An Analysis of the Highest-Level Selection Rule in the Preflow-Push Max-Flow Algorithm

Joseph Cheriyan * Kurt Mehlhorn †

August 20, 1998

Abstract

Consider the problem of finding a maximum flow in a network. Goldberg and Tarjan introduced the preflow-push method for solving this problem. When this method is implemented with the highest-level selection rule, then both the running time and the number of pushes are known to be $O(n^2\sqrt{m})$, where n is the number of nodes and m is the number of edges. We give a new proof based on a potential function argument. Potential function arguments may be preferable for analysing preflow-push algorithms, since they are simple and generic.

1 Introduction

Consider the problem of finding a maximum flow in a network. This is one of the most intensively studied combinatorial problems, and finds applications in transportation, communication networks, VLSI design, etc., see [AMO93]. Goldberg and Tarjan [GT88] introduced the preflow-push method for solving this problem. The complexity of this method depends mainly on the number of push operations, and this in turn depends mainly on the selection rule used in every iteration for choosing an active node.

The highest-level selection rule guarantees that the number of pushes is $O(n^2\sqrt{m})$, where n is the number of nodes and m is the number of edges. This bound was first proved by Cheriyan and Maheshwari [CM89] for a particular implementation of the algorithm that uses so-called "current edges". Tunçel [Tun94] gave a more general proof that applies to the standard algorithm, i.e., the algorithm need not follow the "current edges implementation". Both proofs use the concepts of "flow atom" and "flow path"; a flow atom is flow excess that is moved by a sequence of non-saturating pushes and a flow path is the path traced by a flow atom. The exact definition of both concepts is non-trivial and is different in the two mentioned papers. For many other selection rules, e.g., the arbitrary rule, the LIFO-rule, or the FIFO-rule, a potential function argument can be used to bound the number of pushes. Potential function arguments are simpler and more generic than flow atom and flow path arguments. Ahuja et al [AMO93] gave a potential argument proof for the "current edges implementation" of the highest-level selection rule.

We give a potential argument proof that applies to the standard algorithm. Our proof uses a new potential function that may be regarded as a simplification of the potential function in [AMO93]. Our potential function is "similar in spirit" to the potential functions used by Goldberg and Tarjan for analyzing the LIFO and FIFO selection rules. The running time of the algorithm is determined by the number of pushes and is also $O(n^2\sqrt{m})$.

*Department of Combinatorics and Optimization, University of Waterloo, Canada

†Max-Planck-Institute for Computer Science, Saarbrücken, Germany

For general information, we mention that there are max-flow algorithms that run much faster than the $O(n^2\sqrt{m})$ time bound. However, such algorithms either (i) use the dynamic trees (or similar) data structures that allow flow to be sent across a path of ℓ edges in $O(\log^{O(1)} n)$ time rather than $O(\ell)$ time, or (ii) they apply to the restricted problem where the capacities are integers rather than reals, or (iii) they apply to the restricted problem where the graph underlying the network is undirected rather than directed. Several computational studies have been conducted to compare among algorithms predating the preflow-push method as well as different algorithms arising from the preflow-push method. On almost all of the 20–40 network families used as inputs in these computational studies, the preflow-push algorithm using the highest-level selection rule turned out to be the fastest or the second fastest. (Implementations of the preflow-push method use additional heuristics to quickly solve the given instance of the problem, see [CG97, AKMO97, MN99].)

Section 2 states the max-flow problem and describes the preflow-push method. Readers familiar with this may prefer to skip Section 2. Section 3 has the main result.

2 The Max-Flow Problem and the Preflow-Push Method

The problem

Let $G = (V, E)$ be a directed graph, and let n and m denote the number of nodes and edges, respectively. Let s and t be distinct nodes in G, and let $cap : E \longrightarrow \mathbb{R}$ be a non-negative function on the edges of G. For an edge e, we call $cap(e)$ the *capacity* of e. We use V^+ to denote $V \setminus \{s, t\}$. An (s,t)-*flow* or simply *flow* is a function $f : E \longrightarrow \mathbb{R}$ satisfying the capacity constraints (1) and the flow conservation constraints (2):

$$
\begin{array}{lcl}
(1) & 0 \le f(e) \le cap(e) & \text{for every edge } e \in E \\
(2) & \sum_{e:source(e)=v} f(e) = \sum_{e:target(e)=v} f(e) & \text{for every node } v \in V^+
\end{array}
$$

For a function $f : E \longrightarrow \mathbb{R}$ and a node v

$$
excess(v) = \sum_{e:target(e)=v} f(e) \quad - \sum_{e:source(e)=v} f(e)
$$

is called the *excess* of v. The *value* of a flow is the net flow into t (equivalently, the net flow out of s), namely, $excess(t)$. A flow is maximum if its value is at least as large as the value of any other flow. The max-flow problem asks for the computation of a maximum flow.

The method

Goldberg and Tarjan [GT88] introduced the preflow-push method for solving the maximum-flow problem. It manipulates a preflow that gradually evolves into a flow. We review the method to the extent that is needed for this paper.

A *preflow* f is a function $f : E \longrightarrow \mathbb{R}$ with

$$
\begin{array}{lll}
(1) & 0 \le f(e) \le cap(e) & \text{for every edge } e \in E \text{ and} \\
(2) & excess(v) \ge 0 & \text{for every node } v \in V^+
\end{array}
$$

i.e., the flow conservation constraint is replaced by the weaker constraint that no node in V^+ has negative excess. A node $v \in V^+$ is called *active* if its excess is positive.

The *residual network* G_f with respect to a preflow f has the same node set as G. Every edge of G_f is induced by an edge of G and has a so-called *residual capacity.* Let $e = (v, w)$ be an arbitrary edge of G. Let e^{rev} denote the reverse edge (w, v). If $f(e) < cap(e)$, then e is also an edge of G_f. Moreover, if $f(e) > 0$, then e^{rev} is an edge of G_f. Note that if e is an edge of G_f, then either e or e^{rev} is an edge of G. The residual capacity of an edge e in G_f is $r(e) = cap(e) - f(e) + f(e^{rev})$, where we take $cap(e) = f(e) = 0$ if e is not in G, and similarly we take $f(e^{rev}) = 0$ if e^{rev} is not in G.

The basic operation to manipulate a preflow is a *push.* Let v be an active node, let $e = (v, w)$ be a residual edge out of v, and let $\delta = \min(excess(v), r(e))$. Informally, a push of δ across (v, w) sends δ units of flow from node v to node w by increasing $f(e)$ and/or decreasing $f(e^{rev})$ by a total of δ. A push of δ across e has the following effect on the residual network: $r(e)$ decreases by δ and $r(e^{rev})$ increases by δ. Moreover, if $r(e)$ becomes zero, then e is removed from G_f, and if $r(e^{rev})$ increases from zero to δ, then e^{rev} is added to G_f. A push is called *saturating* if $\delta = r(e)$ and is called *nonsaturating* otherwise. Note that a saturating push removes one edge from G_f, and a nonsaturating push deactivates one node. Either kind of push adds an edge e^{rev} to the residual network (if it is not already there). A push of δ across $e = (v, w)$ increases $excess(w)$ by δ and decreases $excess(v)$ by δ.

```
/* initialization */
set f(e) = cap(e) for all edges e with source(e) = s;
set f(e) = 0 for all other edges e;
set d(s) = n and d(v) = 0 for all other nodes v;

/* main loop */
while there is an active node
{  let v be any active node;
     if there is an eligible edge e = (v,w)
     { push δ across e where δ = min(excess(v), r(e)); }
     else
     { relabel v; }
}
```

Table 1: The preflow-push method.

Goldberg and Tarjan suggested to partition the node set of G (and hence G_f) into layers with t on the bottom-most (zero) layer and to perform only pushes that move excess to a lower layer. Let each node v be assigned an integer label $d(v)$. Informally, $d(v)$ is the (number of the) layer containing v. More precisely, $d(t) = 0$, $d(s) = n$, and $d : V \longrightarrow \mathbb{Z}$ must satisfy the inequality $d(v) \leq 1 + d(w)$, for each residual edge (v, w). (So, if $d(v) < n$, then $d(v)$ is no more than the length of a directed path from v to t in G_f.) An edge $e = (v, w) \in G_f$ is called *eligible* if $d(w) < d(v)$. A push across an edge $e = (v, w) \in G_f$ can be performed only if v is active and e is eligible.

There is another basic operation in the preflow-push method. When v is active and there is no eligible edge out of v, v may be *relabeled* by increasing $d(v)$ by one.

The preflow-push method is given in Table 1. Goldberg and Tarjan [GT88, Lemmas 3.8,3.9] give a short proof of the following result.

Proposition 1 *The number of relabelings is at most $2n^2$, and the number of saturating pushes is at most $2nm$.*

The preflow-push method is general, and different algorithms can be obtained by fixing rules for choosing nodes/edges for applying the basic operations. Several different rules for selecting the active node v in the main loop (see Table 1) have been proposed, see [AMO93]. The number of nonsaturating pushes depends on the rule for selecting active nodes. The rule that guarantees the currently best bound on the number of nonsaturating pushes is the so-called highest-level selection rule.

3 The Highest-Level Selection Rule

The highest-level selection rule always selects an active node on the highest-level, i.e., with the maximum d-label.

Theorem 2 *For the preflow-push algorithm that uses the highest-level selection rule, the number of nonsaturating pushes is $O(n^2\sqrt{m})$.*

Proof: Our proof is based on a potential function. Let $K = \sqrt{m}$; the reason for this choice of K will become clear in the analysis. For a node v, let

$$d'(v) = |\{w \ : \ d(w) \leq d(v)\}|.$$

In words, $d'(v)$ is the number of nodes w such that the label $d(w)$ is no more than $d(v)$. Let

$$\Phi = \sum_{v \ : \ v \text{ is active}} d'(v)/K.$$

Note that $\Phi \leq n^2/K$ initially, and $\Phi \geq 0$ always.

Consider the effect on Φ of relabelings and pushes. Note that $d'(w) \leq n$ always, for all $w \in V$. A relabeling of a node v increases Φ by at most n/K, since $d'(v)$ may increase, but for all other nodes $w \neq v$, $d'(w)$ does not increase. A saturating push increases Φ by at most n/K, since at most one new active node may be created. A nonsaturating push does not increase Φ. To see this, note that a nonsaturating push across an edge (v, w) deactivates v, activates w (if w is not active already), and $d'(w) \leq d'(v)$.

We estimate the total number of nonsaturating pushes by splitting the execution into phases. We define a phase to consist of all pushes between two consecutive changes of

$$d^* = \max\{d(v) \ : \ v \text{ is active } \}.$$

We call a phase *cheap* if it contains at most K nonsaturating pushes, and *expensive* otherwise.

Claim 1: The number of phases is at most $4n^2$. The number of nonsaturating pushes in cheap phases is at most $4n^2K$.

To see that the number of phases is $\leq 4n^2$, we observe that $d^* = 0$ initially, $d^* \geq 0$ always, and every increase of d^* is caused by a relabeling. Since d^* increases at most $2n^2$ times (by Proposition 1), it decreases at most $2n^2$ times, and hence it changes at most $4n^2$ times. The second part of Claim 1 follows immediately.

Claim 2: An expensive phase containing $Q > K$ nonsaturating pushes decreases Φ by at least Q.

To see Claim 2, consider an expensive phase that executes $Q > K$ nonsaturating pushes. Clearly, d^* is constant during the phase, and hence all Q nonsaturating pushes must be out of nodes at level d^*. The phase terminates either when all nodes in level d^* are deactivated, or when a relabeling moves a node from level d^* to level $d^* + 1$. In either case, we conclude that level d^*

contains $Q > K$ (active/inactive) nodes at all times during the phase. Hence, each nonsaturating push in the phase decreases Φ by at least one, since we have $d'(v) \leq d'(u) - 1$ for an edge (u, v) with $|\{w : d(w) = d(u)\}| \geq K$.

To complete the proof, note that the total increase of Φ is at most $(2n^2 + 2nm)n/K$, hence, the total decrease is at most this number plus the initial value of Φ (since $\Phi \geq 0$ always). Therefore, the number of nonsaturating pushes in expensive phases is bounded by $(2n^3 + n^2 + 2n^2m)/K \leq (3n^3 + 2n^2m)/K$. Now, Claim 1 implies that the total number of nonsaturating pushes is at most $(3n^3 + 2n^2m)/K + 4n^2K$. Observing that $n = O(m)$ and that the choice $K = \sqrt{m}$ balances the contributions from expensive and cheap phases, we obtain a bound of $O(n^2\sqrt{m})$. $\square$

The bound on the running time follows easily from the previous theorem. Note that each push operation and each relabel operation can be implemented in $O(1)$ time. The highest-level selection rule can be implemented by a "buckets data structure" with each bucket i containing all nodes w with $d(w) = i$. The total running time overhead for this is $O(nm)$.

Theorem 3 *For the preflow-push algorithm that uses the highest-level selection rule, the running time is* $O(n^2\sqrt{m})$.

References

[AKMO97] R. K. Ahuja, M. Kodialam, A. K. Mishra, and J. B. Orlin. Computational investigation of maximum flow algorithms. *European Journal on Operational Research*, 97:509–542, 1997.

[AMO93] R. K. Ahuja, T. L. Magnanti, and J.B. Orlin. *Network Flows*. Prentice Hall, 1993.

[CG97] B.V. Cherkassky and A.V. Goldberg. On implementing the push-relabel method for the maximum flow problem. *Algorithmica*, pages 390–410, 1997.

[CM89] Cheriyan and Maheshwari. Analysis of preflow push algorithms for maximum network flow. *SIAM Journal of Computing*, 18:1057–1086, 1989.

[GT88] A.V. Goldberg and R.E. Tarjan. A new approach to the maximum-flow problem. *JACM*, 35:921–940, 1988.

[MN99] K. Mehlhorn and S. Näher. *The LEDA Platform for Combinatorial and Geometric Computing*. Cambridge University Press, 1999. `http://www.mpi-sb.mpg.de/~mehlhorn`.

[Tun94] L. Tuncel. On the complexity of preflow-push algorithms for maximum flow problems. *Algorithmica*, 11:353–359, 1994.