Maximum flow problem (part I)

Giovanni Righini

University of Milan

UNIVERSITÀ DEGLI STUDI DI MILANO

Definitions

A flow network is a digraph $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ with two particular nodes s and t acting as *source* and *sink* of a flow.

The flow is a quantity that can traverse the arcs from their tails to their heads, starting from s and reaching t.

The digraph $\mathcal{D}$ is weighted with a capacity $u : \mathcal{A} \mapsto \Re_+^m$.

The capacity of an arc is the upper limit to the amount of flow that can traverse the arc.

- An arc with no flow is empty.
- An arc with a flow equal to its capacity is saturated.

The max-flow problem

An (s, t)-cut is a subset of arcs such that if they are removed from $\mathcal{A}$, the resulting digraph $\mathcal{D}'$ is disconnected and s and t belong to different components $\mathcal{S}'$ and $\mathcal{T}'$.

Every (s, t)-cut C is made by a subset $F(C)$ of *forward arcs* and a subset $B(C)$ of *backward arcs*.

An arc $(i, j) \in C$ is forward if $i \in \mathcal{S}'$ and $j \in \mathcal{T}'$ and backward in the other case.

The capacity of a cut C is $u(C) = \sum_{(i,j)\in\mathcal{F}(C)} u_{ij}$.

Problem (Maximum Flow Problem). Find a maximum flow from s to t not exceeding any arc capacity.

A formulation

We use a continuous and non-negative variable x_{ij} to indicate the amount of flow on each arc $(i, j) \in \mathcal{A}$.

A mathematical model of the max-flow problem is:

$$\text{maximize} \quad z$$

$$\text{s.t.} \sum_{i \in \mathcal{N}:(i,j) \in \mathcal{A}} x_{ij} - \sum_{i \in \mathcal{N}:(j,i) \in \mathcal{A}} x_{ji} = \begin{cases} -z & j = s \\ 0 & \forall j \neq s, t \\ z & j = t \end{cases}$$

$$0 \leq x_{ij} \leq u_{ij} \qquad \forall (i, j) \in \mathcal{A}.$$

Constraints impose:

- flow conservation,
- arc capacity.

Constraint matrix

The coefficients in the left-hand-side of the constraints

$$\sum_{i \in \mathcal{N}:(i,j) \in \mathcal{A}} x_{ij} - \sum_{i \in \mathcal{N}:(j,i) \in \mathcal{A}} x_{ji}$$

define a totally unimodular matrix, since each variable appears exactly once with coefficient 1 and once with coefficient -1.

Therefore if capacities u are all integer, the optimal flows x are also integer.

The dual problem: min $s-t$ cut

As with any LP model, one can write the dual problem:

maximize z

$$\text{s.t.} \sum_{i\in\mathcal{N}:(i,j)\in\mathcal{A}} x_{ij} - \sum_{i\in\mathcal{N}:(j,i)\in\mathcal{A}} x_{ji} = \begin{cases} -z & j=s \\ 0 & \forall j \neq s,t \\ z & j=t \end{cases} \quad [y_j]$$

$$0 \leq x_{ij} \leq u_{ij} \qquad \forall (i,j)\in\mathcal{A} \quad [\omega_{ij}]$$

$$\text{minimize } w = \sum_{(i,j)\in\mathcal{A}} u_{ij}\omega_{ij}$$

$$\begin{aligned} \text{s.t. } & y_j - y_i + \omega_{ij} \geq 0 && \forall (i,j)\in\mathcal{A} && [x_{ij}] \\ & y_s - y_t \geq 1 && && [z] \\ & y_i \text{ unrestricted} && \forall i \in N \\ & \omega_{ij} \geq 0 && \forall (i,j)\in\mathcal{A}. \end{aligned}$$

Primal: max $(s-t)$ flow problem. Dual: min $(s-t)$ cut problem.

Max-flow min-cut: weak duality

Given any feasible flow x of value z and any $(s-t)$-cut C with capacity $u(C)$, we have:

$$z(x) = \sum_{(i,j)\in F(C)} x_{ij} - \sum_{(i,j)\in B(C)} x_{ij}$$

and therefore

$$z(x) \leq u(C).$$

This is the **Max-flow Min-cut theorem** in its weak form.

It is a special case of the weak duality theorem.

Max-flow min-cut: strong duality

The **Max-flow Min-cut theorem** in strong form is:

$$\exists x^*, C^* : z(x^*) = u(C^*).$$

This is a special case of the strong duality theorem for linear programming.

The maximum flow we can send from s to t is equal to the minimum capacity of an (s, t)-cut.

The minimum capacity (s, t)-cut, C^*, is also called the *bottleneck*.

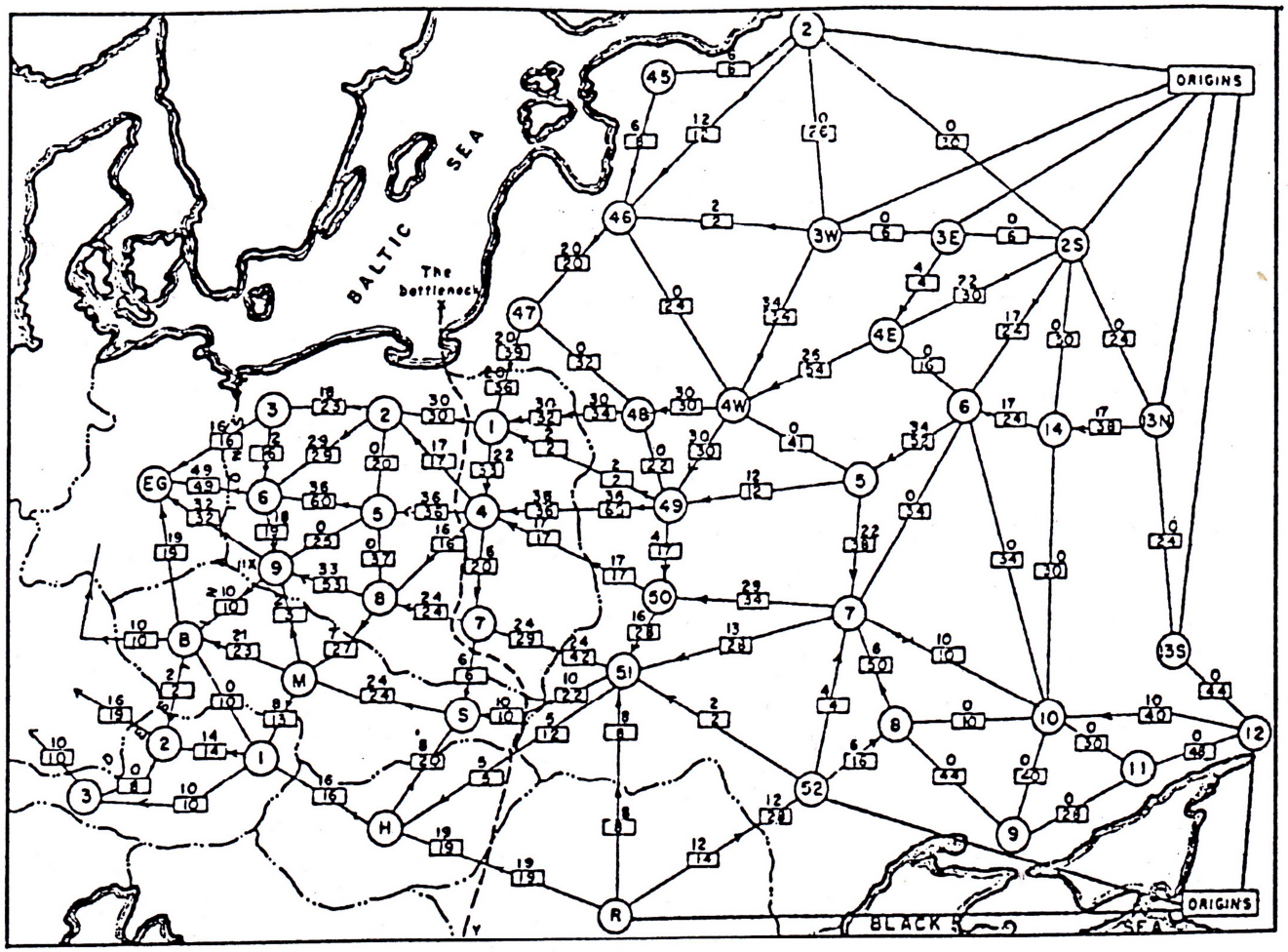


Figure 10.2

From Harris and Ross [1955]: Schematic diagram of the railway network of the Western Soviet Union and East European countries, with a maximum flow of value 163,000 tons from Russia to Eastern Europe, and a cut of capacity 163,000 tons indicated as 'The bottleneck'.

Figure: Source: A.Schrijvers, *Combinatorial optimization*, Springer 2003

The residual digraph

Given a flow x on the weighted digraph $\mathcal{D} = (\mathcal{N}, \mathcal{A})$, we define an auxiliary weighted digraph $\mathcal{R} = (\mathcal{N}, \mathcal{A}_R)$, called the residual digraph, with the following arcs $\mathcal{A}_R$ and weights w:

- an arc $(i, j) \in \mathcal{A}_R$ for each $(i, j) \in \mathcal{A}$ which is not saturated, with $w_{ij} = u_{ij} - x_{ij}$;
- an arc $(j, i) \in \mathcal{A}_R$ for each $(i, j) \in \mathcal{A}$ which is not empty, with $w_{ji} = x_{ij}$.

In the former case we can increase the flow on (i, j); in the latter case we can decrease it. The capacity w in the residual digraph represents the maximum allowed increase/decrease of flow in the original digraph, such that $0 \leq x_{ij} \leq u_{ij}$.

We can increase the overall flow from s to t by sending flow along any augmenting path in the residual digraph.

The flow is maximum if and only if there are no augmenting paths.

An example

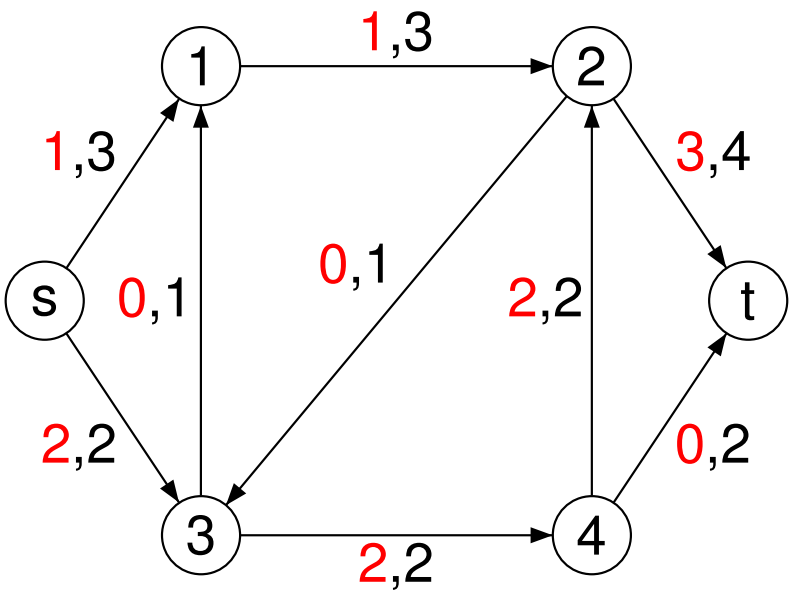


Figure: Flow on the original digraph.

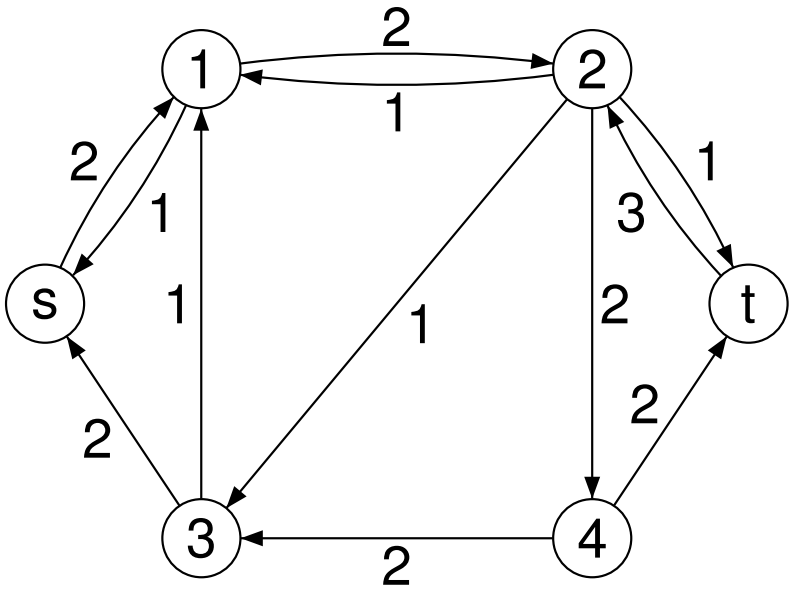


Figure: The residual digraph.

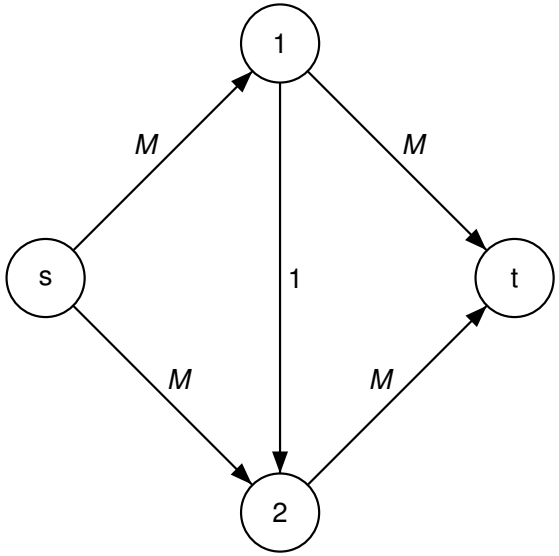
Ford-Fulkerson algorithm (1956)

for $(i,j) \in \mathcal{A}$ **do**
  x_{ij}:=0;
$\mathcal{A}_R := \emptyset$;
for $(i,j) \in \mathcal{A}$ **do**
  $\mathcal{A}_R$:=$\mathcal{A}_R \cup \{(i,j)\}$; λ_{ij} := true; w_{ij}:=u_{ij};
  $\mathcal{A}_R$:=$\mathcal{A}_R \cup \{(j,i)\}$; λ_{ji} := false; w_{ji}:=0;
while $\exists Path(\mathcal{A}_R)$ **do**
  δ:=min$\{w_{ij} : (i,j) \in Path(\mathcal{A}_R)\}$;
  for $(i,j) \in Path(\mathcal{A}_R)$ **do**
    if λ_{ij} **then**
      x_{ij}:=$x_{ij} + \delta$; w_{ij}:=$w_{ij} - \delta$; w_{ji}:=$w_{ji} + \delta$;
    else
      x_{ji}:=$x_{ji} - \delta$; w_{ij}:=$w_{ij} - \delta$; w_{ji}:=$w_{ji} + \delta$;

Computing the augmenting paths

After every iteration one more arc is saturated; δ is always strictly positive. Hence, if capacities are integer, the algorithm converges to an optimal solution in a finite number of steps.

However its computational complexity depends on how the augmenting path is chosen.



For a bad choice of the paths ($(s, 1, 2, t)$ and $(s, 2, 1, t)$ alternately) the number if iterations may depend on the capacities (and it may be very large).

Complexity

Assume U is the maximum capacity:

$$U = \max_{(i,j) \in A} \{u_{ij}\}$$

Then the minimum capacity among all (s, t)-cuts is limited by nU.

Each augmenting path can be detected in $O(m)$.

The update of the residual graph takes $O(n)$, because no more than n arcs can belong to the augmenting path.

Hence the computational complexity of the max-flow algorithm can be as bad as $O(nmU)$.

Maximum capacity augmenting paths

A possible choice is: *choose the augmenting path that maximizes* δ.

This can be done with a variation of Dijkstra's algorithm for shortest paths: instead of minimum cost paths, we look for maximum capacity paths.

The label of each node represents the maximum amount of flow that can be sent from s to that node.

Dijkstra's algorithm for max capacity paths

$T := \emptyset$;
for $i \in \mathcal{N}$ **do**
  $y_i \leftarrow 0$; $\pi_i \leftarrow nil$; $flag(i) \leftarrow 0$
$y_s \leftarrow \infty$; $\pi_s \leftarrow s$; $flag(s) \leftarrow 1$
$last \leftarrow s$;
while $flag(t) = 0$ **do**
  for $i \in \mathcal{N}$ **do**
    if $(flag(i) = 0) \wedge \min\{y_{last}, c_{last,i}\} > y_i)$ **then**
      $y_i \leftarrow \min\{y_{last}, c_{last,i}\}$; $\pi_i \leftarrow last$
  $y^{max} \leftarrow 0$
  for $i \in \mathcal{N}$ **do**
    if $(flag(i) = 0) \wedge (y_i > y^{max})$ **then**
      $last \leftarrow i$; $y^{max} \leftarrow y_i$
  $T \leftarrow T \cup \{(\pi_{last}, last)\}$; $flag(last) \leftarrow 1$

Visualization

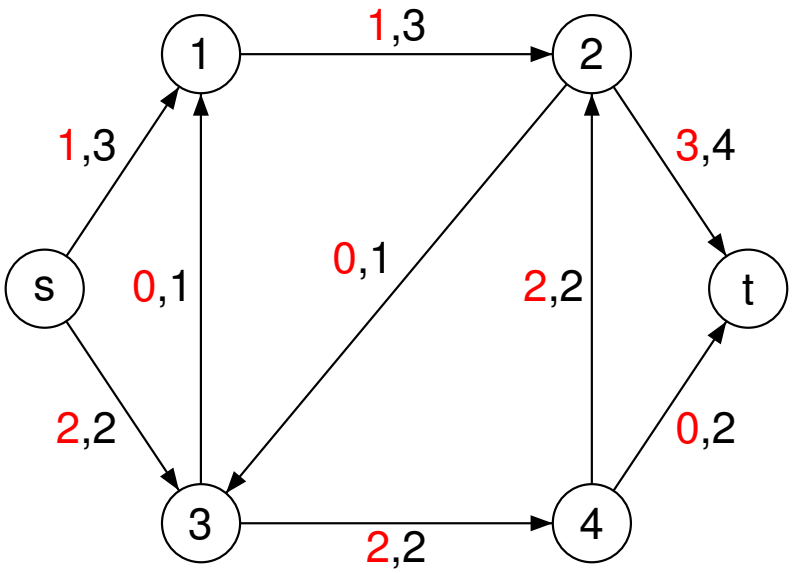


Figure: Flow on the original digraph.

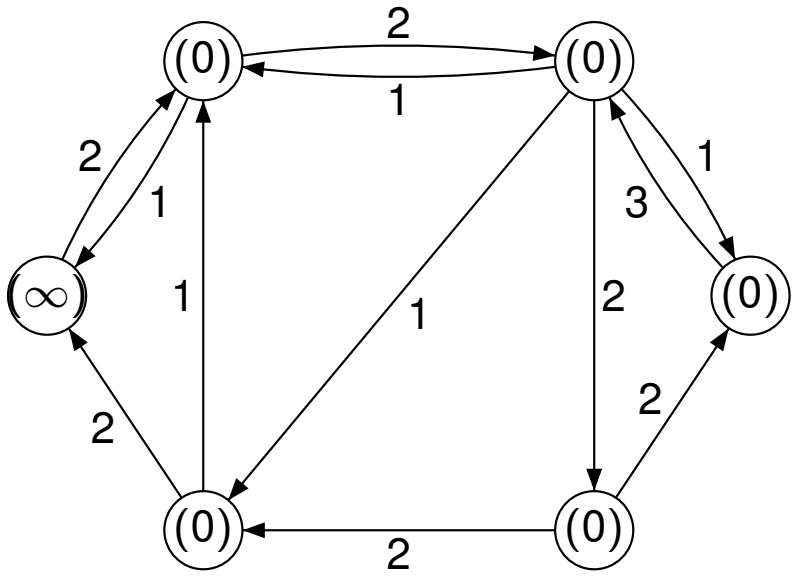


Figure: The residual digraph.

Visualization

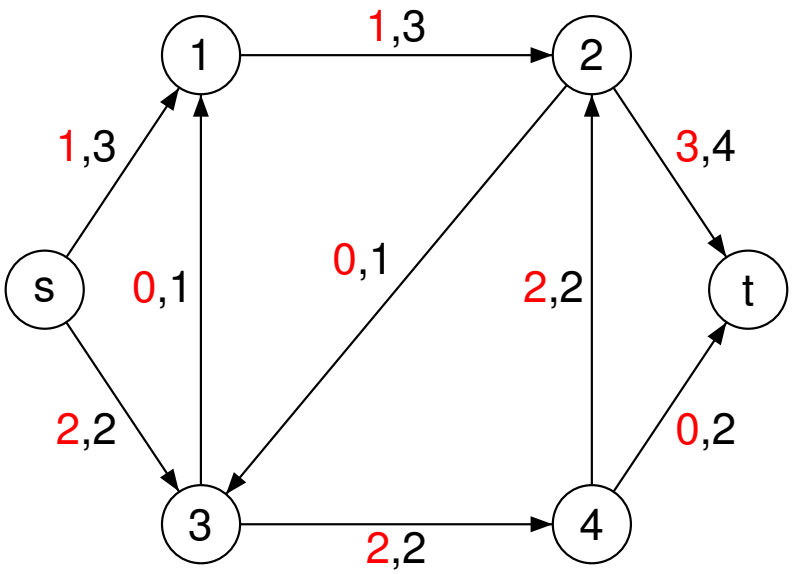


Figure: Flow on the original digraph.

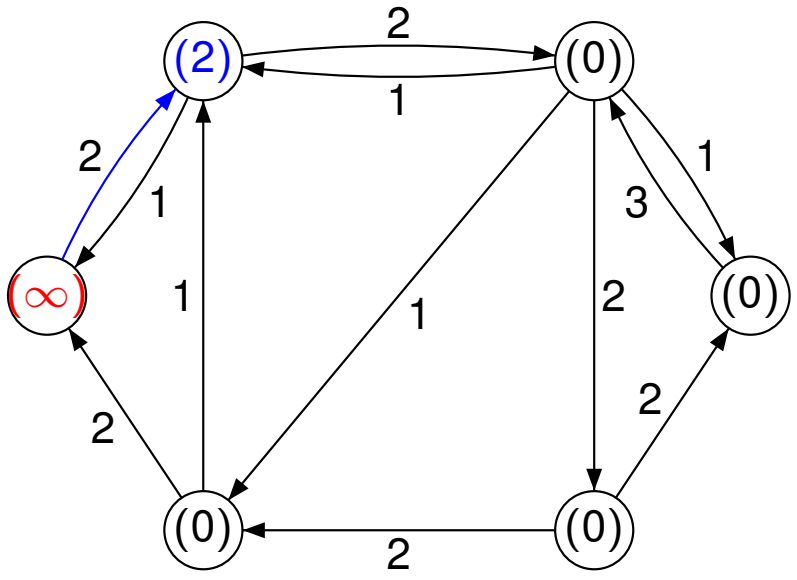


Figure: The residual digraph.

Visualization

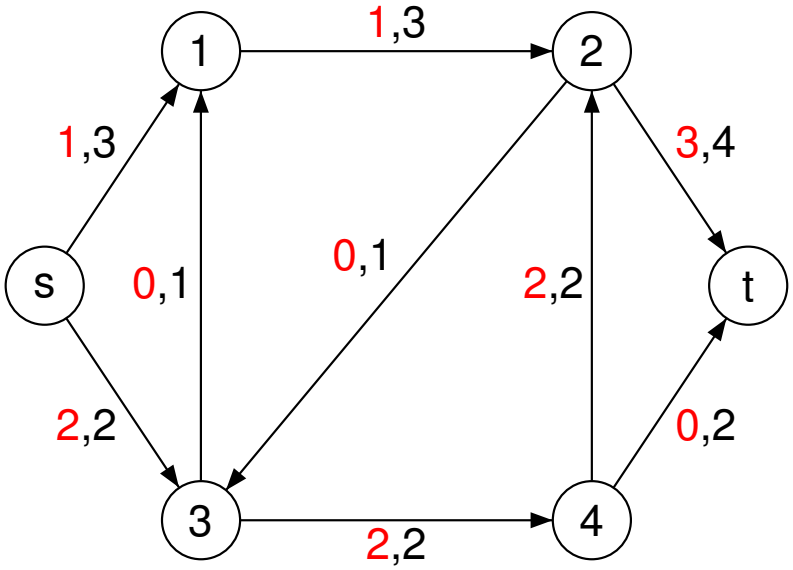


Figure: Flow on the original digraph.

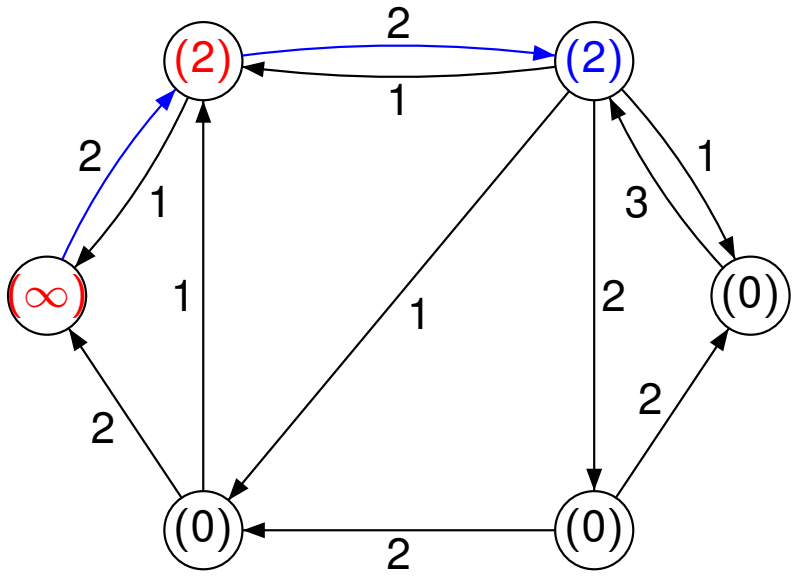


Figure: The residual digraph.

Visualization

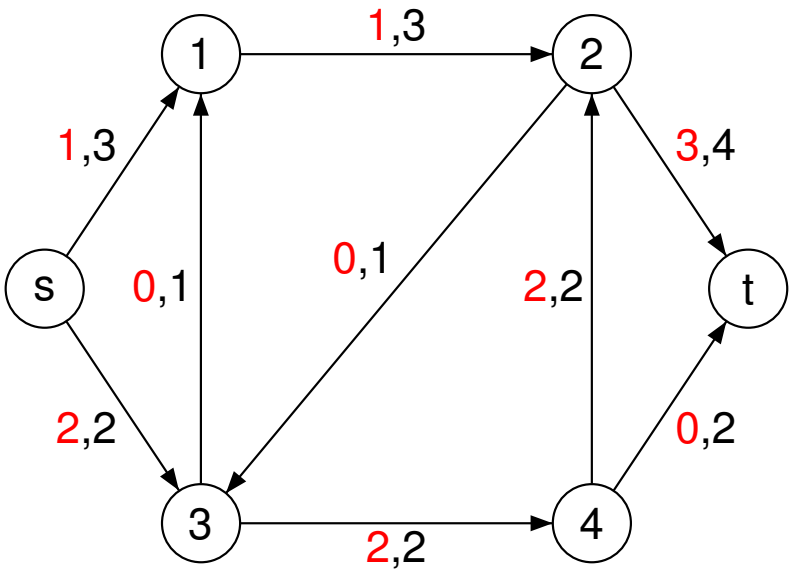


Figure: Flow on the original digraph.

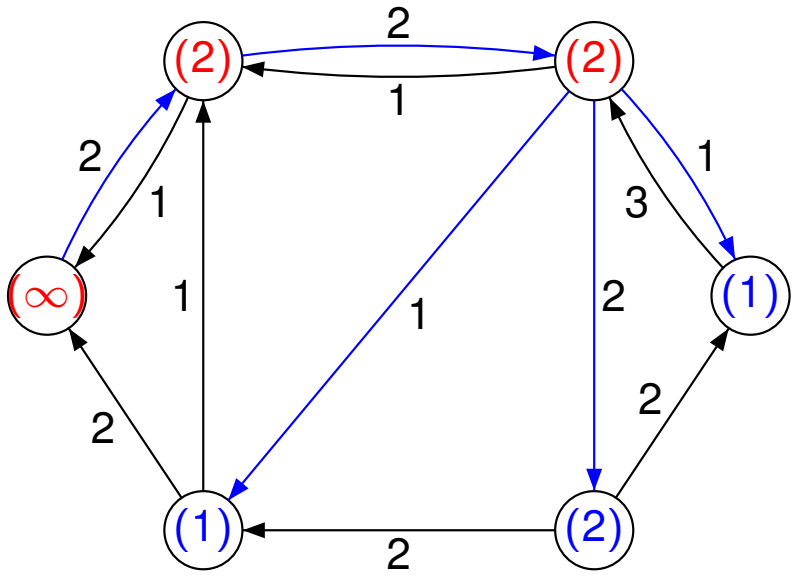


Figure: The residual digraph.

Visualization

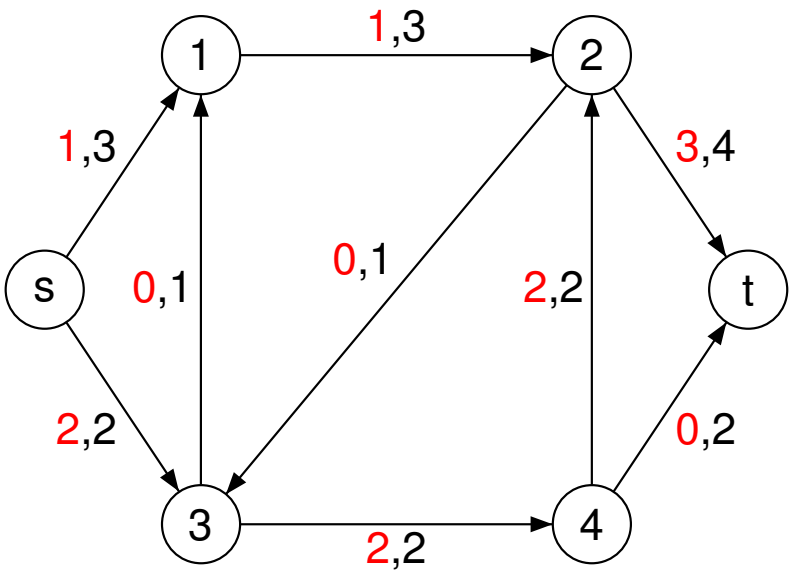


Figure: Flow on the original digraph.

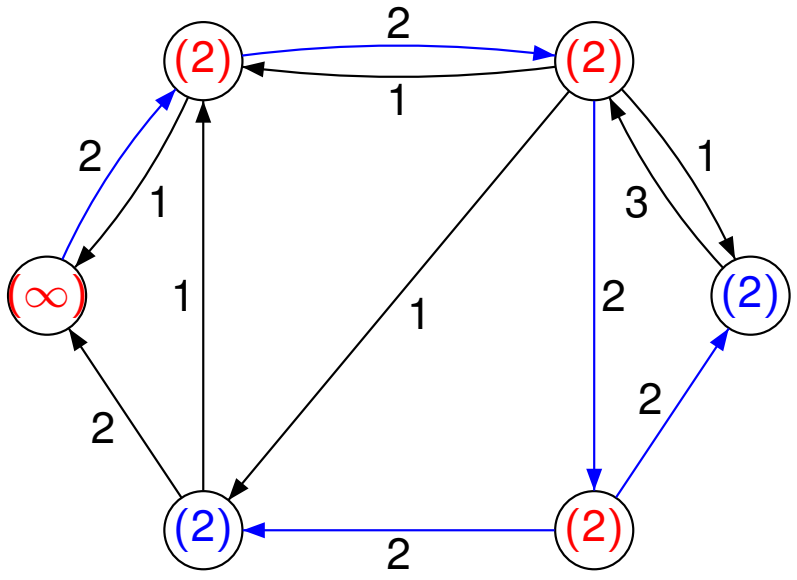


Figure: The residual digraph.

Visualization

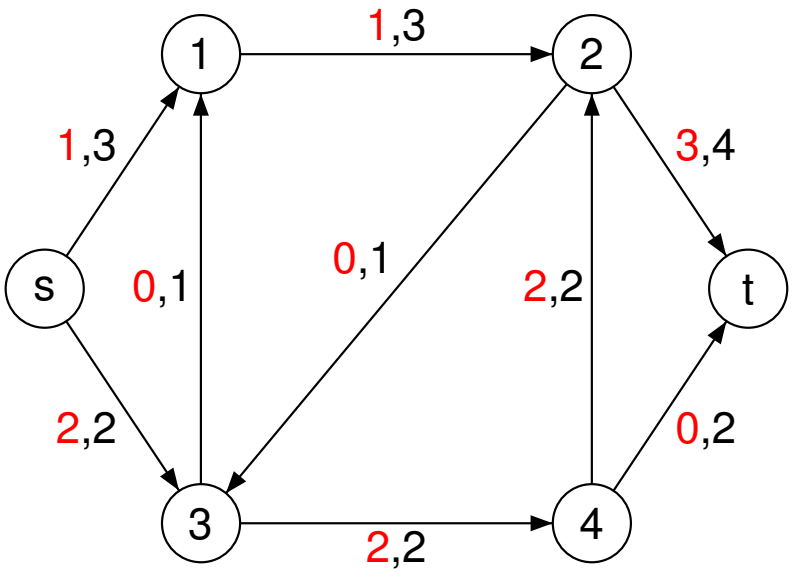


Figure: Flow on the original digraph.

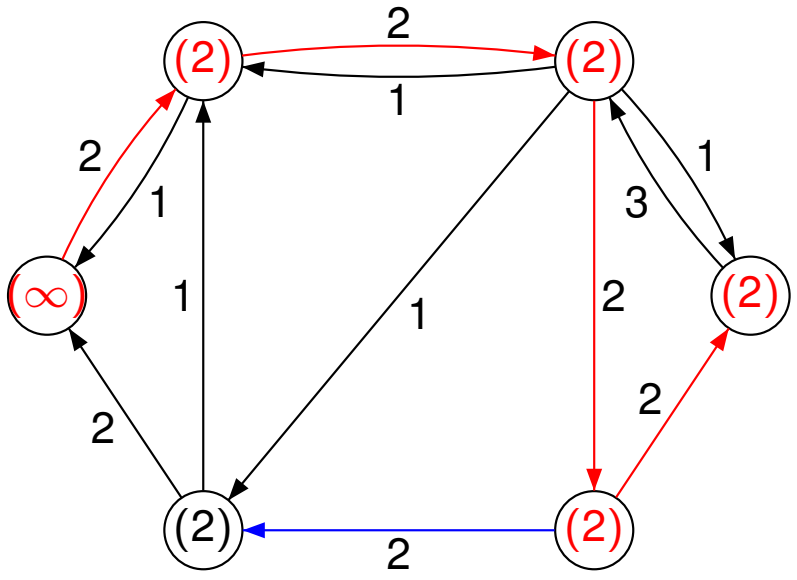


Figure: The residual digraph.

The augmenting path is made by 4 arcs and $\delta = 2$.

Complexity

The complexity of Dijkstra algorithm (basic implementation) is $O(n^2)$.

In order to find the complexity of the overall algorithm, we have to bound the number of iterations, i.e. the number of augmentations needed to reach the maximum flow.

Assume that:

- x is a current flow,
- z is its value,
- z^* is the value of the maximum flow.

Complexity

If there are no more than m (s,t)-paths with overall residual capacity $z^* - z$, then the maximum capacity of a path is

$$\delta \geq \frac{z^* - z}{m}.$$

Consider a sequence of $2m$ consecutive augmentations, indexed by $k = 1, \ldots, 2m$; the corresponding maximum capacities δ_k are obviously non-increasing.

If all of them have $\delta_k \geq \frac{z^* - z}{2m}$, then z^* is reached and the algorithm stops.

Else, the last maximum capacity must be $\delta_{2m} < \frac{z^* - z}{2m} \leq \frac{1}{2}\delta_1$, i.e. the value of the maximum capacity has been at least halved.

Since $1 \leq \delta_k \leq U$ for all k, then $O(m \log U)$ iterations are sufficient to reach the optimum.

Complexity

With respect to the general case

- the number of iterations is $O(m \log U)$ instead of $O(nU)$,
- but the complexity of each iteration is $O(n^2)$

and this yields an overall complexity of $O(mn^2 \log U)$ for the max-flow algorithm.

Improvement: capacity scaling algorithm, where the augmenting path is not chosen as a maximum capacity path but as a path with large enough capacity, i.e. with a capacity not smaller than a suitable threshold.

Capacity scaling

For any given current flow x, the arc set in the residual graph is

$$A_R(x) = \{(i,j) \in N \times N : ((i,j) \in A \wedge x_{ij} < u_{ij}) \vee ((j,i) \in A \wedge x_{ji} > 0)\}$$

Let indicate by r_{ij} the residual capacity or each arc of $A_R(x)$.

Let $\Delta \geq 1$ be an integer parameter.

We define

$$\overline{A}_R(x, \Delta) = \{(i,j) \in A_R(x) : r_{ij} \geq \Delta\}$$

as the set of arcs with a “large enough” residual capacity.

The value of Δ is iteratively decreased; a scaling phase is the set of operations that are done with constant value of Δ.

Capacity scaling: correctness

During each scaling phase only the arcs in $\overline{A}_R(x, \Delta)$ are used to find augmenting paths.

Therefore $\delta \geq \Delta$ for all augmenting paths.

A scaling phase ends when no augmenting paths exist in $\overline{A}_R(x, \Delta)$.

When $\Delta = 1$ all arcs in the residual graph are considered.

Therefore, if a scaling phase with $\Delta = 1$ terminates, then no augmenting paths exist and the flow is maximum.

Capacity scaling: complexity

To establish the complexity we have to prove

- an upper bound on the number of scaling phases;
- an upper bound on the number of augmentations during each scaling phase;
- the complexity of each augmentation.

Number of scaling phases.
The value of Δ is initialized at $2^{\lfloor \log U \rfloor}$ and iteratively decreased to 1 by halving it $\lfloor \log U \rfloor$ times.

Therefore the number of scaling phases is $1 + \lfloor \log U \rfloor$.

Complexity

Number of augmentations during each scaling phase.
Consider a scaling phase p ending with flow x' and value z'.
Let indicate by S the set of nodes reachable from s in $\overline{A}_R(x', \Delta)$.
Let $\overline{S}$ be the complement of S.
Since there are no augmenting paths in $\overline{A}_R(x', \Delta)$, then $t \notin S$ and $r_{ij} < \Delta$ for all arcs (i, j) traversing the cut $(S, \overline{S})$ (from S to $\overline{S}$).

Therefore the residual capacity of the cut $(S, \overline{S})$ is $r(S, \overline{S}) < m\Delta$.

For the max-flow min-cut theorem, we obtain $z^* - z' < m\Delta$.

The next scaling phase $p + 1$ will look for augmenting paths with $\delta \geq \frac{\Delta}{2}$.

Since no more than $z^* - z'$ units of flow can be added in phase $p + 1$, no more than $2m$ augmentations can be done in phase $p + 1$. This bounds holds for all phases.

Complexity

Complexity of each augmentation.
Since we are no longer looking for maximum capacity augmenting paths but to any augmenting path in $\overline{A}_R(x, \Delta)$, any algorithm to visit the residual graph is enough to detect an augmenting path if one exists.

This can be done with complexity $O(m)$.

Therefore we have:

- upper bound on the number of scaling phases: $1 + \lfloor \log U \rfloor$
- upper bound on the number of augmentations during each scaling phase: $2m$
- complexity of each augmentation: $O(m)$.

Hence the overall complexity of the resulting max-flow algorithm is $O(m^2 \log U)$. This is a polynomial complexity because $m^2 \log U$ is polynomial in the number of bits needed to describe a problem instance.

Shortest augmenting path

Improvement: *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

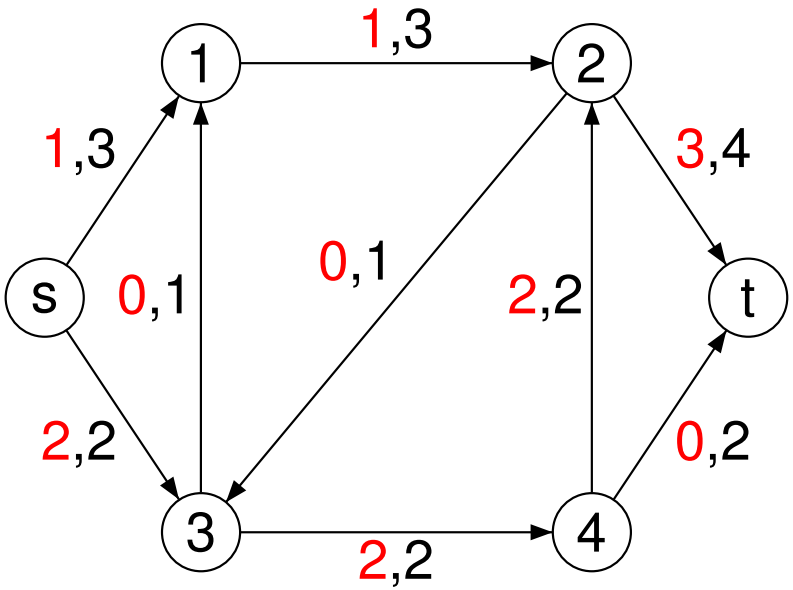


Figure: Flow on the original digraph.

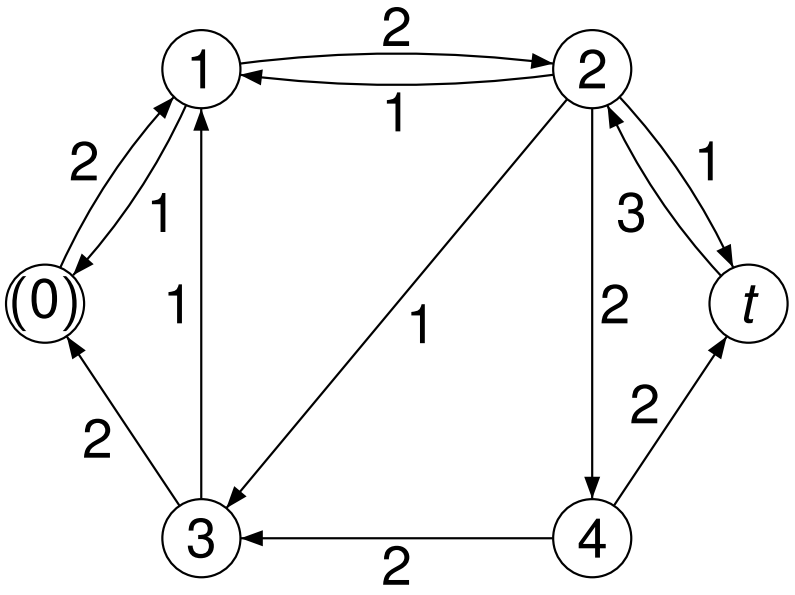


Figure: The residual digraph.

Visualization

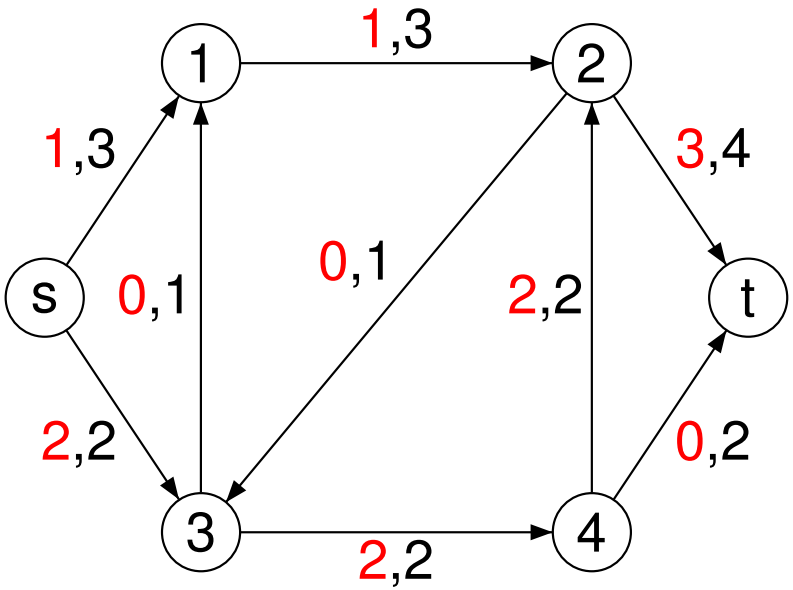


Figure: Flow on the original digraph.

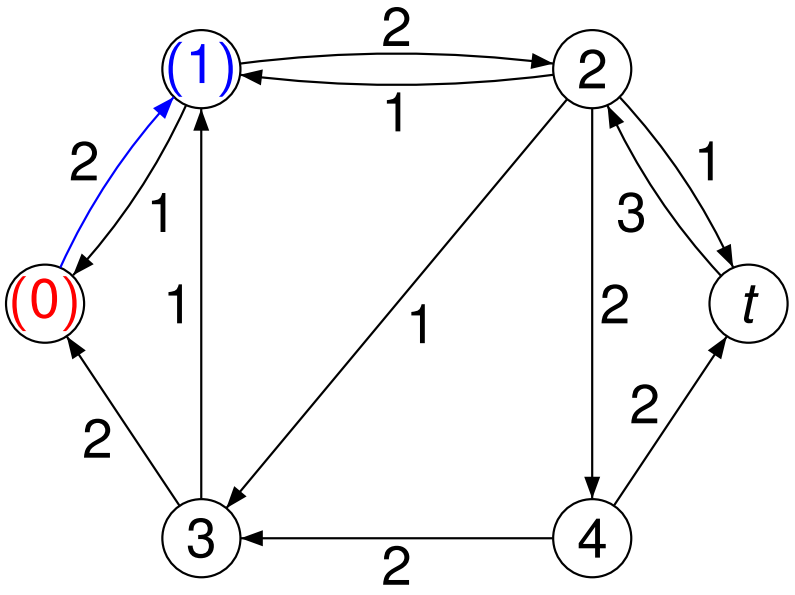


Figure: The residual digraph.

Visualization

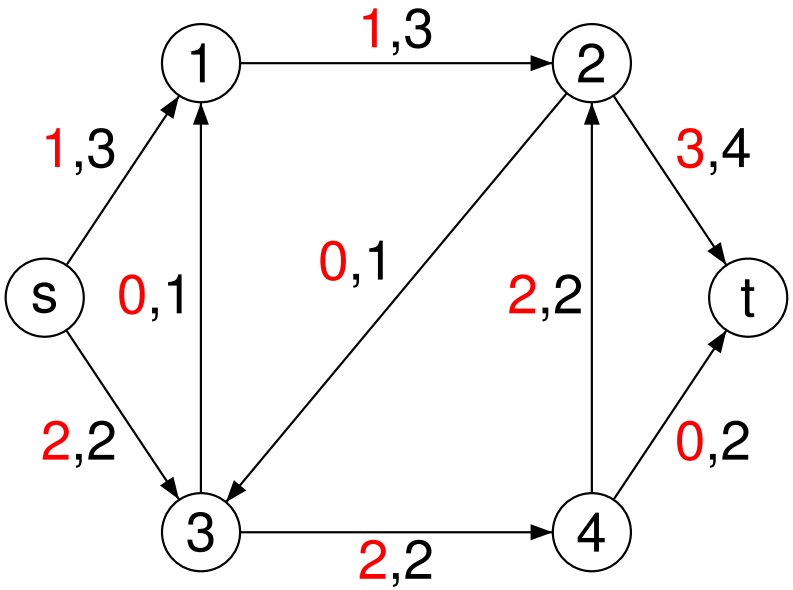


Figure: Flow on the original digraph.

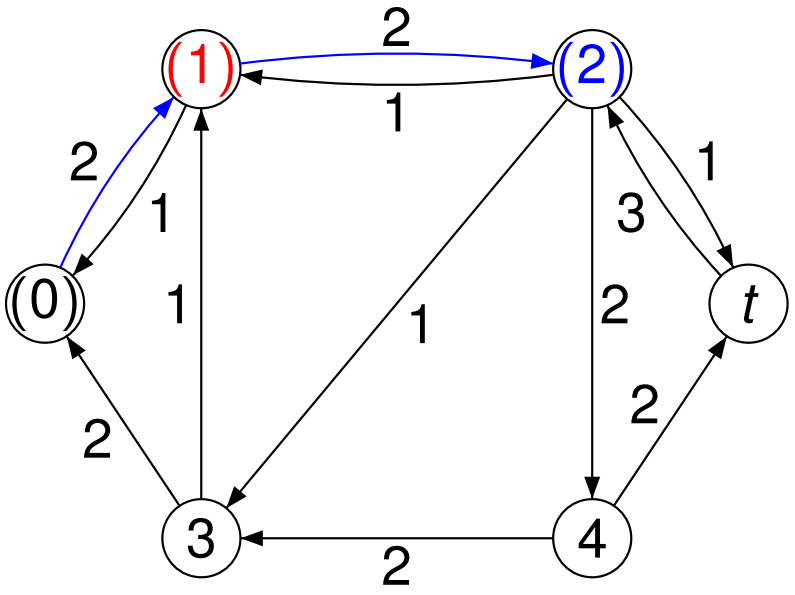


Figure: The residual digraph.

Visualization

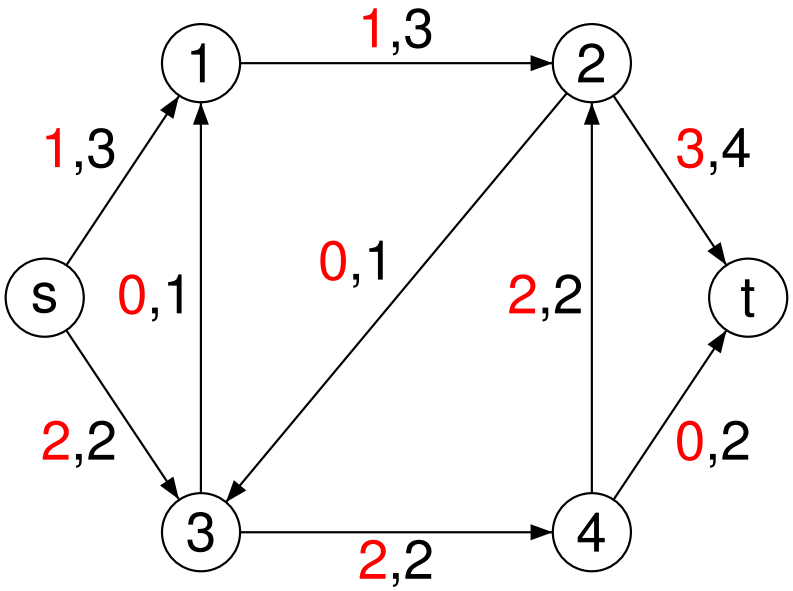


Figure: Flow on the original digraph.

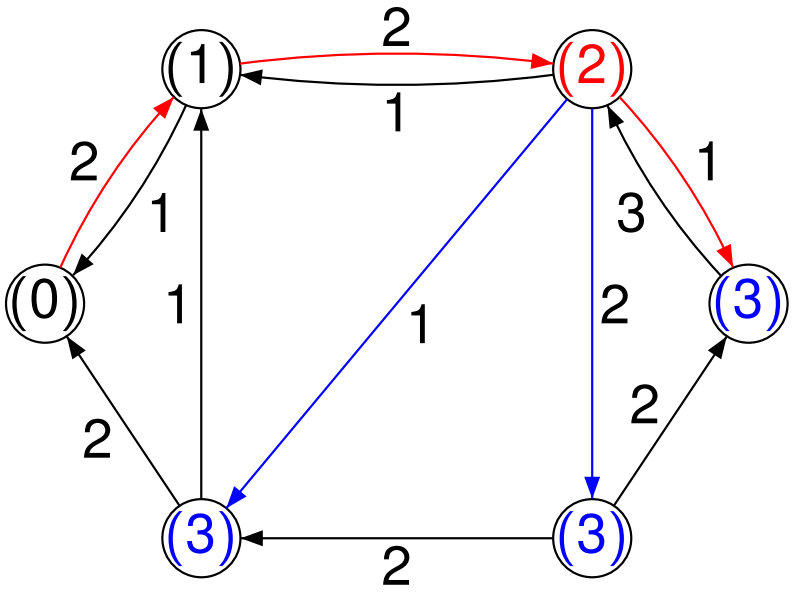


Figure: The residual digraph.

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.

The number of augmentations would be $O(nm)$.

Hence, the resulting complexity would be $O(nm^2)$ (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

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

Procedures

Algorithm 1 Procedure *Advance*(i)

$pred(j) \leftarrow i$
$i \leftarrow j$

Algorithm 2 Procedure *Retreat*(i)

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

Correctness

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

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

Proof (by induction): assume d is valid before an operation and prove that it remains valid after it.

Basis of the induction: the initial labeling computed by *ExactDistance* is valid.

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

Correctness

Procedure *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

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

Data-structure: the outstar A_i of each node i is stored as 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 resumes.

When the end of the list is reached without finding admissible arcs, d_i is recomputed and the search restarts from the head of the list.

Between two relabeling operations, the search for admissible arcs takes $O(|A_i|)$ (each arc in A_i is considered once).

When a relabeling occurs, the time needed to compute d_i' is $O(|A_i|)$ (each arc in A_i 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.

Proof. 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 $O(n^2)$;
- 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 $O(n^2 + n^2 m)$, because

- $O(n^2 m)$ *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 $O(n^2 m)$.

Improved end test

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 algorithm as a subroutine.

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

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.

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

This can be 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.

Dinic algorithm

Dinic algorithm sends flow only along the arcs of the layered network.

When a reverse arc (j, i) is generated by sending flow along (i, j), it is not inserted in the layered network, because $d(j) \neq d(i) + 1$ (actually $d(i) = d(j) + 1$).

Since every (s, t)-path in the layered network is a shortest (s, t)-path, a shortest augmenting path can be found in $O(n)$ time on average (same argument used for the shortest augmenting path algorithm).

Each augmentation saturates at least one arc: there are at most $O(m)$ augmentations.

In $O(nm)$ a blocking flow is found: no further augmentations are possible in the layered network.

Dinic algorithm

When a blocking flow is found, exact distances are recomputed in $O(m)$ and another iterations starts on a new layered network.

At each iteration the distance $d(s)$ increases by at most 1 unit.

When s becomes unreachable from t in the exact distance computation, the algorithm stops.

The number of layered networks is bounded by n.

Therefore the complexity of Dinic algorithm is $O(n^2 m)$.

Special cases of max flow

Unoriented graphs. We can replace each edge $[i, j]$ with a pair of arcs (i, j) and (j, i) with same capacity of the edge.

Arcs in both directions. When $\mathcal{A}$ contains pairs (i, j) and (j, i), the auxiliary graph would contain four arcs between i and j. To avoid this, we replace one of the two original arcs (e.g. (j, i)) by two arcs and an intermediate dummy node (i.e. (j, k) and (k, i)). The two new arcs must have the same capacity of the original arc.

Capacitated nodes. We can split a capacitated node i into two copies, i' and i'' and an arc (i', i'') with the same capacity of node i. All arcs entering i enter i' and all arcs leaving i leave i''.

Given flow. To find a flow equal to a pre-specified amount $\overline{z}$, we can connect s (or t) to a dummy node s' with an arc (s', s) of capacity $\overline{z}$ and then search for a maximum flow from s'.