Shortest augmenting path

An even better way to choose the augmenting path is: *choose the path with the minimum number of arcs*.

We define a distance function $d : N \mapsto \mathbb{Z}$, such that

- $d(t) = 0$
- $d(i) \leq d(j) + 1, \forall (i,j) \in A_R$.

It represents a lower bound to the number of arcs a unit of flow must traverse to go from any node to node t in the residual graph.

If $d(s) \geq n$, then there no augmenting paths in the residual graph.

Exact distance

We say the distance is exact when

- $d(t) = 0$
- $d(i) = 1 + \min_{(i,j) \in A_R} \{d(j)\}$.

The exact distance $d(i)$ represents the minimum number of arcs from node i to node t in the residual graph.

An arc (i, j) is admissible if and only if $d(i) = 1 + d(j)$.
A path is admissible if and only if it contains only admissible arcs.

Visualization

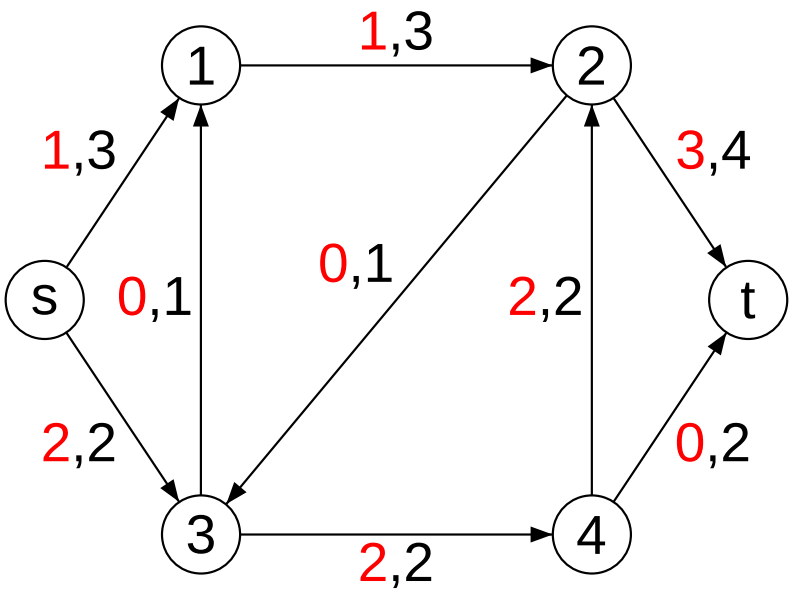

Figure: Flow on the original digraph.

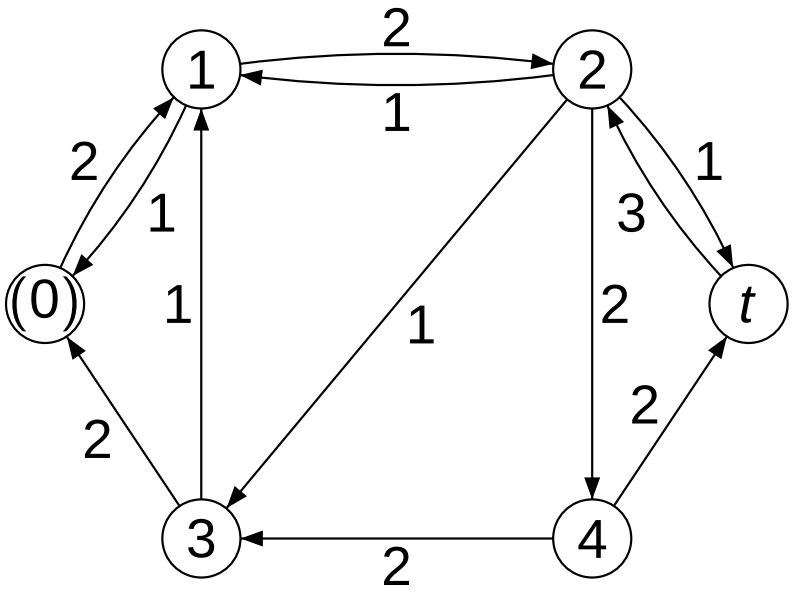

Figure: The residual digraph.

Visualization

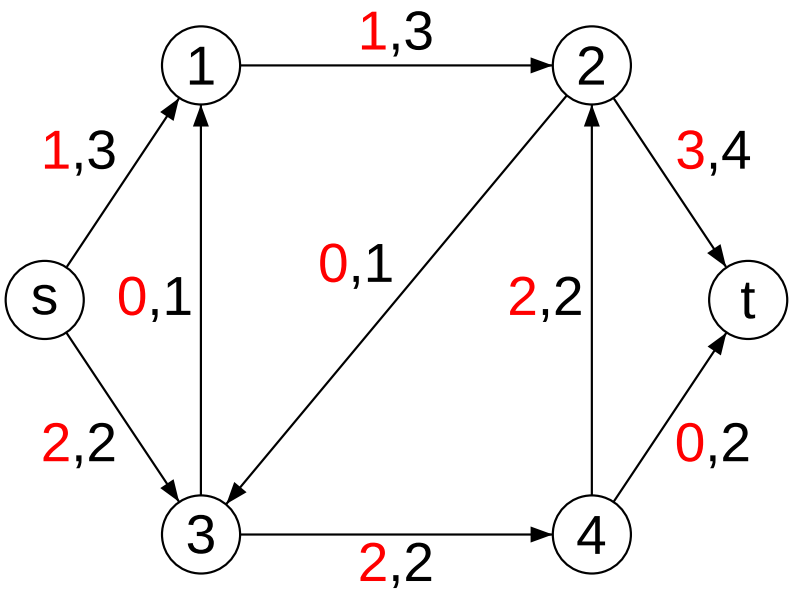

Figure: Flow on the original digraph.

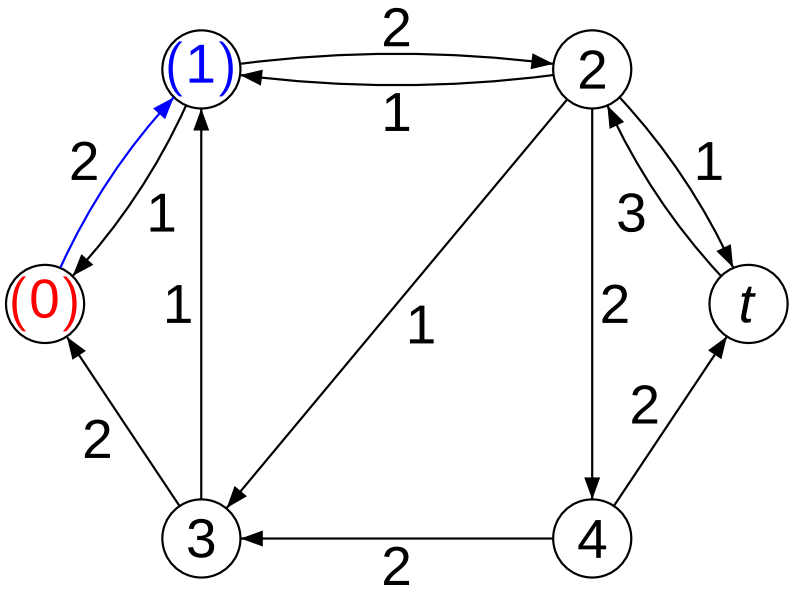

Figure: The residual digraph.

Visualization

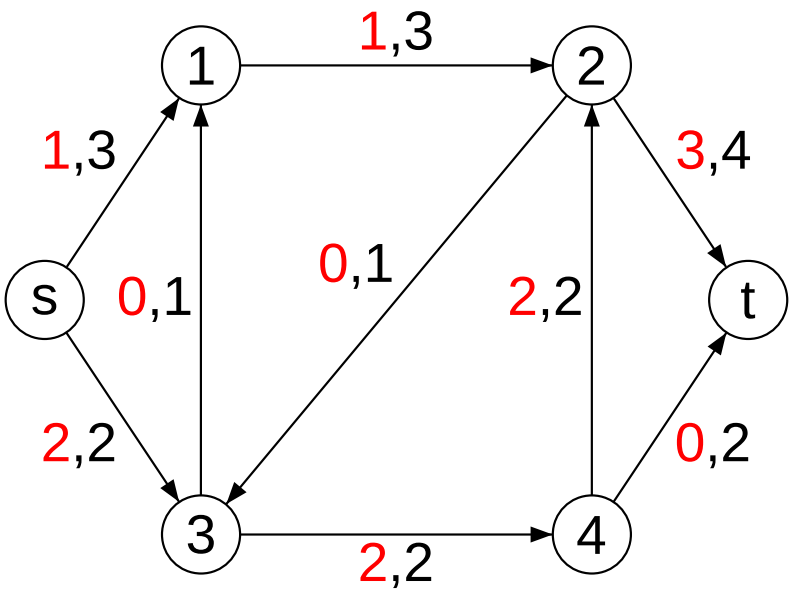

Figure: Flow on the original digraph.

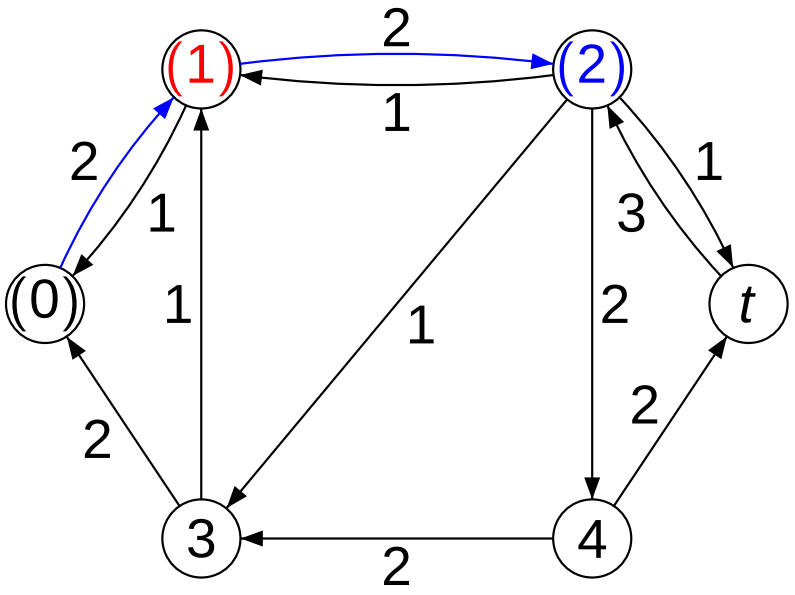

Figure: The residual digraph.

Visualization

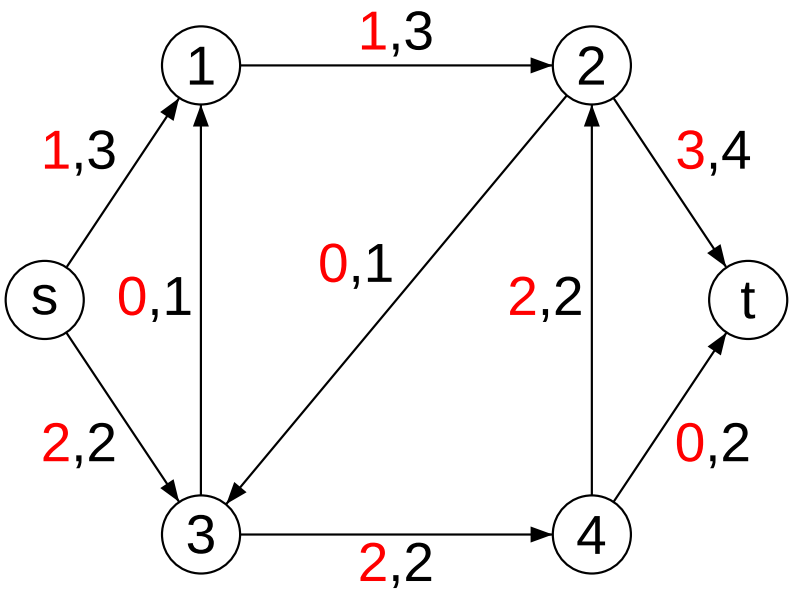

Figure: Flow on the original digraph.

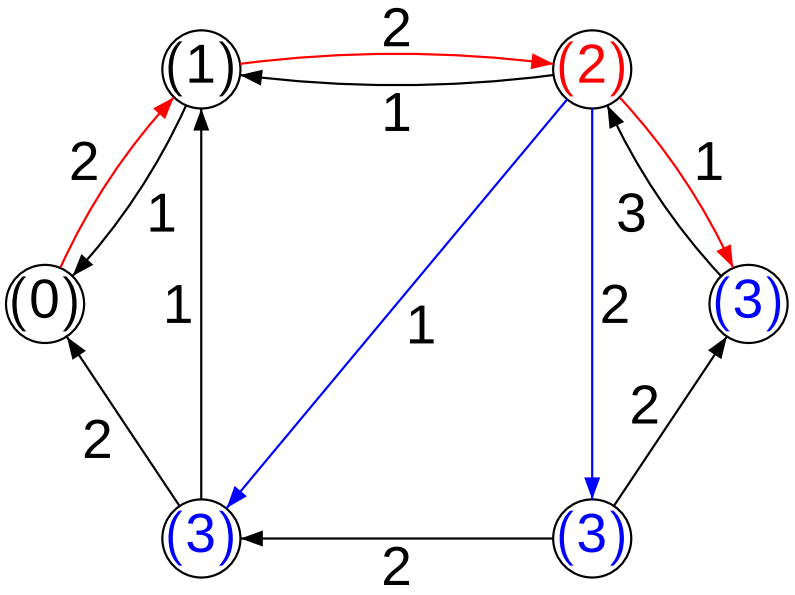

Figure: The residual digraph.

The augmenting path is made of 3 arcs and $\delta = 1$.

Shortest augmenting path algorithm

If BFS were executed for each augmentation, it would take $O(m)$ each time and the number of augmentations would be $O(nm)$. Hence the resulting complexity would be $O(nm^2)$ which is not so good.

The complexity can be improved by a variation of the DFS algorithm to visit the residual graph, that exploits the property of the distance function of being non-decreasing.

This algorithm uses three basic operations:

- *Advance*
- *Retreat*
- *Augment*

Shortest augmenting path algorithm

x:=0;
d:=*ExactDistance*;
i:=s;
while $d(s) < n$ **do**
    if $\exists(i,j)$ admissible **then**
        Advance(i);
        if $i = t$ **then** *Augment*; i:=s;
    else *Retreat*(i);

Procedures

procedure *Advance* (*i*);

pred(*j*):=*i*;
i:=*j*;

procedure *Retreat* (*i*);

$d(i)$:=$1 + \min\{d(j) : r(i,j) > 0\}$;
if $i \neq s$ **then** *i*:=*pred*(*i*);

Correctness

Every *Advance*, *Retreat* or *Augment* operation maintains valid distance values *d*.

Each *Retreat* operation strictly increases the distance of some node.

The proof is by induction, assuming *d* is valid before an operation and proving that it remains valid after it.
The basis of the induction is the initial labeling by *ExactDistance*, which is valid.

Operation *Advance* does not modify distances and capacities; hence it does not affect the validity of *d*.

Correctness

Operation *Augment* modifies residual capacities on the arcs of the augmenting path P.
Such arcs must be admissible: if $(i,j) \in P$ then $d(i) = d(j) + 1$.

When (i,j) is saturated and disappears from the residual graph, this deletes a constraint ($d(i) \leq d(j) + 1$).

A new arc (j,i) may appear in the residual graph and this generates a new constraint: $d(j) \leq d(i) + 1$.

However this is always satisfied because $d(i) = d(j) + 1$ and therefore the constraint is $d(j) \leq (d(j) + 1) + 1$.

Correctness

Operation *Retreat* does not modify residual capacities r, but it implies relabeling some node i, changing $d(i)$ into

$$d'(i) = 1 + \min_{(i,j):r_{ij}>0} \{d(j)\}.$$

This is done when no admissible arc leaves i:

$$\nexists (i,j) : (r_{ij} > 0) \wedge (d(i) = 1 + d(j)).$$

This means that $d(i) < 1 + d(j) \;\; \forall (i,j) : r_{ij} > 0$.
Therefore:

- $d'(i) > d(i)$ (the distance is strictly increasing).
- $d'(i) \leq 1 + d(j) \;\; \forall (i,j) : r_{ij} > 0$ (the distance remains valid for all arcs outgoing from i).

For incoming arcs, we have $d(k) \leq 1 + d(i) \;\; \forall (k,i) : r_{ki} > 0$.
Since $d'(i) > d(i)$ the inequality remains true:

$$d(k) \leq 1 + d'(i) \;\; \forall (k,i) : r_{ki} > 0.$$

Complexity: scanning the out-star

To establish the computational complexity of the algorithm we need to define the data-structure that is used: we assume that all arcs of A_i, i.e. outgoing from each node i are arranged in a list. The list is scanned to search for admissible arcs. Every time the current node is set to i, the search in A_i is resumed from the last reached position. When the search reaches the end of the list without finding admissible arcs, the node distance d_i is recomputed and the search restarts from the head of the list.

Between two relabeling operations the search for admissible arcs takes $O(|A_i|)$ because each outgoing arc is considered once. When a relabeling occurs, the time needed to compute the new distance d_i' is $O(|A_i|)$, because each successor node must be considered once.

If each node is relabeled at most k times, these operations take $O(km)$ overall.

Complexity: saturating arcs

If the algorithm relabels each node at most k times, then it does not saturate arcs more than km times.

Between two consecutive saturations of an arc (i, j), both distances d_i and d_j must have been recomputed.

Therefore if each distance is updated at most k times, then each arc is saturated at most k times.

Complexity

The values of the distances are always bounded by n.
Therefore each distance label increases at most n times.
Therefore the number of *Retreat* operations is at most $O(n^2)$.
Therefore the number of saturations is at most nm.
Therefore the number of *Augment* operations (complexity $O(n)$) is at most nm.
The number of *Advance* operations is $O(n^2 + n^2m)$, because

- $O(n^2m)$ *Advance* operations are needed to build $O(nm)$ augmenting paths of length $O(n)$;
- $O(n^2)$ *Advance* operations are "undone" by $O(n^2)$ *Retreat* operations.

Therefore the shortest augmenting path algorithm runs in $O(n^2m)$.

Improvements

The shortest augmenting path algorithm terminates only when $d(s) \geq n$.

When a minimum $(s-t)$-cut is saturated, it separates the digraph into two subsets of nodes

- S, reachable from s,
- $\overline{S}$, unreachable from s.

The algorithm keeps recomputing $d(i)$ for nodes $i \in S$ until $d(s) \geq n$.
Idea: immediately detect when an $s-t$-cut becomes saturated.

Data-structure

Keep an additional array of integers, *number*, indexed from 0 to $n-1$. The value in $number[k]$ is the number of nodes whose label d is equal to k.

Initialization: with BFS, when computing the initial exact distances.

Update: at each $Relabel(i)$, *number* is updated:
$number[d(i)] \leftarrow number[d(i)] - 1$
EndTest
$d(i) \leftarrow d(j) + 1$
$number[d(i)] \leftarrow number[d(i)] + 1$

The *EndTest* is simply $number[d(i)] = 0$.

End test

When $number[k] = 0$ for some value k such that $S' = \{i \in N : d(i) > k\}$ and $S'' = \{i \in N : d(i) < k\}$ are non-empty, then S' and S'' define the saturated $(s - t)$-cut.

Proof.

- $s \in S'$; otherwise there would be no point in relabeling a node i with $d(i) = k$.
- $t \in S''$, because $d(t) = 0$.
- By the definition of S' and S'', $d(i) > d(j) + 1 \ \ \forall i \in S', \ \forall j \in S''$, because $d(i) > k > d(j)$.
- For the properties of a valid distance function $d(i) \leq d(j) + 1$ for all arcs in the residual graph. Hence $r_{ij} = 0 \ \ \forall i \in S', \ \forall j \in S''$.

Therefore (S', S'') is a saturated (s, t)-cut.

Improvements of the capacity scaling algorithm

We have derived an $O(m^2 \log U)$ bound for the capacity scaling algorithm.

It can be improved to $O(nm \log U)$ using the shortest augmenting path as a subroutine.

We define the distance labels on the restricted residual digraph using only the arcs in $A_R(x, \Delta) = \{(i,j) \in \mathcal{A}_R : r_{ij} \geq \Delta\}$.

Each scaling phase includes only $O(m)$ augmentations.
Hence the overall complexity of the augmentations is $O(nm)$, instead of $O(n^2 m)$.

Data-structure

The worst-case bound $O(nm)$ to the number of augmentations is tight.

Improvements can be obtained only by reducing the time needed for each augmentation.

This is obtained with dynamic trees, that reduce the average time for each augmentation from $O(n)$ to $O(\log n)$ and consequently the complexity of the shortest augmenting path algorithm from $O(n^2 m)$ to $O(nm \log n)$.

Layered network

A layered network consists of the arcs (i, j) of the residual graph satisfying the condition $d(i) = d(j) + 1$.

It can be constructed by BFS or DFS in $O(m)$.

Nodes are partitioned into subsets $N_0, N_1, N_2, \ldots, N_{d^*}$ such that nodes in N_k are at distance k from t.
Arcs of the layered network connect nodes in consecutive layers.

Therefore every (s, t)-path in the layered network is a shortest (s, t)-path.