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-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)

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)

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, f´ be two flows subsequently generated by Edmonds-Karp: Then for all $v \neq s,t$: $\delta_f(s,v) \leq \delta_{f'}(s,v)$

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 Dinic

Notation:

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

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

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