1 Blocking Flows

Idea: In order to find an $s - t$ path in the residual flow network $G_f$, we augment along paths that have a minimum length, as specified by some (binary) length function $\ell$.

1.1 Definitions

Definition 1.1. Given a directed graph $G = (V, E)$, a layered graph $G_L = (V_L, E_L)$ is a subgraph of $G$ that contains vertices of a particular distance from the source, $s$. We let $G_L$ denote a graph of a particular layer (i.e. $G_1$ includes vertices whose minimum distance from $s$ is 1, $G_2$ with a minimum distance of 2, and so on).

Below is a graph $G = (V, E)$ with distance labelings shown on each edge as well as each vertex. We assume unit capacities for all arcs.

![Graph with distance labelings](image)

Definition 1.2. An admissible arc is an arc contained in a minimal length $s - t$ path in the residual flow network $G_f$ that traverse adjacent layers. That is, for any admissible arc $a = (i, j)$, $d(j) = d(i) + 1$.

From the graph in Definition 1.1, we see that the admissible arcs in $G$ are $(s, a)$, $(s, b)$, $(a, c)$, $(a, d)$, $(b, d)$, $(c, t)$, and $(d, t)$.

Definition 1.3. An admissible graph $G_A = (V, E_A)$ is the subgraph of $G$ such that each arc in $E_A$ is an admissible arc.
From our previous example, we construct the admissible graph $G_A = (V, E_A)$.

![Diagram of the admissible graph $G_A$]

**Definition 1.4.** An admissible path $\Gamma$ is an $s-t$ path such that $\forall a \in \Gamma$, $a \in E_A$.

The three admissible paths in $G_A$ are $\Gamma_1 = s - a - c - t$, $\Gamma_2 = s - b - d - t$, and $\Gamma_3 = s - a - d - t$.

**Definition 1.5.** A blocking flow is a flow along admissible paths that saturates at least one arc on each admissible path (that is, $f(a) = u(a)$).

We now will show a blocking flow in the graph $G_A$. Note that the arc $(s, a)$ is contained in $\Gamma_1$ and $\Gamma_3$. Therefore, we can saturate each arc along $\Gamma_1$ and $\Gamma_2$ to achieve a blocking flow in $G_A$ (thus leaving only $(a, d)$ unsaturated).

![Diagram of the admissible graph $G_A$ with blocking flow]

If we augment along the blocking flow $f_b$ found in $G_A$, we are left with the graph:
Which yields the new residual flow network $G_f$:

![Diagram of residual flow network $G_f$]

We see that the distance to the sink $d(t)$ increased from $d(t) = 3$ to $d(t) = \infty$. This observation that the distance to the sink $t$ increases gives rise to the following lemma.

**Lemma 1.1.** Augmenting along a complete blocking flow $f_b$ increases the distance between the source vertex $s$ and the sink $t$.

**Proof.** Suppose $\exists$ an $s - t$ path $\Gamma$ in $G_f$ (that is, $\Gamma \subseteq E_f$), such that $\ell(\Gamma) = d(t)$ in the original graph $G$. If this condition is true, then $\Gamma$ is an admissible path in $G$. Therefore, a valid blocking flow must saturate at least one of its arcs. By the definition of the residual flow network, the saturated edge will not be contained in $G_f$. Therefore, $\Gamma$ no longer exists as a valid $s - t$ path in $G_f$, and therefore the distance from $s$ to $t$ increases. 

To demonstrate our efficient algorithm, we must show that finding a blocking flow in $G_A$ takes $O(m)$ time.

**Lemma 1.2.** A blocking flow $f_b$ can be found in $G_A$ in $O(m)$ time.

**Proof.** Consider the two operations to find a blocking flow (Dinic’s Blocking Flow Algorithm):

**ADVANCE:** Follow an edge forward from the current vertex $i$, and add the traversed edge to the current path $\Gamma$ from $s$.

**RETREAT:** If $\nexists$ an outgoing edge from $i$, backtrack along the previous edge on $\Gamma$ and delete the edge.

We will eventually reach $t$ through repeated application of the ADVANCE and RETREAT, and will therefore get an $s - t$ path $\Gamma$. Once $\Gamma$ has been found, augment along all arcs $a \in \Gamma$, and then delete all edges (that is, reconstruct the residual graph $G_f$ where all the edges will be reversed). In this algorithm, it is clear that each operation (ADVANCE, RETREAT and AUGMENT) will be applied at most once per edge. This is because the unit-capacity arcs will be reversed in the residual network, and will not be considered in the admissible graph $G_A$ which does not contain any of reverse edges in $G_f$. Therefore, finding a blocking flow and augmenting takes at most $O(m)$ time.
Theorem 1.1. In a unit-capacity graph, at most \( n \) augmentations from a blocking flow \( f_b \) results in a max-flow.

Proof. By Lemma 1.1, each blocking flow \( f_b \) increases the \( s-t \) distance by at least 1. Further, the length of any \( s-t \) path is at most \( O(n) \). Therefore, we can conclude that a max-flow will be found after \( n \) iterations of the procedure. By Lemma 1.2, we have shown that the blocking flow can be found in \( O(m) \) steps in \( G_A \). Therefore, we can find the max-flow in a unit capacity graph by the blocking flow method in \( O(mn) \) time. \( \square \)

2 Improved Bounds

Remark: In Section 1, we use Theorem 1.1 to bound the running time of the algorithm described (\( n \) iterations of the procedure in Lemma 1.2). Now, we will prove a lemma which allows us to achieve a tighter bound on the procedure.

Lemma 2.1. If \( s \) and \( t \) are separated by a distance \( k \), the max residual flow is at most \( m/k \).

Proof. The residual network \( G_f \) can be decomposed into edge-disjoint paths by a generic MAX-FLOW procedure. We are given that the distance is \( k \), so each decomposed \( s-t \) path must contain at least \( k \) edges (since \( G \) has unit capacities). Thus, each edge is at most part of one \( s-t \) path. Therefore, the number of flow paths is bounded above by \( m/k \). Since each one contributes a single unit of flow, the max-flow in \( G \) is at most \( m/k \).

Lemma 2.1 allows us to prove a tighter bound on the procedure in section 1. The new bound and the proof is shown below.

Theorem 2.1. The max-flow in a unit capacity graph can be found in \( O(m^{3/2}) \) time.

Proof. Assume that we have already run \( k \) phases of the \( O(m) \) blocking flow procedure from Lemma 1.2. By Lemma 1.1, the distance between \( s \) and \( t \) will be \( k \). Using Lemma 2.1, the max-flow in the residual network \( G_f \) is \( m/k \). We see that at most \( m/k \) more blocking flows are required since each blocking will add at least 1 unit of flow to \( f \). Therefore, \( k + m/k \) blocking flow operations will find a max-flow. We must now choose \( k \) such that \( k + m/k \) is minimized. If we let \( k = m/k \), then we have \( k = m/k \Rightarrow k = m^{1/2} \). Substituting into our previous expression, we get the number of steps as:

\[
    k + m/k = m^{1/2} + \frac{m}{m^{1/2}} = 2 \cdot m^{1/2} = O(m^{1/2})
\]

Combining this with our \( O(m) \) blocking flow procedure (Lemma 1.2), we have now shown that finding a max-flow in a unit capacity graph takes \( O(m^{3/2}) \) steps. \( \square \)

3 The Goldberg-Rao Algorithm

3.1 Intuition

In the previous two sections, a tight \( O(m^{3/2}) \) bound was proved for Dinic’s algorithm. This algorithm relied on finding blocking flows in an admissible graph, and then augmenting along those flows. The correctness of the procedure relied on Lemma 1.1, which demonstrated that the distance between \( s \) and \( t \) increases for each iteration of the algorithm. Now, an attempt at improving this bound will be demonstrated.

Definition 3.1. Let \( \Lambda = \min\{m^{1/2}, n^{2/3}\} \).
We see that when $\Lambda = m^{1/2}$, Dinic’s algorithm runs in $O(\Lambda m)$ (in the form that we have proven it). The other form also holds. In the case we have examined, unit capacity graphs, running $\Lambda$ iterations of a blocking flow algorithm yields the max-flow, and that at any iteration we can add at most $O(\Lambda)$ units of flow.

**Idea:** Assume that arcs from $a_k$ to $a_{k-1}$ have a residual capacity $u_f$ such that $\forall k, u_f(a_k) \leq \Delta$. We see that after $\Lambda$ iterations of the blocking flow procedure we will have a cut with a capacity bounded above by $\Delta \Lambda$ (since we will augment along an edge-disjoint path, which constitutes a complete blocking flow. This means we increase the flow out $s$ by $\Delta$ at each iteration). After this is carried out, we reduce the value of $\Delta$ and repeat the procedure.

**Problem:** How does the algorithm ensure that all arcs $a_k$ to $a_{k+1}$ have residual capacity at most $\Delta$. Furthermore, how does this figure into the notion of “distance” in Dinic’s algorithm?

**Solution:** We modify our notion of “distance” from $s$ as follows. Let:

$$\ell(a) = \begin{cases} 1 & \text{if } u_f(a) \leq \Delta \\ 0 & \text{otherwise} \end{cases}$$

Further, we can define a new distance function $d_{\ell} : V \to \mathbb{Z}^+$ such that $d(i)$ is the distance from $i$ to $t$ using lengths from $\ell$. This leads us to a new definition of admissibility.

**Definition 3.2.** An arc $a = (i, j)$ is admissible if $a \in E_A$ and $d_{\ell}(i) = d_{\ell}(j) + \ell(i, j)$.

However, this definition gives rise to two more problems with this notion of a length function based on residual capacities. Namely, they are:

1. In order for the blocking flow proof argument to work, the $s - t$ distance must increase. This fact is not plainly obvious under the new length function.
2. The graph of admissible arcs may contain cycles that are induced by zero-weight arcs (given that $\ell(a) = 0$ will be true if $u_f(a) > \Delta$). This is problematic, since many of the most efficient blocking flow algorithms of Dinic and Tarjan [1] only work on directed acyclic graphs (DAGs).

It’s easiest to address (2) first. How can we remove zero-length cycles from $G_A$ while preserving the fact that we are finding a max-flow?

**Solution:** We contract, or “shrink,” every strongly-connected component induced by zero-length arcs to a single node, and then run Dinic’s algorithm.¹

¹Tarjan’s Strongly-Connected Components algorithm can decompose a graph into its SCCs in $O(m + n)$ time.
This results in a new contracted admissible graph $G'_A$ with the strongly connected component contracted:

![Graph Diagram]

However, this introduces a new concern. How can we make a valid flow found in the contracted admissible graph $G'_A$ a valid flow in $G_A$? First, it is useful to know that all arcs in the SCC have a capacity of at least $\Delta$ (since $\ell(a) = 0$ for all arcs in question). Suppose, for purposes of developing a technique to handle this, we limit the flow through the SCC at $\frac{\Delta}{4}$.

**Idea:** Build two trees, an in-tree $T_{in}$ and an out-tree $T_{out}$ from the vertices within the contracted SCCs. Use these to route the flow through the component, therefore achieving a valid flow in $G'_A$ that corresponds to a valid flow in $G_A$.

The in-tree and out-tree allow us to route at most $\frac{\Delta}{4}$ on each tree. Therefore, we are only using $\frac{\Delta}{2}$ on each arc within the SCC, which is acceptable since each arc has a capacity $u_f = \Delta$. In order to limit the utilization of the in and out-tree to $\frac{\Delta}{4}$, we first change the goal of each improvement phase. Now, either a blocking flow is found, or we find a flow with value $\Delta$.

### 3.2 Pseudocode

1. function $GOLDBERG$-$RAO(G, s, t, u)$
   \[ G = (V, A), s, t \in V, u : A \rightarrow \mathbb{R}^+ \]
2. let $U = \max_{a \in A} u(a)$, $F = mU$, and $f = 0$
3. let $\Lambda = \min\{m^{1/2}, n^{3/2}\}$
4. while $F \geq 1$ do
5.     $\Delta = F/2\Lambda$
6.     for 1 to $5\Lambda$ do
7.         let $\ell(i, j) = \begin{cases} 
1 & \text{if } u_f(i, j) \leq \Delta \\
0 & \text{otherwise} 
\end{cases}$
8.         let $d_\ell$ be the distance to sink $t$ on $\ell$
9.         construct the admissible graph $G_A$
10.    contact strongly-connected components induced by 0-length arcs in $G_A$ into $G'_A$
11.    run $BLOCKING$-$FLOW$ or $MAX$-$FLOW$ for $|f| = \frac{\Delta}{4}$ to find flow $\tilde{f}$
12.    let $\hat{f}$ be $\tilde{f}$ routed through contracted SCCs
13.    augment $f$ along $\hat{f}$
14. end for
15. $F = F/2$
16. end while
17. end function

### 3.3 Running Time

We must now bound the running time of this procedure. The bulk of the algorithm executes within the while loop. But how can we bound the amount of time it takes for $F < 1$ to be true? We will establish a theorem that allows us to bound the running time based on the relationship between $F$ and the value of the max-flow in $G_f$. 

6
Theorem 3.1. \( F \) is an upper bound on the value of the max-flow in \( G_f \).

Proof. The theorem will be proven by induction on the while-loop, the main body of the algorithm. First, we establish a basis; \( F = mU \) is initially an upper bound on the total amount of flow (it assumes \( \forall a \in A, u(a) = U \)). Next, we consider the inductive step. We know that an iteration of the while loop will terminate after either (1) a blocking flow is found \( \Lambda \) times, or (2) a flow of value \( \frac{\Delta}{2} \) is found at least \( 4\Lambda \) times.

If (1) is true, then we see by our observation about the in and out-trees that we increase the flow by \( \frac{\Delta}{2} \cdot 4\Lambda = 2\Lambda\Delta = \frac{F}{2} \). Since there was at most \( F \) flow in \( G_f \) at the beginning of this iteration of the while-loop, the flow remaining after this iteration is at most \( \frac{F}{2} \).

If (2) is true, then assuming the validity of Lemma 3.1, we know that \( d_f(s) \geq \Lambda \). This implies that \( \exists \) an \( (S, T) \) cut in the residual graph of capacity \( \Lambda\Delta = \frac{F}{2} \).

This gives us an important corollary which allows us to guarantee the termination of the algorithm.

Corollary 3.1. **GOLDBERG-RAO** terminates due to Theorem 3.1.

Proof. The fact that the algorithm terminates is a direct consequence of Theorem 3.1. Since we know that \( F \) is reduced in value to \( F/2 \) during each iteration of the while-loop, we see that the sequence of values formed by the successive reduction in magnitude of \( F \) is decreasing. Therefore, we can see that the condition \( F \geq 1 \) will eventually fail when enough successive divisions have taken place, and therefore we see that the procedure terminates.

Finally, we must bound the running time of the algorithm. In order to achieve a tight bound, we must prove a final theorem.

Theorem 3.2. The overall running time of **GOLDBERG-RAO** is \( \tilde{O}(\Lambda m) \).\(^2\)

Proof. Initialization will take \( O(1) \) time. Now, we see that the main body of the loop will execute \( \lg(mU) \) times (since we repeatedly halve \( F \)). During each improvement phase, we run either our blocking flow procedure or our max-flow procedure in the contracted graph \( 5\Lambda \) times. Each inner loop will require \( O(m \lg n) \) time to find a blocking flow or find a valid \( \frac{\Delta}{4} \) flow through the contracted graph. Therefore, this gives us an overall running time of \( O(\Lambda m \lg(mU) \lg(n)) = \tilde{O}(\Lambda m) \). This also implies a ballpark bound of \( o(mn) \).

References


\(^2\)The \( \tilde{O} \) notation ignores logarithmic factors in the overall running time.