Algorithmica (1994) 11: 353–359

Algorithmica
© 1994 Springer-Verlag New York Inc.

On the Complexity of Preflow-Push Algorithms for Maximum-Flow Problems[1]

Levent Tunçel[2]

Abstract. We study the maximum-flow algorithm of Goldberg and Tarjan and show that the largest-label implementation runs in $O(n^2\sqrt{m})$ time. We give a new proof of this fact. We compare our proof with the earlier work by Cheriyan and Maheswari who showed that the largest-label implementation of the preflow-push algorithm of Goldberg and Tarjan runs in $O(n^2\sqrt{m})$ time when implemented with current edges. Our proof that the number of nonsaturating pushes is $O(n^2\sqrt{m})$, does not rely on implementing pushes with current edges, therefore it is true for a much larger family of largest-label implementation of the preflow-push algorithms.

Key Words. Graph theory, Network flows, Algorithms, Complexity, Maximum flow.

1. Introduction. Goldberg and Tarjan [GT2] gave a family of algorithms, called preflow-push algorithms, for the maximum-flow problem. They also proved that without using sophisticated data structures, the algorithm runs in $O(n^3)$ time. Using the *dynamic-tree data structure* enabled them to obtain an $O(nm \log(n^2/m))$-time bound.

Cheriyan and Maheswari [CM] showed that the *largest-label* implementation actually runs in $O(n^2\sqrt{m})$ time when the current edges are used. It is reasonable to believe that the key point in decreasing the number of nonsaturating pushes is to push through the same edge repeatedly until it gets saturated. Here, we give a different proof that the *largest-label* algorithm runs in $O(n^2\sqrt{m})$. We show that the number of nonsaturating pushes is $O(n^2\sqrt{m})$ even if the algorithm is implemented without using current edges. In other words, any arbitrary choice of the edge to push through cannot give a worse bound than $O(n^2\sqrt{m})$. We note that to the best of our knowledge this is the best bound proven on the number of nonsaturating pushes for any version of the preflow-push algorithms.

Throughout this paper we assume that the reader is familiar with the maximum-flow algorithm of Goldberg and Tarjan. For a survey on the algorithm see, for instance, the survey by Goldberg *et al.* [GTT].

---

[1] Research performed while the author was a Ph.D. student at Cornell University and was partially supported by the Ministry of Education of the Republic of Turkey through the scholarship program 1416.

[2] Department of Combinatorics and Optimization, Faculty of Mathematics, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada.

Received June 29, 1990; revised June 14, 1991. Communicated by Harold N. Gabow.

2. Maximum-Flow Problem. Let $G = (V, E)$ be a digraph. Let s and t be two distinguished nodes of G. Node s is called the source, and node t is called the sink. $u(v, w) \geq 0$ is a real-valued capacity for v, $w \in V$. We assume that $u(v, w) = 0$ if $(v, w) \notin E$. Throughout this paper $n := |V|$ and $m := |E|$.

DEFINITION 2.1. A *preflow* f is a real-valued function on the node pairs that satisfies the following constraints:

$$f(v, w) \leq u(v, w), \qquad \forall (v, w) \in E \qquad \text{(capacity constraints)},$$

$$f(v, w) = -f(w, v), \qquad \forall (v, w) \in E \qquad \text{(antisymmetry constraints)},$$

$$e_f(v) = \sum_{w \in V} f(w, v) \geq 0, \qquad \forall v \in V - \{s, t\}.$$

DEFINITION 2.2. A *flow* on G is a preflow which also satisfies the following:

$$e_f(v) = 0 \qquad \text{(flow conservation constraints)}, \qquad \forall v \in V - \{s, t\}.$$

The *value* of a flow f is defined as $e_f(t)$ and the maximum-flow problem is to find a flow of maximum value from s to t.

DEFINITION 2.3. Given a preflow f, the *residual capacity* of a node pair is defined as $u_f(v, w) = u(v, w) - f(v, w)$, $\forall v, w \in V$. The *residual graph* is $G_f = (V, E_f)$ where $E_f = \{(v, w) \in V \times V \mid u_f(v, w) > 0\}$.

DEFINITION 2.4. A *distance labeling* d for a preflow f is a function from V to the nonnegative integers such that $d(s) = n$, $d(t) = 0$, and $d(v) \leq d(w) + 1$, $\forall (v, w) \in E_f$.

The generic preflow-push algorithm maintains a preflow f and a distance labeling function d for the preflow f, and updates f and d by using push and relabel operations. Now we give the generic preflow-push maximum-flow algorithm:

INITIALIZE:
$\forall (v, w) \in E$ **do**
begin
  $f(v, w) := 0$;
  if $v = s$ **then** $f(v, w) := u(v, w)$;
  if $w = s$ **then** $f(v, w) := -u(w, v)$;
end
$\forall v \in V$ **do**
begin
  Calculate $e_f(v)$
  if $v = s$ **then** $d(v) := n$ **else** $d(v) := 0$;
end

PUSH(v, w):
{ *push*(v, w) is applicable if $e_f(v) > 0$, $u_f(v, w) > 0$, and $d(v) = d(w) + 1$. }
begin
  $\delta := \min\{e_f(v), u_f(v, w)\}$;
  $f(v, w) := f(v, w) + \delta$;
  $f(w, v) := f(w, v) - \delta$;
end

RELABEL(v):
{ *relabel*(v) is applicable if
  (i) $e_f(v) > 0$, and
  (ii) $v \notin \{s, t\}$
  (iii) $\forall w \in V$ either $u_f(v, w) = 0$ or $d(w) \geq d(v)$. }
begin
  $d(v) := \min_{(v, w) \in E_f}\{d(w)\} + 1$;
end

MAIN LOOP OF THE ALGORITHM:
while $\exists v$ such that $e_f(v) > 0$ **do**
  select an applicable operation (either *push*(v, w) or *relabel*(v)) and apply it
end.

From the description of the generic algorithm it is clear that there are many possible alternatives for the choice of the next operation to apply, the next edge to push through, and the next node to relabel. Different implementations use different rules to select the next node with excess, and there might be different rules to choose the next edge to push through.

The *largest-label preflow-push algorithm* selects v in the main algorithm so that $e_f(v) > 0$ and $d(v) = \text{Max}_{w \in V}\{d(w) | e_f(w) > 0\}$. In other words, the *largest-label preflow-push algorithm* tries to push the excess away from a node with the largest label.

DEFINITION 2.5. The *push*(v, w) operation is called *saturating* if $u_f(v, w) = 0$ after the *push*(v, w) operation, and called *nonsaturating* otherwise.

The proofs of the following lemmas (Lemmas 2.1–2.5) can be found in [G], [GT2], and [GTT].

LEMMA 2.1. *When the algorithm terminates, the preflow f is a maximum flow.*

LEMMA 2.2. *The number of relabeling operations is at most $2n - 1$ per vertex and at most $(2n - 1)(n - 2) < 2n^2$ overall.*

LEMMA 2.3. *The number of saturating pushes is at most nm.*

LEMMA 2.4. *If k is the number of nonsaturating pushes throughout the algorithm, then the largest-label preflow-push algorithm can be implemented in $O(nm + k)$ time.*

Note that Lemma 2.4 has been proven for the implementation with current edges. Since we want to prove the same bound for a larger family of implementations, we should generalize it a little bit more. For instance, it is not difficult to see that any implementation, that does not spend more than $O(nm + k)$ time over all to find an edge to push along or to conclude that there are no eligible edges, will satisfy the claim of the lemma. We show that the number of nonsaturating pushes is $O(n^2\sqrt{m})$ for *all* pushing rules. So, in terms of the bound on the number of pushes, all pushing rules are covered by our result. We give some examples of natural pushing rules that are covered by our result:

(i) Push to the node with minimum excess.
(ii) Push to the node with maximum residual capacity (i.e., push to a node w such that $\sum_{r \in V} u_f(w, r)$ is maximized among all eligible neighbors of v).

The first rule is something reasonable to do because it tries to spread the current excess among its neighbors evenly. The second rule is slightly more involved. It tries to push to a neighbor which has the best potential (locally) for pushing that excess away.

DEFINITION 2.6. Let $d_{\max} = \text{Max}_{w \in V}\{d(w) | e_f(w) > 0\}$. We define a *phase* of the algorithm as a maximal interval of time during which $d_{\max}$ remains constant.

LEMMA 2.5. *The number of phases for the largest-label preflow-push algorithm is at most $4n^2$.*

Cheriyan and Maheswari [CM] showed that the *largest-label preflow-push algorithm* runs in $O(n^2\sqrt{m})$ time when the current edges are used. Their version is a particular implementation of the largest-label preflow-push algorithm. After a *push*(v, w) operation, if the edge (v, w) is not saturated it becomes the *current edge* of v. The next time v is selected for a *push* operation, if $d(v)$ is still equal to $d(w) + 1$, the algorithm applies a *push*(v, w) operation. Using current edges forces the algorithm to push along the same edge for a given node as long as the edge is not saturated. Intuitively, this implementation should reduce the number of nonsaturating pushes. However, we prove that even if we do not use current edges, the number of pushes throughout the algorithm is $O(n^2\sqrt{m})$.

Similar to the definition of Cheriyan and Maheswari [CM], we define a *flow atom* as a flow excess that travels through a sequence of nonsaturating pushes without getting relabeled.

DEFINITION 2.7.

(i) A flow atom is born at node v when a *relabel*(v) operation is done (and the atom which was previously at this node dies).

(ii) During a saturating *push*(v, w) operation a flow atom which will travel through edge (v, w) is born and if $e_f(v) > u_f(v, w)$ another flow atom is born at node v (in either case the atom which was previously at node v dies).
(iii) If w had excess before a *push*(v, w) operation, then after the push operation a flow atom is born at node w (and the two merged atoms die).

The following lemma can be found in [CM]. Due to the slight difference in the definition of a flow atom, we give a proof.

LEMMA 2.6. *The number of flow atoms born throughout the algorithm is* $O(nm)$.

PROOF. We consider all cases separately:
(i) The number of *flow atoms* born when a relabelling is done is at most $O(n^2)$ (by Lemma 2.2).
(ii) The number of *flow atoms* born by saturating pushes is at most $O(nm)$ (by Lemma 2.3).
(iii) The number of *flow atoms* born by merging is at most the sum of the number of *flow atoms* born by cases (i) and (ii), because merge decreases the number of alive *flow atoms* by one. □

For the proof of the $O(n^2\sqrt{m})$-time bound for the algorithm without using current edges, we introduce the concept of *future course* of a flow atom.

DEFINITION 2.8. The *future course* of a flow atom is the path consisting of the sequence of nonsaturating pushes that this atom will travel through until it dies.

LEMMA 2.7. *Suppose that at some time during the algorithm we have* $e_f(v_1) > 0$ *and* $e_f(v_2) > 0$ (v_1 *and* v_2 *are distinct*), *then the future courses of the flow atoms at* v_1 *and* v_2 *are node-disjoint, except potentially the last node.*

PROOF. Suppose not. Let the *flow atom* at v_1 be α_1 and the *flow atom* at v_2 be α_2. Let w be a common node of the future courses of α_1 and α_2. Without loss of generality, assume that α_1 arrived at w first. If we push α_2 to w when α_1 is there, they both die, so it is necessary that we push α_1 away from w before we push α_2 to w. Note that the atoms travel monotone downward, because, *push*(v, w) applies only if $d(v) = d(w) + 1$ and relabeling kills the atom at the relabeled node. So, when we push α_1 away from w, α_2 is at a node with a larger label. Therefore, α_1 cannot be pushed away from w. This is a contradiction.

THEOREM 2.1. *The number of nonsaturating pushes throughout the algorithm is* $O(n^2\sqrt{m})$.

PROOF. Let a *special push* be pushing a flow atom which has a future course of length strictly greater than $n/\sqrt{m}$. If we consider the nonspecial pushes, we see

that each atom dies after at most $n/\sqrt{m}$ such pushes. There are $O(mn)$ atoms, so there are at most $O(n^2\sqrt{m})$ nonsaturating pushes of this type. To establish an upper bound on the number of special pushes consider all such pushes during a phase. Let $d_{\max}$ be the maximum label during a particular phase. All pushes made during that phase are on the atoms which are at the nodes with label $d_{\max}$ at the beginning of the phase (because we are using the largest-label algorithm). Lemma 2.7 implies that the number of special pushes per phase is at most $\sqrt{m}$. Using Lemma 2.5 we conclude that the number of special pushes is $O(n^2\sqrt{m})$ overall. □

COROLLARY 2.1. *The largest-label preflow-push algorithm runs in* $O(n^2\sqrt{m})$ *time.*

Note that in [CM] instead of *future course*, the *trace* of a flow atom is used. The *trace* of a flow atom is defined as the sequence of nonsaturating pushes that this atom has traveled through since it was born. Their analysis is based on the observation that if two flow atoms are simultaneously alive at the nodes with largest label, then they have node-disjoint *traces*. Our observation is that if *any* two flow atoms are simultaneously alive, then they have node-disjoint *future courses*. Our result is strong enough to prove that any arbitrary choice of the next edge to push along gives an $O(n^2\sqrt{m})$-time bound.

3. Conclusion. As was pointed out before, it is intuitively clear that the usage of current edges decreases the number of nonsaturating pushes. However, in terms of the worst-case performance, we proved that no matter how you choose the edge to push you will not get a worse bound than $O(n^2\sqrt{m})$ on the number of nonsaturating pushes. We believe that this result encourages the attempts to get a better running time by using preflow-push algorithms, because this result provides flexibility in choosing which edge to push along. Consider for instance the results of Cheriyan *et al.* [CHM], they start the execution of their algorithm with edges that have relatively large capacities and then add the other edges in the order of decreasing capacity as the execution progresses. Using their algorithm on dense graphs, they were able to get an $o(mn)$-time bound on running time. We believe that using more sophisticated rules for choosing the next edge, it might be possible to improve the $O(n^2\sqrt{m})$ bound on the number of pushes for preflow-push algorithms. We also believe that some analog of this result might be extended to some version of the successive approximation algorithm of Goldberg and Tarjan [GT1] for the minimum-cost circulation problem. Also note that since the complexity bound applies no matter how the next edge is chosen, the result of the paper might yield more efficient time bounds on applications of the maximum-flow model where there is more than one criterion for optimality.

Acknowledgment. We thank Èva Tardos for her very valuable and helpful discussions.

References

[CHM] J. Cheriyan, T. Hagerup, and K. Mehlhorn, Can a Maximum Flow be Computed in $o(nm)$ Time?, *Proceedings of the 17th International Colloquium on Automata, Languages, and Programming*, London, 1990, pp. 235–248.

[CM] J. Cheriyan and S. N. Maheswari, Analysis of Preflow Push Algorithms for Maximum Network Flow, *SIAM J. Comput.*, **18** (1989), 1057–1086.

[G] A. V. Goldberg, Efficient Graph Algorithms for Sequential and Parallel Computers, Ph.D. thesis MIT/LCS/TR-374, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 1987.

[GT1] A. V. Goldberg and R. E. Tarjan, Finding Minimum-Cost Circulations by Successive Approximations, Technical Report MIT/LCS/TM-333, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 1987.

[GT2] A. V. Goldberg and R. E. Tarjan, A New Approach to the Maximum Flow Problem, *J. Assoc. Comput. Mach.*, **35** (1988), 921–940.

[GTT] A. V. Goldberg, È. Tardos, and R. E. Tarjan, Network Flow Algorithms, Technical Report No. 860, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY, 1989.