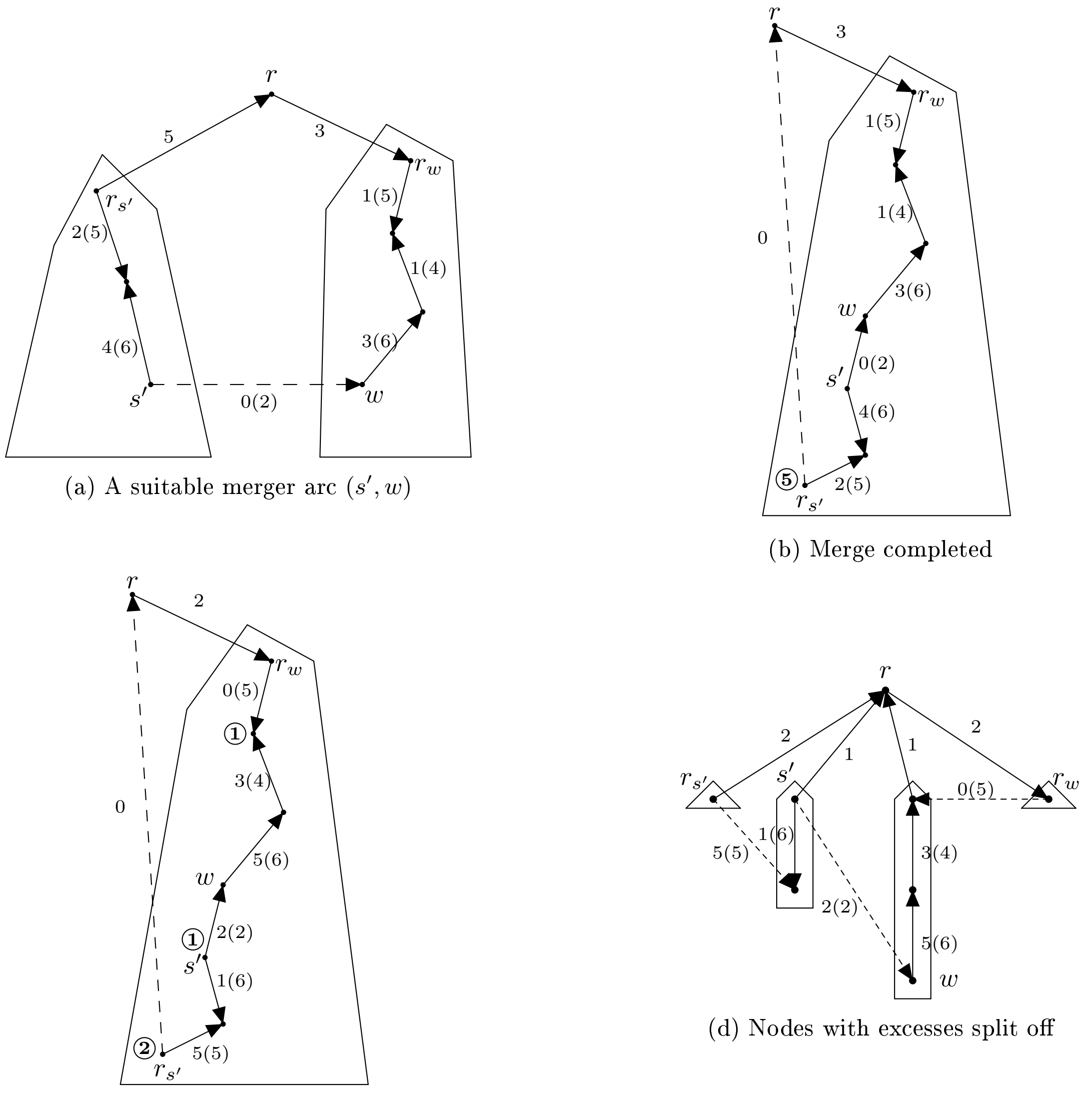
Let T be an initial normalized tree with initial pseudoflow $\mathbf{x}$.
while there are arcs with residual capacity in (S, W) **do**
  Select $(s', w) \in (S, W)$ with residual capacity.
  Let $r_{s'}$ and r_w be the roots of the branches containing s' and w respectively.
  Let $\delta = e(r_{s'}) = \mathbf{x}_{r_{s'},r}$.
  Merge $\{(s', w)\}$:
  Let $T = T \cup (s', w) \backslash (r_{s'}, r)$.
  Let $\mathbf{x}_{r_{s'},r} = 0$.
  Renormalize:
  Let $(v_1, \ldots, v_k) = (r_{s'}, \ldots, s', w, \ldots, r_w, r)$.
  Let $i = 1$.
  repeat
    if the residual capacity $c^r_{v_i,v_{i+1}}$ of (v_i, v_{i+1}) is at least δ **then**
      Augment flow on (v_i, v_{i+1}) by δ
    else
      Split $\{(v_i, v_{i+1}), \delta - c^r_{v_i,v_{i+1}}\}$:
      Let $T = T \cup (v_i, r) \backslash (v_i, v_{i+1})$.
      Let $\mathbf{x}_{v_i,r} = \delta - c^r_{v_i,v_{i+1}}$.
      Let $\delta = c^r_{v_i,v_{i+1}}$.
      Augment (v_i, v_{i+1}) by δ.
    end if
    Let $i = i + 1$
  until $v_{i+1} = r$
end while

The algorithm can be easier understood through an example. In Figure 2 we can follow an iteration of the algorithm. The numbers on arcs show the current flow value, with the upper bound in parentheses (lower bounds are zero), while the encircled numbers show the excess of a node. Subfigure 2a shows a suitable arc (s', w) from a strong branch to a weak branch with residual capacity. Subfigure 2b shows the merging of the two branches, with $\mathbf{x}_{r_{s'},r} = 0$, and 5 excess at $r_{s'}$. In Subfigure 2c we pushed as much flow from $r_{s'}$ upwards as the bounds allowed, creating nodes with positive excesses. Subfigure 2d shows how the splitting procedure turns nodes with positive excesses into roots of new strong branches.

When the algorithm terminates, the cut between the strong and weak branches can be showed to be a minimum cut, and a maximum flow can be recovered using a procedure based on flow decomposition (see [12] section 8).

The algorithm is very similar to a pivot algorithm, as the pseudoflow after every iteration can be turned into a basic feasible flow, using the procedure mentioned above. These basic solutions are, however, not neighboring in the sense that getting from one such basic solution to the next one is not possible with a single pivot.

In fact, there is a simplex variant described in [12] section 10, that after the merging operation calculates the minimum residual capacity along the $[r_{s'}, \ldots, s', w, \ldots, r_w, r]$ path, pushes only this much upwards, and lets the first arc on the path where this bottleneck capacity is attained leave the tree. It can be showed that the normalized tree structure is retained, making the algorithm a primal simplex algorithm on the original graph.



(a) A suitable merger arc (s', w)

(b) Merge completed

(c) Flow pushed upwards

(d) Nodes with excesses split off

Figure 2: An iteration of the pseudoflow algorithm

However, Algorithm 2 can be also described as a pivot algorithm, where a single iteration consists of multiple pivots: one pivot for the merge operation, and one pivot for each split operation. The "Merge $\{(s', w)\}$" operation has (s', w) entering and $(r_{s'}, r)$ leaving the basis with the whole $\delta = e(r_{s'}) = \mathbf{x}_{r_{s'},r}$ flow pushed around the cycle, making some basic arcs infeasible. Then the split operations recover a feasible flow by taking the first infeasible arc (v_i, v_{i+1}) on the path, making it leave the basis with (v_i, r) entering, and the infeasibility *pushed back* around this cycle, and repeating this process until no infeasible arcs remain.

This way Algorithm 2 is also a pivot algorithm traversing bases that are neither primal nor dual feasible. The entering arc is dual infeasible, but the leaving arc is not chosen according to a primal ratio test, therefore primal feasibility is lost. Then a structured sequence of pivots making primal infeasible arcs leave the basis, and the artificial (v_i, r) arcs entering it restores feasibility. The advantage of this method is that a larger augmentation can be performed in the merger pivot than with the ratio test.

Comparatively, Algorithm 1 chooses a dual infeasible variable, but does not let it enter until it is safe to do so, in the sense that no new dual infeasible arcs are created. The price of this "safety" is that the pivots leading up to the driving variable finally entering the basis are dual degenerate, and so the objective function stagnates in the meantime.

4. Proving Strong Polynomiality

First, we need to note that it is sufficient to establish the polynomiality of making a single (g, h) dual infeasible arc feasible. This statement relies on the primal MBU SA having the property that a dual feasible variable never becomes infeasible during the algorithm, and so the number of outer cycles is bounded by the number of dual infeasible variables in the initial basic solution.

To bound the number of pivots needed to get the driving variable into the basis, we use a few features of the labelling technique. First, we prove that the label of every node is monotonically non-decreasing ("monotonicity lemma"). This tells us that the algorithm is progressing in a certain sense. The lemma's appropriate version appears in the proofs of polynomiality for both the primal and the dual simplex variant ([11] and [3]).

As the label of a node represents the length of a shortest path, it is bounded by the number of vertices. After the monotonicity lemma we show that not only the labels do not decrease, but strict increases must happen regularly. Thus we will be able to derive an upper bound on the number of pivots.

The structure is based on Lemma 3. of [3]. We show in Lemma 5 ("main lemma") that for every arc $p = (p_v, p_w)$ the sum of its labels $d(p_v) + d(p_w)$ must increase between subsequent enterings into the basis. This is split into two cases according to whether p leaves the basis having the same direction that it had when entering or not, their proofs aided by Lemmas 3 and 4 respectively. Due to S^i and Z^i not behaving like a dual feasible basis in [3], these lemmas are somewhat weaker, and their proofs somewhat more convoluted.

Lemma 2 (Monotonicity lemma). *Assume that a MFP is solved by Algorithm 1. For any $v \in V$ and iteration i during a single outer cycle $d^{i+1}(v) \geq d^i(v)$ holds.*

Proof: Assume indirectly that there exists z and i such that $d^{i+1}(z) < d^i(z)$. We can assume that z is a counterexample with minimal $d^{i+1}(z)$ label.

As $d^{i+1}(z) < d^i(z)$, the shortest PAP from z to g after the ith pivot must use an arc in a direction that was not available before. Let us take a look at how the arcs change with respect to labelling:

- arcs that are in the basis both before and after the pivot ($T^i_g \cap T^{i+1}_g$) can be used in both directions for calculating $d^i(z)$ and $d^{i+1}(z)$,
- arcs not in either basis ($E \backslash (T^i_g \cup T^{i+1}_g)$) didn't have their flow value changed, so they can be used in the same directions both before and after the pivot,
- the entering arc p was usable in one direction before entering the basis, and is usable in both directions after the pivot,
- conversely, the leaving arc q was usable in both directions before the pivot, and in only one direction after it.

Let $v \in T^i_q$ and $w \notin T^i_q$ be the two vertices of p, the only new possibility for labelling after the pivot is using p in the $w \to v$ direction, so the new shortest PAP from z must use that.

As $v \in T^i_q$, this PAP must leave T^i_q after using p via some p' arc, with $v' \in T^i_q$ and $w' \notin T^i_q$ its two vertices. Note that $p' \neq q$, because after leaving the basis q can only be used from its vertex not in T^i_q. Therefore $p' \notin T^i$, and could have been used to leave T^i_q, so it was a candidate for entering the basis at the ith pivot. However, we chose p over p', so $d^i(w) \leq d^i(w')$ must hold.

Therefore the shortest PAP from z to g first uses the shortest PAP from z to w, uses (w, v), uses the shortest PAP from v to v', uses (v', w'), and finally the shortest PAP from w' to z. Denoting the shortest PAP from v_1 to v_2 before pivot k by $d^k(v_1, v_2)$, $d^{i+1}(z)$ can be written as:

$$\begin{aligned} d^{i+1}(z) &= d^{i+1}(z,w) + 1 + d^{i+1}(v,v') + 1 + d^{i+1}(w') \\ &\geq d^i(z,w) + d^i(w') + 2 \geq d^i(z,w) + d^i(w) + 2 \geq d^i(z) + 2. \end{aligned}$$

Where we used:

- $d^{i+1}(z,w) \geq d^i(w,z)$, as the shortest PAP from z to w can not use p in the $w \to v$ direction.
- $d^{i+1}(v',v) \geq 0$.
- $d^{i+1}(w') \geq d^i(w')$, as otherwise w' would be a counterexample to the lemma, with $d^{i+1}(w') < d^{i+1}(z)$, contradicting the minimality of z.
- $d^i(w') \geq d^i(w)$ from the choice of p as the entering variable (see above).

- $d^i(w) + d^i(w,z) \geq d^i(z)$ is a triangle inequality for PAPs.

We assumed indirectly that $d^{i+1}(z) < d^i(z)$, but we concluded $d^{i+1}(z) \geq d^i(z) + 2$, a contradiction. ■

To proceed, we will prove an inequality that will help us in both cases of the main lemma.

Lemma 3 (Subtree lemma). *Assume that a MFP is solved by Algorithm 1. If (v,w) entered the basis at the ith pivot, with w being the upper vertex, and this remains true throughout, even after the jth pivot, then for all $z \in T_v^{j+1}$: $d^{j+1}(z) \geq d^i(w) + 1$.*

Proof: Case $j = i$. Note that $T_v^{i+1} = T_q^i$. Take a shortest reverse pseudo-augmenting path from g to z. As $z \in T_v^{i+1}$, this path must contain an arc leading into T_q^i. This arc can not be q, as it left the basis on the wrong bound for that.

If it is p, then $d^{i+1}(z) \geq d^{i+1}(w) + 1 \geq d^i(w) + 1$.

Otherwise that arc could have entered the basis at pivot i, but we chose p instead, so $d^i(w') \geq d^i(w)$ for its $w' \notin T_q^i$ vertex. Then $d^{i+1}(z) \geq d^{i+1}(w') + 1 \geq d^i(w') + 1 \geq d^i(w) + 1$.

This finishes the proof for $j = i$.

For $j > i$ we use induction, so let us assume that the lemma is true for $j - 1$, and let $z \in T_v^{j+1}$.

If $z \in T_v^j$ as well, then monotonicity and the induction hypothesis gives $d^{j+1}(z) \geq d^j(z) \geq d^i(w) + 1$.

Otherwise, z entered T_v during the jth pivot. Let the entering arc of that pivot be p with $p_v \in T_v^j$ and $p_w \notin T_v^j$ vertices. Then $d^{j+1}(z) \geq d^j(p_v) + 1$ using the $i = j$ case of this lemma for p, and $d^j(p_v) \geq d^i(w) + 1$ by the induction hypothesis, giving $d^{j+1}(z) \geq d^i(w) + 2$. This completes the proof. ■

The next lemma shows that if p changes direction since entering the basis, then a strict increase in his labels must already have happened.

Lemma 4 (Reversal lemma). *Assume that a MFP is solved by Algorithm 1. If (v,w) entered the basis at the ith pivot, with w being the upper vertex, this remains true throughout, but changes to v being upside with the jth pivot, then $d^{j+1}(w) \geq d^i(w) + 1$.*

Proof: We claim that the following inequalities hold:

$$d^{j+1}(w) \geq d^j(p_w) + 1 \geq d^j(p_v) \geq d^i(w) + 1$$

Let the entering arc at pivot j be p with $p_w \notin T_v^j$ and $p_v \in T_v^j$ vertices. The leaving arc q must be on the path in the spanning tree from w to g for (v,w) to change directions. This also means that $w \in T_{p_w}^{j+1}$, so by the subtree lemma we have $d^{j+1}(w) \geq d^j(p_w) + 1$.

As p was a candidate for entering, it could be used for labelling from the p_w end, which means $d^j(p_v) \leq d^j(p_w) + 1$.

Finally, $p_v \in T_v^j$, so using the subtree lemma we get $d^j(p_v) \geq d^i(w) + 1$. ■

Now we are ready to state our main lemma, describing the growth behavior of the labels.

Lemma 5 (Main lemma). *Assume that a MFP is solved by Algorithm 1. If (v,w) entered the basis at pivot i, left it at pivot j, and entered it again at pivot j, then $d^{k+1}(v) + d^{k+1}(w) \geq d^i(v) + d^i(w) + 2$ holds.*

Proof: Without loss of generality we might assume that w is the upper vertex of (v,w) in T_g^{i+1}.

Case a: w is the upper vertex in T_g^j as well. We claim that

$$d^{k+1}(v) + d^{k+1}(w) \geq 2d^k(v) + 1 \geq 2d^{i+1}(v) + 1 \geq d^i(v) + d^i(w) + 2$$

After leaving a base (v,w) can be used for labelling only from its v end, therefore v will be the upper vertex after pivot k. According to the subtree lemma $d^{k+1}(w) \geq d^k(v) + 1$, and using $d^{k+1}(v) \geq d^k(v)$ we get the first inequality.

The second inequality is the monotonicity lemma.

In the third inequality we bound one of the $d^{i+1}(v)$ with the subtree lemma: $d^{i+1}(v) \geq d^i(w) + 1$, and the other one with monotonicity: $d^{i+1}(v) \geq d^i(v)$.

Case b: v is the upper vertex in T_g^j. We explain

$$d^{k+1}(v) + d^{k+1}(w) \geq 2d^k(w) + 1 \geq 2d^{l+1}(w) + 1 \geq 2d^i(w) + 3 \geq d^i(v) + d^w + 2$$

where l is the first pivot when the direction of (v, w) changes in the spanning tree ($i < l < j < k$).

As v is the upper vertex when leaving the basis, w is the upper vertex after pivot k, and we get the first inequality by using the subtree lemma and monotonicity similar to the previous case.

The second inequality is the monotonicity lemma.

The third inequality is the reversal lemma: $d^{l+1}(w) \geq d^i(w) + 1$.

In the fourth inequality we use that before pivot i we could use (v, w) for labelling from the side of w, therefore $d^i(v) \leq d^i(w) + 1$. ■

Finally, we deduce the strong polinomiality of the algorithm from the previous lemma.

Theorem 1. *Algorithm 1 solves a MFP in at most $2nm^2$ pivots.*

Proof: The number of dual infeasible arcs at the start of the algorithm is less than m. As the primal MBU simplex algorithm does not create new dual infeasible arcs, the inner cycle can happen at most m times. Let us then examine the number of pivots it takes to "fix" an infeasible arc.

For any (v, w) arc we have $1 \leq d(v) + d(w) \leq 2n - 5$ (we have 2 vertices with label 0, so the maximum label is $n - 2$, and there can be at most one such vertex). By the monotonicity lemma $d(v) + d(w)$ is not decreasing, and by the previous lemma it increases by at least 2 if it enters the basis twice. Therefore (v, w) can enter the basis at most $2n - 5$ times. As every pivot has an entering arc, we can thus have at most $2nm$ pivots, even counting the final pivot that lets the dual infeasible arc enter the basis. ■

5. Conclusions and further directions

Building upon the techniques used for proving the polynomiality of certain variants of the primal and dual simplex algorithms [11, 3] on the maximum flow problem, we have shown that the primal MBU SA also has such a strongly polynomial variant. This variant has an interesting structure: the algorithm makes at most m dual nondegenerate steps, each two separated by at most $2nm$ dual degenerate steps. The corresponding flow becomes primal feasible after every nondegenerate step, but this property may not hold in between them.

It remains an open problem if similar results can be reached with other non-primal, non-dual pivot algorithms, such as the dual MBU SA, exterior point simplex algorithms [17], or criss-cross type algorithms [18]. Another interesting question is whether the results can be generalized to minimum cost flow problems.

Acknowledgements. We are grateful for the work of the two anonymous reviewers, and also for the suggestion of including lemma 1 in the article.

Tibor Illés acknowledges the research support obtained from Strathclyde University, Glasgow under the *John Anderson Research Leadership Program.*

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows.* Prentice Hall, New Jersey, 1993.

[2] K. M. Anstreicher and T. Terlaky. A monotonic build-up simplex algorithm for linear programming. *Operations Research*, 42:556–561, 1994.

[3] R. D. Armstrong, W. Chen, D. Goldfarb, and Z. Jin. Strongly polynomial dual simplex methods for the maximum flow problem. *Mathematical Programming*, 80:17–33, 1998.

[4] F. Bilen, Zs. Csizmadia, and T. Illés. Anstreicher–Terlaky type monotonic simplex algorithms for linear feasibility problems. *Optimisation Methods and Software*, 22(4):679–695, 2007.

[5] G. B. Dantzig. *Linear Programming and Extensions.* Princeton University Press, Princeton, NJ., 1963.

[6] E. A. Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Math. Doklady*, (11):1277–1280, 1970.

[7] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of ACM*, 19:248–264, 1972.

[8] L. R. Ford and D. R. Fulkerson. *Flows in Networks.* Princeton University Press, Princeton, NJ., 1962.

[9] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum flow problem. In *Proceedings of the 18th ACM Symposium on the Theory of Computing*, pages 136–146, 1986.

[10] D. Goldfarb and J. Hao. A primal simplex algorithm that solves the maximum flow problem in at most nm pivots and $\mathcal{O}(n^2m)$ time. *Mathematical Programming*, 47:353–365, 1990.

[11] D. Goldfarb and J. Hao. On strongly polynomial variants of the network simplex algorithm for the maximum flow problem. *Operations Research Letters*, 10:383–387, 1991.

[12] D. S. Hochbaum. The pseudoflow algorithm: A new algorithm for the maximum-flow problem. *Operations Research*, 56(4):992–1009, 2008.

[13] T. Illés and R. Molnár-Szipai. On strongly polynomial variants of the MBU-simplex algorithm for a maximum flow problem with non-zero lower bounds. *Optimization*, 63:39–47, 2014.

[14] A. V. Karzanov. Nakhozhdenie maksimal'nogo potoka v seti metodom predpotokov ("determining the maximal flow in a network by the method of preflows"). *Doklady Akademii Nauk SSSR*, 215(1):49–52, 1974.

[15] J. B. Orlin. Genuinely polynomial simplex and non-simplex algorithms for the minimum cost flow problem. Technical report, Technical Report 1615-84, Sloan School of Management, MIT, Cambridge, MA., 1984.

[16] J. B. Orlin. A polynomial time primal network simplex algorithm for minimum cost flows. *Mathematical Programming*, 78:109–129, 1997.

[17] K. Paparizzos, N. Samaras, and A. Sifaleras. Exterior point simplex-type algorithms for linear and network optimization problems. *Annals of Operations Research*, 229(1):607–633, 2015.

[18] T. Terlaky and S. Zhang. Pivot rules for linear programming: a survey on recent theoretical developments. *Annals of Operations Research*, 46(1):203–233, 1993.