INTRODUCTION OF ALGORITHMS ON NETWORK FLOWS

JACKY LI

Abstract. In this paper, the algorithms for network flows are explored. Firstly, a network and its max-flow are defined, then by proving the Max-flow Min-cut Theorem, we derive Ford and Fulkerson’s Algorithm to find a max-flow on any network. After that, a more recent algorithm developed by Goldberg and Tarjan using a new idea of pre-flows is introduced and derived.

Contents

1. Introduction 1
   1.1. Graphs 1
   1.2. Flow Networks 2
2. Algorithm Computing Maximum Flow of a Network - Ford and Fulkerson 3
3. A Different Algorithm from Goldberg and Tarjan 7
   Acknowledgments 13
   References 13

1. Introduction

In the field of algorithms, there is an important topic that has half a century of research history and is an active area of research even now: network flows[1]. The maximum-flow problem is a specific subject under this topic that was investigated for tens of years.[8] The answer to this problem is very useful - it can model the routing of traffic through a transportation network, packets through a data network, or oil through a distribution network, etc. Even bipartite matching and image segmentation can also be reduced to the maximum-flow problem[2]. Thus, it has plenty of charm to be studied in detail.

1.1. Graphs.

Definition 1.1. A graph is a pair $G = (V, E)$ of sets such that the elements of $E$ are 2-element subsets of $V$, and $E \subseteq [V]^2$. The elements of $V$ are called vertices of the graph $G$, the elements of $E$ are its edges. An element $e = \{a, b\} = \{b, a\}$ of $E$ is called an edge with end vertices $a$ and $b$. We say that $a$ and $b$ are incident with $e$ and that $a$ and $b$ are adjacent or neighbors of each other. $G$ is also called a non-directional graph.

Definition 1.2. A directional graph, or digraph for short, is a pair $G = (V, E)$ of sets such that the elements of $E$ are ordered 2-element subsets of $V$, i.e. $\{a, b\} \neq \{b, a\}$, and $E \subseteq [V]^2$. The elements of $V$ are vertices of the graph $G$, the elements of $E$ are its directional edges. An element $e = \{a, b\}$ of $E$ is an edge with the start vertex $a$ and end vertex $b$, vice versa. Intuitively, one can consider digraph’s directional edges as one-way bridges between two vertices whereas non-directional edges are normal two-way roads.

Definition 1.3. $G$ is a symmetric digraph if $\{v, w\}$ and $\{w, v\}$ are both edges of $G$. Intuitively, symmetric digraph behaves the same as a non-directional graph, except the edge numbers are doubled as there are two edges connecting two vertices.

Date: Aug. 31, 2019.
Definition 1.4. Let \((e_1, e_2, ..., e_n)\) be a sequence of edges in a graph \(G\). If there are vertices \(v_0, v_1, ..., v_n\) such that \(e_i = \{v_{i-1}, v_i\}\) or \(e_i = \{v_i, v_{i-1}\}\) for \(i = 1, 2, ..., n\), the sequence is called a non-directional walk. If \(v_i\) are distinct, then the walk is a non-directional path. If there are vertices \(v_0, v_1, ..., v_n\) such that \(e_i = \{v_{i-1}, v_i\}\) for \(i = 1, 2, ..., n\), the sequence is called a directional walk. If \(v_i\) are distinct, then the walk is a directional path.

The usual way to picture a graph is by drawing a dot for each vertex and joining two of these dots by a line if the corresponding two vertices form an edge. Note that how these dots and lines are drawn is considered irrelevant as only information of which two pairs of vertices form an edge and which do not matters.

1.2. Flow Networks. Now, let us view a graph as a real-life network: its edges carry some kind of flow - of water, electricity, data, traffics or something similar. How much can be transported in a network from a source \(s\) to a sink \(t\) if the capacities of the connections are given? Let’s rewrite the question in mathematical terms.

Definition 1.5. Let \(G = (V, E)\) be a digraph, and let \(c : E \rightarrow \mathbb{R}^+\) be a mapping. \(c(e)\) is defined as the capacity of the edge \(e\). Let \(s\) and \(t\) be the special vertices of source and sink respectively. For every \(e = \{a, b\} \in E\), we define element \(e' = \{b, a\}\), and refer to \(e'\) as backward edge, \(e\) as forward edge. We define \(c(e') = c(e)\), and the total flow on \(e\) and \(e'\) adds up to \(c(e)\). Then, \(N = (G, c, s, t)\) is called a flow network with source \(s\) and sink \(t\).

The following graph is an example of a flow network with \(s\) being the source, \(t\) being the sink, and each edge’s capacity is identified with an number in brackets:
Definition 1.6. A flow on $N$ is a mapping $f : E \to \mathbb{R}^+$ satisfying the following two conditions:

(F1) $0 \leq f(e) \leq c(e)$ for each edge $e$; (Feasibility condition)
(F2) $\sum_{e^+ = v} f(e) = \sum_{e^- = v} f(e)$ for each vertex $v \neq s, t$, where $e^-$ and $e^+$ denote the start and end vertex of $e$, respectively. (Kirchhoffs Law)

Now, from Definition 1.5, we have $f(e') + f(e) = c(e)$.

The constraint (F1) requires that each edge carries a non-negative amount of flow which may not exceed the capacity of the edge and (F2) means that the flows are preserved: at each vertex except the source and the sink, the amount that flows in is equal to the amount that flows out. Intuitively, the total flow coming out of $s$ should be the same as the total flow going in to $t$; let us check if this is true.

Lemma 1.7. Let $N = (G, c, s, t)$ be a flow network with flow $f$. Then

$$w \sum_{e^- = s} f(e) - \sum_{e^+ = t} f(e) = \sum_{e^- = s} f(e) - \sum_{e^+ = t} f(e)$$

Proof.

$$\sum_{e} f(e) = \sum_{e^+ = s} f(e) + \sum_{e^- = t} f(e) + \sum_{v \neq s, t} \sum_{e^+ = v} f(e)$$

$$\sum_{e} f(e) = \sum_{e^- = s} f(e) + \sum_{e^+ = t} f(e) + \sum_{v \neq s, t} \sum_{e^- = v} f(e)$$

by (F2),

$$\sum_{v \neq s, t} \sum_{e^+ = v} f(e) = \sum_{v \neq s, t} \sum_{e^- = v} f(e)$$

therefore,

$$\sum_{e^+ = s} f(e) + \sum_{e^- = t} f(e) = \sum_{e^- = s} f(e) + \sum_{e^+ = t} f(e)$$

The quantity in (1.8) is called the value of $f$, the flow through the network; it is denoted by $w(f)$. A flow $f$ is said to be maximal if $w(f) \geq w(f')$ holds for every flow $f'$ on $N$. The original problem is successfully translated into finding the $w(f)$ of a given network.

Definition 1.9. Let $N = (G, c, s, t)$ be a flow network. A cut of $N$ is a partition $V = S \cup T$ of the vertex set $V$ of $G$ into two disjoint sets $S$ and $T$ with $s \in S$ and $t \in T$. The capacity of a cut $(S, T)$ is defined as $c(S, T) = \sum_{e \in S, e+ \in T} c(e)$. The cut $(S, T)$ is called minimal if $c(S, T) \leq c(S', T')$ holds for every cut $(S', T')$.

Definition 1.10. Let $f$ be a flow in the network $N = (G, c, s, t)$. A path $W$ from $s$ to $t$ is called an augmenting path with respect to $f$ if $f(e) < c(e)$ holds for every forward edge $e \in W$, whereas $f(e) > 0$ for every backward edge $e' \in W$.

2. Algorithm Computing Maximum Flow of a Network - Ford and Fulkerson

In 1900s, Ford and Fulkerson proved several theorems that yielded an algorithm that finds the $w(f)$ of a given network[6].

Lemma 2.1. Let $N = (G, c, s, t)$ be a flow network $(S, T)$ a cut, and $f$ is a flow. Then

$$w(f) = \sum_{e^- \in S, e+ \in T} f(e) - \sum_{e^+ \in S, e^- \in T} f(e)$$

In particular, $w(f) \leq c(S, T)$. 
Proof. Summing (F2) over all \( v \in S \) gives
\[
w(f) = \sum_{v \in S} \left( \sum_{e^- = v} f(e) - \sum_{e^+ = v} f(e) \right)
= \sum_{e^- \in S, e^+ \in S} f(e) - \sum_{e^- \in S, e^- \in T} f(e) + \sum_{e^+ \in S, e^- \in S} f(e) - \sum_{e^+ \in S, e^+ \in T} f(e)
= \sum_{e^- \in S, e^- \in T} f(e) - \sum_{e^+ \in S, e^- \in S} f(e)
\]
Note \( f(e) \leq c(e) \) for all edges \( e \) with \( e^- \in S \) and \( e^+ \in T \), and \( f(e) \geq 0 \) for all edges \( e \) with \( e^+ \in S \) and \( e^- \in T \). \( \square \)

**Theorem 2.3** (Augmenting path theorem). A flow \( f \) on a flow network \( N = (G, c, s, t) \) is maximal if and only if there are no augmenting paths with respect to \( f \).

Proof. First let \( f \) be a maximal flow. Suppose there is an augmenting path \( W \). Let \( d \) be the minimum of all values \( (c(e) - f(e)) \) taken over all forward edges \( e \) in \( W \), and all values \( f(e') \) taken over the backward edges in \( W \). Then \( d > 0 \), by definition of an augmenting path. Now we define a mapping \( f' : E \to \mathbb{R}^+ \) as follows:
\[
f'(e) = \begin{cases} f(e) + d & \text{if } e \text{ is a forward edge in } W \\ f(e) - d & \text{if } e \text{ is a backward edge in } W \\ f(e) & \text{otherwise} \end{cases}
\]
It is easily checked that \( f' \) is a flow on \( N \) with value \( w(f') = w(f) + d > w(f) \), contradicting the maximality of \( f \).

Conversely, suppose there are no augmenting paths in \( N \) which respect to \( f \). Let \( S \) be the set of all vertices \( v \) such that there exists an augmenting path from \( s \) to \( v \) including \( s \), and put \( T = V \setminus S \). By hypothesis, \( (S, T) \) is a cut of \( N \). Thus, each edge \( e \) with \( e^- \in T \) and \( e^+ \in S \) has to be void: \( f(e) = 0 \). By Lemma 2.1, \( w(f) = c(S, T) \), so \( f \) is maximal. \( \square \)

**Corollary 2.4.** A flow \( f \) on a flow network \( N = (G, c, s, t) \) is maximal if and only if the set \( S \) of all vertices accessible from \( s \) on a augmenting path with respect to \( f \) is a proper subset of \( V \). In this case, \( w(f) = c(S, T) \), where \( T = V \setminus S \).

Proof. Follows from proof of Theorem 2.3. \( \square \)

**Theorem 2.5** (Integral flow theorem). Let \( N = (G, c, s, t) \) be a flow network where all capacities \( c(e) \) are integers. Then there is a maximal flow on \( N \) such that all values \( f(e) \) are integral.

Proof. By setting \( f_0(e) = 0 \) for all \( e \), we obtain an integral flow \( f_0 \) on \( N \) with value 0. If this trivial flow is not maximal, then there exists an augmenting path with respect to \( f_0 \). In that case the number \( d \) appearing in the proof of Theorem 2.3 is a positive integer, and we can construct an integral flow \( f_1 \) of value \( d \) as in the proof of Theorem 2.3. We continue in the same manner. As the value of the flow is increased in each step by a positive integer and as the capacity of any cut is an upper bound on the value of the flow (by Lemma 2.1), after a finite number of steps we reach an integral flow \( f \) for which no augmenting path exists. By Theorem 2.3, this flow \( f \) is maximal. \( \square \)

**Theorem 2.6** (Max-flow Min-cut theorem). The maximal value of a flow on a flow network is equal to the minimal capacity of a cut for \( N \).

Proof. If all capacities are integers, the assertion follows from Theorem 2.5 and Corollary 2.4. The case where all capacities are rational can be reduced to the integral case by multiplying all numbers by their common denominator. Then real-valued capacities may be treated using a continuity argument, since the set of flows is a compact subset of \( R^{E} \) and since \( w(f) \) is a continuous function of \( f \). \( \square \)

Notice that, sometimes we allow networks on directed multigraphs (graph with parallel edges being assigned with multiple capacities) - this is not really more general, because parallel edges can be replaced by a
single edge whose capacity is the sum of the corresponding capacities of the parallel edges. Non-directional graphs can also be solved by adding edges to the directional graphs.

Now, after the proof, the algorithm is at hand.

**Algorithm 1 Algorithm of Ford and Fulkerson[4]**

Let $N = (G, c, s, t)$ be a flow network, and capacity function $c$ only takes integral or rational values.

1: procedure FORDFULK($N; f, S, T$)
2: for $e \in E$ do
3: \hspace{1em} $f(e) \leftarrow 0$;
4: \hspace{1em} Label $s$ with $(-, \infty)$;
5: for $v \in V$ do
6: \hspace{2em} $u(v) \leftarrow \text{false}; d(v) \leftarrow \infty$;
7: \hspace{1em} while $u(v) \neq \text{true}$ for all vertices $v$ which are labelled do
8: \hspace{2em} choose a vertex $v$ which is labelled and satisfies $u(v) = \text{false}$;
9: \hspace{2em} for $e \in e \in E : e^- = v$ do
10: \hspace{3em} if $w = e^-$ is not labelled then and $f(e) < c(e)$
11: \hspace{4em} $d(w) \leftarrow \min\{c(e) - f(e), d(u)\}$;
12: \hspace{4em} label $w$ with $(v, -, d(w))$;
13: \hspace{2em} $u(v) \leftarrow \text{true}$;
14: \hspace{2em} if $t$ is labelled then
15: \hspace{3em} let $d$ be the last component of the label of $t$;
16: \hspace{3em} $w \leftarrow t$;
17: \hspace{2em} while $w \neq s$ do
18: \hspace{3em} find the first component $v$ of the label of $w$;
19: \hspace{3em} if the second component of the label of $w$ is $+$ then
20: \hspace{4em} set $f(e) \leftarrow f(e) + d$ for $e = \{v, w\}$;
21: \hspace{4em} else
22: \hspace{5em} set $f(e) \leftarrow f(e) - d$ for $e = \{w, u\}$;
23: \hspace{3em} $w \leftarrow v$;
24: \hspace{2em} delete all labels except for the label of $s$;
25: for $v \in V$ do
26: \hspace{2em} $d(v) \leftarrow \infty$;
27: \hspace{2em} $u(v) \leftarrow \text{false}$;
28: let $S$ be the set of vertices which are labelled and put $T \leftarrow V \setminus S$;

Let’s run this algorithm through the network of Figure 2 (pictures read from left to right)[4]:
Figure 3
The Ford and Fulkerson algorithm presented so far constructs a maximal flow starting with the zero flow by augmenting the flow iteratively along a single augmenting path. The algorithm of Goldberg and Tarjan is based on a completely different concept: it uses pre-flows. These are mappings for which flow excess is allowed: the amount of flow entering a vertex may be larger than the amount of flow leaving it. This preflow property is maintained throughout the algorithm; it is only at the very end of the algorithm that the preflow becomes a flow which is then already maximal.

The main idea of the algorithm is to push flow from vertices with excess flow toward the sink \( t \), using paths which are not necessarily shortest paths from \( s \) to \( t \), but merely current estimates for such paths. Of course, it might occur that excess flow cannot be pushed forward from some vertex \( v \); in this case, it has to be sent back to the source on a suitable path. The choice of all these paths is controlled by a certain labelling function on the vertex set. We will soon make all this precise. Altogether, the algorithm will be quite intuitive and comparatively simple to analyze.

In this section, we define flows in a formally different although, of course, equivalent way; this notation will simplify the presentation of the algorithm.

**Definition 3.1 (Redefine).** It is convenient to consider \( c \) and \( f \) as functions from \( V \times V \) to \( \mathbb{R} \). Thus we do not distinguish between \( f(e) \) and \( f(u,v) \), where \( e = \{u,v\} \) is an edge of \( G \); we put \( f(u,v) = 0 \) whenever \( \{u,v\} \) is not an edge of \( G \); and similarly for \( c \). Then we drop the condition that flows have to be non-negative, and define a flow \( f : V \times V \to R \) by the following requirements:

1. \( f(v,w) \leq c(v,w) \) for all \( (u,w) \in V \times V \)
2. \( f(v,w) = -f(w,v) \) for all \( (u,w) \in V \times V \)
3. \( \sum_{u \in V} f(u, v) = 0 \) for all \( v \in V \setminus \{s,t\} \)

The anti-symmetry condition (2) makes sure that only one of the two edges in a pair \( \{v,w\} \) and \( \{w,v\} \) of anti-parallel edges in \( G \) may carry a positive amount of flow. Condition (2) also simplifies the formal description in one important respect: we will not have to make a distinction between forward and backward edges anymore. Moreover, the formulation of the flow conservation condition (3) is easier. The definition of the value of a flow becomes a little easier, too:

\[
 w(f) = \sum_{u \in V} f(v,t)
\]

For an intuitive interpretation of flows in the new sense, we should consider only the non-negative part of the flow function: this part is a flow as originally defined in Section 1.2. Also, notice that the anti-symmetry of \( f \) implies that condition (3) is equivalent to the earlier condition (F2).

**Definition 3.2.** Now we define a preflow as a mapping \( f : V \times V \to \mathbb{R} \) satisfying conditions (1) and (2) above and the following weaker version of condition (3):

\[
 (3') \sum_{u \in V} f(u, v) \geq 0 \text{ for all } v \in V \setminus \{s\}
\]

Using the intuitive interpretation of flows, condition \((3')\) means that the amount of flow entering a vertex \( v \neq s \) no longer has to equal the amount leaving \( v \); it suffices if the in-flow is always at least as large as the out-flow.

**Definition 3.3.** The value

\[
 e(v) = \sum_{u \in V} f(u,v)
\]

is called the flow excess of the preflow \( f \) in \( v \).

As mentioned before, the algorithm of Goldberg and Tarjan tries to push flow excess from some vertex \( v \) with \( e(v) > 0 \) forward towards \( t \). We first need to specify which edges may be used for pushing flow.

**Definition 3.4.** Given a preflow \( f \), let us define the residual capacity \( r_f : V \times V \to R \) as follows:

\[
 r_f(v,w) = c(v,w) - f(v,w)
\]

**Definition 3.5.** If an edge \( \{v,w\} \) satisfies \( r_f(v,w) > 0 \), we may move some flow through this edge; such an edge is called a residual edge.
In our intuitive interpretation, this corresponds to two possible cases. Either the edge \( \{v, w\} \) is a forward edge which is not yet saturated: \( 0 \leq f(v, w) < c(v, w) \); or it is a backward edge, that is, the anti-parallel edge \( \{w, v\} \) is non-void: \( 0 < f(w, v) \leq c(w, v) \), and hence \( f(v, w) = -f(w, v) < 0 \leq c(v, w) \).

**Definition 3.6.** The residual graph with respect to \( f \) is defined as \( G_f = (V, E_f) \), where \( E_f = \{\{v, w\} \in E : r_f(v, w) > 0\} \).

**Definition 3.7.** A mapping \( d : V \to \mathbb{N}_0 \cup \{\infty\} \) is called a valid labelling with respect to a given preflow \( f \) if the following two conditions hold:

1. \( d(v) \leq d(w) + 1 \) for all \( \{v, w\} \in E_f \).

2. The algorithm of Goldberg and Tarjan starts with some suitable preflow and a corresponding valid labelling. Usually, one saturates all edges emanating from \( s \), and puts \( d(s) = |V| \) and \( d(v) = 0 \) for all \( v \in V \setminus \{s\} \).

More precisely, the initial preflow is given by \( 4f(s, v) = -f(v, s) = c(s, v) \) for all \( v \neq s \) and \( f(v, w) = 0 \) for all \( v, w \neq s \).

Then the algorithm executes a series of operations which we will specify later. These operations change either the preflow \( f \) (by pushing the largest possible amount of flow along a suitable residual edge) or the labelling \( d \) (by raising the label of a suitable vertex); in both cases, the labelling will always remain valid. As mentioned before, \( d \) is used to estimate shortest paths in the corresponding residual graph. In particular, \( d(v) \) is always a lower bound for the distance from \( v \) to \( t \) in \( G_f \) provided that \( d(v) < |V| \); and if \( d(v) \geq |V| \), then \( t \) is not accessible from \( v \), and \( d(v) - |V| \) is a lower bound for the distance from \( v \) to \( s \) in \( G_f \).

The algorithm terminates as soon as the preflow has become a flow (which is then actually a maximal flow). We need one more notion to be able to write down the algorithm in its generic form.

**Definition 3.8.** A vertex \( v \) is called active provided that \( v \neq s, t; e(v) > 0 \); and \( d(v) < \infty \).

Algorithm 2 [4]

Let \( N = (G, c, s, t) \) be a flow network on symmetric digraph, where \( c : V \times V \to \mathbb{R}^+ \), that is, for \( (u, w) \notin E \) we have \( c(u, w) = 0 \).

1. **procedure** GOLDBERGER\( \left(N; f\right)\)
2.   for \( (u, v) \in (V \setminus \{s\}) \times (V \setminus \{s\}) \) do
3.     \( f(u, v) \leftarrow 0; r_f(v, w) \leftarrow c(v, w); \)
4.     \( d(s) \leftarrow |V|; \)
5.   for \( v \in V \setminus \{s\} \) do
6.     \( f(v, s) \leftarrow c(s, v); r_f(s, v) \leftarrow 0; \)
7.     \( f(v, s) \leftarrow -c(s, v); r_f(v, s) \leftarrow c(v, s); \)
8.     \( d(v) \leftarrow 0; \)
9.     \( e(v) \leftarrow c(s, v) \)
10. while there exists an active vertex \( v \) do
11.     choose an active vertex \( v \) and execute an admissible operation

In step (11), one of the following operations may be used, provided that it is admissible:

1. **procedure** PUSH\( \left(N, f, v, w; f\right)\)
2.   \( \delta \leftarrow \min(e(v), r_f(v, w)); \)
3.   \( f(v, w) \leftarrow f(v, w) + \delta; f(w, v) \leftarrow f(w, v) - \delta; \)
4.   \( r_f(v, w) \leftarrow r_f(v, w) - \delta; r_f(w, v) \leftarrow r_f(w, v) + \delta; \)
5.   \( e(v) \leftarrow e(v) - \delta; e(w) \leftarrow e(w) + \delta. \)

The procedure \( \text{PUSH}(N, f, v, w; f) \) is admissible provided that \( v \) is active, \( r_f(v, w) > 0 \), and \( d(v) = d(w) + 1 \).
The procedure RELABEL($N, f, v, d; d$) is admissible provided that $v$ is active and $r_f(v, w) > 0$ always implies $d(v) \leq d(w)$. The minimum in step (2) is defined to be $\infty$ if there does not exist any $w$ with $r_f(v, w) > 0$. However, we will see that this case cannot occur.

Let us look more closely at the conditions for admissibility. If we want to push some flow along an edge $\{v, w\}$, three conditions are required. Two of these requirements are clear: the start vertex $v$ has to be active, so that there is positive flow excess $e(v)$ available which we might move; and $\{v, w\}$ has to be a residual edge, so that it has capacity left for additional flow. It is also not surprising that we then push along $\{v, w\}$ as much flow as possible, namely the smaller of the two amounts $e(v)$ and $r_f(v, w)$. The crucial requirement is the third one, namely $d(v) = d(w) + 1$. Thus we are only allowed to push along residual edges $\{v, w\}$ for which $d(v)$ is exactly one unit larger than $d(w)$, that is, for which $d(v)$ takes its maximum permissible value - see condition (5) from Definition 3.7. We may visualize this rule by thinking of water cascading down a series of terraces of different height, with the height corresponding to the labels. Obviously, water will flow down, and condition (5) has the effect of restricting the layout of the terraces so that the water may flow down only one level in each step. Now assume that we are in an active vertex $v$ so that some water is left which wants to flow out and that none of the residual edges leaving $v$ satisfies the third requirement. In our watery analogy, $v$ would be a sort of local sink: $v$ is locally on the lowest possible level, and thus the water is trapped in $v$. It is precisely in such a situation that the RELABEL-operation becomes admissible: we miraculously raise $v$ to a level which is just one unit higher than that of the lowest neighbor $w$ of $v$ in $G_f$; then a PUSH becomes permissible, that is, some of the water previously trapped in $v$ can flow down to $w$. Please note that, these remarks in no way constitute a proof of correctness; nevertheless, they help to obtain a feeling for the strategy behind the Goldberg-Tarjan algorithm.

Now we turn to the formal proof. This will allow us to show that Algorithm 2 indeed constructs a maximal flow on $N$ in finitely many steps, no matter in which order we select the active vertices and the admissible operations. This is in remarkable contrast to the situation for the algorithm of Ford and Fulkerson; recall the discussion in Section 2. To get better estimates for the complexity, however, we will have to specify appropriate strategies for the choices to be made. We begin by showing that the algorithm is correct under the assumption that it terminates at all. Afterwards, we will estimate the maximal number of admissible operations executed during the while-loop and use this result to show that the algorithm really is finite. Our first lemma is just a simple but important observation; it states a result which we have already emphasized in our informal discussion.

**Lemma 3.9.** Let $f$ be a preflow on $N$, $d$ a valid labelling on $V$ with respect to $f$, and $v$ an active vertex. Then either a PUSH-operation or a RELABEL-operation is admissible for $v$.

**Proof.** As $d$ is valid, we have $d(v) \leq d(w) + 1$ for all $w$ with $r_f(v, w) > 0$. If PUSH($v, w$) is not admissible for any $w$, we must even have $d(v) \leq d(w)$ for all $w$ with $r_f(v, w) > 0$, as $d$ takes only integer values. But then RELABEL is admissible. □

**Lemma 3.10.** During the execution of Algorithm 2, procedure GOLDBERG, $f$ always is a preflow and $d$ always is a valid labelling with respect to $f$.

**Proof.** We use induction on the number $k$ of admissible operations already executed. The assertion holds for the induction basis $k = 0$: $f$ is initialized as a preflow in procedure GOLDBERG steps (6) and (7); and the labelling $d$ defined in (4) and (8) is valid for $f$, since $d(v) = 0$ for $v \neq s$ and since all edges $\{s, v\}$ have been saturated in step (6); also, the residual capacities and the flow excesses are clearly initialized correctly in steps (6), (7), and (9). For the induction step, suppose that the assertion holds after $k$ operations have been executed. Assume first that the next operation is a PUSH($v, w$). It is easy to check that $f$ remains a preflow, and that the residual capacities and the flow excesses are updated correctly. Note that the labels are kept unchanged, and that $\{v, w\}$ and $\{w, v\}$ are the only edges for which $f$ has changed. Hence we only need to worry about these two edges in order to show that $d$ is still valid. By definition, $\{v, w\} \in E_f$ before the

```plaintext
1: procedure RELABEL($N, f, v, d; d$)
2: \[ d(v) \leftarrow \min\{d(w) + 1 : r_f(v, w) > 0\}; \]
```
PUSH. Now \{v, w\} might be removed from the residual graph \(G_f\) which happens if it is saturated by the PUSH; but then the labelling stays valid trivially. Now consider the anti-parallel edge \{w, v\}. If this edge already is in \(G_f\), there is nothing to show. Thus assume that \{w, v\} is added to \(G_f\) by the PUSH; again, \(d\) stays valid, since the admissibility conditions for the PUSH(\(v, w\)) require \(d(w) = d(v) - 1\). It remains to consider the case where the next operation is a RELABEL(\(v\)). Then the admissibility requirement is \(d(v) \leq d(w)\) for all vertices \(w\) with \(r_f(v, w) > 0\). As \(d(v)\) is increased to the minimum of all the \(d(w) + 1\), the condition \(d(v) \leq d(w) + 1\) holds for all \(w\) with \(r_f(v, w) > 0\) after this change; all other labels remain unchanged, so that the new labelling \(d\) is still valid for \(f\).

As mentioned before, the valid labelling \(d\) allows us to estimate distances in the corresponding residual graph:

**Lemma 3.11.** Let \(f\) be a preflow on \(N\), let \(d\) be a valid labelling with respect to \(f\), and let \(v\) and \(w\) be two vertices of \(N\) such that \(w\) is accessible from \(v\) in the residual graph \(G_f\). Then \(d(v) - d(w)\) is a lower bound for the distance of \(v\) and \(w\) in \(G_f\):

\[
d(v) - d(w) \leq d(v, w).
\]

**Proof.** Let \(P : v = v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_k = w\) be a shortest path from \(v\) to \(w\) in \(G_f\). Since \(d\) is a valid labelling, \(d(v_i) \leq d(v_{i+1}) + 1\) for \(i = 0, \ldots, k - 1\). As \(P\) has length \(k = d(v, w)\), we obtain the desired inequality. \(\square\)

**Corollary 3.12.** Let \(f\) be a preflow on \(N\) and \(d\) a valid labelling with respect to \(f\). Then \(t\) is not accessible from \(s\) in the residual graph \(G_f\).

**Proof.** Assume otherwise. Then \(d(s) \leq d(t) + d(s, t)\), by above lemma. But this contradicts \(d(s) = |V|\), \(d(t) = 0\), and \(d(s, t) \leq |V| - 1\). \(\square\)

**Theorem 3.13.** If Algorithm 2 terminates with all labels finite, then the preflow \(f\) constructed is in fact a maximal flow on \(N\).

**Proof.** By Lemma 3.9, the algorithm can only terminate when there are no more active vertices. As all labels are finite by hypothesis, \(e(v) = 0\) has to hold for each vertex \(v \neq s, t\); hence the preflow constructed by the final operation is indeed a flow on \(N\). By Corollary 3.12, there is no path from \(s\) to \(t\) in \(G_f\), so that there is no augmenting path from \(s\) to \(t\) with respect to \(f\). Now the assertion follows from Theorem 2.3 (Augmenting path theorem). \(\square\)

It remains to show that the algorithm indeed terminates and that the labels stay finite throughout. We need several further lemmas.

**Lemma 3.14.** Let \(f\) be a preflow on \(N\). If \(v\) is a vertex with positive flow excess \(e(v)\), then \(s\) is accessible from \(v\) in \(G_f\).

**Proof.** We denote the set of vertices accessible from \(v\) in \(G_f\) via a directed path by \(S\), and put \(T := V \setminus S\). Then \(f(u, w) \leq 0\) for all vertices \(u, w\) with \(u \in T\) and \(w \in S\), since \(0 = r_f(w, u) = c(w, u) - f(w, u) \geq 0 + f(u, w)\).

Using the anti-symmetry of \(f\), we get

\[
\sum_{w \in S} e(w) = \sum_{u \in V, w \in S} f(u, w) = \sum_{u \in V, w \in S} f(u, w) + \sum_{u, w \in S} f(u, w) = \sum_{u \in T, w \in S} f(u, w) \leq 0.
\]

Now the definition of a preflow requires \(e(w) \geq 0\) for all \(w \neq s\). But \(e(v) > 0\), and hence \(\sum_{w \in S} e(w) \leq 0\) implies \(s \in S\). \(\square\)

**Lemma 3.15.** Throughout Algorithm 2 \(d(v) \leq 2|V| - 1\) for all \(v \in V\).

**Proof.** Obviously, the assertion holds after the initialization phase in procedure GOLDBERG steps (1) to (9). The label \(d(v)\) of a vertex \(v\) can only be changed by an operation RELABEL(\(v\)), and such an operation is admissible only if \(v\) is active. In particular, \(v \neq s, t\), so that the claim is trivial for \(s\) and \(t\); moreover, \(e(v) > 0\). By Lemma 3.14, \(s\) is accessible from \(v\) in the residual graph \(G_f\). Now Lemma 3.11 gives

\[
d(v) \leq d(s) + d(v, s) \leq d(s) + |V| - 1 = 2|V| - 1.
\]

\(\square\)
Lemma 3.16. During the execution of Algorithm 2, at most \(|V| - 1\) RELABEL-operations occur for any given vertex \(v \neq s, t\). Hence the total number of RELABEL-operations is at most \((|V| - 1)(|V| - 2) < 2|V|^2\).

Proof. Each RELABEL\((v)\) increases \(d(v)\). Since \(d(v)\) is bounded by \(|V| - 1\) throughout the entire algorithm as stated in Lemma 3.15, the assertion follows. It is more difficult to estimate the number of PUSH-operations. We need to distinguish two cases: a PUSH\((v, w)\) will be called a saturating PUSH if \(r_f(v, w) = 0\) holds afterwards that is, for \(\delta = r_f(v, w)\) in step (2) of the PUSH, and a non-saturating PUSH otherwise.

Lemma 3.17. During the execution of Algorithm 2, fewer than \(|V||E|\) saturating PUSH-operations occur.

Proof. By definition, any PUSH\((v, w)\) requires \(\{v, w\} \in E_f\) and \(d(v) = d(w) + 1\). If the PUSH is saturating, a further PUSH\((v, w)\) can only occur after an intermediate PUSH\((w, v)\), since we have \(r_f(v, w) = 0\) after the saturating PUSH\((w, v)\). Note that no PUSH\((w, v)\) is admissible before the labels have been changed in such a way that \(d(w) = d(v) + 1\) holds; hence \(d(w)\) must have been increased by at least 2 units before the PUSH\((w, v)\). Similarly, no further PUSH\((v, w)\) can become admissible before \(d(v)\) has also been increased by at least 2 units. In particular, \(d(v) + d(w)\) has to increase by at least 4 units between any two consecutive saturating PUSH\((v, w)\)-operations. On the other hand, \(d(v) + d(w) \geq 1\) holds as soon as the first PUSH from \(v\) to \(w\) or from \(w\) to \(v\) is executed. Moreover, \(d(v), d(w) \leq 2|V| - 1\) throughout the algorithm, by Lemma 3.15; hence \(d(v) + d(w) \leq 4|V| - 2\) holds when the last PUSH-operation involving \(v\) and \(w\) occurs. Therefore there are at most \(|V| - 1\) saturating PUSH\((v, w)\)-operations, so that the total number of saturating PUSH-operations cannot exceed \((|V| - 1)|E|\).

Lemma 3.18. During the execution of Algorithm 2, there are at most \(2|V|^2|E|\) non-saturating PUSH-operations.

Proof. Let us introduce the potential

\[
\Theta = \sum_{v=\text{active}} d(v)
\]

and investigate its development during the course of Algorithm 2. After the initialization phase, \(\Theta = 0\); and at the end of the algorithm, we have \(\Theta = 0\) again. Note that any non-saturating PUSH\((v, w)\) decreases \(\Theta\) by at least one unit: because \(r_f(v, w) > e(v)\), the vertex \(v\) becomes inactive so that \(\Theta\) is decreased by \(d(v)\) units; and even if the vertex \(w\) has become active due to the PUSH, \(\Theta\) is increased again by only \(d(w) = d(v) - 1\) units, as the PUSH must have been admissible. Similarly, any saturating PUSH\((v, w)\) increases \(\Theta\) by at most \(2|V| - 1\), since the label of the vertex \(w\) which might again have become active due to this PUSH satisfies \(d(w) \leq 2|V| - 1\), by Lemma 3.15. Let us put together what these observations imply for the entire algorithm. The saturating PUSH-operations increase \(\Theta\) by at most \((2|V| - 1)|E|\) units altogether, by Lemma 3.17; and the RELABEL-operations increase \(\Theta\) by at most \((2|V| - 1)(|V| - 2)\) units, by Lemma 3.15. Clearly, the value by which \(\Theta\) is increased over the entire algorithm must be the same as the value by which it is decreased again. As this happens for the non-saturating PUSH-operations, we obtain an upper bound of \((2|V| - 1)(|V| + |V| - 2)\) for the total number of non-saturating PUSH-operations. Now the bound in the assertion follows easily, using that \(G\) is connected.

After all the Lemmas proved above, we finally have the desired result:

Theorem 3.19. Algorithm 2 terminates after at most \(O(|V|^2|E|)\) admissible operations with a maximal flow.

Let’s run Goldberg and Tarjan’s algorithm through the network given in Section 1.2 again (pictures read from left to right):
Figure 4

It yields same answer as using Ford and Fulkerson’s Algorithm.
ACKNOWLEDGMENTS

It is a pleasure to thank my mentor, Chloe Avery for supervising this project. I would like to thank Tony Qing for inspiring me toward this topic. I would also like to thank Prof. Peter May for organizing this REU and providing valuable feedback and advice, and Daniil Rudenko for running the Apprentice Program.

REFERENCES