Algorithmics

Sebastian Iwanowski
FH Wedel

4. Graph algorithms
4.3 Computation of maximum flows in s/t-networks

Algorithmics 4

4.3 Computation of maximum flows in s/t-networks

Notation

Def.: s/t-network (q/s-Netzwerk):

Complete directed graph (V,E) with nonnegative edge capacities c(e) for all edges e and a selected source vertex s (Quelle q) and a selected target vertex t (Senke s)

Def.: flow f: function $E \rightarrow \mathbb{N}$ where

- $f(e) \leq c(e)$ for all edges e
- $f(u,v) = - f(v,u)$
- For all vertices $v \neq s,t$ the following holds:
  The sum of all flows from v to all neighbors is 0.

Def.: value |f| of a flow:

net flow out of s resp. net flow into t (both values must be equal)

References:

Cormen, ch. 26.1 (flow networks)
Alt, Kap. 4.5.1
Turau, Kap. 6.1 (siehe auch Ausarbeitung und Vortrag Seminararbeit Claudia Padberg)

Algorithmics 4

4.3 Computation of maximum flows in s/t-networks

Notation

Def.: Augmenting path (Erweiterungsweg) of a flow f:

Path vom s to t where the following holds for each edge (u,v): $f(u,v) < c(u,v)$
The value $c(u,v) – f(u,v)$ is called the remainder capacity.

Note: $f(u,v)$ may be negative which means that $f(v,u) > 0$.
In this case, $f(v,u) = c(v,u)$ is permitted.

Def.: Residual network (Restegraph, Restnetz) G_f:

For each edge (u,v) with positive remainder capacity in G, insert an edge $(u,v) \in G_f$ where the capacity is equal to that remainder capacity.

For each edge (u,v) with positive flow f(u,v) in G, insert an edge $(v,u) \in G_f$ where $c(v,u)= f(u,v)$

Prop. 1: A path p is an augmenting path in G $\Leftrightarrow$ p is a directed path from s to t in G_f

Prop. 2: A flow f may be increased by the *residual flow* (Restfluss) whose value is the minimum capacity of a directed path from s to t in G_f.

References:

Cormen, ch. 26.2 (Ford-Fulkerson method)
Alt, Kap. 4.5.2
Turau, Kap. 6.1, 6.3 (Restegraph) (siehe auch Ausarbeitung und Vortrag Seminararbeit C. Padberg)

Algorithmics 4

4.3 Computation of maximum flows in s/t-networks

Notation

Def.: s/t-cut (X,Y) (q/s-Schnitt):

Partition of vertices in G such that $s \in X$ und $t \in Y$

Def.: capacity c(X,Y) of an s/t-cut:

Sum of all capacities c(u,v) where $u \in X$ and $v \in Y$

Def.: flow f(X,Y) of an s/t-cut:

Sum of all flows f(u,v) where $u \in X$ and $v \in Y$

Prop. 1: For each s/t-cut (X,Y) the following holds: $|f| = f(X,Y)$

Prop. 2: $|f| \leq \min \{c(X,Y); (X,Y) \text{ is s/t-cut}\}$

References:

Cormen, ch. 26.2 (Ford-Fulkerson method)
Turau, Kap. 6.1 (siehe auch Ausarbeitung und Vortrag Seminararbeit Claudia Padberg)

Algorithmics 4

4.3 Computation of maximum flows in s/t-networks

Max-flow min-cut theorem (Ford-Fulkerson theorem)

The following propositions are equivalent:

- f is a maximum flow in G
- There is no augmenting path for f in G
- There is an s/t-cut (X,Y) where |f| = c(X,Y)

Proof:

Circular argument:

1) => 2) trivial
2) => 3) will be shown in class (according to Cormen)
3) => 1) follows by Prop.2 of last slide

References:

Cormen, ch. 26.2 (Ford-Fulkerson method)
Turau, Kap. 6.2 (anderer Beweis)

Algorithmics 4

4.3 Computation of maximum flows in s/t-networks

Algorithm of Edmonds-Karp: (using the notation of Skript Alt)

1) Initialize f by 0 for all edges.
Repeat
    2a) Compute residual graph G_f
    2b) Find augmenting path in G_f **with breadth first search**
    3) Increase f by the residual flow of the augmenting path (Prop. 2, slide 3)
until no augmenting path exists

Correctness: follows by Ford-Fulkerson theorem

Time complexity: $O(nm^2)$

Outline of time complexity proof:

Each operation of type 2a), 2b) and 3) costs time O(m) (easy to see)

There are O(nm) loop iterations:
Each augmenting path has got a critical edge. Each edge can be critical at most O(n) times.
There are m edges.

References:

Cormen, ch. 26.2 (Ford-Fulkerson method)
Alt, Kap. 4.5.4
Turau, Kap. 6.3 (mit Pseudocode) (siehe auch Seminararbeit Claudia Padberg)

Algorithmics 4

4.3 Computation of maximum flows in s/t-networks

Algorithm of Edmonds-Karp:

Details of time complexity proof:

Def.: Let $\delta_f(u,v)$ be the minimum number of edges between u and v in the residual network G_f

For a *breadth first search*, a source s and a target t, the following holds:

Lemma 1: Each path in a graph found by breadth first search starting at a source s has got the minimum number of edges.

Lemma 2: For each edge (u,v) of a path P_f in the residual network G_f found by breadth first search, The following holds: $\delta_f(s,v) = \delta_f(s,u) + 1$

Lemma 4.5.8 / 26.8: (Monotonicity) Let f_1, f_2 be two flows subsequently generated by Edmonds-Karp: Then for all $v \neq s,t$: $\delta_{f_1}(s,v) \leq \delta_{f_2}(s,v)$

Lemma 4.5.9 / 26.9: (O(n) theorem) Each edge will be at most n/2 times a critical one.

References:

Cormen, ch. 26.2 (Ford-Fulkerson method)
Alt, Kap. 4.5.4
Turau, Kap. 6.3 (anderer Beweisaufbau und Notation)

Algorithmics 4

4.3 Computation of maximum flows in s/t-networks

Algorithm of Edmonds-Karp:

Proof of Lemma 4.5.8:

Let for v ≠ s,t the following hold: $\delta_{f_1}(s,v) > \delta_{f_2}(s,v)$ **(*)**

Let v be the node with property (*) having minimal distance from s in G_{f_2},
i.e. for all u with $\delta_{f_2}(s,u) < \delta_{f_2}(s,v)$, the following holds: $\delta_{f_1}(s,u) \le \delta_{f_2}(s,u)$ **(**)**

Let P_2 be the shortest path from s to v in G_{f_2}, and let u be the predecessor of v on that path.
Thus, $(u,v) \in G_{f_2}$, and u satisfies (**).

Consider the following two cases:

a) $f_1(u,v) < c(u,v)$.

This implies: $(u,v) \in G_{f_1}$ => $\delta_{f_1}(s,v) \le \delta_{f_1}(s,u) + 1 \le \delta_{f_2}(s,u) + 1 = \delta_{f_2}(s,v)$ contradicting (*)

(first ≤: because v can be reached via (u,v) in G_{f_1}; second ≤: (**); =: because (u,v) is part of the shortest path in G_{f_2})

b) $f_1(u,v) = c(u,v)$.

This implies: $(u,v) \notin G_{f_1}$ Since $(u,v) \in G_{f_2}$, $f_2(v,u) > 0$ and (since $(u,v) \notin G_{f_1}$) $f_1(v,u) = 0$

Thus, (v,u) is part of an augmenting path which was used in order to increase f_1 to obtain f_2.

By Lemma 2, $\delta_{f_1}(s,v) = \delta_{f_1}(s,u) - 1 \le \delta_{f_2}(s,u) - 1 = \delta_{f_2}(s,v) - 2 < \delta_{f_1}(s,v)$ contradiction !

(≤: (**); =: Lemma 2; <: (*))

In either case, we get a contradiction
which proves that for all v ≠ s,t the following holds: $\delta_{f_1}(s,v) \le \delta_{f_2}(s,v)$

Algorithmics 4

4.3 Computation of maximum flows in s/t-networks

Algorithm of Edmonds-Karp:

Proof of Lemma 4.5.9:

Let (u,v) be a critical edge in an augmenting path for flow f_1.

By Lemma 2, $\delta_{f_1}(s,v) = \delta_{f_1}(s,u) + 1$ **(**)**

If (u,v) becomes a critical edge again implies:
(v,u) is in an augmenting path some time in between for a flow f_2.

Condider the following:

$$\delta_{f_2}(s,u) \underset{\text{Lemma 2}}{=} \delta_{f_2}(s,v) + 1 \underset{\text{Lemma 4.5.8}}{\geq} \delta_{f_1}(s,v) + 1 \underset{(**)}{=} \delta_{f_1}(s,u) + 2$$

Thus, the distance from u to the source s has increased by at least 2

This can happen at most n/2 times, because the distance is never greater than n. **q.e.d.**

Algorithmics 4

4.3 Computation of maximum flows in s/t-networks

Algorithm of Dinic

Notation:

Def.: Level graph L_f: **(Turau: Niveaugraph G'_f)**

Delete all edges (u,v) from G_f where $\delta_f(s,v) \leq \delta_f(s,u)$

Def.: blocking flow:

A flow where each path from s to t has got a critical edge.

Theorem: f is maximal $\Rightarrow$ f is blocking

Def. (Increase of a flow f by a flow r in L_f):

Let r be a flow in L_f. For each edge e, let $f'(e) = f(e) + r(e) - \overleftarrow{r}(e)$

Theorem: $|f'| = |f| + |r|$

References:

Cormen, ch. 26.4 (push relabel algorithms)
Turau, Kap. 6.4 (siehe auch Ausarbeitung und Vortrag Seminararbeit C. Padberg)
Alt, Kap. 4.7

Algorithmics 4

4.3 Computation of maximum flows in s/t-networks

Algorithm of Dinic

1) Initialize f by 0 for all edges.
Repeat
2a) Compute L_f
2b) Search for a blocking flow r in L_f
3) Increase f by the blocking flow r
until no blocking flow exists (t cannot be reached anymore in L_f from s)

Difference to Edmonds-Karp:
Maximize each path in the flow, not just one.

Time complexity: $O(n^2m)$ Improvement in Turau: $O(n^3)$

Outline of time complexity proof:

In each iteration, $\delta_f(s,t)$ is increased by at least 1 $\Rightarrow$ there are O(n) loop iterations

2a) and b) may be combined with a repeated depth first search: O(nm)
Improvement in Turau: $O(n^2)$

References for the details:

Cormen, ch. 26.4 (push relabel algorithms: with proof of correctness)
Turau, Kap. 6.4 (siehe auch Ausarbeitung und Vortrag Seminararbeit C. Padberg)
Alt, Kap. 4.7