Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Informatik V

Pearls of Algorithms

Bipartite matching and network flows

Prof. Dr. Norbert Blum

Lecture Notes WS 2014/15

prepared by
Simon Schneider, Daniel Weber

Bonn, November 12, 2014

Contents

1 Bipartite Matching **2**
1.1 Definitions and the general method 2
1.2 The unweighted case 5
1.3 The weighted case 7
1.4 The primal-dual method for the weighted bipartite matching problem . . 11

2 Network flows **17**
2.1 The Max-Flow Min-Cut Theorem 20
2.2 The Algorithm of Ford and Fulkerson 22
2.3 Dinic's Method 24
2.4 Karzanov's Improvement to Dinic's Method 27

List of Figures **32**

List of Algorithms **33**

Index **34**

Preface

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 occurs often as a subproblem witch 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.

References

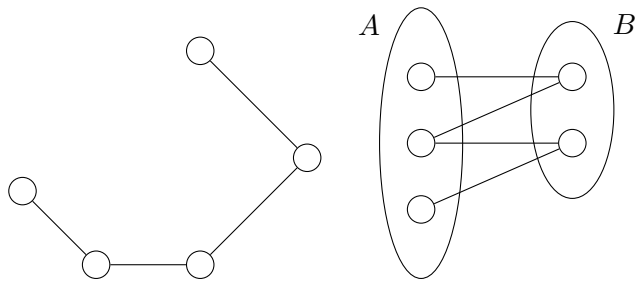
C.H. Papadimitriou, K.Steiglitz, Combinatorial Optimization: Algorithms and Complexity, Prentice-Hall 1982

N. Blum, Algorithmen und Datenstrukturen, 2. Aufl. Oldenburg 2013 (in German).

1
Bipartite Matching

1.1 Definitions and the general method

A *graph* $G = (V, E)$ consists of a finite non-empty set of *nodes* V and a set *edges* E. G is either *directed* or *undirected*. In the (un-)directed case, each edge is an (un-)ordered pair of distinct nodes. We call G *bipartite* if V can be partitioned into disjoint non-empty 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 $G = (A, B, E)$. graph bipartite graph



(a) bipartite graph G **(b)** partition of V

Figure 1.1: Some graphs

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

$$P := (v = v_0, v_1, \ldots, v_k = w)$$

witch 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 $0 \leq i \leq k$.

If there exists a path from v to w (of length 1) v is called 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 is no $e \in E \setminus M$ such that $M \cup \{e\}$ is a matching. A matching M is *maximum* if there is no matching M' of larger size. matching

Note that a maximum matching is always also a maximal matching. But there are maximal matchings which are not maximum.

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

A path $P = v_0, \ldots, v_k$ is M*-alternating* if it contains alternately edges in M and in $E \setminus M$. M-alternating

A node $v \in V$ is M*-free* if v is not incident to an edge in M. M-free

Let $P = v_0, \ldots, v_k$ be a simple M-alternating path. P is M*-augmenting* if v_0 and v_k are M-free. M-augmenting

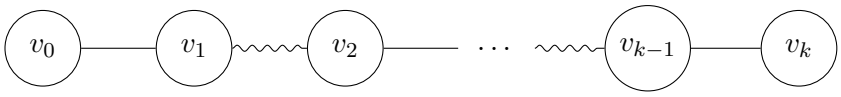


Figure 1.2: A M-augmenting path from v_0 to v_k

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, of size $|M| + 1$.

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

Proof:
„$\Rightarrow$“
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.

„$\Leftarrow$“
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 that $|M| = |M'|$.

Assume $|M| < |M'|$. Let us consider the subgraph $G' := (V, M \oplus M')$. The fact that M and M' are matchings implies that for all $v \in V$, $\deg_{G'}(v) \leq 2$. Furthermore, $\deg_{G'}(v) = 2$

implies that 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 alternating edges in M and M', or

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

$|M'| > |M|$ implies that at least one of the connected components is a path P with more edges in M' than in M. Hence, the first edge of P and also the last edge of P are in M' such that both end nodes of P are M-free. Therefore, P is an M-augmenting path. $\square$[1]

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

Algorithm 1 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;
$M := M \oplus P$
done;
$M_{\max} := M$.

The key problem now is this: How to find an M-augmenting path P, if such a path exists?

Before considering this key problem, 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

weighted undirected graph

weight of a matching

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

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

maximum weighted matching problem

Note that a matching of maximum weight is maximal, but not necessarily maximum.

[1] This symbol usually marks the end of a proof, but feel free to tick off the biscuits you'll have allowed yourself having understood each proof. It's hard work, we know.

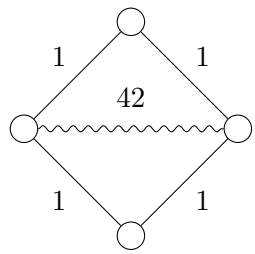


Figure 1.3: A maximum weighted matching that is not maximum

1.2 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.

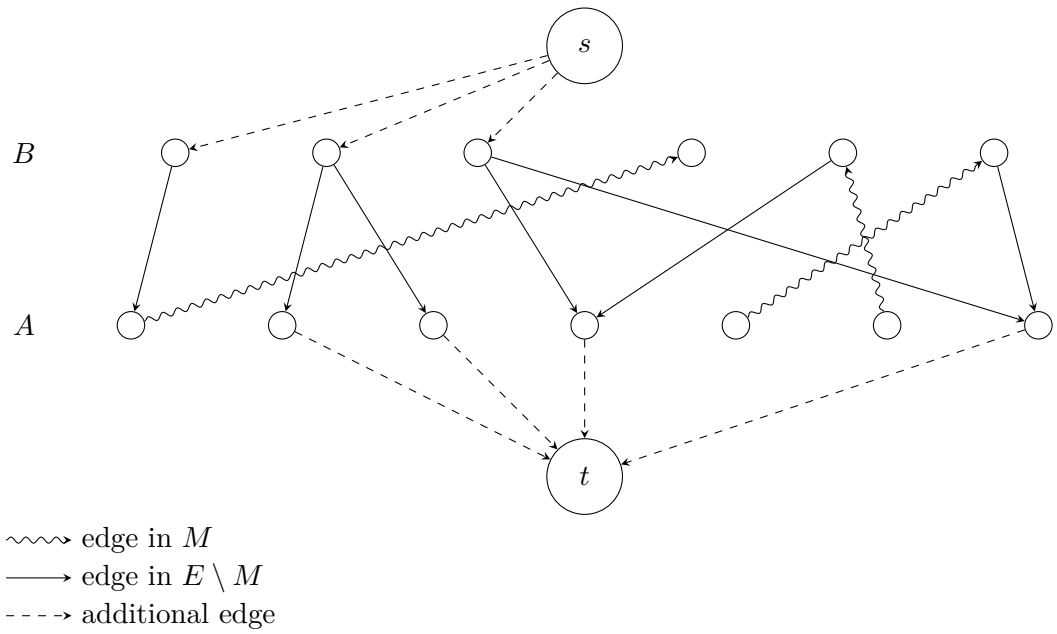


Figure 1.4: The graph $G_M = (A', B', E_M)$

Therefore, the graph $G_M = (A', B', E_M)$ is defined by

$$\begin{aligned}
&A' = A \cup \{s\}, \quad B' = B \cup \{t\}, \quad s, t \notin A \cup B, \quad s \neq t, \quad \text{and} \\
&E_M = \{(u, v) \mid \{u, v\} \in M : u \in A, v \in B\} \\
&\qquad \cup \{(x, y) \mid \{x, y\} \in E \setminus M : x \in B, y \in A\} \\
&\qquad \cup \{(s, x) \mid x \in B \text{ and } M\text{-free}\} \\
&\qquad \cup \{(y, t) \mid y \in A \text{ and } M\text{-free}\}\,.
\end{aligned}$$

Lemma 1.1: *Let $G = (A, B, E)$ be a bipartite undirected graph, $M \subseteq E$ 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:
„⇒“
Let $P = v_0, \ldots, v_k$ be an M-augmenting path in G. W. l. o. g. we can assume $v_0 \in B$.

Then there hold:

1. $v_i \in A$ for odd i,
2. $v_i \in B$ for even i,
3. $v_0 \in B$, $v_k \in A$ and both are M-free,
4. $(v_i, v_{i+1}) \in M$ for odd i, and
5. $(v_i, v_{i+1}) \in E \setminus M$ for even i or $i = 0$.

Hence, by the construction of G_M, the path $P' := s, v_0, \ldots, v_k, t$ is a simple path from s to t in G_M.

„⇐“ analogously. □

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

Algorithm 2 FindAugPath

Input: A bipartie graph $G = (A, B, E)$, and
a matching $M \subseteq E$
Output: An M-augmenting path P, if such path exists.
Method:
Construct $G_M = (A', B', E_M)$;
Apply a depth-first search with start node s to G_M;
if DFS reaches node t **then:**
return the path P contained in the stack without nodes s and t
fi.

Next, we shall analyse the time used by the algorithm FindAugPath. Let $n := |A| + |B|$ and $m := |E|$.

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

The symmetric difference $M := M \oplus P$ can be performed in $\mathcal{O}(|P|) = \mathcal{O}(n)$ 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 an undirected bipartite graph which contains no isolated nodes. Then we can compute a maximum matching of G in $\mathcal{O}(n \cdot m)$ where $n = |A| + |B|$ and $m = |E|$*

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

1.3 The weighted case

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

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

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

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

Proof:
„⇒“
Let $M \subseteq E$ be a matching of maximum weight. Obviously, neither of the two cases can be fulfilled. Otherwise, $M \oplus P$ would yield a matching of larger weight.

„$\Leftarrow$“
Let $M' \subseteq E$ be any matching of G. Consider the graph $G' := (V, M \oplus M', w)$. Each connected component of G' which is not an isolated node is either a 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: $w(M) = w(M \oplus M' \oplus M) \leq w(M)$

In case you don't know, let us quickly show that the symmetric difference is its own inverse operation. To prove that

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

we 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

$$\begin{aligned} \bar{M} \cup M &= (M \cup M') \setminus (M \cap M') \cup M \\ &= M' \cup M \end{aligned}$$

and

$$\begin{aligned} \bar{M} \cap M &= (M \cup M') \setminus (M \cap M') \cap M \\ &= M \setminus (M \cap M'), \end{aligned}$$

we obtain

$$\begin{aligned} \bar{M} \oplus M &= (M' \cup M) \setminus (M \setminus (M \cap M')) \\ &= M'. \end{aligned}$$

To prove that

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

we consider the connected component of G' which are not isolated nodes. Note that

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

Since for each connected component P of G' which is not an isolated node $\Delta(P)$ is non-positive, 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. It follows that:

$$\begin{aligned} w(M) + \Delta(R) &= w(M \oplus R) \leq w(M \oplus P) \\ w(M'') - \Delta(R) &= w(M'' \oplus R) \leq w(M) \end{aligned}$$

After the addition of both inequalities, we obtain:

$$\begin{aligned} w(M) + w(M'') &\leq w(M \oplus P) + w(M) \\ w(M'') &\leq w(M \oplus P) \end{aligned}$$

□

Applying Theorem 1.3 and Theorem 1.4, we obtain the following general method for the computation of a maximum weighted matching of a graph $G = (V, E, w)$.

Algorithm 3 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 P with maximal gain $\Delta(P)$;
    $M := M \oplus P$
  done;
  $M_{\max} := M$.

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 $t := |M_{\max}|$. Then by Theorem 1.4, for $1 \leq k \leq t$, the matching M_k, computed during the k-th 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 implies that one of the following cases is fulfilled:

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

If there is an $M_{\max}$-alternating cycle P of even length 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) $(v, w) \in M_{\max} \implies (v, w) \in P$ 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. $\square$

The key problem now is this: How do we find an M-augmenting path P with maximum gain $\Delta(P) > 0$ if such a path exists?

Note that the considerations above also hold for non-bipartite graphs. Our goal is now the 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 zero.

Observe that a depth-first search on G_M with start node s finds an M-augmenting path P if such a path exists. But $\Delta(P)$ may not be maximal within all M-augmenting paths. It follows that we need a mechanism which guarantees that $\Delta(P)$ is maximal within all M-augmenting paths.

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

The primal-dual method can be separated into rounds. Each round divides into two steps, the *search step* and the *extension step*. The input of a search step will always be a subgraph $G_M^\star$ 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 $G_M^\star$, and
3) $G_M^\star$ contains all M-augmenting paths P of G_M with $\Delta(P) = U$.

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

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

where $E_\emptyset$ is defined similar to E_M on page 6. Let $W := \max\{w_{ij} \mid (i, j) \in E\}$. We initialize the upper bound U and the input graph $G_\emptyset^\star$, for the first search step as follows:

$$\begin{aligned} U &:= W \\ G_\emptyset^\star &:= (A \cup \{s\}, B \cup \{t\}, E_\emptyset^\star, w) \end{aligned}$$

where

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

It is easy to prove that U and $G^\star_\emptyset$ fulfill the Properties 1–3.

Exercise: Prove that U and $G^\star_\emptyset$ 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 $G^\star_M = (A \cup \{s\}, B \cup \{t\}, E^\star_M, w)$ of the last unsuccessful depth-first search.

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

Question: How do we get the reduced upper bound and the input graph for the next search step?

For getting an answer to this question, let us consider the depth-first search tree T which has been constructed during the last unsuccessful depth-first search.

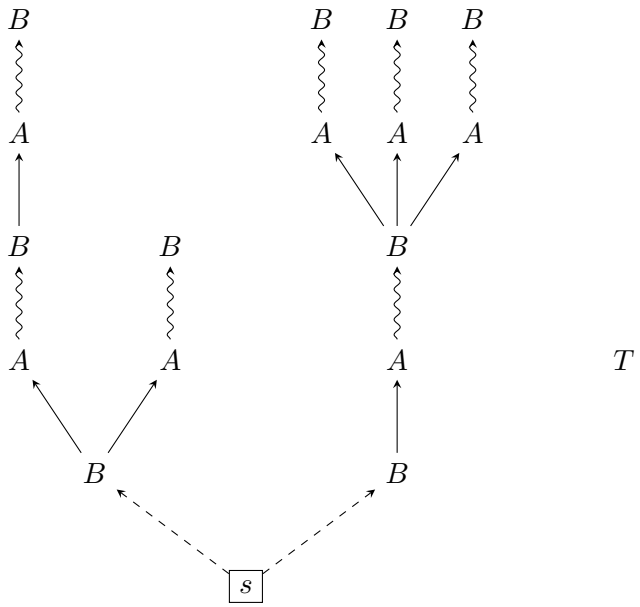


Figure 1.5: Example of an unsuccessful depth-first search

Let $B_T := B \cap T$ and $A_T := A \cap T$.

Our current goal is the reduction of the current upper bound U by an appropriate value δ and the extension of $E^\star_M$ by edges from $E_M \setminus E^\star_M$ such that the Properties 1–3 will be

fulfilled with respect to the upper bound $U - \delta$ and the constructed input graph for the next search step.

So, how do we get the appropriate δ and in dependence of δ those edges which we have to add to $E_M^\star$? The idea is that 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 the following invariants are always fulfilled:

1) $u_i + v_j \geq w_{ij}$ for all $(i, j) \in E \setminus M$,
2) $v_j + u_i = w_{ij}$ 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 $E_M^\star := M \cup \{(i, j) \in E_M \setminus M \mid u_i + v_j = w_{ij}\}$.

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: edge unbalance

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

We now want to prolong the 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 $(i, j) \notin E_M^\star \implies u_i + v_j > w_{ij} \implies \pi_{ij} > 0$

How do we change the node weights u_i and v_j such that the edge (i, j) is added to $E_M^\star$; 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 $E_M^\star$. Since we do not wish to delete some edge from $E_M^\star$, we have to modify the node weights of the other nodes in T in an appropriate manner.

Remark: We fix $\delta := \pi_{ij}$ before (i, j) enters $E_M^\star$, so that in the next search step π_{ij} will be zero.

We perform

$$\begin{aligned} u_i &:= u_i - \delta \quad \forall i \in B_T \quad \text{and} \\ v_j &:= v_j + \delta \quad \forall j \in A_T. \end{aligned}$$

Furthermore, we reduce the upper bound.

$$U := U - \delta$$

Since at the beginning $u_i = U$ and always $i \in B_T$ for all M-free nodes $i \in B$, after each extension step there holds $u_i = U$, as u_i is decreased by the same value as U. Moreover, at the beginning $v_j = 0$ and always $j \notin A_T$ for all M-free nodes $j \in A$. Hence, after each extension step, there holds $v_j = 0$ for all M-free nodes $j \in A$, as v_j is not changed.

Since δ should be chosen such that at least one edge is added to $E_M^\star$ but the decrease of U should be as small as possible, we define

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

If $\delta \geq U$, the algorithm terminates.

Exercise: Work out the algorithm in detail.

Let P be a path in G_M The *unbalance* $\pi(P)$ of the path P is defined by path unbalance

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

Next we shall 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 fulfilled.

Lemma 1.2: *Let $P = v_0, \ldots, v_k$ 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:
From the definition follows:

$$\begin{aligned} \Delta(P) &= \sum_{e \in P \cap E_M \setminus M} w_e - \sum_{e \in P \cap M} w_e \\ &= \sum_{(i,j) \in P \cap E_M \setminus M} (u_i + v_j - \pi_{ij}) - \sum_{(j,i) \in P \cap M} (v_j + u_i - \pi_{ji}) \end{aligned}$$

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

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

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 always fulfills 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:
It follows from the definition, that at the beginning; i.e. when $M = \emptyset$, all invariants are fulfilled.

We shall prove now the following: If the invariants are fulfilled before the search and extension steps, respectively, then the invariants are fulfilled after the search and extension steps, respectively as well.

The search step never changes the node weights or the upper bound. Hence, the invariants remain 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 that the modification of the node weights never violates the invariant iii). □

Theorem 1.6: *The algorithm* MaxWeiMatching*, implemented as described above, terminates with a maximum weighted matching M_{max}.*

Proof:
It is clear that the algorithm terminates with a matching $M_{\max}$. Because of Theorem 1.4, 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 augments the M-augmenting path P with $\Delta(P) > 0$ although an M-augmenting path Q with $\Delta(Q) > \Delta(P)$ exists. Let

$$P = i, \ldots, j \quad \text{and} \quad Q = p, \ldots, k.$$

Lemma 1.2 implies that $\Delta(Q) = u_p + v_k - \pi(Q)$ and $\Delta(P) = u_i + v_j - \pi(P)$. Lemma 1.3 implies that $\pi(Q) \geq 0$, $u_i = u_p$ 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 $j \in A$. Lemma 1.2 and Lemma 1.3 imply that

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

for all $M_{\max}$-augmenting paths $P = i, \ldots, j$. It follows that there is no $M_{\max}$-augmenting path P with $\Delta(P) > 0$. □

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

2
Network 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 ist the maximal quantity of bananas which can be transported to the target seaports?
3) How to ship the bananas?

For getting an answer to those questions, the manager transforms his economic problem into a mathematical problem which can be solved with a well-known algorithm. For doing this he constructs a directed graph $G = (V, E)$ where

$$\begin{aligned} V &:= \{A_1, \ldots, A_p, B_1, \ldots, B_q\} \text{ and} \\ E &:= \{(A_i, B_j) \mid 1 \leq i \leq p, 1 \leq j \leq q\}\,. \end{aligned}$$

To ensure that at most $c(A_i, B_j)$ bananas can be shipped 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, $1 \leq i \leq p$, ships at most

r_i bananas and that each seaport B_j, $1 \leq j \leq q$, obtains at most d_j bananas, two new nodes s and t, the edges (s, A_i), $1 \leq i \leq p$ with capacities $c(s, A_i) := r_i$, and the edges (B_j, t), $1 \leq j \leq q$, with capacities $c(B_j, t) := d_j$ are added to G.

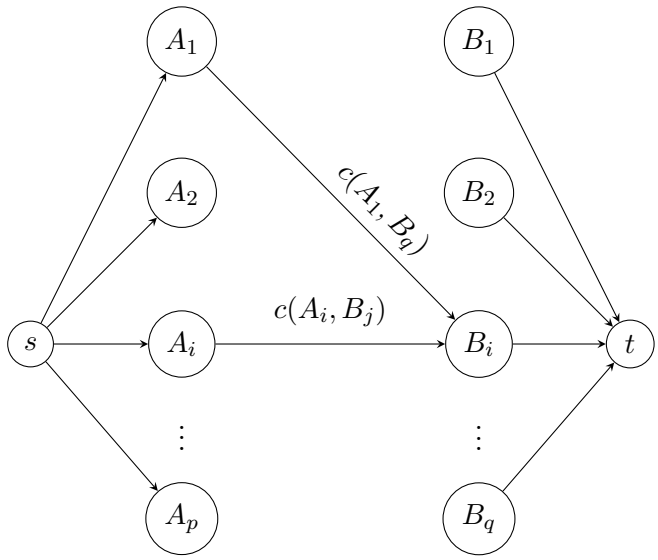


Figure 2.1: The directed graph G.

The solution of the following problem would give us 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 also has 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* $c\colon E \to \mathbb{R}^+$ which assigns to each edge $e = (v, w)$ a positive capacity $c(v, w)$. flow network

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

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

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

1) $0 \leq f(v,w) \leq c(v,w) \quad \forall (v,w) \in V \times V$ (*capacity requirement*)

2) $\sum_{e \in \mathrm{IN}(v)} f(e) = \sum_{e \in \mathrm{OUT}(v)} f(e) \quad \forall v \in V \setminus \{s,t\}$ (*Kirchhoff's Law*)

The *size* $|f|$ of a 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 *cut* $(S, V \backslash S)$ of a flow network $G = (V, E, c, s, t)$ is the partition of V into two nonempty sets S and $V \backslash S$. Note that the definition implies $\emptyset \neq S \neq V$. A cut $(S, V \backslash S)$ *separates* s and t if $s \in S$ and $t \in V \backslash S$. cut

Let $E(S, V \backslash S)$ denote the set of edges $(x,y) \in E$ with $x \in S$ and $y \in V \backslash S$. The *capacity* $c(S, V \backslash S)$ of the cut $(S, V \backslash S)$ is defined by

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

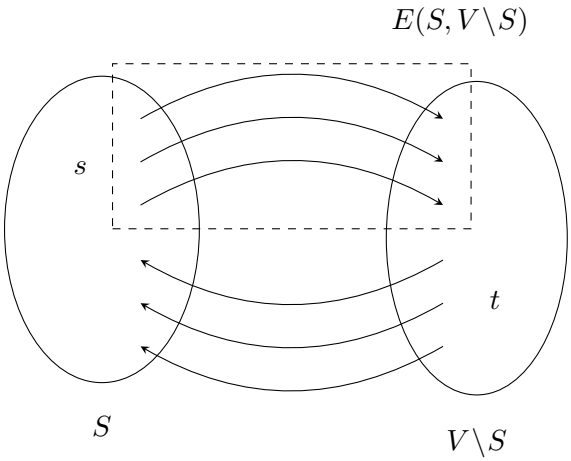


Figure 2.2: Example of a cut $(S, V \backslash S)$ separating s and t

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

2.1 The Max-Flow Min-Cut Theorem

Theorem 2.1 (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 cut $(S, V\setminus S)$ separating s and t with $c(S, V\setminus S) = F_{\max}$.

For doing this we shall develop an algorithm, which given a flow f with $|f| = F_{\max}$ computes a cut $(S, V\setminus S)$ separating s and t with $c(S, V\setminus S) = F_{\max}$. We start with $S = \{s\}$ and extend the current set S during each step by a node $y \in V\setminus S$ which has to be added to S to fulfill the assertion.

$S := \{s\}$;
while $\exists x \in S, y \in V\setminus S$: $c(x, y) > f(x, y)$ **or** $f(y, x) > 0$ **do:**
  $S := S \cup \{y\}$
done.

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

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

Since $x_{l-1} \in S$ there holds $x_{l-1} = s$ or $\exists x_{l-2} \in S$ which causes the addition of x_{l-1} to S and so on. This implies that there exists an undirected path $P := (s = x_0, x_1, \ldots, x_{l-1}, x_l = t)$ with $x_j \in S$ for $0 \leq j \leq l$.

Let for $0 \leq i < l$: $\varepsilon_i := \max\{c(x_i, x_{i+1}) - f(x_i, x_{i+1}), f(x_{i+1}, x_i)\}$. Observe, that by construction: $\varepsilon_i > 0,\ 0 \leq i \leq l$.

Let $\varepsilon := \min\{\varepsilon_i \mid 0 \leq i \leq l\}$. We shall obtain a contradiction by the construction of a flow $f^\star$ such that $|f^\star| = F_{\max} + \varepsilon$. For obtaining the flow $f^\star$, we change the flow on the undirected edges $(x_i, x_{i+1}),\ 0 \leq i < l$ by

$$\begin{cases} f^\star(x_i, x_{i+1}) := f(x_i, x_{i+1}) + \varepsilon & \text{if } \varepsilon_i = c(x_i, x_{i+1}) - f(x_i, x_{i+1}) \\ f^\star(x_{i+1}, x_i) := f(x_{i+1}, x_i) - \varepsilon & \text{otherwise.} \end{cases}$$

The flow on all other edges is not changed.

The choice of ε implies that $f^\star$ does not destroy the capacity requirement. Next, we shall prove that $f^\star$ fulfills Kirchhoff's Law, too. For doing this let us consider any node $x_i,\ 0 < i < l$. The following four cases can arise:

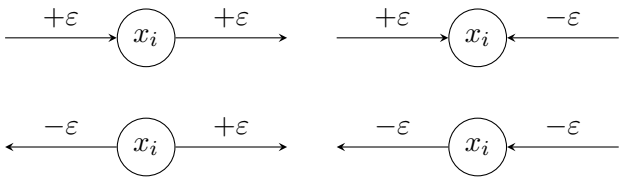


Figure 2.3: Changed flow on edges with end node x_i on P

In all cases, the validity of Kirchhoff's Law before the flow changes implies the validity of Kirchhoff's Law after the flow changes. Therefore, $f^\star$ is a flow. Furthermore,

$$\begin{aligned}
|f^\star| &= \sum_{v\in V} f^\star(v,t) - \sum_{v\in V} f^\star(t,v) \\
&= \sum_{v\in V\setminus\{x_{l-1}\}} f(v,t) + f^\star(x_{l-1},t) - \sum_{v\in V\setminus\{x_{l-1}\}} f(t,v) - f^\star(t,x_{l-1}) \\
&= \sum_{v\in V\setminus\{x_{l-1}\}} f(v,t) + f(x_{l-1},t) + \varepsilon - \sum_{v\in V\setminus\{x_{l-1}\}} f(t,v) - f(t,x_{l-1}) \\
&= \sum_{v\in V} f(v,t) - \sum_{v\in V} f(t,v) + \varepsilon \\
&= |f| + \varepsilon \\
&= F_{\max} + \varepsilon
\end{aligned}$$

This is a contradiction to the maximality of f. Hence, $t \notin S$, i.e. $(S, V\setminus S)$ separates s and t. By the construction, $\forall x \in S\, \forall y \in V\setminus S\colon f(x,y) = c(x,y)$ and $f(y,x) = 0$.

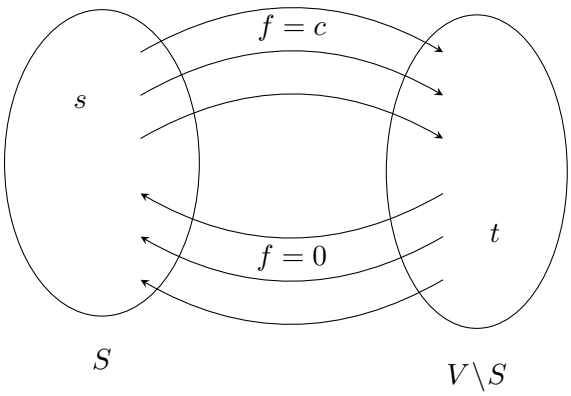


Figure 2.4: A minimum cut with a maximum flow from s to t

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. □

2.2 The Algorithm of Ford and Fulkerson

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. We obtain the following algorithm for the construction of a maximum flow in a flow network:

augmenting path

Algorithm 4 FORD-FULKERSON

Input: A flow network $G = (V, E, c, s, t)$.
Output: A maximum flow f in G.
Method:
  Start with the zero flow f_0, i.e. $\forall (x,y) \in E\colon f(x,y) = 0$.
  $f := f_0$;
  $found := false$;
  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;
  done;
  return f.

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

Theorem 2.2 (Integrality Theorem): *Let $G = (V, E, c, s, t)$ be a flow network where all capacities are integers. Then the 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. Thus, 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

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

Even with integral capacities, the algorithm Ford-Fulkerson may perform many augmentations. Therefore, we shall look for an answer to the following 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? 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 *augmenting path* P is a path from s to t in G' such that augmenting path

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

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

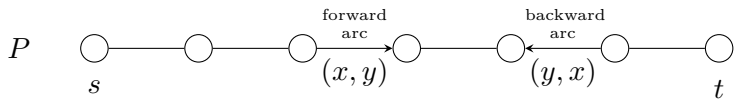


Figure 2.5: Backward and forward arcs on a path P

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 *augment the flow on P by ε*. augment the flow

Our goal is the development of rules which choose the augmenting paths in such a way that the method of Ford and Fulkerson terminates after the augmentation of „few“ augmenting paths.

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 $m = |E|$. In total, it runs in $\mathcal{O}(m^2 \cdot n)$ time. Independently, Dinic observed that it would be useful to consider all shortest augmenting paths simultaneously. As long as such a path exists, one augments such a path.

2.3 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 blocking flow useful edge

$$e = (x, y) \text{ and } f(x, y) < c(x, y) \qquad \text{or} \qquad e = (y, x) \text{ and } f(y, x) > 0.$$

On an augmenting path, all edges are useful.

Dinic's algorithm separates into phases. A phase obtains as input a flow network $G = (V, E, c, s, t)$ and a current flow $f\colon E \to \mathbb{R}$ 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 5 CONSTRUCTION G_f

Input: A flow network $G = (V, E, c, s, t)$, and
a flow f in G.
Output: The layered network $G_f = (V', E', \tilde{c}, s, t)$ or
„f is already maximum“.
Method:
$V_0 := \{s\}$;
$i := 0$;
loop:
$H := \{v \in V \setminus (\bigcup_{j \le i} V_j) \mid \exists u \in V_i$ and edge $e \in E$ which is useful from u to v $\}$;
$E(H) := \{(u, v) \mid u \in V_i, v \in H$ and $e \in E$ which is useful from u to v $\}$;
if $H = \emptyset$ **then:**
return „f is already maximum“
fi;
if $t \in H$ **then:**
$l := i + 1$;
$V_l := \{t\}$;
$E_l := \{(u, t) \mid u \in V_i$ and $\exists e \in E$ which is useful from u to t $\}$;
return $G_f := (\bigcup_j V_j, \bigcup_j E_j, \tilde{c}, s, t)$ ▷ $\tilde{c}$ is defined below
else:
$V_{i+1} := H$;
$E_{i+1} := E(H)$;
$i := i + 1$;
goto loop
fi.

Lemma 2.1: *If* CONSTRUCTION G_f *terminates because of $H = \emptyset$, then the current flow f is already a maximum flow.*

Proof:
Let $S := \bigcup_{j=0}^{i} V_j$. Then by construction $s \in S$ and $t \in V \backslash S$. This implies that $(S, V \backslash S)$ is an s,t-separating cut. All edges from S to $V \backslash S$ are saturated and all edges from $V \backslash S$ to S have flow zero. Hence, the assertion follows from the Max-Flow Min-Cut Theorem. □

For $(u, v) \in E_i,\ 1 \leq i \leq l$, we define its *reduced capacity* $\tilde{c}(u, v)$ by

$$\tilde{c}(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{if } u \in V_{i-1} \\ f(v,u) & \text{if } u \in V_i. \end{cases}$$

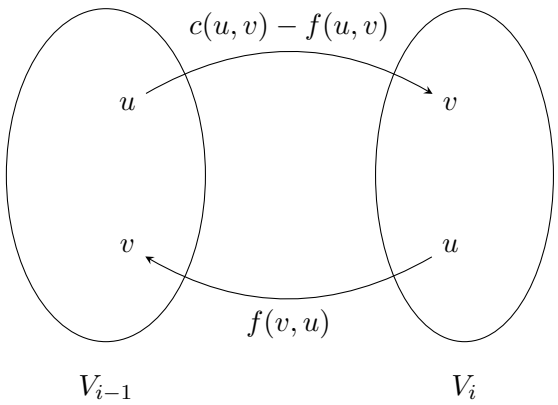


Figure 2.6: Reduced capacity of edges between V_{i-1} and V_i

Note that the edges in G_f are directed from V_{i-1} to V_i 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 2.2: *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'.

We shall prove now that the number of phases is bounded by $|V| = n$. It suffices to 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 k-th phase.

Lemma 2.3: *If the k-th phase is not the last phase then $l_{k+1} > l_k$.*

Proof:
Assume that after the $(k-1)$-th phase, there is an augmenting path. We shall prove that after the k-th phase, the length of a shortest augmenting path increases. Let us consider the layered network constructed during the k-th phase.

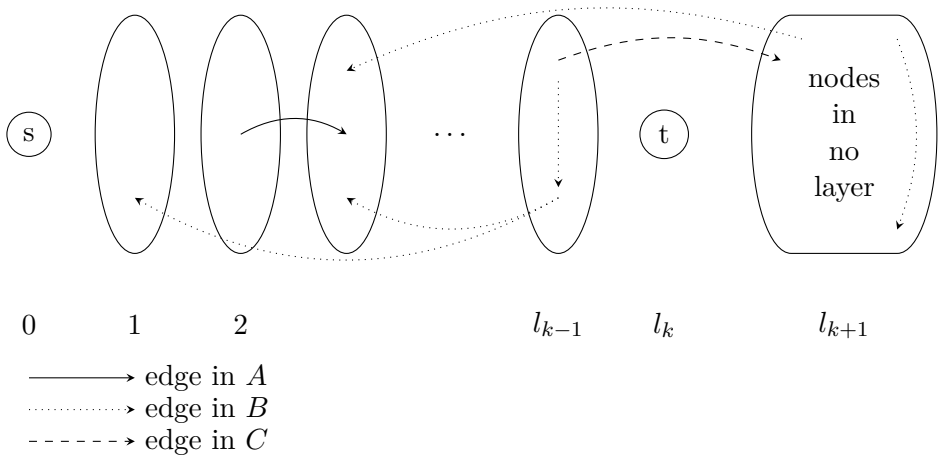


Figure 2.7: The layered network during the k-th phase with example edges in A, B, C

We divide the useful edged in G into three classes:

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

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

C: useful edges from layer l_{k-1} or l_k to layer l_{k+1}.

From the construction follows:

1) $E' = A$; i.e. G_f contains exactly the edges in A. This implies that after the k-th phase, the edges in B and in C are useful in the same manner as before. An edge in A remains 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 k-th phase, a shortest augmenting path P has to pass through each layer; i.e. $|P| \geq l_k$.

3) Since during the k-th phase, a blocking flow has been constructed and the corresponding augmentations have been performed, there is an edge e on P 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 $i < l_k$.

In both cases, the edge e prolongs the length of P by at least one. Hence, $|P| \geq l_k+1$.

□

Corollary: *The number of phases is at most n.*

Altogether, we have proved the following theorem.

Theorem 2.3: *Let $G = (V, E, c, s, t)$ be a flow network. Then a maximum flow f_{max} can be constructed on $\mathcal{O}(n \cdot m + n \cdot b)$ 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.
   If during the DFS a backtrack is performed then we can delete the corresponding node and all incident edges from G_f.

2) Augment this path P.
   If after the augmentation $\bar{f}(e) = \tilde{c}(e)$ 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.

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 $\mathcal{O}(m+n)$ 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 $\mathcal{O}(m+n)$. An augmenting path has length $\leq n$. Hence, the total time used for the augmentations is bounded by $\mathcal{O}(m \cdot n)$. Altogether, we have proved the following theorem.

Theorem 2.4: *Dinic's algorithm computes a maximum flow in time $\mathcal{O}(n^2 \cdot m)$.*

2.4 Karzanov's Improvement to Dinic's Method

Can we improve Dinic's method for the computation of a blocking flow? In 1974 Karzanov realized the following idea: During each step, saturate a node instead of an edge. Our goal is to develop Karzanov's method. First, we need some notations.

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

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

A flow f *saturates a node* v if

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

A node $v \in V$ is *blocked* if on each path from v to the sink t there is a saturated edge. The idea now is the following: 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 $\sum_{e \in \mathrm{OUT}(v)} c(e)$. In that case, we reduce the surplus of v, and we look for another way to transport this surplus to t. We take care that a blocked node remains blocked. First, we need some further notations.

Let $G = (V, E, c, s, t)$ be a flow network. A *preflow* $p\colon E \to \mathbb{R}_{\geq 0}$ is a function which fulfills the following requirements: preflow

1) $0 \leq p(e) \leq c(e) \quad \forall e \in E$ and
2) $\underbrace{\sum_{e \in \mathrm{IN}(v)} p(e)}_{p_{\mathrm{in}}(v)} \geq \underbrace{\sum_{e \in \mathrm{OUT}(v)} p(e)}_{p_{\mathrm{out}}(v)} \quad \forall v \in V \setminus \{s, t\}.$

A node $v \in V$ is called *unbalanced* if $p_{\mathrm{in}}(v) > p_{\mathrm{out}}(v)$. A preflow p is a flow if 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 an 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 always blocked. We balance the preflow p at an unbalanced node v such that $p_{\mathrm{in}}(v) = p_{\mathrm{out}}(v) = c(v)$. Hence, the node v remains blocked. After balancing a node v, we do not allow any further change of the incoming preflow of v.

An edge is *open* if it is allowed to change $p(e)$. Otherwise, e is *closed*. At the beginning, all edges are open.

To realize the first step, for all nodes $v \in V$, we number the outgoing edges of v in any fixed order. The idea is now to transport the incoming preflow of a node v 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 $\mathrm{OUT}(v)$, the following invariant will always be 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 $k(v)$. All further edges have zero preflow.

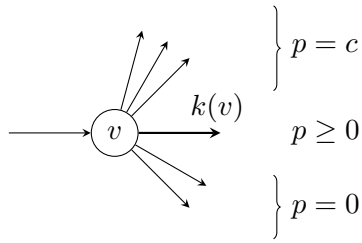


Figure 2.8: Illustration of the invariant, with $k(v)$ in bold

For the organisation of the second step, we use a stack $\mathcal{K}_w$ for the management of the incoming edges $e \in \mathrm{IN}(w)$ of a node w with $p(e) > 0$. At the moment when an edge $e \in \mathrm{IN}(w)$ obtains a preflow $p(e) > 0$, the edge e is pushed. If during Step 2 the incoming preflow of the node w has to be reduced, the preflow of the uppermost edge in $\mathcal{K}_w$ is reduced, then the preflow on the next edge on $\mathcal{K}_w$, and so on.

Altogether, we obtain the following realizations of the steps of Karzanov's method.

Algorithm 6 KARZANOV - 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 $e \in \mathrm{OUT}(s)$ **do:**
    $p(e) := \tilde{c}(e)$
  done;
  $START := 1$;

(2) **for** $i := START$ **to** $l - 1$ **step** 1 **do:**
    for all $v \in V_i$ **do:**
      $P := p_{\mathrm{in}}(v)$;
      while $P > 0$ **and** $k(v)$ defined **do:**
        $e := k(v)$;
        $P' := \min\{\tilde{c}(e) - p(e), P\}$;
        $p(e) := p(e) + P'$;
        $P := P - P'$
      done
    done
  done.

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

1) $p_{\text{in}}(v) = p_{\text{out}}(v)$, or

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

Algorithm 7 KARZANOV - STEP 2 Balancing of the preflow in G_f

Let V_r denote the layer of highest index which contains an unbalanced node.

(1) **for all** unbalanced nodes $v \in V_r$ **do:**
Using $\mathcal{K}_v$, reduce the incoming preflow of v until v is balanced,
i.e. $p_{\text{in}}(v) = p_{\text{out}}(v) = \tilde{c}(v)$;
Close the incoming edges of v
done;

(2) $START := r - 1$;
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 2.4: *During the whole algorithm, a blocked node remains blocked.*

Proof:
After the first performance of STEP 1, all outgoing edged of s are saturated. Hence, s is blocked.

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

Claim: Each node which is blocked before STEP 2, remains blocked after the performance of STEP 2.

Proof of claim: For all nodes $v \in V_j,\ j \geq r$, the preflow $p(e)$ does not change for all edges e on a path from v to t in G_f. Therefore, if v is blocked before the performance of STEP 2 then it is also blocked after the performance of STEP 2. Consider $v \in V_r$ with v unbalanced. Note that v is blocked.

STEP 2 reduces the incoming preflow of v. Assume that the preflow $p(e)$ for $e = (w,v)$ and $w \in V_{r-1}$ is reduced. Consider $x \in V_j,\ j < r$ with x 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. Therefore, x remains blocked after the reduction of $p(e)$. $\square$

Since the source node s is blocked after the first performance of Step 1, Lemma 2.4 implies that Karzanov's method computes a blocking flow. It remains the analysis of the used time.

Lemma 2.5: *Karzanov's method finds a blocking flow in $\mathcal{O}(n^2)$ time.*

Proof:
Note that after the reduction of the preflow on an edge e, the edge e is closed. Hence, the number of preflow reductions 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 balanced node has at most one edge e such that $p(e)$ is increased and e is not saturated. It follows that 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 $\mathcal{O}(n^2)$. $\square$

List of Figures

1.1 Some graphs 2
1.2 A M-augmenting path from v_0 to v_k 3
1.3 A maximum weighted matching that is not maximum 5
1.4 The graph $G_M = (A', B', E_M)$ 5
1.5 Example of an unsuccessful depth-first search 12

2.1 The directed graph G. 18
2.2 Example of a cut $(S, V \setminus S)$ separating s and t 19
2.3 Changed flow on edges with end node x_i on P 21
2.4 A minimum cut with a maximum flow from s to t 21
2.5 Backward and forward arcs on a path P 23
2.6 Reduced capacity of edges between V_{i-1} and V_i 25
2.7 The layered network during the k-th phase with example edges in A, B, C 26
2.8 Illustration of the invariant, with $k(v)$ in bold 29

List of Algorithms

1 Maximum Matching 4
2 FindAugPath 6
3 MaxWeiMatching 9

4 Ford-Fulkerson 22
5 Construction G_f 24
6 Karzanov - Step 1 Forward movement of preflow in G_f 29
7 Karzanov - Step 2 Balancing of the preflow in G_f 30

Index

blocking flow, 24

capacity function, 18
cut, 19

edge unbalance, 13

flow, 18
- blocked node, 28
- blocking flow, 24
- capacity, 19
- capacity requirement, 19
- Kirchhoff's Law, 19
- node capacity, 28
- reduced capacity, 25
- saturated node, 28
- separating flow, 19
- size, 19

flow network, 18
- augmenting path, 23
  - augment the flow, 23
  - backward arc, 23
  - forward arc, 23
- capacity function, 18
- saturated edge, 24
- sink node, 18
- source node, 18
- useful edge, 24

graph, 2
- bipartite, 2
- directed, 2
- edge, 2
- node, 2
- undirected, 2
- weighted graph, 4

Kirchhoff's Law, 19

matching, 3
- M-alternating path, 3
- M-augmenting path, 3
- M-free node, 3
- maximal, 3
- maximum, 3
- maximum weight, 4

maximum matching problem, 3
maximum weighted matching problem, 4

path, 2
- direct predecessor, 3
- direct successor, 3
- gain, 7
- length, 2
- M-alternating, 3
- M-augmenting, 3
- predecessor, 3
- simple, 2
- successor, 3
- unbalance, 14

preflow, 28
- closed node, 28
- open node, 28
- unbalanced node, 28

symmetric difference, 3