



Chapter 6 Graph Algorithms

Algorithm Theory WS 2016/17

Fabian Kuhn

Flow Network





Residual Graph



Given a flow network G = (V, E) with capacities c_e (for $e \in E$)

For a flow f on G, define directed graph $G_f = (V_f, E_f)$ as follows:

- Node set $V_f = V$
- For each edge e = (u, v) in E, there are two edges in E_f :
 - forward edge e = (u, v) with residual capacity $c_e f(e)$
 - backward edge e' = (v, u) with residual capacity f(e)





Residual Graph G_f



FREBURG

Definition:

An augmenting path P is a (simple) s-t-path on the residual graph G_f on which each edge has residual capacity > 0.

bottleneck(P, f): minimum residual capacity on any edge of the augmenting path P

Augment flow f to get flow f':

• For every forward edge (u, v) on P:

 $f'((u,v)) \coloneqq f((u,v)) +$ **bottleneck**(P, f)

• For every backward edge (u, v) on P:

 $f'((v, u)) \coloneqq f((v, u)) - \text{bottleneck}(P, f)$

Ford-Fulkerson Algorithm

- Improve flow using an augmenting path as long as possible:
- 1. Initially, f(e) = 0 for all edges $e \in E$, $G_f = G$
- 2. while there is an augmenting s-t-path P in G_f do
- 3. Let P be an augmenting s-t-path in G_f ;
- 4. $f' \coloneqq \operatorname{augment}(f, P);$
- 5. update f to be f';
- 6. update the residual graph G_f
- 7. **end**;



Ford-Fulkerson Running Time



Theorem: If all edge capacities are integers, the Ford-Fulkerson algorithm can be implemented to run in O(mC) time.

Proof:

s-t Cuts



Definition:

An *s*-*t* cut is a partition (A, B) of the vertex set such that $s \in A$ and $t \in B$



Ford-Fulkerson Gives Optimal Solution



Lemma: If f is an s-t flow such that there is no augmenting path in G_f , then there is an s-t cut (A^*, B^*) in G for which

 $|f| = c(A^*, B^*).$

Theorem: The flow returned by the Ford-Fulkerson algorithm is a maximum flow.

Theorem: Given a flow \underline{f} of maximum value, we can compute an \underline{s} - \underline{t} cut of minimum capacity in O(m) time.

Theorem: (Max-Flow Min-Cut Theorem)

In every flow network, the maximum value of an s-t flow is equal to the minimum capacity of an s-t cut.



Theorem: (Integer-Valued Flows)

If all capacities in the flow network are integers, then there is a maximum flow f for which the flow f(e) of every edge e is an integer.

Improved Algorithm

FREBURG

Idea: Find the best augmenting path in each step

- best: path P with maximum bottleneck(P, f)
- Best path might be rather expensive to find
 → find almost best path
- Scaling parameter Δ : (initially, $\Delta = \text{"max } c_e$ rounded down to next power of 2")
- As long as there is an augmenting path that improves the flow by at least Δ, augment using such a path
- If there is no such path: $\Delta \coloneqq \Delta /_2$

Running Time: Scaling Max Flow Alg.



Theorem: The number of augmentations of the algorithm with scaling parameter and integer capacities is at most $O(m \log C)$. The algorithm can be implemented in time $O(m^2 \log C)$.

 $((\mathbf{W} \cdot \mathbf{W}))$

Maximum Flow Applications



- Maximum flow has many applications
- Reducing a problem to a max flow problem can even be seen as an important algorithmic technique
- Examples:
 - related network flow problems
 - computation of small cuts
 - computation of matchings
 - computing disjoint paths
 - scheduling problems
 - assignment problems with some side constraints

- ...

Undirected Edges and Vertex Capacities



ov.

5

Undirected Edges:

• Undirected edge {*u*, *v*}: add edges (*u*, *v*) and (*v*, *u*) to network

Vertex Capacities:

- 4 00 8 00 M
- Not only edges, but also (or only) nodes have capacities
- Capacity $\underline{c_v}$ of node $v \notin \{\underline{s,t}\}$: $f^{\text{in}}(v) = f^{\text{out}}(v) \le \underline{c_v}$
- Replace node v by edge $e_v = \{v_{in}, v_{out}\}$:



Minimum *s*-*t* Cut





s-*t* cut: Partition (A, B) of V such that $s \in A, t \in B$

Size of cut (A, B): number of edges crossing the cut



Edge Connectivity



Definition: A graph G = (V, E) is k-edge connected for an integer $k \ge 1$ if the graph $G_X = (V, E \setminus X)$ is connected for every edge set



Goal: Compute edge connectivity $\lambda(G)$ of G (=) (and edge set X of size $\lambda(G)$ that divides G into ≥ 2 parts)

- minimum set X is a minimum s-t cut for some $s, t \in V$
 - Actually for all s, t in different components of $G_X = (V, E \setminus X)$ Funning Huma: $O(m n^2)$
- Possible algorithm: fix s and find min s-t cut for all $t \neq s$

Algorithm Theory, WS 2016/17

Fabian Kuhn

Minimum s-t Vertex-Cut



Given: undirected graph G = (V, E), nodes $s, t \in V$

<u>s-t vertex cut</u>: Set $X \subset V$ such that $s, t \notin X$ and s and t are in different components of the sub-graph $G[V \setminus X]$ induced by $V \setminus X$

Size of vertex cut: |X|

Objective: find *s*-*t* vertex-cut of minimum size

- Replace undirected edge $\{u, v\}$ by (u, v) and (v, u)
- Compute max s-t flow for edge capacities <u>and node capacities</u>

$$c_v = 1$$
 for $v \neq s, t$

- Replace each node v by v_{in} and v_{out} :
- Min edge cut corresponds to min vertex cut in G

Vertex Connectivity



Definition: A graph G = (V, E) is k-vertex connected for an integer $k \ge 1$ if the sub-graph $G[V \setminus X]$ induced by $V \setminus X$ is connected for every edge set



Goal: Compute vertex connectivity $\kappa(G)$ of G(and node set X of size $\kappa(G)$ that divides G into ≥ 2 parts)

• Compute minimum <u>s-t vertex</u> cut for one fixed s and all $t \neq s$?

Edge-Disjoint Paths



Given: Graph G = (V, E) with nodes $s, t \in V$ unweighted, dir. or under.

Goal: Find as many edge-disjoint *s*-*t* paths as possible



Solution:

• Find max s-t flow in G with edge capacities $c_e = 1$ for all $e \in E$

Flow f induces |f| edge-disjoint paths:

- Integral capacities \rightarrow can compute integral max flow f
- Get |f| edge-disjoint paths by greedily picking them
- Correctness follows from flow conservation $f^{in}(v) = f^{out}(v)$



Algorithm Theory, WS 2016/17

9

Vertex-Disjoint Paths



Given: Graph G = (V, E) with nodes $\underline{s, t} \in V$

Goal: Find as many internally vertex-disjoint *s*-*t* paths as possible

Solution:



• Find max *s*-*t* flow in *G* with node capacities $c_v = 1$ for all $v \in V$

Flow f induces |f| vertex-disjoint paths:

- Integral capacities \rightarrow can compute integral max flow f
- Get |f| vertex-disjoint paths by greedily picking them
- Correctness follows from flow conservation $f^{in}(v) = f^{out}(v)$



B

• +

Theorem: (edge version)

For every graph G = (V, E) with nodes $s, t \in V$, the size of the minimum s-t (edge) cut equals the maximum number of pairwise edge-disjoint paths from s to t.

Theorem: (node version)

For every graph G = (V, E) with nodes $s, t \in V$, the size of the minimum <u>s-t vertex</u> cut equals the maximum number of pairwise internally vertex-disjoint paths from s to t



 Both versions can be seen as a special case of the max flow min cut theorem

Baseball Elimination



	l	<u> </u>	<u>ل</u>					
Team	Wins	Losses	To Play	Against = r_{ij}				
i	w _i	l _i	r _i	NY	Balt.	Т. Вау	Tor.	Bost.
New York	81	70	11	-	2	5	2	3
Baltimore	79	77	6	2	-	2	1	1
Tampa Bay	79	75	8	5	2	-	1	1
Toronto	76	80	6	2	1	1	-	2
Boston	71	84	7	3	1	1	2	-

- Only wins/losses possible (no ties), winner: team with most wins
- Which teams can still win (as least as many wins as top team)?
- Boston is eliminated (cannot win):

- Boston can get at most 78 wins, New York already has 81 wins

- If for some $i, j: w_i + r_i < w_j \rightarrow \text{team } i$ is eliminated
- Sufficient condition, but not a necessary one!

Algorithm Theory, WS 2016/17

Fabian Kuhn

Baseball Elimination



Team	Wins	Losses	To Play	Against = r_{ij}				
i	W _i	ℓ_i	r _i	NY	Balt.	Т. Вау	Tor.	Bost.
New York	81	70	2 22 (2	-	2	5	2	3
Baltimore	79	77	6	2	-	2	1	1
Tampa Bay	79	75	\$9	5	2	-	1	1
Toronto	76	80	6	2	1	1	-	2
Boston	71	84	7	3	1	1	2	-

- Can Toronto still finish first?
- Toronto can get <u>82</u> > 81 wins, but: NY and Tampa have to play 5 more times against each other
 → if NY wins two, it gets 83 wins, otherwise, Tampa has 83 wins
- Hence: Toronto cannot finish first
- How about the others? How can we solve this in general?

Max Flow Formulation $\omega_3 + \delta$



• Can team 3 finish with most wins?



• Team 3 can finish first iff all source-game edges are saturated

Reason for Elimination



AL East: Aug 30, 1996

Team	Wins	Losses	To Play	Against = r_{ij}				
i	wi	ℓ_i	r _i	NY	Balt.	Bost.	Tor.	Detr.
New York	75	59	28	-	3	8	7	3
Baltimore	71	63	28	3)/	2	7	4
Boston	69	66	27	8	2		0	0
Toronto	63	72	27	7	7_	0	-/0	0
Detroit	<u>49</u>	86	27	3	4	0	0	-

- Detroit could finish with 49 + 27 = 76 wins
- Consider $R = \{NY, Bal, Bos, Tor\}$
 - Have together already won w(R) = 278 games
 - Must together win at least r(R) = 27 more games
- On average, teams in R win $\frac{278+27}{4} = 76.25$ games

Algorithm Theory, WS 2016/17

Fabian Kuhn



Certificate of elimination:



Team $x \in X$ is eliminated by R if

$$\frac{w(R) + r(R)}{|R|} > w_x + r_x.$$