MIT/LCS/TM-80

THE MAX FLOW ALGORITHM OF DINIC AND KARZANOV: AN EXPOSITION

Shimon Even

December 1976

The Max Flow Algorithm of Dinic and Karzanov
An Exposition

by

Shimon Even[*]
Department of Computer Science
Technion
Haifa, Israel

December 1976

Abstract: Recently A.V. Karzanov improved Dinic's algorithm to run in time $O(n^3)$ for networks of n vertices. For the benefit of those who do not read Russian, the Dinic-Karzanov algorithm is explained and proved.

In addition to being the best algorithm known for network flow, this algorithm is unique in that it does not use path augmentation.

[*] Research for this paper was completed while the author was visiting at the Laboratory for Computer Science, M.I.T. This report was prepared with the support of the National Science Foundation under grant no. DCR74-12997-A01.

I. Introduction

A network consists of the following data:

1) A directed finite graph $G(V,E)$ with no parallel edges. Let n be the number of vertices ($=|V|$).

2) Two vertices s and t are specified; s is the source and t is the sink.

3) Each edge $e \in E$ is assigned a positive real number $c(e)$ called the capacity of e.

A legal flow function f is an assignment of a real number $f(e)$ to each edge e which satisfies two conditions:

C 1) For every edge $e \in E$, $0 \leq f(e) \leq c(e)$.

C 2) For every vertex $v \in V$ let $\alpha(v)$ and $\beta(v)$ be the sets of edges incoming to v and outgoing from v respectively. For every vertex $v \neq s,t$

$$\sum_{e\in\alpha(v)} f(e) = \sum_{e\in\beta(v)} f(e) \qquad (1)$$

The total flow F of f is defined by

$$F = \sum_{e\in\alpha(t)} f(e) - \sum_{e\in\beta(t)} f(e) \qquad (2)$$

Let $S \subseteq V$ such that $s \in S$ and $t \notin S$; also $\bar{S} = V-S$. Let $(S;\bar{S})$ be the set of all edges which start in a vertex of S and end in a vertex of $\bar{S}$. The set $(\bar{S};S)$ is defined similarly.

Lemma 1: For every S

$$F = \sum_{e\in(S;\bar{S})} f(e) - \sum_{e\in(\bar{S};S)} f(e). \qquad (3)$$

The Lemma is easily proven by summing up Eq. (2) and Eq. (1) for all $v \in \bar{S} - \{t\}$.

Let $c(S:\bar{S})$ be defined by

$$c(S;\bar{S}) = \sum_{e\in(S;\bar{S})} c(e) .$$

(This is called the capacity of the cut defined by S.)

Lemma 2: Let f be any legal flow whose total flow is F. Then for every S

$$F \leq c(S;\bar{S}). \qquad (4)$$

Proof: By Lemma 1

$$F = \sum_{e \in (S;\bar{S})} f(e) - \sum_{e \in (\bar{S};S)} f(e).$$

Since f(e) ≥ 0 by (C1) and since f(e) ≤ c(e) again by (C1), we get

$$F \leq \sum_{e \in (S;\bar{S})} c(e) = C(S;\bar{S}).$$

Q.E.D.

Corollary 1: If f and S satisfy (4) by equality, then F is maximum (and $c(S;\bar{S})$ is minimum).

We are interested in constructing an efficient algorithm for finding a maximum F in a given network. Ford and Fulkerson [1] have suggested an algorithm which achieved this goal by repeated improvements of F through paths, called augmenting paths, by which the flow can be increased. The number of steps of their algorithm is not bounded by any function of n, even if we allow one step operations on reals; that is, a comparison of two real numbers or their addition is considered a primitive operation. In fact they described an example for which their algorithm may not terminate. Yet, their algorithm was used successfully for the last 20 years. Edmonds and Karp [2] showed in 1969 that if one used Breadth First Search (BFS) in the labeling algorithm of Ford and Fulkerson, then the number of steps is bounded by $O(n^5)$. Dinic [3] described a different algorithm in 1970, which uses BFS too and has a time bound $O(n^4)$. Finally, Karzanov [4] improved Dinic's algorithm in 1974 and the time bound for the new algorithm is $O(n^3)$. The new algorithm does not use augmenting paths and in this respect it is revolutionary.

I do not read Russian and could not read the more detailed available material on the work of Dinic and Karzanov [5]. The short descriptions in Soviet Math. Dokl. arehard to read because they are translations and lack details. I have rediscovered Dinic's algorithm with J. Hopcroft in 1972 and have reconstructed Karzanov's result with the help of A. Itai. The proofs are my own and I do not know if they differ from those of Dinic and Karzanov. In any case I alone am responsible for any mistakes that may be in my exposition.

II. Maximum Flow Through Maximal Flows in Layered Networks

As in previous algorithms the new algorithm describes how to improve existing legal flow, and when no improvement is possible the algorithm halts.

If we have a legal present flow f, an edge e, connecting vertices u and v, can be used to transfer flow from u to v in any one of the following two cases:

1) $u \overset{e}{\rightarrow} v$ (e is directed from u to v) and $f(e) < c(e)$.
2) $u \overset{e}{\leftarrow} v$ (e is directed from v to u) and $f(e) > 0$.

(Clearly an edge can be used to both transfer flow from u to v and from v to u if $0 < f(e) < c(e)$). We say that e is useful from u to v if (1) or (2) hold.

The layered network of G(V,E) with flow f is defined by the following procedure:

```
1) $V_0 \leftarrow \{s\}$ and $i \leftarrow 0$.
2) Construct $T \leftarrow \{v \mid v \notin V_j$ for $j \leq i$ and there is a useful edge from
   a vertex of $V_i$ to $v\}$
3) If T is empty, the existing F is maximum, halt.
4) If T contains t then $\ell = i + 1$,  $V_\ell \leftarrow \{t\}$ and halt.
5) Let $V_{i+1} \leftarrow T$, increment i and return to (2).
```

For every $0 < i \leq \ell$, E_i is the set of edges useful from a vertex of V_{i-1} to a vertex of V_i.

Clearly the procedure investigated each edge at most twice; once in each direction. Thus, the number of steps in the procedure is bounded by $O(|E|) \le O(n^2)$.

Lemma 3: If the procedure halts in step (3) the present flow is indeed maximum.

Proof: Let $S = \bigcup_{j=0}^{i} V_j$. Every edge $u \overset{e}{\rightarrow} v$ in $(S:\bar{S})$ is saturated, i.e., $f(e) = c(e)$, or else e is useful from u to v and T is not empty. Also, every edge $u \overset{e}{\leftarrow} v$ in $(\bar{S},S)$ has $f(e) = 0$, or again e is useful from u to v, etc. Thus,

$$F = \sum_{e\in(S:\bar{S})} f(e) - \sum_{e\in(\bar{S}:S)} f(e) = \sum_{e\in(S:\bar{S})} c(e) = c(S:\bar{S}),$$

and by Corollary 1, F is maximum.

Q.E.D.

For every edge e in E_j let $\tilde{c}(e)$ be defined as follows:

1) If $u \in V_{j-1}, v \in V_j$ and $u \overset{e}{\rightarrow} v$ then $\tilde{c}(e) = c(e) - f(e)$.

2) If $u \in V_{j-1}, v \in V_j$ and $u \overset{e}{\leftarrow} v$ then $\tilde{c}(e) = f(e)$.

We now consider all edges of E_j to be directed from V_{j-1} to V_j, even if in $G(V,E)$ they may have the opposite direction (in case (2)). Also, the initial flow in the new network is $\tilde{f}(e) = 0$ everywhere. We seek a maximal flow $\tilde{f}$ in the layered network; by a <u>maximal flow</u> $\tilde{f}$ we mean that $\tilde{f}$ satisfies the condition that for every path $s \xrightarrow{e_1} v_1 \xrightarrow{e_2} v_2 — \cdots v_{\ell-1} \xrightarrow{e_\ell} t$, where $v_j \in V_j$ and $e_j \in E_j$ there is at least one edge e_j such that $\tilde{f}(e_j) = \tilde{c}(e_j)$. Clearly, a maximal flow is not necessarily maximum as the example of Figure 1 shows.

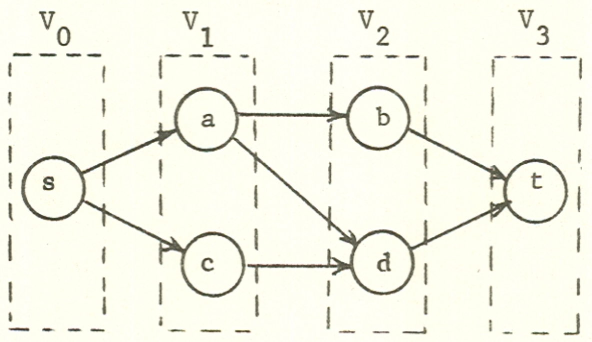

Figure 1

If for all edges $\tilde{c} = 1$ and we push one unit flow from s to t via a and d then the resulting flow is maximal in spite of the fact that the total flow is 1 while a total flow of 2 is possible.

In the next section we shall describe how one can find a maximal flow function $\tilde{f}$ efficiently. For now, let us assume that such a flow function has been found and its total value is $\tilde{F}$. The flow f in G(V,E) is changed into f' as follows:

1) If $u \xrightarrow{e} v$, $u \in V_{j-1}$ and $v \in V_j$ then $f'(e) = f(e) + \tilde{f}(e)$.

2) If $u \xleftarrow{e} v$, $u \in V_{j-1}$ and $v \in V_j$ then $f'(e) = f(e) - \tilde{f}(e)$.

It is easy to see that the new flow f' satisfies both C1 (due to the choice of $\tilde{c}$) and C2 (because it is the superposition of two flows which satisfy C2). Clearly $F' = F + \tilde{F} > F$.

Let us call the part of the algorithm which starts with f, finds its layered network, finds a maximal flow $\tilde{f}$ in it and improves the flow in the original network to become f' - a <u>phase</u>. We want to show that the number of phases is bounded by n. For that purpose we shall prove that the length of the layered network increases from phase to phase; by <u>length</u> is meant the index of the last layer, which we called ℓ in step (4) of the procedure. Thus, ℓ_k denotes the length of the layered network of the k^{th} phase.

<u>Lemma 4</u>: If the $(k+1)^{st}$ phase is not the last then $\ell_{k+1} > \ell_k$.

<u>Proof</u>: There is a path of length k+1 in the (k+1)-st layered graph which starts with s and ends with t: $s \xrightarrow{e_1} v_1 \xrightarrow{e_2} \cdots v_{\ell_{k+1}-1} \xrightarrow{e_{\ell_{k+1}}} t$.

First, let us assume that all the vertices of the path appear in the k-th layered network. Let V_j be the j-th layer of the k-th layered network. We claim that if $v_a \in V_b$ then $a \geq b$. This is proved by induction on a. For $a = 0$, $(v_0 = s)$ the claim is obviously true. Now, assume $v_{a+1} \in V_c$. If $c \leq b+1$ the inductive step is trivial. But if $c > b+1$ then the edge e_{a+1} has not been used in the k-th phase since it is not even in the k-th layered network, in which only edges between adjacent layers appear. If e_{a+1} has not been used and is useful from v_a to v_{a+1} in the beginning of phase k+1, then it was useful from v_a to v_{a+1} in the beginning of phase k. Thus, v_{a+1} cannot belong to V_c (by the procedure). Now, in particular, $t = v_{\ell_{k+1}}$ and $t \in V_{\ell_k}$. Therefore, $\ell_{k+1} \geq \ell_k$. Also, equality cannot hold, because in this case the whole path is in the k-th layered network, and if all its edges are still useful in the beginning of phase k+1 then the $\tilde{f}$ of phase k was not maximal.

If not all the vertices of the path appear in the k-th layered network then let $v_a \xrightarrow{e_{a+1}} v_{a+1}$ be the first edge such that for some b $v_a \in V_b$ but v_{a+1} is not in the k-th layered netowrk. Thus, e_{a+1} was not used in phase k. Since it is useful in the beginning of phase k+1, it was also useful in the beginning of phase k. The only possible reason for v_{a+1} not to belong to V_{b+1} is that $b+1 = \ell_k$. By the argument of the previous paragraph $a \geq b$. Thus $a+1 \geq \ell_k$, and therefore $\ell_{k+1} > \ell_k$.

Q.E.D.

<u>Corollary 2</u>: The number of phases is less than or equal to n-1.

<u>Proof</u>: Since $\ell \leq n-1$, Lemma 4 implies the corollary.

Q.E.D.

The remaining task is to describe an efficient algorithm to construct a maximal flow in a layered network. Dinic used here Depth First Search and built $\tilde{f}$ through augmenting paths. The best bound for this method is $O(|V|\cdot|E|)$, yielding a bound $O(|V|^2\cdot|E|)$ for the whole algorithm. In the next section we describe Karzanov's contribution, which brings the number of steps down to $O(n^2)$ per phase, yielding an overall complexity of $O(n^3)$.

III. <u>Construction of a Maximal Flow in a Layered Network.</u>

Karzanov's method uses in intermediate steps illegal flow functions which he calls preflow. A <u>preflow</u> function $\tilde{f}$ satisfies C1; that is, for every e in the layered network $0 \leq \tilde{f}(e) \leq \tilde{c}(e)$. However, it may not satisfy C2. Instead it satisfies the following weaker condition:

(C3) For every v in the layered network let $\tilde{\alpha}(v)$ and $\tilde{\beta}(v)$ be the sets of edges incoming to v and outgoing from v in the layered network, respectively. For every $v \neq s,t$

$$\sum_{e\in\tilde{\alpha}(v)} \tilde{f}(e) \geq \sum_{e\in\tilde{\beta}(v)} \tilde{f}(e). \qquad (5)$$

The algorithm uses two procedures alternately. They are called advance and balance.

The _advance_ procedure will push forward additional preflow. It will start with vertices of some V_j; at that time no forwarding of preflow will be possible from vertices of lower layers. If preflow is pushed into some vertices of V_{j+1}, we shall try to push the preflow further, and the procedure will stop only when all the vertices into which preflow has been pushed have been processed. The vertices will be considered by moving from one layer to the next after all the pertinent vertices in the layer have been processed. The order by which the vertices in one layer are processed is irrelevant. However, for each vertex v the edges in $\tilde{\beta}(v)$ are ordered in some fixed order: $\tilde{\beta}(v) = \{e_{v1}, e_{v2},\ldots,e_{vd}\}$, where d is the outgoing degree of v in the layered network. Initially, all the edges of the layered network are _open_, but as the algorithm proceeds, some edges will be declared _closed_, and the flow through them will remain unchanged to the end of the algorithm. Once the advance chooses an _unbalanced_ vertex, (one for which (5) is satisfied with inequality), it first pushes flow through e_{v1} if it is open, next through e_{v2}, if it is open, etc. It advances through the picked edge e as much as possible. If e becomes saturated (i.e., $\tilde{f}(e) = \tilde{c}(e)$) it moves to the next open edge. If the vertex v becomes balanced, the advance from v ends. Also, if all the edges of $\tilde{\beta}(v)$ are either closed or saturated, the advance from v ends, even though v may still be unablanced. Typically, $e_{v1},e_{v2},\ldots,e_{vb}$ are either saturated or closed, $e_{v,(b+1)}$ is only partly used (i.e., $\tilde{f}(e_{v,(b+1)} < \tilde{c}(e_{v,(b+1)}))$

and for $c > b+1$, $\tilde{f}(e_{vc}) = 0$.

For every vertex v there is a stack (push-down store) on which we record the additions of incoming flow into v. Each addition is an ordered pair of an edge $e \in \tilde{\alpha}(v)$ and a positive real number r. It specifies through which edge the additional incoming of r units of flow is produced. Also, for each vertex we keep in two registers the sum of the incoming flow (which is equal to the sum of the r's stored in the stack) and the sum of the outgoing flow. Clearly, if these two registers contain the same number then the vertex is balanced.

The balancing procedure is the tool through which unbalanced vertices become balanced. It is applied to all the unbalanced vertices of one layer, and this layer is chosen to be the highest which contains unbalanced vertices. We balance a vertex v by canceling most recent additions, and as many of them as necessary so that the incoming flow will equal the outgoing flow. Clearly, the last canceled addition may be only partial, if only a part of the quantity specified by it must be canceled. In this case we restore the corrected addition (same edge, reduced quantity) in the stack.* After an unbalanced vertex v is balanced, all edges of $\tilde{\alpha}(v)$ are declared closed.

Algorithm K:

```
1) Assign zero flow to all edges and all vertex flow registers.  Empty
   the stacks of all the vertices.
2) $i \leftarrow 0$
3) Perform advance starting from $V_i$.
4) If there are no unbalanced vertices other than s and t, halt:
   the present preflow is a maximal flow.
5) Let $V_j$ $(0 < j < \ell)$ be the highest layer which contains unbalanced
   vertices.  Perform balancing for the unbalanced vertices in $V_j$.
6) $i \leftarrow j-1$ and go to (3).
```

A vertex is called blocked if every directed path from it to t contains at least one saturated edge. Clearly, s becomes blocked on the first application of Step (3), since all the edges in $\tilde{\beta}(s)$ become saturated.

* This step is actually superfluous.

Lemma 5: Blocked vertices remain blocked and with each completion of the advance procedure every unbalanced vertex is blocked.

Proof: The proof is by induction on the number of applications of the main loop (Steps (4), (5), (6), and (3)).

After the first application of the advance procedure, if a vertex v is unbalanced then all its outgoing edges are saturated and it is obviously blocked.

Assume now that by the end of the m-th application of the advance procedure the Lemma holds and let us show that it holds by the end of the (m+1)-st application.

First the balance procedure is applied. Let us show that it cannot unblock a vertex. Assume that $v \in V_j$ has been unbalanced and it is balanced during the procedure. That means, by the inductive hypothesis, that v has been blocked before balancing it and must still be blocked since no flow has been canceled on the paths between v and t. If a blocked vertex u belongs to a lower layer and there is a path from u to t via v then since v remains blocked, the cancellation of incoming flow into v does not unblock u.

Now, a vertex $u \in V_{j-1}$ may become unbalanced by balancing the vertices of V_j. During the next advance procedure u may become balanced again. If it remains unblanaced then each of its outgoing edges either becomes saturated or it is closed and therefore leads into a vertex $v \in V_j$ which is blocked.

Q.E.D.

Lemma 6: Every vertex v is being balanced at most once.

Proof: Since all incoming edges are closed during the first balancing of v, it cannot become unbalanced by new incoming flow. It remains to be shown that it will not become unbalanced by cancellation of outgoing flow.

If $v \in V_j$, then when v is being balanced all vertices of higher layers are balanced. Let us call all the additions of these vertices old. The claim is that old additions are never cancelled, and therefore v remains balanced. Let u be a vertex for which old additions are cancelled later, and let u belong to the highest layer which contains such vertices. Since the old flow on the edges outgoing from u is not changed, and since u has been balanced when balancing has been performed on v, some new flow has entered u since, and during u's balancing the new flow is cancelled first. If all its new additions are cancelled, u cannot be out of balance any more. This contradicts the assumption that old additions into u are cancelled.

Q.E.D.

__Theorem 1__: Algorithm K terminates and the resulting preflow is a maximal flow.

__Proof__: By Lemma 6 the number of times Step (5) is applied is bounded by n-2. Since every one of the procedures takes a bounded number of steps, the algorithm terminates. Also, vertex s is blocked immediately after the first application of Step (3). Since all the vertices are balanced upon termination, the preflow is a flow, and since s remains blocked (by Lemma 5) the flow is maximal.

Q.E.D.

__Theorem 2__: The number of steps Algorithm K takes is bounded by $O(n^2)$.

__Proof__: The total number of steps used in the balance procedures is bounded by the number of additions since this is an upper bound on the number of cancellations of additions. Also, the total number of steps in the advance procedures is bounded by the number of additions. Let us show that the number of additions is bounded by $O(n^2)$.

Let us charge each addition which saturates the edge to the edge, and there is at most one such addition per edge. The remaining additions, which are not saturating the edge are charged to the vertex from which this edge emanates. There can be only one such addition for each vertex in every advance procedure, and since the number of such procedures is bounded by n-1, the number of these additions is bounded by (n-1)(n-2). Thus, the total number of additions is bounded by $O(n^2)$.

Clearly the remaining number of steps in (1), (2), (4), and (6) is bounded by $O(n^2)$ too.

Q.E.D.

__Theorem 3__: The total number of steps of the Dinic-Karzanov algorithm is bounded by $O(n^3)$.

Proof: Since the number of phases is bounded by n-1 (Corollary 2), and since each phase takes at most $0(|E|)$ to construct the layered network and at most $0(n^2)$ to find a maximal flow (Theorem 2) then the whole algorithm takes at most $0(n^3)$ steps.

Q.E.D.

References

[1] L.R. Ford, Jr. and D.R. Fulkerson, Flows in Networks, Princeton Univ. Press, 1962.

[2] J. Edmonds and R.M. Karp, "Theoretical Improvements in Algorithm Efficiency for Network Flow Problems," JACM Vol. 19, No. 2, 1972, 248-264.

[3] E.A. Dinic, "Algorithm for Solution of a Problem of Maximum Flow in a Network with Power Estimation," Soviet Math. Dokl., Vol. 11, 1970, 1277-1280.

[4] A.V. Karzanov, "Determining the Maximal Flow in a Network by the Method of Preflows," Soviet Math. Dokl., Vol. 15, 1974, 434-437.

[5] G.M. Adelson-Velsky, E.A. Dinic and A.V. Karzanov, Flow Algorithms, Moscow, Nauka, 1975 (in Russian).