Exposition of the Dinitz algorithm

Yefim Dinitz and Avraham A. Melkman
Department of Computer Science
Ben Gurion University of the Negev, 84105 Beer-Sheva, Israel
{dinitz,melkman}@cs.bgu.ac.il

June 6, 2006

1 The Dinitz algorithm

The textbook presents the Edmonds-Karp algorithm for finding the maximum flow in a network. It retains the framework of the Ford-Fulkerson algorithm, but instead of arbitrarily finding an augmenting path it finds a *shortest* augmenting path. Such a path can be found in $\mathcal{O}\big(|E|\big)$ time (how?). Somewhat surprisingly, *the lengths of the shortest augmenting paths found by the algorithm form a non-decreasing sequence*. The textbook shows further that for each given length the number of shortest path of that length encountered by the algorithm cannot exceed $|E|$ (we prove analogous results in Lemma 3.1 and Theorem 3.2). Therefore the Edmonds-Karp algorithm exhausts its finding of shortest augmenting paths from s to t after $\mathcal{O}\big(|E|^2\big)$ time. Since the length of a shortest path does not exceed $|V|-1$, the number of different lengths is no more than $|V|-1$. Consequently the Edmonds-Karp algorithm runs in $\mathcal{O}\big(|V||E|^2\big)$ time.

Like the Edmonds-Karp algorithm, the Dinitz algorithm proceeds by finding shortest augmenting paths, but does so more efficiently: it determines each such path in $\mathcal{O}\big(|V|\big)$ time, resulting in a $\mathcal{O}\big(|V|^2|E|\big)$ algorithm. The Dinitz algorithm achieves this efficiency by reusing the information gathered in a BFS to find not just one but a number of shortest paths. To this end, when looking for shortest augmenting paths of a fixed length ℓ, it maintains an auxiliary network, the **layered network**, which contains precisely **all** current shortest paths from s to t, all of length ℓ. The construction of this layered network, at a total cost of $\mathcal{O}\big(|E|\big)$, is detailed in section 2.

Denote by $d_f(v)$ the shortest distance from s to v in G_f. The Dinitz algorithm is outlined in Algorithm 2. It can be viewed as being based on

the slightly modified version of the Ford-Fulkerson algorithm outlined in Algorithm 1.

Algorithm 1 GENERALIZED FORD-FULKERSON(G, c, s, t)

1: $f \leftarrow 0$;
2: **while** there is a path from s to t in G_f **do**
3: $\quad \triangleright$ *invariant assertion:* f *is a flow in* $\mathcal{N}$*;*
4: $\quad$ find a flow g in $\mathcal{N}_f = (G_f, c_f, s, t)$;
5: $\quad f \leftarrow f + g$;
6: $\triangleright$ *Post-condition of the while loop:* f *is a maximum flow in* $\mathcal{N}$*;*

Note that in the usual version the flow g found in line 4 is the maximum flow along a single path from s to t. As a finger exercise, prove the correctness of this version.

Algorithm 2 DINITZ(G, c, s, t)

1: $f \leftarrow 0$; construct the residual network $\mathcal{N}_f = (G_f, c_f, s, t)$;
2: **while** there is a path from s to t in G_f **do**
3: $\quad \triangleright$ *invariant assertion:* f *is a flow in* $\mathcal{N}$*;*
4: $\quad$ construct the layered network $\mathcal{L}_f = (L_f, c_f, s, t)$;
5: $\quad$ find a blocking flow b for $\mathcal{L}_f$; $\triangleright$ *assertion:* $d_{f+b}(t) > d_f(t)$*;*
6: $\quad f \leftarrow f + b$;
7: $\quad$ construct the residual network $\mathcal{N}_f$;
8: $\triangleright$ *Post-condition of the while loop:* f *is a maximum flow in* $\mathcal{N}$*;*

The blocking flow found in line 5 is constructed by repeatedly extracting an unsaturated path from the layered network, and adding as much flow as possible along the path. Since each such path saturates at least one edge, ablocking flow is found after at most $|E|$ paths have been extracted. Finding a path from s to t in a layered network of length ℓ takes only $\mathcal{O}(\ell) = \mathcal{O}(|V|)$ time, by repeatedly stepping back, starting from t, until s is reached. To ensure that s is indeed reached, the algorithm 4 for finding a blocking flow maintains the invariant that all vertices in the layered network are reachable from s. Thus, when a flow along an augmenting path saturates some edges, teh algorithm removes from the layered network any vertex which cannot be reched from s. This clean-up operation, Algorithm 5, takes a total of $\mathcal{O}(|E|)$ time until a blocking flow is found. Consequently the total cost of finding a blocking flow for the layered network is $\mathcal{O}(|V||E|)$.

2 Construction of the layered network

Definition 2.1. *Let $\mathcal{N}_f = (G_f, c_f, s, t)$ be a residual network which contains a path from s to t. Denote by $d_f(u, v)$ the length of the shortest path from u to v in G_f, and set $\ell = d_f(s, t)$. Let $V_0 = \{s\}$, and let $V_i = \{u : d_f(s, u) = i\}$, $E_i = (V_{i-1} \times V_i) \cap E_f$, $1 \leq i \leq \ell$. The layered network corresponding to $\mathcal{N}_f$ is its subnetwork whose vertices are $V_0 \cup V_1 \cup \ldots \cup V_\ell$ and whose edges are $E_1 \cup \ldots \cup E_\ell$.*

The layered network is constructed by performing a modified BFS on the residual network, which keeps all edges from one layer to the next (instead of only the first edge that leads to some vertex).

Algorithm 3 LayeredNetwork_Construction($\mathcal{N}_f$)

$V_0 \leftarrow \{s\}$; i $\leftarrow 0$;
while $(V_i \neq \emptyset)$ **and** $(t \notin V_i)$ **do**
  $V_{i+1} \leftarrow \emptyset$; $E_{i+1} \leftarrow \emptyset$;
  for each $u \in V_i$ **do**
    for each $v \in V$ such that $(\langle u, v\rangle \in E_f)$ **and** $(v \notin V_j,\ \forall j \leq i)$ **do**
      if $(v \notin V_{i+1})$ **then**
        add v to V_{i+1};
      add $\langle u, v\rangle$ to E_{i+1};
  i $\leftarrow$ i+1
if $(V_i = \emptyset)$ **then**
  return $\mathcal{L}_f = (\emptyset, c_f, s, t)$; $\triangleright$ *assertion: there is no path from s to t in $\mathcal{N}_f$*
$L_f \leftarrow (V_0 \cup \ldots \cup V_i, E_1 \cup \ldots \cup E_i)$;
return $\mathcal{L}_f = (L_f, c_f, s, t)$;

Post-condition: *if t is reachable from s in $\mathcal{N}_f$, then $\mathcal{L}_f$ is the layered network for $\mathcal{N}_f$*

Lemma 2.2. *1. $\mathcal{L}_f$ contains all shortest paths from s to t in $\mathcal{N}_f$ (of length $d_f(t)$).*

2. A path from s to t in $\mathcal{L}_f$ is found in $\mathcal{O}(\ell) = \mathcal{O}(|V|)$ time, by stepping back from t, layer to layer, taking any edge in $\mathcal{L}_f$ until s is reached.

3 The number of iterations is $\mathcal{O}(|V|)$

Lemma 3.1. *CLRS 26.8. Let f be a flow in $\mathcal{N}$, and g a flow in $\mathcal{L}_f$. Then $d_f(t) \leq d_{f+g}(t)$, with equality if and only if each shortest path from s to t in*

$\mathcal{N}_{f+g}$ is also contained in $\mathcal{N}_f$.

Proof. For the proof we will pretend that the construction of $\mathcal{L}_f$, by the modified BFS, is continued after the layer which contains t, until the network contains *all* vertices, even those not on shortest paths from s to t in $\mathcal{N}_f$, and denote the layered network so obtained by $\bar{\mathcal{L}}_f$.

Let $p = \langle u_0, u_1, \ldots, u_k \rangle$ be a shortest path from $s = u_0$ to $t = u_k$ in $\mathcal{N}_{f+g}$. Thus $d_{f+g}(u_i) = i$. We will prove by induction that $d_f(u_i) \leq i$, and that there is equality only if $\langle u_0, u_1, \ldots, u_i \rangle$ is in $\mathcal{N}_f$. This is clearly true for $u_0 = s$.

For the induction step it has to be proven that $d_f(u_{i+1}) \leq i + 1$, with equality holding if and only if the path $\langle u_0, u_1, \ldots, u_{i+1} \rangle$ is also a path in $\bar{\mathcal{L}}_f$. To this end we establish that $d_f(u_{i+1}) - d_f(u_i) \leq 1$, with equality if and only if $\langle u_i, u_{i+1} \rangle \in \bar{\mathcal{L}}_f$.

case 1 $\langle u_i, u_{i+1} \rangle \in \bar{\mathcal{L}}_f$. Then $d_f(u_{i+1}) - d_f(u_i) = 1$.
Simply because all edges in $\bar{\mathcal{L}}_f$ go from one layer to the next.

case 2 $\langle u_i, u_{i+1} \rangle \notin \bar{\mathcal{L}}_f$.

case a $\langle u_i, u_{i+1} \rangle \in \mathcal{N}_f$. Then $d_f(u_{i+1}) - d_f(u_i) \leq 0$.
The only reason that an edge in $\mathcal{N}_f$ does not appear in $\bar{\mathcal{L}}_f$ is that it goes from a layer to the same layer or to a previous one.

case b $\langle u_i, u_{i+1} \rangle \notin \mathcal{N}_f$, *i.e.*, $c_f(u_i, u_{i+1}) = 0$. Then $d_f(u_{i+1}) - d_f(u_i) = -1$.
The fact that $\langle u_i, u_{i+1} \rangle$ is present in $\mathcal{N}_{f+g}$ although it is not present in $\mathcal{N}_f$, means that it is a regret edge resulting from a positive flow of g on the edge $\langle u_{i+1}, u_i \rangle$: $0 < c_{f+g}(u_i, u_{i+1}) = c_f(u_i, u_{i+1}) - g(u_i, u_{i+1}) = g(u_{i+1}, u_i)$. Since g is a flow in $\mathcal{L}_f$, the edge $\langle u_{i+1}, u_i \rangle$ necessarily goes from one layer of $\mathcal{L}_f$ to the next, $d_f(u_i) - d_f(u_{i+1}) = 1$.

∎

Theorem 3.2. *Algorithm* DINITZ *terminates after at most $|V| - 1$ passes through the while-loop.*

Proof. It suffices to prove the assertion of line 5 in Algorithm Dinitz, after a flow b blocking $\mathcal{L}_f$ has been found, $d_f(t) < d_{f+b}(t)$. Since by Lemma 3.1, $d_f(t) \leq d_{f+b}(t)$, let's assume by contradiction that $d_f(t) = d_{f+b}(t)$. According to Lemma 3.1 this happens only if there is a path $p =< u_0, \ldots, u_k >$, with $u_0 = s$ and $u_k = t$, that is a shortest path in both $\mathcal{N}_f$ and $\mathcal{N}_{f+b}$. Since

p is a shortest path in $\mathcal{N}_{f+g}$, $c_{f+b}(u_i, u_{i+1}) > 0, i = 0, \ldots, k-1$. In other words, $c_f(u_i, u_{i+1}) - b(u_i, u_{i+1}) > 0,\ i = 0, \ldots, k-1$. But then the path p is in $\mathcal{N}_f$ yet is not blocked by the flow b, a contradiction to b being a blocking flow. ∎

4 Construction of a blocking flow

Algorithm 4 BLOCKINGFLOW($\mathcal{L}_f$)

```
 1: b ← 0; M ← L_f; c ← c_f;
 2: repeat
 3:    ▷ assertion: s is the only vertex in M with zero indegree;
 4:    find a path p from s to t in M; let its bottleneck capacity be c(p);
 5:    for each edge ⟨u, v⟩ ∈ p do
 6:       b(u, v) ← b(u, v) + c(p); b(v, u) ← −b(u, v);
 7:       c(u, v) ← c(u, v) − c(p);
 8:       if c(u, v) = 0 then
 9:          remove ⟨u, v⟩ from M;
10:          ▷ cleanup of M;
11:          if indegree(v) = 0 then
12:             CLEANFORWARD(v, M);
13: until indegree(t) = 0:
```

Algorithm 5 CLEANFORWARD($u, \mathcal{M}$)

Pre-condition: $indegree(u) = 0$.

```
1: for each v such that ⟨u, v⟩ ∈ M do
2:    remove ⟨u, v⟩ from M;
3:    if (indegree(v) = 0) then
4:       CLEANFORWARD(v, M);
```

Post-condition: Let x be any vertex such that at the start of CLEANFORWARD all *paths back from x in $\mathcal{M}$ end at u. Then all edges going out of x have been removed.*

5 Analysis of the running time