Pearls of Algorithms

1. Bipartite matching and network flows

Our goal is to learn efficient solutions of two important problems:

The computation of

1) a maximum (weighted) matching in bipartite graphs, and

2) a maximum flow in a network.

In practice, each problem occur often as a subproblem which has to be solved.

Moreover, we shall be concerned with fundamental methods of the development of algorithms as

- the augmenting solution method and
- the primal-dual method.

1.1 Bipartite Matching

References

C. H. Papadimitriou, K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity Prentice-Hall 1982.

N. Blum, Algorithmen und Datenstrukturen, Oldenbourg 2004 (in German).

1.1.1 Definitions and the general method

A graph $G = (V, E)$ consists of a finite, nonempty set of nodes V and a set of edges E. G is either directed or undirected. In the (un-)directed case, each edge is an (un-)ordered pair of distinct nodes. A graph $G = (V, E)$ is bipartite if V can be partitioned into disjoint nonempty sets A and B such that for all $(u, v) \in E$, $u \in A$ and $v \in B$, ~~or~~ vice versa. Then we often write $G = (A, B, E)$.

A path P from $v \in V$ to $w \in V$ is a sequence of nodes

$$P = v = v_0, v_1, \ldots, v_k = w$$

which satisfies $(v_i, v_{i+1}) \in E$, for $0 \leq i < k$. The length $|P|$ of P is the number k of edges on P.

P is simple if $v_i \neq v_j$, for $0 \leq i < j \leq k$.

If there exists a path from v to w (of length 1) v is call a (direct) predecessor of w, and w is called a (direct) successor of v.

Let $G = (V, E)$ be an undirected graph. $M \subseteq E$ is a matching of G if no two edges in M have a common node. A matching M is maximal if there exists no $e \in E \setminus M$ such that $M \cup \{e\}$ is a matching. A matching M is maximum if there exists no matching $M' \subseteq E$ of larger size.

Given an undirected graph $G = (V, E)$, the maximum matching problem is finding a maximum matching $M \subseteq E$.

A path $P = v_0, v_1, \ldots, v_k$ is M-alternating if it contains alternately edges in M and in $E \setminus M$. A node $v \in V$ is M-free if v is not incident to an edge in M.

Let $P = v_0, v_1, \ldots, v_k$ be a simple M-alternating path. P is M-augmenting if v_0 and v_k are M-free. Let P be an M-augmenting path in G. Then $M \oplus P$ denotes the symmetric difference of M and P; i.e.

$$M \oplus P := M \setminus P \cup P \setminus M.$$

Note that $M \oplus P$ is a matching of G, and $|M \oplus P| = |M| + 1$.

Theorem 1.1 (Berge 1957)
Let $G = (V, E)$ be an undirected graph and $M \subseteq E$ be a matching of G. Then M is maximum if and only if there exists no M-augmenting path in G.

Proof:

„⇒"

If M is maximum then no M-augmenting path P can exist. Otherwise, $M \oplus P$ would be a matching of size $|M| + 1$, a contradiction.

„⇐"

Let M be a matching of G. Assume that in G no M-augmenting path exists.

Let M' be any maximum matching of G. We have to prove $|M| = |M'|$

Assume $|M| < |M'|$.

Let us consider the subgraph

$$G' := (V, M \oplus M').$$

M and M' matchings $\Rightarrow$

$$\deg_{G'}(v) \leq 2 \quad \forall v \in V.$$

Furthermore

$$\deg_{G'}(v) = 2 \Rightarrow$$

One of the two edges with end node v is in M and the other such an edge is in M'.

Hence, the connected components of G' are of the following kinds.

i) isolated nodes,

ii) cycles of even length with alternately edges in M and in M', or

iii) paths with alternately edges in M and in M'.

$|M'| > |M| \Rightarrow$

At least one of the connected components is a path P with more edges in M' than in M.

$\Rightarrow$

The first edge of P and also the last edge of P are in M'.

$\Rightarrow$

Both end nodes of P are M-free.

$\Rightarrow$

P is an M-augmenting path.

This is a contradiction to the assumption that no M-augmenting path in G exists. ∎

Berge's Theorem implies the following general method for finding a maximum matching in a graph G.

Algorithm MAXIMUM MATCHING

Input: An undirected graph $G = (V, E)$, and a matching $M \subseteq E$ (possibly $M = \emptyset$)

Output: A maximum matching M_{max}

Method:

```
while there exists an M-augmenting path
    do
        construct such a path P;
        M := M ⊕ P
    od;
M_max := M.
```

The key problem is now this:

How to find an M-augmenting path P, if such a path exists?

Next we will define the maximum weighted matching problem.

Let $G = (V, E)$ be an undirected graph. If we associate with each edge $(i,j) \in E$ a weight $w_{ij} > 0$ then we obtain a weighted undirected graph $G = (V, E, w)$.

The weight $w(M)$ of a matching M is the sum of the weights of the edges in M. A matching $M \subseteq E$ has maximum weight if

$$\sum_{(i,j) \in M'} w_{ij} \leq \sum_{(i,j) \in M} w_{ij} \quad \forall \text{ matchings } M' \subseteq E.$$

Given a weighted undirected graph $G = (V, E, w)$, the maximum weighted matching problem is finding a matching $M \subseteq E$ of maximum weight.

1.1.1 The unweighted case

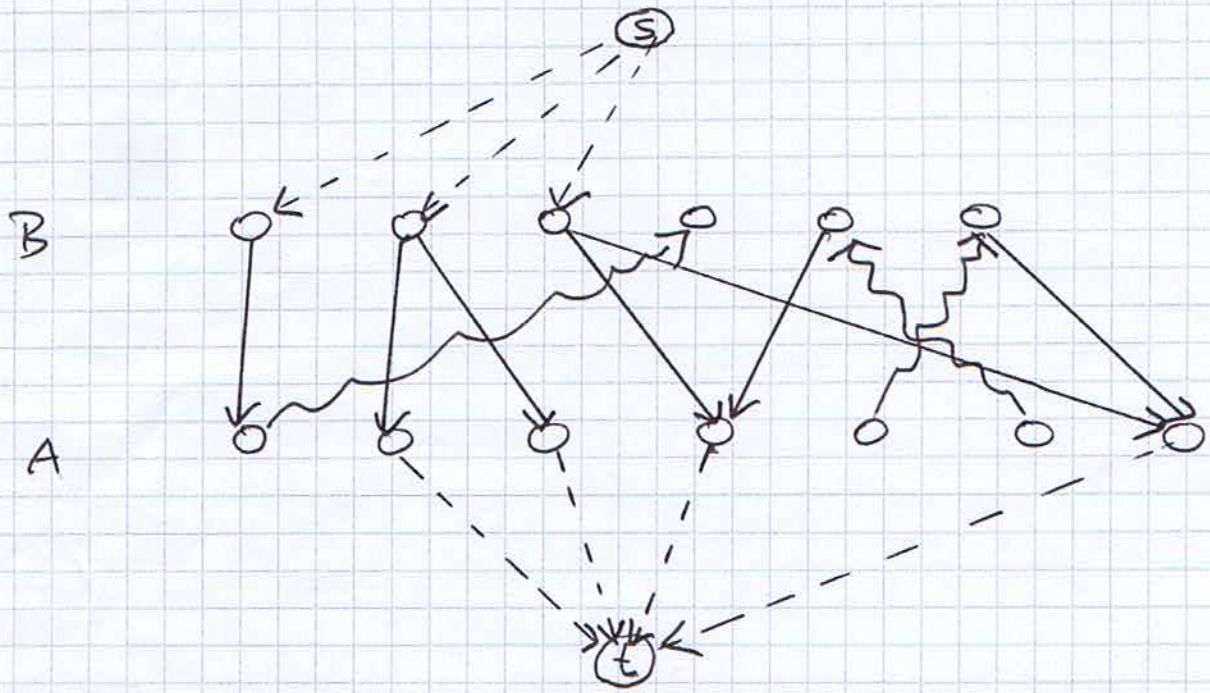
We solve the key problem above in the following way.

1) We reduce the key problem to a reachability problem in a directed, bipartite graph $G_M = (A', B', E_M)$.

2) We solve this reachability problem constructively.

Given the bipartite graph $G = (A, B, E)$ and the matching $M \subseteq E$, we direct the edges in M from A to B and the edges in $E \setminus M$ from B to A. Additionally, we add two new nodes s and t to $A \cup B$, add for each M-free node $b \in B$ the edge (s, b) to E_M, and add for each M-free node $a \in A$ the edge (a, t) to E_M

$\rightsquigarrow$



$\rightsquigarrow$ edge in M

$\longrightarrow$ edge in $E \setminus M$

$\dashrightarrow$ additional edge

$G_M = (A', B', E_M)$ where

$A' = A \cup \{s\}$, $B' = B \cup \{t\}$, $s, t \notin A \cup B$, $s \neq t$

and

$$E_M = \{(u,v) \mid (u,v) \in M \text{ and } u \in A, v \in B\}$$
$$\cup \{(x,y) \mid (x,y) \in E \setminus M \text{ and } x \in B, y \in A\}$$
$$\cup \{(s,b) \mid b \in B \text{ } M\text{-free}\}$$
$$\cup \{(a,t) \mid a \in A \text{ } M\text{-free}\}.$$

Lemma 1.1

Let $G = (A, B, E)$ be a bipartite graph, $M \subseteq E$ be a matching and $G_M = (A', B', E_M)$ constructed as above. Then there exists an M-augmenting path in G if and only if there is a simple path from s to t in G_M.

Proof:

"$\Rightarrow$"

Let $P = v_0, v_1, v_2, \ldots, v_k$ be an M-augmenting path in G.

W.l.o.g. we can assume $v_0 \in B$.

$\Rightarrow$

1) $v_i \in A$ for i odd,
2) $v_i \in B$ for i even,
3) $v_0 \in B$ M-free and $v_k \in A$ M-free,
4) $(v_i, v_{i+1}) \in M$ if i odd, and
5) $(v_i, v_{i+1}) \in E \setminus M$ if i even.

$\Rightarrow$

By the construction of G_M, the path

$$P' := s, v_0, v_1, v_2, \ldots, v_k, t$$

is a simple path from s to t in G_M.

„$\Leftarrow$" analogously ∎

The following algorithm constructs an M-augmenting path if such a path exists.

Algorithm FIND AUG PATH

Input: A bipartite graph $G = (A, B, E)$ and a matching $M \subseteq E$.

Output: An M-augmenting path P if such a path exists

Method:

(1) Construct $G_M = (A', B', E_M)$.
(2) Apply a depth first search with start node s to G_M.
(3) If depth first search reaches the node t then put out the path P contained in the stack without the nodes s and t.

Next we shall analyze the time used by the algorithm FIND AUG PATH.

Let
$$n = |A| + |B| \quad \text{and} \quad m = |E|.$$

Obviously, G_M can be constructed in $O(n+m)$ time. Depth first search uses also only $O(n+m)$ time. Hence, the algorithm FINDAUGPATH constructs an M-augmenting path in $O(n+m)$ time if such a path exists.

The symmetric difference $M := M \oplus P$ can be performed in $O(|P|) = O(n)$ time. At most $\lfloor \frac{n}{2} \rfloor$ augmentations can be performed. Altogether, we have proven the following theorem.

Theorem 1.2.
Let $G = (A, B, E)$ be a bipartite graph which contains no isolated nodes. Then we can compute a maximum matching of G in $O(n \cdot m)$ time where $n = |A| + |B|$ and $m = |E|$.

In general graphs $G = (V, E)$, the solution of the key problem above is much more complicated.

1.1.2 The weighted case

Let $G = (V, E, w)$ be a weighted undirected graph and $M \subseteq E$ be a matching of G. Let P be a simple M-alternating path.

The gain $\Delta(P)$ with respect to P denotes the weight change of the matching after the performance of the symmetric difference $M := M \oplus P_j$ i.e.,

$$\Delta(P) = \sum_{(i,j) \in P \wedge E \setminus M} w_{ij} - \sum_{(i,j) \in P \wedge M} w_{ij}.$$

The following theorem gives an exact characterization of a maximum weighted matching of a weighted undirected graph.

Theorem 1.3

Let $G = (V, E, w)$ be a weighted undirected graph and $M \subseteq E$ be a matching of G. Then M is of maximum weight if and only if none of the following two cases is fulfilled:

1) There is an M-alternating simple cycle P of even length with $\Delta(P) > 0$.
2) There is a simple M-alternating path P such that
   i) $(v,w) \in M \Rightarrow (v,w) \in P$ or $v, w \notin P$ and
   ii) $\Delta(P) > 0$.

Proof:

"$\Rightarrow$"

Let $M \subseteq E$ be a matching of maximum weight. Obviously, none of the both cases can be

fulfilled. Otherwise, after $M := M \oplus P$ we would obtain a matching of larger weight.

$\Leftarrow$ "

Let $M' \subseteq E$ be any matching of G. Consider the graph

$$G' := (V, M \oplus M', \omega).$$

Each connected component of G' which is not an isolated node is either an simple M-alternating cycle P of even length or a simple M-alternating path P. Hence, for each connected component P we have

$$\Delta(P) \leq 0.$$

Claim: $\omega(M') = \omega(M \oplus M' \oplus M) \leq \omega(M)$

Proof of claim:

To prove that

$$M' = M \oplus M' \oplus M$$

consider

$$M \oplus M' = \underbrace{(M \cup M') \setminus (M \cap M')}_{\bar{M}}$$

and

$$\bar{M} \oplus M = (\bar{M} \cup M) \setminus (\bar{M} \cap M)$$

Since

$$\bar{M} \cup M = (M \cup M') \setminus (M \cap M') \cup M = M' \cup M$$

and

$$\bar{M} \cap M = (M \cup M') \setminus (M \cap M') \cap M = M \setminus (M \cap$$

we obtain

$$\bar{M} \oplus M = (M' \cup M) \setminus (M \setminus (M \cap M')) = M'$$

To prove that

$$w(M \oplus M' \oplus M) \leq w(M)$$

consider the connected components of G' which are not an isolated node.

Note that

$$w(M) = w(M \setminus (M \cap M')) + w(M \cap M')$$

and

$$w(M') = w(M' \setminus (M \cap M')) + w(M \cap M')$$

Since for each connected component P of G' which is not an isolated node $\Delta(P) \leq 0$ there holds

$$w(M' \setminus (M \cap M')) \leq w(M \setminus (M \cap M'))$$

and hence

$$\begin{aligned} w(M') &= w(M' \setminus (M \cap M')) + w(M \cap M') \\ &\leq w(M \setminus (M \cap M')) + w(M \cap M') \\ &= w(M). \end{aligned}$$

■

Applying Theorem 1.3, we have to consider M-augmenting paths, also the other simple M-alternating paths, and also the simple M-alternating cycles.

The following theorem gives us the possibility to restrict us to the consideration of M-augmenting paths.

Theorem 1.4

Let $G = (V, E, w)$ be a weighted undirected graph and $M \subseteq E$ be a matching of G such that $|M| = k$ and $w(M)$ is maximum with respect to matchings of size k. Let P be an M-augmenting path with maximal gain $\Delta(P)$. Then $M' := M \oplus P$ is a matching with maximum weight within the matchings of size $k+1$ of G

Proof:

Let M'' be any matching of G with $|M''| = k+1$. It suffices to prove

$$w(M'') \leq w(M \oplus P).$$

Consider the subgraph

$$G' := (V, M'' \oplus M, w).$$

Let R be any M-augmenting path in G'.
$|M''| > |M|$ implies that R exists.

$\Rightarrow$

$$w(M) + \Delta(R) = w(M \oplus R) \leq w(M \oplus P)$$
$$w(M'') - \Delta(R) = w(M'' \oplus R) \leq w(M)$$

After the addition of both inequalities, we obtain

$$w(M) + w(M'') \leq w(M \oplus P) + w(M)$$
$\Leftrightarrow$
$$w(M'') \leq w(M \oplus P).$$

■

Applying Theorems 1.3 and 1.4 we obtain the following general method for

the computation of a maximum weighted matching of a graph $G = (V, E, w)$.

Algorithm MAXWEIMATCHING

Input: A weighted undirected graph $G = (V, E, w)$

Output: A maximum weighted matching M_{max} of G.

Method:

$M := \emptyset$;
while there exists an M-augmenting path P with $\Delta(P) > 0$
do
construct such a path with maximal gain $\Delta(P)$;
$M := M \oplus P$
od;
$M_{max} := M$.

The following theorem proves the correctness of the algorithm above.

Theorem 1.5

The algorithm MAXWEIMATCHING computes a matching of maximum weight of G.

Proof:

Let $|M_{max}| = t$. Then by Theorem 1.4

- For $1 \leq k \leq t$, the matching M_k computed during the kth performance of the body of the while-loop has maximum weight within all matchings of size k of G.

Assume that M_{max} has not maximum weight.

Theorem 1.3 $\Rightarrow$

One of the following cases is fulfilled:

1) $\exists$ M_{max}-alternating cycle P of even length with $\Delta(P) > 0$.
2) $\exists$ simple M_{max}-alternating path P such that
   i) $(v,w) \in M_{max} \Rightarrow (v,w) \in P$ or $v, w \notin P$ and
   ii) $\Delta(P) > 0$.

If there is an M_{max}-alternating cycle P of even with $\Delta(P) > 0$ then $M_{max} \oplus P$ would be a matching of size t with

$$w(M_{max} \oplus P) > w(M_{max}).$$

This contradicts that M_{max} is a matching of size t of maximum weight within all matchings of size t.

If there is an M_{max} - alternating path P such that

i) $(v,w) \in M_{max} \Rightarrow (v,w) \in P$ or $v, w \notin P$ and
ii) $\Delta(P) > 0$

then P cannot be M_{max}-augmenting. Otherwise, the algorithm would not terminate with the matching M_{max}.

If the length of P is even then $M_{max} \oplus P$ would be a matching of size t with

$$w(M_{max} \oplus P) > w(M_{max}),$$

a contradiction.

If the length of P is odd then $M_{max} \oplus P$ would be a matching of size $t-1$ with

$$w(M_{max} \oplus P) > w(M_{t-1}),$$

a contradiction.

Altogether, Theorem 1.3 implies that the computed matching M_{max} is of maximum weight. ∎

The key problem is now this:

How to find an M-augmenting path P with maximum gain $\Delta(P) > 0$ if such a path exists?

Note that the considerations above holds also for nonbipartite graphs.

Goal:
Development of a solution of the key problem for bipartite graphs.

Let $G = (A, B, E, w)$ be a weighted undirected (bipartite) graph and $M \subseteq E$ be a matching of G.

Analogously to the unweighted case, we construct the weighted directed graph

$$G_M = (A \cup \{s\}, B \cup \{t\}, E_M, w)$$

where the new edges (s, b) and (a, t) obtain the weight 0.

Observation:
A depth first search on G_M with start node s finds an M-augmenting path P if such a path exists. But $\Delta(P)$ must not be maximal within all M-augmenting paths.

$\Rightarrow$

We need a mechanism which guarantees that $\Delta(P)$ is maximal within all M-augmenting paths.

$\rightsquigarrow$

The primal-dual method for the weighted bipartite matching problem.

We initialize the upper bound U and the input graph $G_\emptyset^*$ for the first search step as follows:

$$U := W \quad \text{and}$$
$$G_\emptyset^* := (A \cup \{s\}, B \cup \{t\}, E_\emptyset^*, w)$$

where

$$E_\emptyset^* = \{(i,j) \mid (i,j) \in E_\emptyset \text{ and } w_{ij} = W\} \cup \{(s,i) \mid i \in B\} \cup \{(j,t) \mid j \in A\}.$$

It is easy to prove that U and $G_\emptyset^*$ fulfill the Properties 1 - 3.

Exercise:
Prove that U and $G_\emptyset^*$ fulfill the properties 1 - 3.

Assume that the current search step terminates with

- current upper bound U,
- a matching M,
- a weighted directed graph $G_M = (A \cup \{s\}, B \cup \{t\}, E_M, w)$ and
- the input graph $G_M^* = (A \cup \{s\}, B \cup \{t\}, E_M^*, w)$ of the last unsuccessful depth first search.

Note that every M-augmenting path P in G_M has gain $\Delta(P) < U$.

The primal-dual method can be separated into rounds. Each round decides into two steps, the <u>search step</u> and the <u>extension step</u>.

The input of a search step will always be a subgraph G_M^* of G_M and an upper bound U such that

1) $\Delta(P) \leq U$ for all M-augmenting paths P in G_M
2) $\Delta(P) = U$ for all M-augmenting paths P in G_M^* and
3) G_M^* contains all M-augmenting paths P of G_M with $\Delta(P) = U$.

For the construction of an M-augmenting path in G_M^* if such a path exists the search step will use depth first search. If G_M^* does not contain any M-augmenting path, the extension step will compute the input for the next search step.

Given G_M^*, the search step will be performed in the same way as in the unweighted case. It remains the description of the extension step.

Given $G = (A, B, E, w)$ we construct the graph

$$G_\phi = (A \cup \{s\}, B \cup \{t\}, E_\phi, w).$$

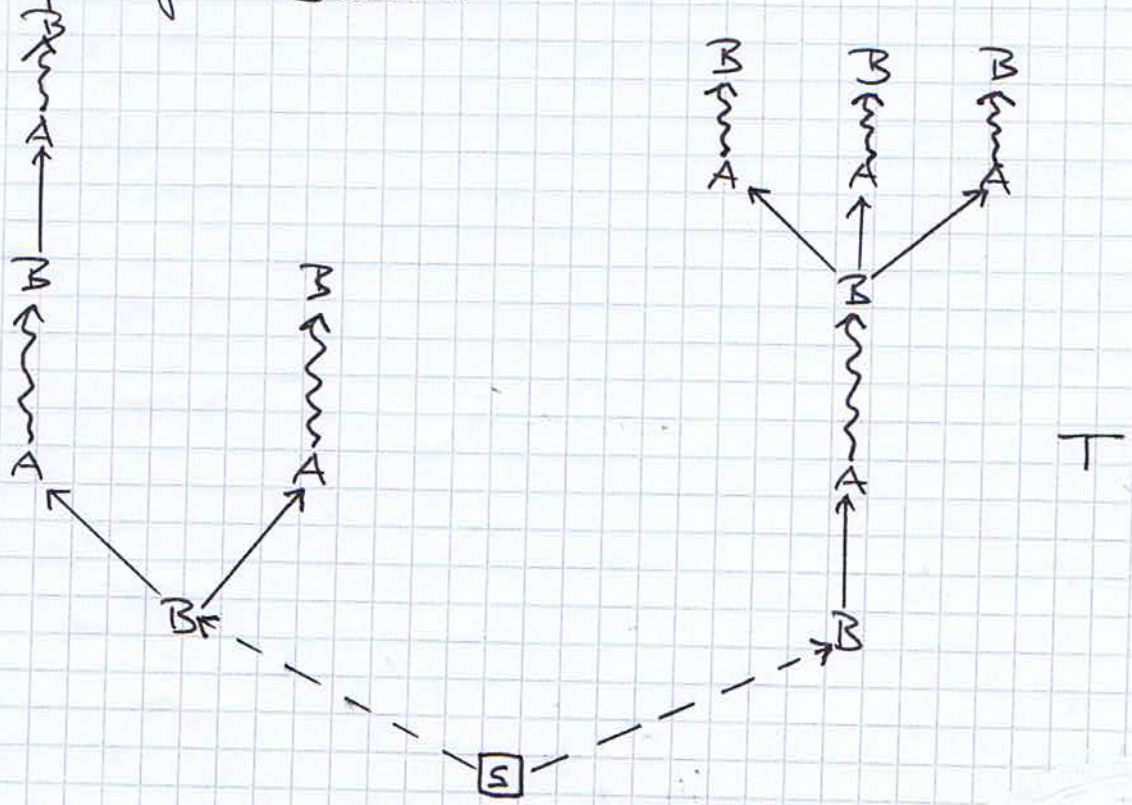
Let

$$W := \max \{ w_{ij} \mid (i,j) \in E \}.$$

Question:

How to get the reduced upper bound and the input graph for the next search step?

For getting an answer of this question let us consider the depth first search tree T which has been constructed during the last unsuccesful depth first search.



Let

$$B_T := B \cap T \quad \text{and} \quad A_T := A \cap T.$$

Goal:

The reduction of the current upper bound u by the appropriate value δ and the extension of E_M^* by edges from $E \setminus E_M^*$ such that the

Properties 1-3 will be fulfilled with respect to the new upper bound $u-\delta$ and the constructed input graph for the next search step.

Question:
How to get the appropriate δ and in dependence of δ those edges which we have to add to E_M^*?

Idea
We associate with all nodes of the graph node weights such that the gain of a path P can be computed only by the consideration of the end nodes of the path P.

More exactly, we associate with each node $j \in A$ a node weight v_j and with each node $i \in B$ a node weight u_i such that always the following invariants are fulfilled:

1) $u_i + v_j \geq w_{ij}$ for all $(i,j) \in E \setminus M$,
2) $v_j + u_i = w_{ji}$ for all $(j,i) \in M$,
3) $u_i = u$ for all M-free $i \in B$, and
4) $v_j = 0$ for all M-free $j \in A$.

Then we define

$$E_M^* := M \cup \{(i,j) \in E_M \setminus M \mid u_i + v_j = w_{ij}\}.$$

For the computation of the appropriate value δ we define for $i \in B$ and $j \in A$ unbalances

π_{ij} and π_{ji} in the following way.

$$\pi_{ij} := u_i + v_j - w_{ij} \quad \text{and}$$

$$\pi_{ji} := v_j + u_i - w_{ji}.$$

Goal:
Prolongation of paths in the depth first search tree T by adding appropriate edges (i,j) with $i \in B_T$ and $j \in A \setminus A_T$. The upper bound U should be reduced as few as possible.

Note that $(i,j) \notin E_M^*$.

$\Rightarrow$

$u_i + v_j > w_{ij} \quad \Rightarrow \quad \pi_{ij} > 0.$

Question:
How to change the node weights u_i and v_j such that the edge (i,j) is added to E_M^*; i.e., $\pi_{ij} = 0$?

After the reduction of u_i by $\delta := \pi_{ij}$ we obtain $(u_i - \pi_{ij}) + v_j = w_{ij}$ such that the edge (i,j) would be added to E_M^*. Since we do not wish to delete some edge from E_M^*, we have to modify the node weights of the other nodes in T in an appropriate manner.

We perform

$$u_i := u_i - \delta \quad \forall\, i \in B_T \quad \text{and}$$
$$v_j := v_j + \delta \quad \forall\, j \in A_T$$

Furthermore, we reduce the upper bound.

$$u := u - \delta.$$

Properties:

- At the beginning

$$u_i = u \quad \forall \text{ M-free nodes } i \in B.$$

Always we have $i \in B_T \ \forall$ M-free $i \in B$.

$\Rightarrow$

After the extension step, we have

$$u_i = u \quad \forall \text{ M-free nodes } i \in B.$$

- At the beginning

$$v_j = 0 \quad \forall \text{ M-free nodes } j \in A.$$

Always we have $j \notin A_T \ \forall$ M-free $j \in A$

$\Rightarrow$

After the extension step, we have

$$v_j = 0 \quad \forall \text{ M-free nodes } j \in A$$

Since δ should be chosen such that

i) at least one edge is added to E_M^* and
ii) δ should be chosen as small as possible,

we define

$$\delta := \min \{ \pi_{ij} \mid i \in B_T \text{ and } j \in A \setminus A_T \}.$$

If $\delta \geq u$ then the algorithm terminates.

Exercice

Work out the algorithm in detail.

Let P be a path in G_M. The unbalance $\pi(P)$ of the path P is defined as follows.

$$\pi(P) := \sum_{e \in P} \pi_e.$$

Next we will show that after the extension step with respect to the new upper bound and the new input graph for the next search step the invariants 1-3 are pulfilled.

Lemma 1.2

Let $P = v_0, v_1, \ldots, v_k$ with $v_0 \neq s$ and $v_k \neq t$ be a simple alternating path in G_M. Then

$$\Delta(P) = \begin{cases} u_{v_0} + v_{v_k} - \pi(P) & \text{if } v_0 \in B, v_k \in A \\ u_{v_0} - u_{v_k} - \pi(P) & \text{if } v_0, v_k \in B \\ -v_{v_0} - u_{v_k} - \pi(P) & \text{if } v_0 \in A, v_k \in B \\ -v_{v_0} + v_{v_k} - \pi(P) & \text{if } v_0, v_k \in A. \end{cases}$$

Proof:

Definition $\Rightarrow$

$$\Delta(P) = \sum_{e \in P \cap E_M \setminus M} w_e - \sum_{e \in P \cap M} w_e$$

$$= \sum_{(i,j) \in P \cap E_M \setminus M} (u_i + v_j - \pi_{ij})$$

$$- \sum_{(j,i) \in P \cap M} (v_j + u_i - \pi_{ji})$$

Since $\pi_{ji} = 0$ for $(j,i) \in M$ there holds

$$= \underbrace{\left(\sum_{(i,j) \in P \cap E_M \setminus M} (u_i + v_j) - \sum_{(j,i) \in P \cap M} (v_j + u_i) \right)}_{D} - \pi(P)$$

Since P is an alternating path, up to the first summand with respect to the first edge on P and the second summand with respect to the last edge on P all summands vanish. Hence

$$D = \begin{cases} u_{v_0} + v_{v_k} & \text{if } v_0 \in B,\ v_k \in A \\ u_{v_0} - u_{v_k} & \text{if } v_0, v_k \in B \\ -v_{v_0} - u_{v_k} & \text{if } v_0 \in A,\ v_k \in B \\ -v_{v_0} + v_{v_k} & \text{if } v_0, v_k \in A \end{cases}$$

This proves the lemma. ∎

Lemma 1.3

The algorithm fulfills always the following invariants:

i) $u_i = u$ for all M-free nodes $i \in B$,
ii) $v_j = 0$ for all M-free nodes $j \in A$, and
iii) $\pi_e \geq 0$ for all edges $e \in E_M$.

Proof:

Definition $\Rightarrow$

At the beginning, i.e., $M = \emptyset$ all invariants are fulfilled.

Goal:

To prove the following:

invariants fulfilled before the search and the extension step, respectively

$\Rightarrow$

the invariants are fulfilled after the search and the extension step, respectively.

The search step never changes the node weights or the upper bound. Hence, the invariants remains fulfilled after the search step.

We have shown above that the extension step maintains the invariants i and ii).

δ is chosen in such a way such that the modification of the node weights never violate the invariant (iii).

■

Theorem 1.6

The algorithm MAXWEIGHTMATCHING implemented as described above terminates with a maximum weighted matching M_{max}.

Proof:

It is clear that the algorithm terminates with a matching M_{max}.

Theorem 1.4 $\Rightarrow$

It suffices to show that always an augmenting path P with maximal gain $\Delta(P)$ is augmented by the algorithm and that after the termination of the algorithm no augmenting path P with $\Delta(P) > 0$ exists.

Assume that the algorithm augment the M-augmenting path P with $\Delta(P) > 0$ although an M-augmenting path Q with $\Delta(Q) > \Delta(P)$ exists. Let

$P = i, \dots, j$ and $Q = p, \dots, k$.

Lemma 1.2 $\Rightarrow$

$$\Delta(Q) = u_p + v_k - \pi(Q) \quad \text{and}$$
$$\Delta(P) = u_i + v_j - \pi(P).$$

Lemma 1.3 $\Rightarrow$

$$\pi(Q) \geq 0, \quad u_i = u_p \text{ and } v_j = v_k = 0.$$

Since $\pi(P) = 0$ we obtain

$$\Delta(Q) + \pi(Q) = \Delta(P)$$

and hence,

$$\Delta(Q) \leq \Delta(P)$$

a contradiction.

After the termination of the algorithm there hold

$u_i \leq 0$ for all M_{max}-free nodes $i \in B$ and
$v_j = 0$ for all M_{max}-free nodes $j \in A$.

Lemmas 1.2 and 1.3 $\Rightarrow$

$$\Delta(P) = u_i + v_j - \pi(P) \leq -\pi(P) \leq 0$$

for all M_{max}-augmenting paths $P = i, \dots, j$.

$\Rightarrow$

$\nexists$ M_{max}-augmenting path P with $\Delta(P) > 0$. ∎

Exercise:
Show that the algorithm MAXWEIMATCHING can be implemented such that its runtime is $O((|A|+|B|)^3)$.

1.2 Networks flows

A manager of a company has the following situation:

- At the seaports $A_1, A_2, \ldots, A_p$ there are bananas waiting for their shipment. r_i, $1 \leq i \leq p$, denotes the quantity of bananas at seaport A_i.
- $B_1, B_2, \ldots, B_q$ are the target seaports. d_j, $1 \leq j \leq q$, denotes the request for bananas of seaport B_j.
- At most $c(A_i, B_j)$, $1 \leq i \leq p$, $1 \leq j \leq q$, bananas can be transported from the seaport A_i to the seaport B_j.

The manager has to answer the following questions:

1) Is it possible to satisfy all requests?
2) If not, what is the maximal quantity of bananas which can be transported to the target seaports?
3) How to ship the bananas?

For getting an answer to these questions, the manager transforms its economic problem to a mathematical problem which he can solve with a well known algorithm.

$\rightsquigarrow$

Construction of a directed graph $G = (V, E)$ where

$$V := \{A_1, A_2, \ldots, A_p, B_1, B_2, \ldots, B_q\} \text{ and}$$
$$E := \{(A_i, B_j) \mid 1 \le i \le p, 1 \le j \le q\}.$$

To ensure that at most $c(A_i, B_j)$ bananas can be shiped from A_i to B_j, each edge (A_i, B_j) obtains the capacity $c(A_i, B_j)$.
To ensure that each seaport A_i, $1 \le i \le p$, ships at most r_i bananas and that each seaport B_j, $1 \le j \le q$, obtains at most d_j bananas, two new nodes s and t, the edges (s, A_i), $1 \le i \le p$, with capacities $c(s, A_i) := r_i$, and the edges (B_j, t), $1 \le j \le q$, with capacities $c(B_j, t) := d_j$ are added to G.

The solution of the following problem would give an answer to all questions posed above:

- Compute a maximum flow from s to t in G, where

  i) the amount of flow on an edge is bounded by the capacity of this edge and

  ii) each flow which enters a node A_i or B_j has also to leave this node.

Now, we shall define the network flow problem in general.

A flow network is a directed graph $G = (V, E, c, s, t)$ with source node $s \in V$, sink node $t \in V \setminus \{s\}$, and capacity function $c: E \to \mathbb{R}^+$ which assigns to each edge $e = (v,w)$ a positive capacity $c(v,w)$.

We extend the capacity function to $V \times V$ by

$$c(v,w) := 0 \quad \forall\, (v,w) \in V \times V \setminus E.$$

Let

$$\begin{aligned} IN(v) &:= \{ e \in E \mid e = (w,v),\ w \in V \}, \\ OUT(v) &:= \{ e \in E \mid e = (v,w),\ w \in V \}, \\ \Gamma^-(v) &:= \{ w \in V \mid (w,v) \in IN(v) \}, \text{ and} \\ \Gamma^+(v) &:= \{ w \in V \mid (v,w) \in OUT(v) \}. \end{aligned}$$

A flow $f: V \times V \to \mathbb{R}$ is a function which fulfills the following requirements:

1) $f(v,w) \leq c(v,w) \quad \forall\, (v,w) \in V \times V$

(capacity requirement)

2) $\sum_{e \in IN(v)} f(e) = \sum_{e \in OUT(v)} f(e) \quad \forall\, v \in V \setminus \{s,t\}$

(Kirchhoff's Law)

The size $|f|$ of the flow is the amount of flow which enters the sink t and does not leave t; i.e.,

$$|f| = \sum_{v \in V} f(v,t) - \sum_{v \in V} f(t,v).$$

A <u>cut</u> $(S, V \setminus S)$ of a flow network $G = (V, E, c, s, t)$ is the partition of V into two nonempty sets S and $V \setminus S$. Note that the definition implies $\emptyset \neq S \neq V$.

A cut $(S, V \setminus S)$ <u>separates</u> s and t if $s \in S$ and $t \in V \setminus S$.

Let $E(S, V \setminus S)$ denote the set of edges $(x, y) \in E$ with $x \in S$ and $y \in V \setminus S$. The <u>capacity</u> $c(S, V \setminus S)$ of the cut $(S, V \setminus S)$ is defined by

$$c(S, V \setminus S) := \sum_{(x,y) \in E(S, V \setminus S)} c(x, y).$$

By definition, the flow from s to t cannot be larger than the capacity of any cut $(S, V \setminus S)$ which separates s and t.

The following theorem characterizes the maximal possible flow from the source s to the sink t in a flow network G.

<u>Theorem 1.7</u> (Max-Flow Min-Cut Theorem)

Let $G = (V, E, c, s, t)$ be a flow network. Then there holds

$$\max \{ |f| \mid f \text{ is a flow in } G \}$$
$$= \min \{ c(S, V \setminus S) \mid (S, V \setminus S) \text{ separates } s \text{ and } t \}.$$

Proof:

Let

$$F_{max} := \max \{ |f| \mid f \text{ is a flow in } G \}.$$

It is clear that

$$F_{max} \leq \min \{ c(S, V \setminus S) \mid (S, V \setminus S) \text{ separates } s \text{ and } t \}.$$

Hence, it suffices to prove that there is a s and t separating cut $(S, V \setminus S)$ with

$$c(S, V \setminus S) = F_{max}.$$

Idea:

We shall develop an algorithm which, given a flow f with $|f| = F_{max}$, computes a s and t separating cut $(S, V \setminus S)$ with $c(S, V \setminus S) = F_{max}$.

We start with $S = \{s\}$ and extend during each step the current set S by a node $y \in V \setminus S$ which has to be added to S to fulfill the assertion.

↝

(1) $S := \{s\}$;

(2) **while** $\exists\, x \in S,\ y \in V \setminus S$ with

$$c(x,y) > f(x,y) \text{ or } f(y,x) > 0$$

do

$$S := S \cup \{y\}$$

od.

First, we shall prove that the computed cut $(S, V \setminus S)$ separates s and t.

Assume that $(S, V \setminus S)$ does not separate s and t; i.e., $t \in S$.

$\Rightarrow$

$\exists\, x_{e-1} \in S$ which causes the addition of t to S; i.e.,

$$c(x_{e-1}, t) > f(x_{e-1}, t) \text{ or } f(t, x_{e-1}) > 0.$$

Since $x_{e-1} \in S$ there holds

$x_{e-1} = s$ or $\exists\, x_{e-2} \in S$ which causes the addition of x_{e-1} to S

$\vdots$

and so on.

$\Rightarrow$

$\exists$ an undirected path

$$P = s = x_0, x_1, \dots, x_{e-1}, x_e = t$$

with

$$x_j \in S \text{ for } 0 \leq j \leq e.$$

Let for $0 \leq i < e$

$$\varepsilon_i := \max \{ c(x_i, x_{i+1}) - f(x_i, x_{i+1}),\ f(x_{i+1}, x_i) \}.$$

Construction ⇒

$$\varepsilon_i > 0 \quad , \quad 0 \le i < e.$$

Let

$$\varepsilon := \min \{ \varepsilon_i \mid 0 \le i < e \}.$$

Goal:

The construction of a flow f^* such that

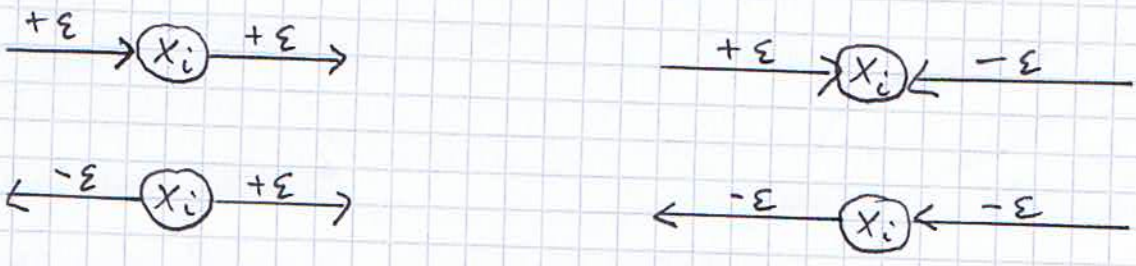
$$|f^*| = F_{max} + \varepsilon.$$

For obtaining the flow f^*, we change the flow on the undirected edges (x_i, x_{i+1}), $0 \le i < e$ in the following way:

$$\begin{cases} f^*(x_i, x_{i+1}) := f(x_i, x_{i+1}) + \varepsilon & \text{if } (x_i, x_{i+1}) \in P \\ f^*(x_{i+1}, x_i) := f(x_{i+1}, x_i) - \varepsilon & \text{if } (x_{i+1}, x_i) \in P \end{cases}$$

The choice of ε implies that f^* does not destroy the capacity requirement. Next, we shall prove that f^* fulfills Kirchhoff's Law, too.

For doing this let us consider any node x_i, $0 < i < e$. The following four cases can arise:



In all cases, the validity of Kirchhoff's Law before the flow changes implies the validity of Kirchhoff's Law after the flow changes.

$\Rightarrow$

f^* is a flow.

Furthermore,

$$|f^*| = \sum_{v \in V} f^*(v,t) - \sum_{v \in V} f^*(t,v)$$

$$= \sum_{v \in V \setminus \{x_{e-1}\}} f(v,t) + f^*(x_{e-1},t) - \sum_{v \in V} f(t,v)$$

$$= \sum_{v \in V \setminus \{x_{e-1}\}} f(v,t) + f(x_{e-1},t) + \varepsilon - \sum_{v \in V \setminus \{x_{e-1}\}} f(t,v)$$

$$= |f| + \varepsilon$$

$$= F_{max} + \varepsilon.$$

This is a contradiction to the maximality of f.

$\Rightarrow$

$t \notin S$; i.e., $(S, V \setminus S)$ separates s and t.

Construction $\Rightarrow$

$\forall x \in S \; \forall y \in V \setminus S$: $f(x,y) = c(x,y)$ and $f(y,x) = 0$

Hence,

$$|f| = \sum_{x \in S, y \in V \setminus S} f(x,y) = \sum_{x \in S, y \in V \setminus S} c(x,y) = c(S, V \setminus S).$$

This proves the Max-Flow Min-Cut Theorem. ∎

The path P constructed in the proof of the Max-Flow Min-Cut theorem is called an augmenting path.

If the current flow in the network is not maximum, the proof of the Max-Flow Min-Cut theorem gives us a method for the construction of an augmenting path P which can be used for the increase of the current flow. Hence, we obtain the following algorithm for the construction of a maximum flow in a flow network:

Algorithm FORD-FULKERSON

Input: Flow network $G = (V, E, c, s, t)$.

Output: A maximum flow f in G.

Method:

(1) Start with the zero flow f_0; i.e., $f(x,y) = 0$ for all $(x,y) \in E$.

(2) $f := f_0$; found := false;

(3) while $\neg$ found
do
As in the proof of the Max-Flow Min-Cut theorem, construct the cut $(S, V \setminus S)$ with respect to the current flow f;

if $t \in S$

then

As described in the proof of the Max-flow Min-Cut theorem, increase the flow by ε

else

found := true

fi

od;

(4) Output := f.

Ford and Fulkerson have given an example which shows that in the case of irrational ~~capacities~~ capacities the algorithm above must not terminate. The next theorem shows that in the case of integer capacities, the algorithm always terminate.

Theorem 1.8 (Integrality Theorem):

Let $G = (V, E, c, s, t)$ be a flow network where all capacities are integers. Then Algorithm Ford-Fulkerson terminates with input G after at most $\sum_{(x,y) \in E} c(x,y)$ augmentations with an integral flow.

Proof:

The algorithm starts with the zero flow and all

capacities are integral.

$\Rightarrow$

During the performance of the algorithm, the current flow f is always integral.

Each augmentation increases the current flow at least by one.

Hence, the assertion follows from the fact that

$$f_{max} \leq \sum_{(x,y) \in E} c(x,y).$$

■

Also with integral capacities, the algorithm FORD-FULKERSON can perform many augmentations.

Question:
Is it possible to develop a method for the choice of the augmenting paths such that the method of Ford and Fulkerson is efficient independently of the sizes of the capacities?

Goal:
The development of such a method for the choice of the augmenting paths.

First, we need some notations:

Let $G = (V, E, c, s, t)$ be a flow network and f be a flow from the source s to the sink t. Let G' be that undirected graph which we obtain from G by ignoring the direction of the edges.

An <u>augmenting path</u> P is a path from s to t in G', such that

a) $f(x,y) < c(x,y)$ for each edge $(x,y) \in E$ which is on P in forward direction (<u>forward arc</u>)

b) $f(y,x) > 0$ for each edge $(y,x) \in E$ which is on P in backward direction (<u>backward arc</u>)

If we increase the flow on each forward arc of P by $\varepsilon > 0$ and decrease the flow on each backward arc by ε then we <u>augment the flow on P by ε</u>.

In 1969 Edmonds and Karp added the following rule to the algorithm of Ford and Fulkerson:

- In each step, augment a with respect to the number of edges shortest augmenting path.

They have proved that this algorithm performs at most $m \cdot \frac{n}{2}$ augmentations where $n = |V|$ and $m = |E|$.

Independently, Dinic observed that it would be useful to consider simultaneously all shortest augmenting paths. As long as such a path exists one augments such a path.

Goal:

The development of Dinic's method.

First, we need some notations:

A flow f in a flow network $G = (V, E, c, s, t)$ is a blocking flow if on each path from s to t there is an edge (x,y) with $f(x,y) = c(x,y)$. We call such an edge saturated. An edge $e \in E$ is called useful from x to y if

$e = (x,y)$ and $f(e) < c(x,y)$ or
$e = (y,x)$ and $f(e) > 0$.

Note that on an augmenting path all edges are useful.

Dinics algorithm separates into phases. A phase obtains as input a flow network $G = (V, E, c, s, t)$ and a current flow $f: E \to \mathbb{R}$ in G.

First, a layered network G_f which contains exactly the shortest augmenting paths is constructed. Then, a blocking flow in G_f is computed.

G_f is constructed by applying a breadth-first search to G.

Algorithm CONSTRUCTION G_f

Input: flow network $G = (V, E, c, s, t)$ and a flow f in G.

Output: The layered network $G_f = (V', E', \tilde{c}, s, t)$.

Method:

(1) $V_0 := \{s\}$; $i := 0$;

(2) $H := \{v \in V \setminus (\bigcup_{j \le i} V_j) \mid \exists\, u \in V_i$ and edge $e \in E$ which is useful from u to $v.\}$

(3) $E(H) := \{(u,v) \mid u \in V_i, v \in H$ and there is an edge $e \in E$ which is useful from u to $v.\}$

(4) if $H = \emptyset$
then
STOP
fi;

(5) if $t \in H$
then
$\ell := i+1$; $V_\ell := \{t\}$;
$E_\ell := \{(u,t) \mid u \in V_i$ and $\exists e \in E$ which is useful from u to $t.\}$;
STOP
else
$V_{i+1} := H$; $E_{i+1} := E(H)$; $i := i+1$;
goto (2)
fi.

Lemma 1.4

If CONSTRUCTIONG$_f$ terminates in Step 4 then the current flow f is already a maximum flow.

Proof:

Let $S := \bigcup_{j=0}^{i} V_j$. Then by construction

$$s \in S \text{ and } t \in V \setminus S.$$

$\Rightarrow$

$(S, V \setminus S)$ is an S,t-separating cut.

All edges from S to $V \setminus S$ are saturated and all edges from $V \setminus S$ to S have flow zero.

Hence, the assertion follows from the Max-Flow Min-Cut Theorem. ∎

For $(u,v) \in E_i$, $1 \le i \le \ell$, we define its <u>reduced capacity</u> $\tilde{c}(u,v)$ as follows:

$$\tilde{c}(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{if } (u,v) \in E \text{ useful from } u \text{ to } v \\ f(v,u) & \text{if } (v,u) \in E \text{ useful from } u \text{ to } v \end{cases}$$

Note that the edges in G_f are directed from V_i to V_{i+1} independently of their direction in G.

Given a blocking flow $\bar{f}$ in G_f, our goal is to increase the flow f in G to a flow f' in G.

For doing this, we define the flow $f'(x,y)$, $e=(x,y) \in E$, as follows:

$$f'(e) := \begin{cases} f(e) + \bar{f}(e) & \text{if } x \in V_{j-1},\ y \in V_j,\ (x,y) \in E_j \text{ for a } j. \\ f(e) - \bar{f}(e) & \text{if } x \in V_j,\ y \in V_{j-1},\ (y,x) \in E_j \text{ for a } j. \\ f(e) & \text{otherwise} \end{cases}$$

Lemma 1.5

f' is a flow in G.

Proof:

exercise

■

Altogether, a phase in Dinic's algorithm consists of

1. the computation of the layered network $G_f = (V', E', \tilde{c}, s, t)$,
2. the computation of a blocking flow $\bar{f}$ in G_f, and
3. the increase of the flow f in G to a flow f'.

Goal:

To prove that the number of phases is bounded by $|V| = n$.

Idea:

Prove that the number of layers in the layered network increases strictly from phase to phase

Let l_k denote the index of the last layer during the kth phase.

Lemma 1.6

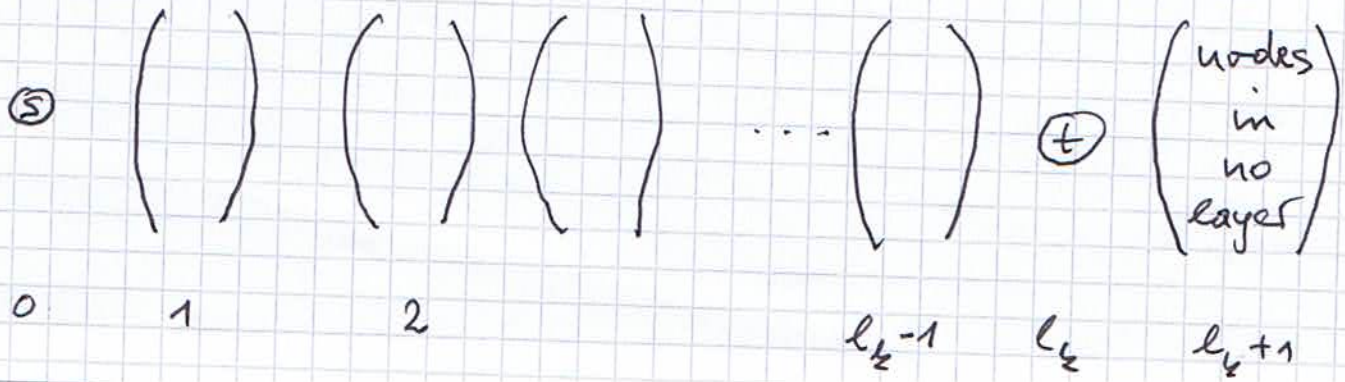
If the kth phase is not the last phase then $l_{k+1} > l_k$.

Proof:

Assume that after the kth phase, there is an augmenting path.

We shall prove that after the kth phase, the length of a shortest augmenting path strictly increases.

Let us consider the layered network constructed during the kth phase.



We divide the useful edges in G_f into three classes.

A: useful edges from layer i to layer $i+1$ for an $i < \ell_k$,

B: useful edges from layer i, $1 \leq i \leq \ell_k + 1$, to layer j for a $j \leq i$, and

C: useful edges from layer $\ell_k - 1$ to layer $\ell_k + 1$.

Construction $\Rightarrow$

1) $E' = A$; i.e., G_f contains exactly the edges in A. This implies that after the kth phase, the edges in B and in C are useful in the same manner as before. An edge in A remains to be useful from layer i to layer $i+1$ or becomes useful from layer $i+1$ to layer i instead of from layer i to layer $i+1$.

2) After the kth phase, a shortest augmenting path P has to pass through each layer; i.e., $|P| \geq \ell_k$

3) Since during the kth phase, a blocking flow has been constructed and the corresponding augmentations have been performed, there is on P an edge e such that

i) $e \in B \cup C$ or

ii) $e \in A$ and after the k-th phase the edge e is useful from layer $i+1$ to layer i and not from layer i to layer $i+1$ for an $i < l_2$.

In both cases the edge e prolongs the length of P at least by one.

$\Rightarrow \quad |P| \geq l_2 + 1.$

□

Corollary 1.1

The number of phases is at most $n-1$.

Altogether, we have proved the following theorem:

Theorem 1.9

Let $G = (V, E, c, s, t)$ be a flow network. Then a maximum flow f_{max} can be constructed in $O(n \cdot m + n \cdot b)$ time where $n = |V|$, $m = |E|$ and b is the time used for the computation of a blocking flow in a layered network with $\leq n$ nodes and $\leq m$ edges.

Dinic's idea for the construction of a blocking flow was the following:

(1) Compute with DFS a path P from s to t in G_f.

(2) Augment this path P. If after the augmentation $\tilde{f}(e) = \tilde{c}(e)$ for an edge e on P then delete this edge from G_f.

(3) As long as there is a path from s to t in G_f, repeat the whole.

If during the DFS a backtrack is performed then we can delete the corresponding node and all incident edges from G_f.

Dinic's method terminates if no path from s to t in G_f exists. Hence, a blocking flow is computed. Since the augmentation of a path P saturates at least one edge on P, at most m augmentations are performed. The total time used for all pop-operations is bounded by $O(m+n)$ since the corresponding node and all incident edges are removed from G_f. A push-operation which does not correspond to an augmenting path causes the corresponding pop-operation. Hence, the total time used for such push-operations is bounded by $O(m+n)$. An augmenting path has length $\leq n$. Hence, the total time used for the augmentations is bounded by $O(m \cdot n)$. Altogether, we have proved

Theorem 1.10

Dinic's algorithm computes a maximum flow in time $O(n^2 \cdot m)$.

Question:
Can we improve Dinic's method for the computation of a blocking flow?

Idea (Karzanov 1974):
During each step saturate a node instead of an edge.

For $v \in V$ we define the capacity $c(v)$ of the node v by

$$c(v) := \min \left\{ \sum_{e \in IN(v)} c(e), \sum_{e \in OUT(v)} c(e) \right\}.$$

A flow f saturates a node v if

$$\sum_{e \in IN(v)} f(e) = \sum_{e \in OUT(v)} f(e) = c(v).$$

A node $v \in V$ is blocked if on each path from v to the sink t there is a saturated edge.

Idea:

- Starting in s, we transport as large as possible flow into the network. It is possible that the amount of flow which reaches a node v is larger than $\sum_{e \in OUT(v)} c(e)$.
- In that case, we reduce the surplus of v.

If $v \neq s$
then we look for another way to transport this surplus to t.

- We take care that a blocked node remains to be blocked.

First, we need some further notations.

Let $G = (V, E, c, s, t)$ be a flow network. A <u>preflow</u> $p : E \to \mathbb{R}$ is a function which fulfills the following requirements:

1) $p(e) \leq c(e) \qquad \forall\, e \in E$

2) $\underbrace{\sum_{e \in IN(v)} p(e)}_{p_{in}(v)} \geq \underbrace{\sum_{e \in OUT(v)} p(e)}_{p_{out}(v)} \qquad \forall\, v \in V \setminus \{s,t\}$

A node $v \in V$ is called <u>unbalanced</u> if

$$p_{in}(v) > p_{out}(v).$$

A preflow p is a flow iff all nodes $v \in V \setminus \{s,t\}$ are balanced.

Let $G_f = (V', E', \tilde{c}, s, t)$ be the layered network constructed by the algorithm CONSTRUCTION G_f

Karzanov's method for the computation of a blocking flow separates into two steps which are iterated as long as there is a

unbalanced node. After the last step, the network contains a blocking flow.

⇝

Step 1: Forward movement of preflow in G_f

Step 2: Balancing of preflow in G_f.

Note that an unbalanced node is blocked. We balance the preflow p at an unbalanced node v such that

$$p_{in}(v) = p_{out}(v) = c(v).$$

Hence, the node v remains to be blocked.

After balancing a node v we do not allow any further change of the incoming flow of v.

An edge e is open if it is allowed to change $p(e)$. Otherwise, e is closed.

At the beginning, all edges are open.

Step 1:

For all nodes $v \in V$, we number the outgoing edges of v in any fixed order.

Idea:

The incoming preflow of a node v is transported to the next layer in the

following way:

- Transport the largest possible amount of preflow over the first open, not saturated outgoing edge of v.
- Then transport the largest possible amount of preflow over the second open, not saturated outgoing edge of v

⋮

and so on.

With respect to the open edges in OUT(v), the following invariant will be always fulfilled:

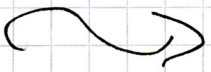
- Invariant:

We have a sequence of saturated edges followed by one not saturated edge e with $p(e) \geq 0$. This edge is called $h(v)$. All further edges have zero preflow.

For the organisation of Step 2, we use a stack K_w for the management of the incoming edges $e \in IN(w)$ of a node w with $p(e) > 0$.

- At the moment when an edge $e \in IN(w)$ obtains a preflow $p(e) > 0$, the edge e is pushed.

- If during Step 2, the ingoing preflow of the node w has to be reduced, the preflow of the uppermost edge in K_w is reduced, then the preflow on the next edge on K_w, and so on.



Step 1: Forward movement of preflow in G_f:

(1) Start in s and transport as much as possible preflow to layer 1; i.e.,

```
for all e ∈ OUT(s)
    do
        p(e) := c̃(e)
    od ;
START := 1 ;
```

(2)
```
for i := START step 1 until e-1
    do
        for all v ∈ V_i
            do
                P := p_in(v) ;
                while P > 0 and k(v) defined
                    do
                        e := k(v) ;
                        P' := min { c̃(e) - p(e), P } ;
                        p(e) := p(e) + P' ;
                        P := P - P'
                    od
        od.
```

After the termination of Step 1, for all nodes $v \in V' \setminus \{s,t\}$ exact one of the following two cases is fulfilled:

1. $p_{in}(v) = p_{out}(v)$.
2. $p_{in}(v) > p_{out}(v) = \tilde{c}(v)$ (i.e., v is unbalanced).

Step 2: Balancing of the preflow in G_f:

Let V_{max} denote the layer of highest index which contains an unbalanced node.

(1) For all unbalanced nodes $v \in V_{max}$
do
Using K_v reduce the incoming preflow of v until v is balanced; i.e.,
$p_{in}(v) = p_{out}(v) = \tilde{c}(v)$;
Close the incoming edges of v.
od;

(2) START := max − 1;
Apply (2) of Step 1.

Exercise:

Work out the algorithm of Karzanov.

For the proof of the correctness of Karzanov's method, it suffices to prove that a blocking flow is constructed. This is a direct consequence of the following lemma.

Lemma 1.7

During the whole algorithm, a blocked node remains to be blocked.

Proof:

After the first performance of Step 1, all outgoing edges of s are saturated

$\Rightarrow$

s is blocked.

Furthermore, each outgoing edge of an unbalanced node v is saturated

$\Rightarrow$

v is blocked.

Claim:
Each node which is blocked before Step 2 remains to be blocked after the performance of Step 2.

Proof of claim:

- For all nodes $v \in V_j$, $j \geq$ max the preflow

$p(e)$ does not change for all edges e on a path from v to t in G_f.

$\Rightarrow$

If v is blocked before the performance of Step 2 then v is also blocked after the performance of Step 2.

- Consider $v \in V_{max}$ with v is unbalanced. Note that v is blocked, too.

  Step 2 reduces the incoming preflow of v.

Assume that the preflow $p(e)$ for $e=(w,v)$ and $w \in V_{max-1}$ is reduced by Step 2.

Consider $x \in V_j$, $j < max$ with x is blocked before the reduction of $p(e)$. Then

- the preflow on all paths from x to t which do not contain the edge e does not change.

and

- all paths from x to t which contain the edge e contain the blocked node v.

$\Rightarrow$

x remains to be blocked after the reduction of $p(e)$.

■

Since the source node s is blocked after the first performance of Step 1, Lemma 1.7 implies that Karzanov's method computes a blocking flow.

It remains the analysis of the used time.

Lemma 1.8
Karzanov's method finds a blocking flow in $O(n^2)$ time.

Proof:

- Note that after the reduction of the preflow on an edge e, the edge e is closed. Hence, the number of preflow reduction is bounded by m.
- Each edge e is saturated at most once. Hence, the number of increases of preflow on an edge e such that e is saturated is bounded by m.
- Between two subsequent balancing steps each not blocked node has at most one edge e such that p(e) is increased and e is not saturated.

  $\Rightarrow$

  The number A of such increases is bounded by:

$$A \leq \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}.$$

Altogether, we have shown that the used time is $O(n^2)$.

A clever modification of Karzanov's method has been done by Malhotra, Pramodh Kumar and Maheshwari (1978):

Let f be a flow in a flow network $G = (V, E, c, s, t)$ and let $v \in V$. The flow potential $\rho_f(v)$ of v is defined by

$$\rho_f(v) := \tilde{c}(v) - \sum_{(u,v) \in E'} f(u,v).$$

$r \in V'$ is called reference node if

$$\rho_f(r) = \min \{ \rho_f(v) \mid \rho_f(v) > 0, v \in V' \setminus \{s,t\} \}.$$

where $G_f = (V', E', \tilde{c}, s, t)$ is the corresponding layered flow network.

Lemma 1.9

Let r be a reference node in the layered flow network $G_f = (V', E', \tilde{c}, s, t)$. Let $\rho_f(v) > 0 \ \forall v \in V' \setminus \{s,t\}$. Furthermore, $\forall v \in V' \setminus \{s,t\}$: $\exists$ paths from s to v and from v to t in G_f. Then, the flow f can

be increased by $\rho_f(r)$ to a flow f' such that $\rho_{f'}(r) = 0$.

Proof:

Note that $\forall v \in V' \setminus \{s,t\}$ there holds

$$\rho_f(r) \leq \rho_f(v).$$

$\Longrightarrow$

Starting in r, we can transport an additional flow of size $\rho_f(r)$ layer by layer to t.

Analogously, starting in r we can obtain backwards an additional flow of size $\rho_f(r)$ from s to r.

$\rightsquigarrow$

Algorithm MPM

Input: layered network $G_f = (V', E', \tilde{c}, s, t)$
Output: blocking flow $\overline{f}$ in G_f.
Method:

(1) Determine a reference node r.
(2) Transport an additional flow of size $\rho_f(r)$ from r to t.
(3) Starting in r, transport backwards an additional flow of size $\rho_f(r)$ from s to r.
(4) Remove all edges and all nodes such that no additional flow can be transported on these edges and through these nodes.

(5) If the network is not empty, goto (1).

Time analysis:

	One performance of (1)-(5)	in total
(1)	$\leq n-1$ comparisons	
(2)	n (unsaturated)	$+$ m (saturated)
(3)	n	$+$ m
(4) + (5)		$O(n+m)$
Σ		$O(n^2)$

Exercise:

a) Work out the algorithm of Karzanov.
b) Work out the algorithm MPM and its time analysis.