Maximum flow problem (part II)

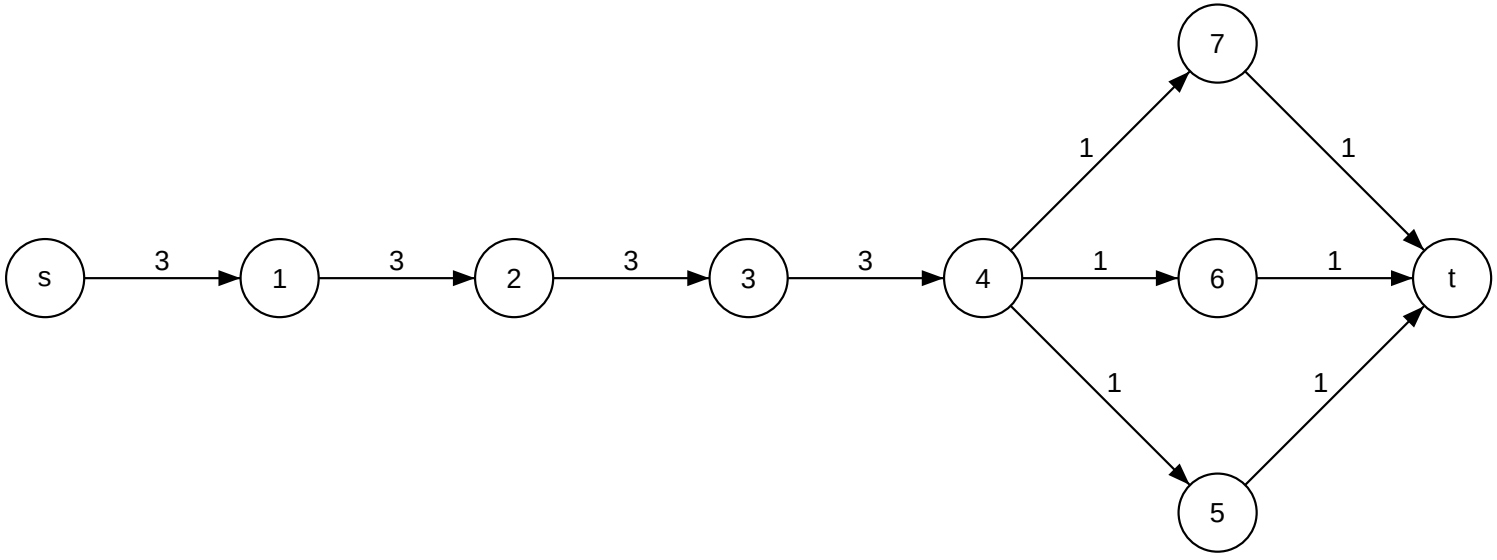
Combinatorial Optimization

Giovanni Righini

Università degli Studi di Milano

A major drawback

Algorithms based on augmenting paths send flow multiple times along non-saturated arcs.



Each augmenting path sends one unit of flow along arcs $\{(s,1),(1,2),(2,3),(3,4)\}$.

Preflow

To overcome this drawback, we consider preflow instead of flow.

A preflow is an assignment of values to the arc flow variables x that can violate the flow conservation constraints, but satisfies them as inequalities:

$$\sum_{i \in \mathcal{N}:(i,j) \in \mathcal{A}} x_{ij} - \sum_{i \in \mathcal{N}:(j,i) \in \mathcal{A}} x_{ji} \geq 0.$$

The preflow entering each node (different from s and t) is not less than the preflow leaving the node. The excess flow e_i at node $i \in N$ is

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

Node s has negative excess; all the others have non-negative excess.

Preflow push

Any node $i \in N \setminus \{s, t\}$ with $e_i > 0$ is an active node.

If there are active nodes, the solution is infeasible.

The preflow push algorithm iteratively pushes excess flow along arcs from active nodes to the sink t, until it achieves feasibility.

Flow is sent only along admissible arcs, defined as in the shortest augmenting path algorithm.

When this is not possible, the distance of the active node is updated to create at least one admissible arc.

Preflow push algorithm

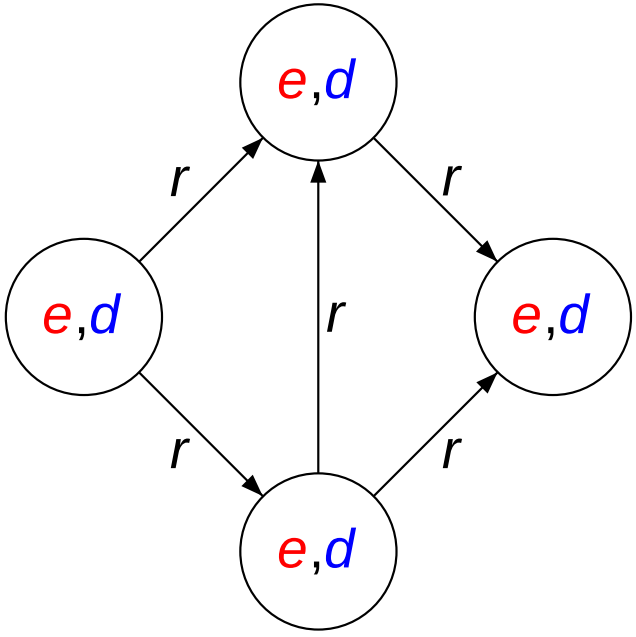
Algorithm 1 Preflow push

$x \leftarrow 0$
ExactDistance
for $(s,j) \in \mathcal{A}$ **do**
  $x_{sj} \leftarrow u_{sj}$
$d_s \leftarrow n$
while *Activenodes* $\neq \emptyset$ **do**
  Select$(i \in$ *Activenodes*$)$
  PushRelabel(i)

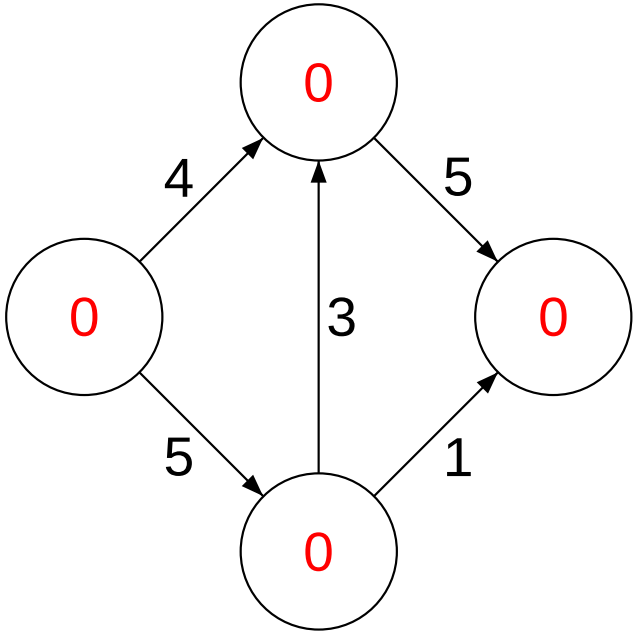
Algorithm 2 *PushRelabel*(i)

if $\exists(i,j)$ admissible **then**
  $\delta \leftarrow \min\{e_i, r_{ij}\}$
  Push(δ, i, j)
else
  $d_i \leftarrow 1 + \min_{(i,j)\in A_R}\{d_j\}$

An example

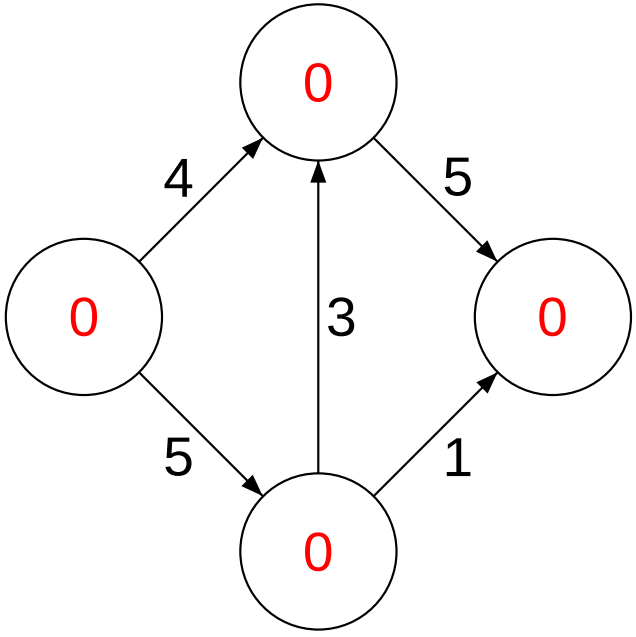


Notation

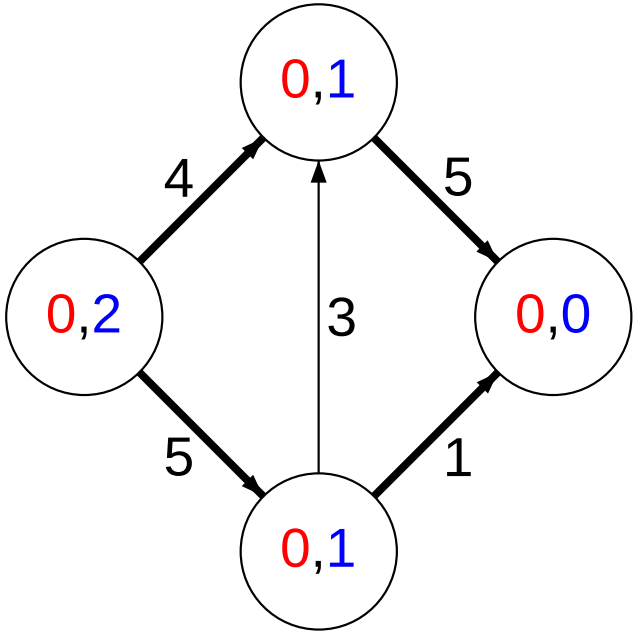


The digraph

An example: initialization

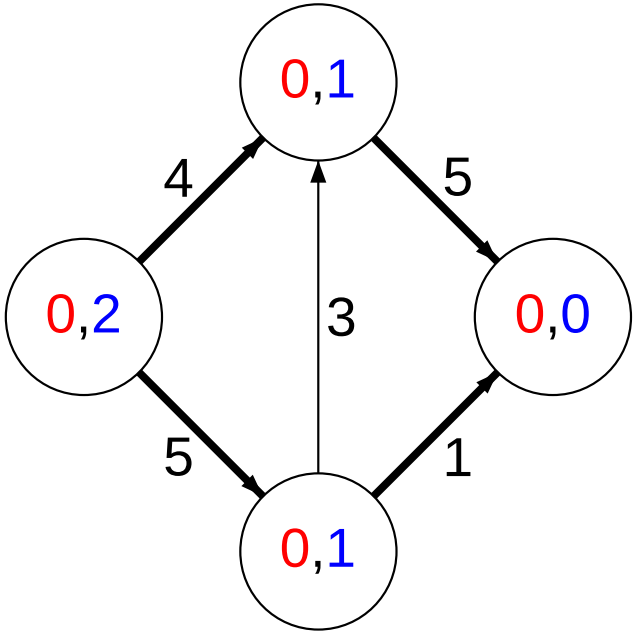


The digraph

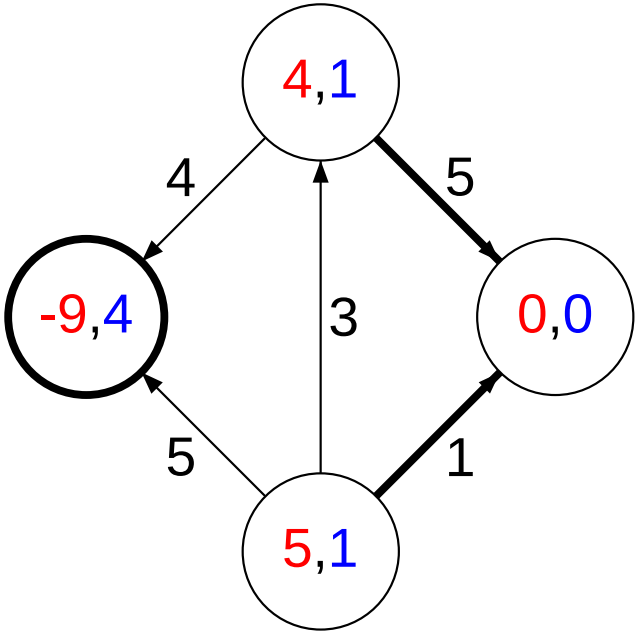


Initialization: compute *d*

An example: iteration 1

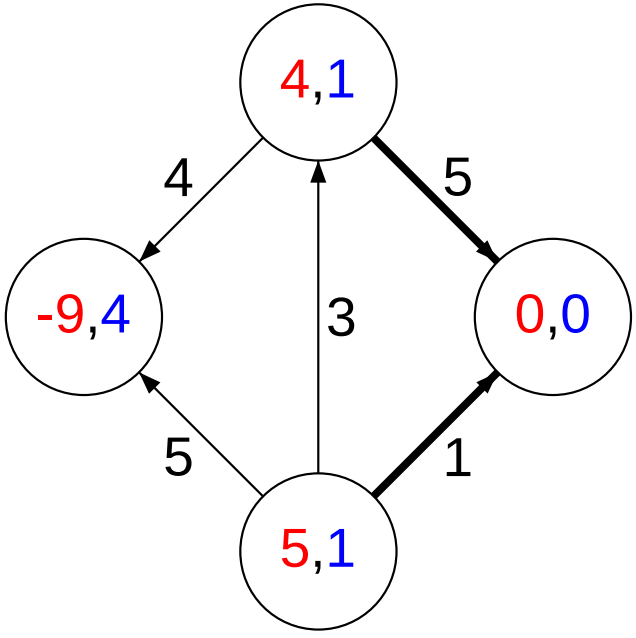


Initialization

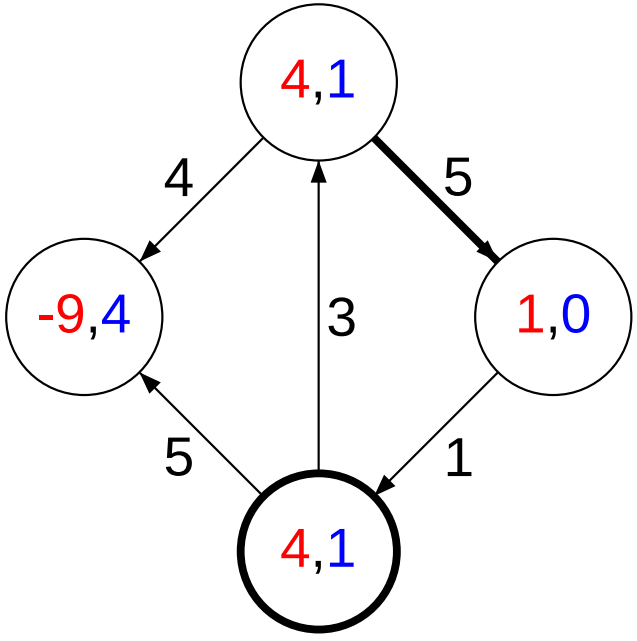


Iteration 1: send preflow from node 1.

An example: iteration 2

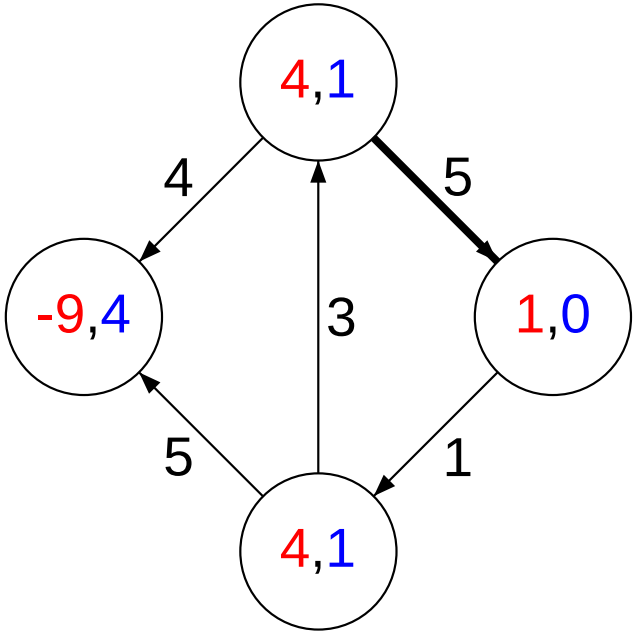


Iteration 1

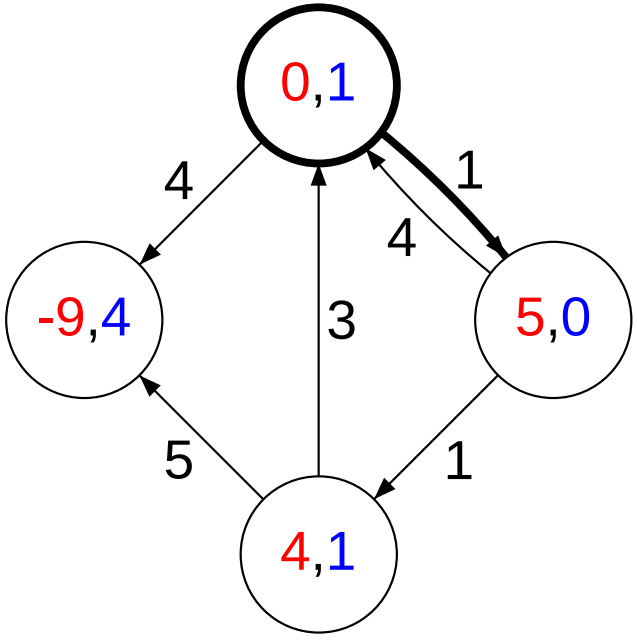


Iteration 2: send preflow from node 2

An example: iteration 3

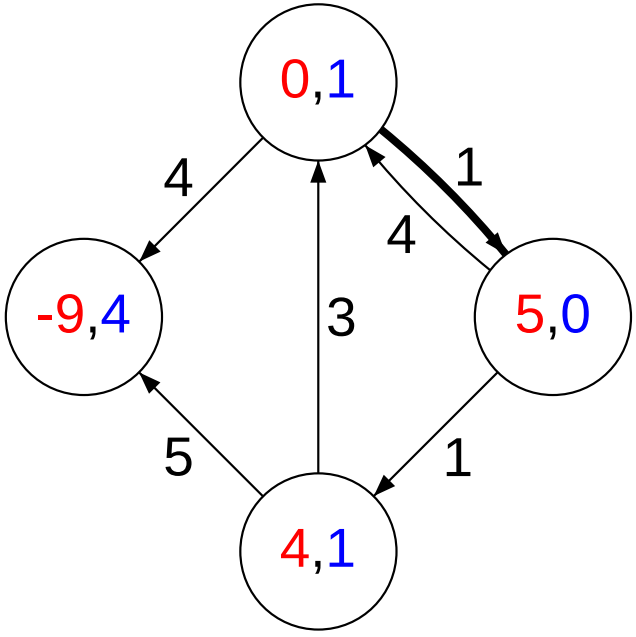


Iteration 2

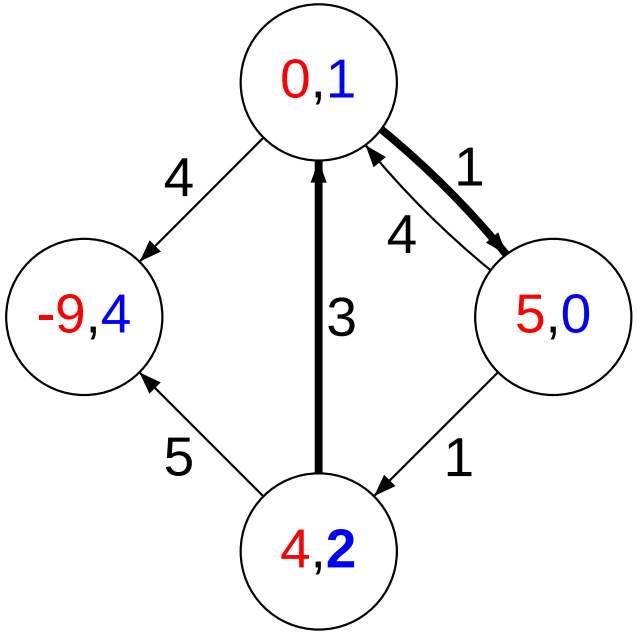


Iteration 3: send preflow from node 3

An example: iteration 4

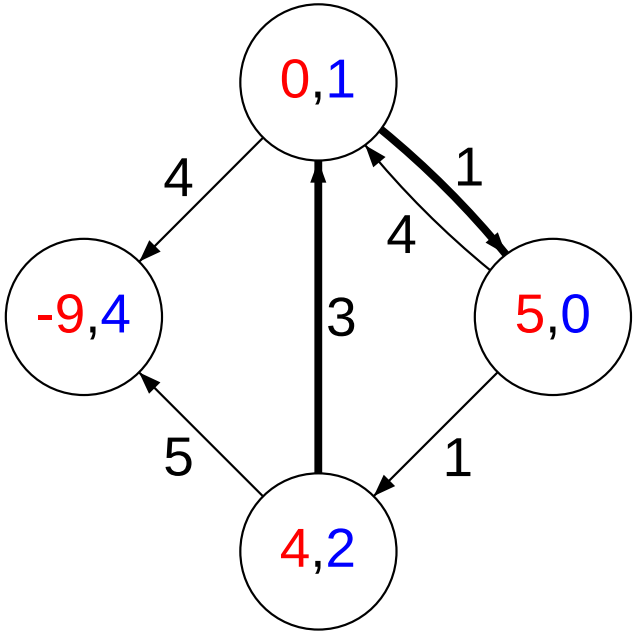


Iteration 3

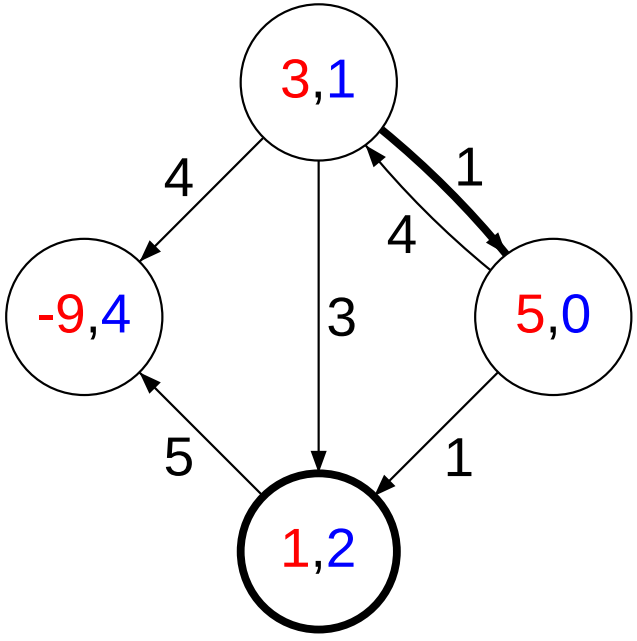


Iteration 4: update d_2

An example: iteration 5

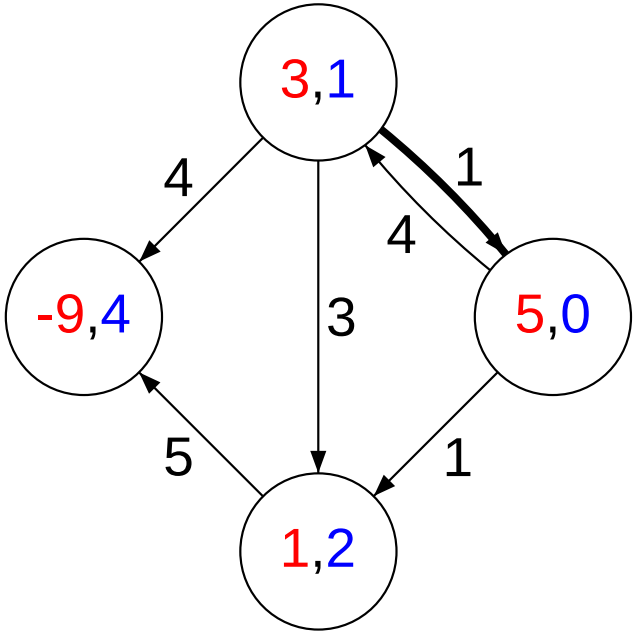


Iteration 4

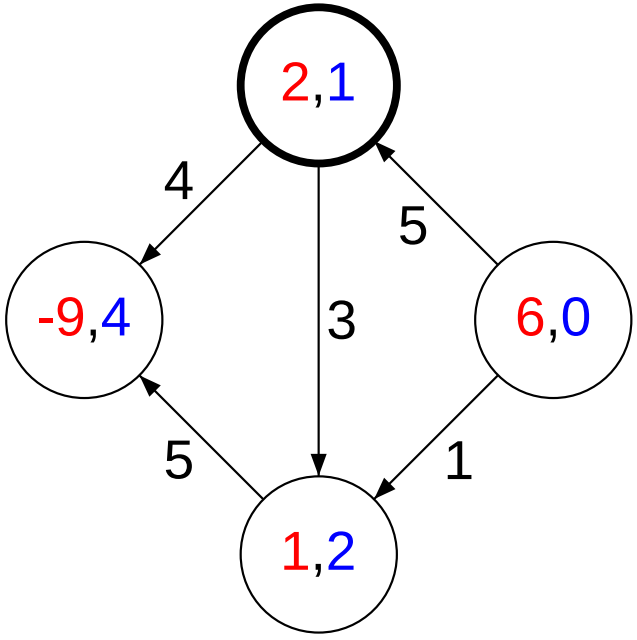


Iteration 5: send preflow from node 2

An example: iteration 6

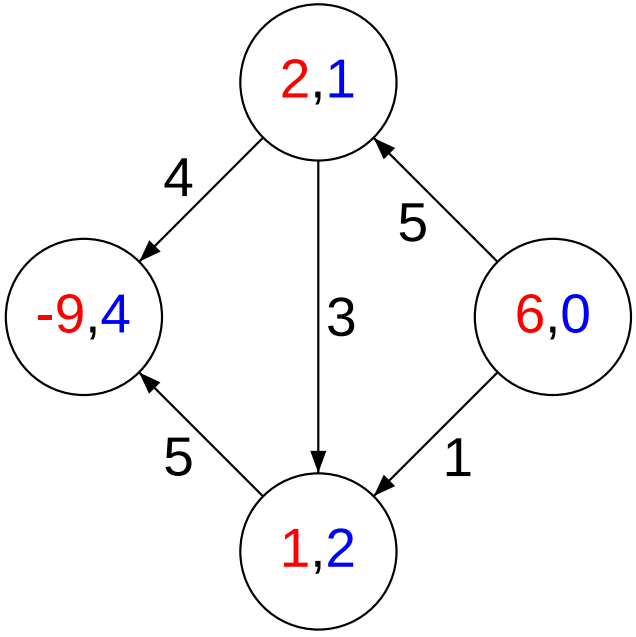


Iteration 5

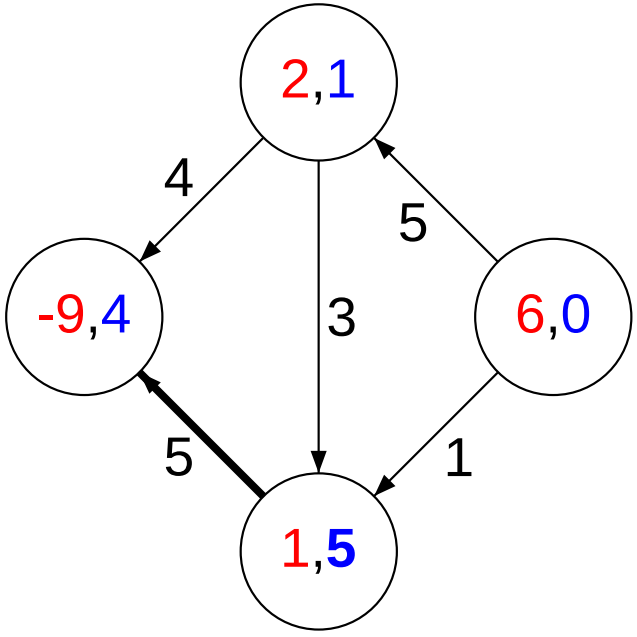


Iteration 6: send preflow from node 3

An example: iteration 7

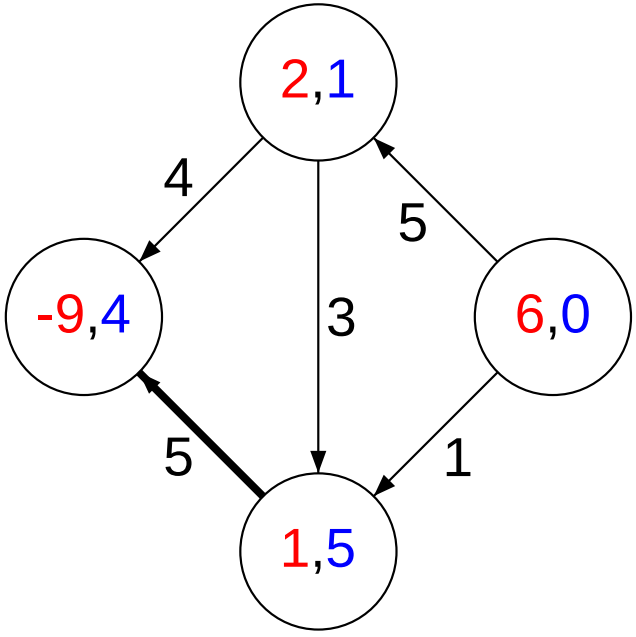


Iteration 6

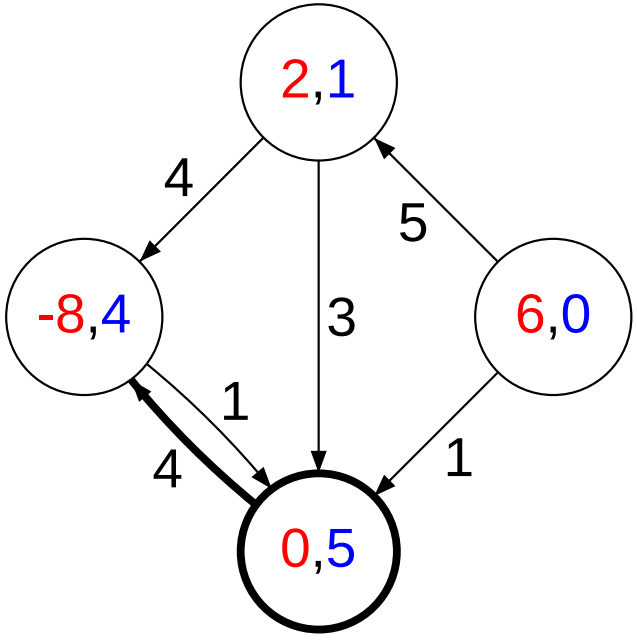


Iteration 7: update d_2

An example: iteration 8

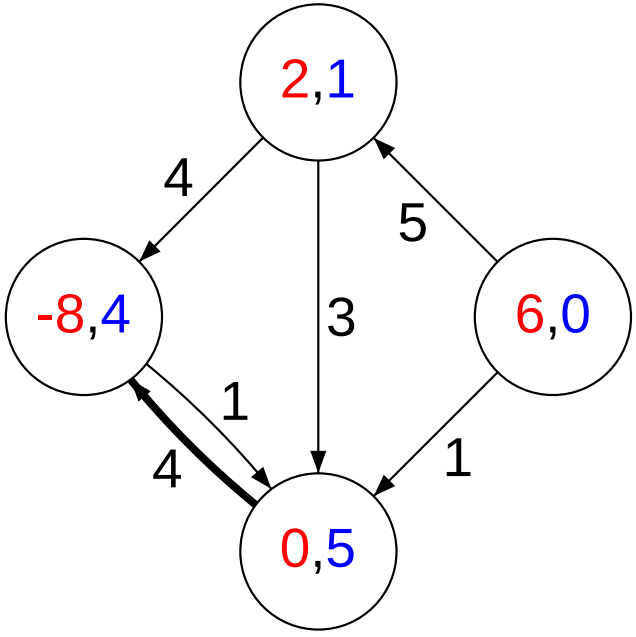


Iteration 6

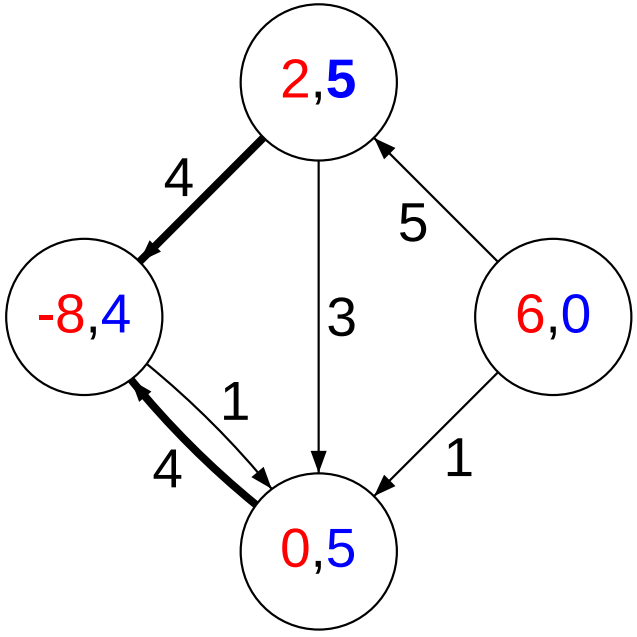


Iteration 8: send preflow from node 2

An example: iteration 9

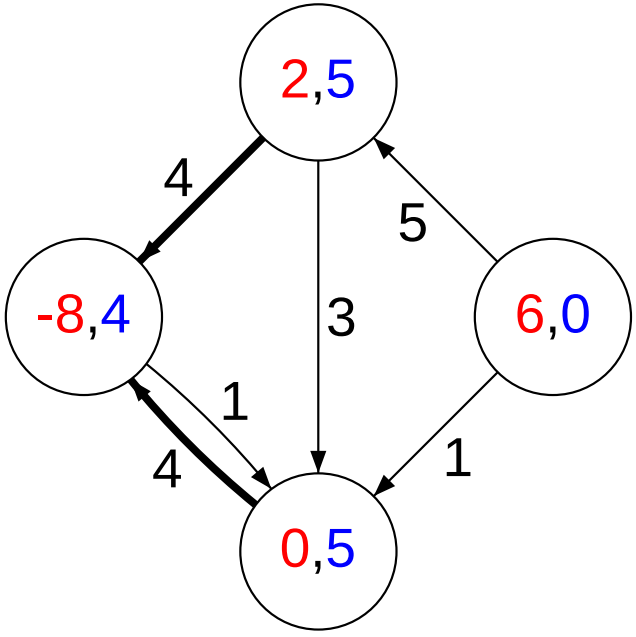


Iteration 8

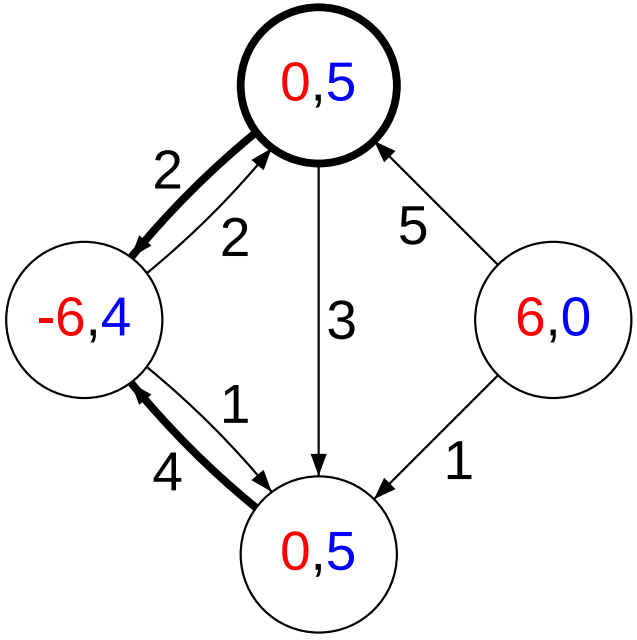


Iteration 9: update d_3

An example: iteration 10



Iteration 9



Iteration 10: send preflow from node 3

Correctness

Property 1. Distance labels are always valid.

Property 2. At any point during the execution of the algorithm, every node i with positive excess $e_i > 0$ is connected to the source by an $i - s$ path in the residual graph.

This guarantees that the *Relabel* operation is always possible.

Complexity

It is also possible to establish a bound on the number of times every node is relabeled.

Property 3. For each node $i \in N$, $d_i < 2n$.

Proof. The last time a node i is relabeled is has a positive excess; hence there exists a path P from i to s in the residual network. Such a path has at most $n - 1$ nodes (including i and s) and $n - 2$ arcs. Since $d_s = n$ and $d_k \leq d_h + 1$ for each (k, h) in P, it follows that $d_i \leq n + (n - 2) < 2n$.

Complexity

Property 4. Each distance label increases at most $2n$ times.

This follows directly from Property 3, because labels increase by at least one unit each time.

Property 5. Consequently, the total number of *Relabel* operations is at most $2n^2$.

We had already proven that the total time required for all *Relabel* operations (search for an admissible arc, computation of the new value for the distance label) is $O(km)$ if each node is relabeled at most k times. Here $k = 2n$.

Therefore the overall time required for relabeling is $O(nm)$.

Complexity

The amount of flow that is pushed at each iteration along an arc (i, j) is $\delta = \min\{e_i, r_{ij}\}$.

Definition.

- A push is saturating, if $\delta = r_{ij}$.
- A push is non-saturating, if $\delta = e_i < r_{ij}$.

Property 6. The algorithm performs at most $2nm$ saturating pushes.

The proof relies upon the lemma stating that if each node is relabeled at most k times then each arc is saturated no more than km times. In this case $k = 2n$ and Property 6 follows.

Complexity

To establish a bound on the number of non-saturating pushes, we use a potential function $\Phi = \sum_{i \in I} d_i$, where I is the set of active nodes.

The initial value of Φ is at most $2n^2$, because

- the number of active nodes is less than n;
- $d_i < 2n$ for every active node.

The final value of Φ is 0, because no active nodes are left.

We now prove bounds on the maximum possible increase and the minimum possible decrease of Φ in case of saturating and non-saturating pushes.

Complexity

Case I: Relabel. The distance of an active node i is increased by at least one unit and this increases Φ by the same amount. The total increase of d_i is bounded by $2n$. Hence, the total increase of Φ due to all *Relabel* operations is bounded by $2n^2$.

Case II: Saturating push. A saturating push along an arc (i,j) can create a new active node j. In this case Φ is increased by d_j which is at most $2n$. Since the number of saturating pushes is at most nm, then the total increase of Φ due to saturating pushes is at most $2n^2m$.

Case III: Non-saturating push. A non-saturating push along an arc (i,j) makes i inactive and therefore it decreases Φ by d_i. If j was inactive, then Φ is increased by $d_j = d_i - 1$. Therefore Φ is decreased by at least one unit.

Complexity

Summarizing:

- The initial value of Φ is at most $2n^2$.
- The final value of Φ is 0.
- The value of Φ is always non-negative.
- The maximum possible increase of Φ is $2n^2 + 2n^2m$.
- The decrease of Φ is at least 1 for each non-saturating push.

Property 7. The generic preflow push algorithm performs $O(n^2m)$ non-saturating pushes.

Complexity

The complexity of inserting and deleting elements from a data-structure containing the active nodes can be done in $O(1)$ (e.g. using a doubly linked list).

Summarizing:

- The overall time required for relabeling is $O(nm)$.
- The overall time required for $O(nm)$ saturating pushes is $O(nm)$.
- The overall time required for $O(n^2m)$ non-saturating pushes is $O(n^2m)$.

Hence the overall complexity of the generic preflow push algorithm is $O(n^2m)$.

Specific implementations

The bottleneck is the number of non-saturating pushes and this depends on how the active node is selected.

We consider three specific implementations:

- FIFO preflow push algorithm: $O(n^3)$ (tight).
- Highest label preflow push algorithm: $O(n^2\sqrt{m})$ (tight).
- Excess scaling algorithm: $O(nm + n^2 \log U)$.

In all cases we assume that the active node remains active after every saturating push.

FIFO preflow push algorithm

Active nodes are kept in a queue.

Starting from the queue obtained after initialization (preflow sent from *s*), a *stage* is the set of iterations when the nodes in the queue are examined once.

The state of the queue at the end of each stage determines the set of nodes to be examined in the next stage.

To establish the complexity we determine a bound on the number of stages.

FIFO preflow push algorithm

Consider a potential function $\psi = \max_{i \in I}\{d_i\}$, where I is the set of active nodes.

We consider stages with at least one *Relabel* operation and stages with no *Relabel* operations and we bound them separately.

Case I: stages that increase the potential.
The potential can grow only in stages including at least one *Relabel* operation. We have already proven that at most $2n^2$ *Relabel* operations can occur. So, at most $2n^2$ stages of this type can occur.

Since all distance labels are non-decreasing and they always remain bounded by $2n$, then the total increase of ψ is at most $2n$ for each node, i.e. $2n^2$.

FIFO preflow push algorithm

Case II: stages that decrease the potential.
If a stage does not include any *Relabel* operation, then all the excess is pushed from active nodes to other nodes with smaller distance values. Hence ψ decreases by at least 1 unit during the stage.

The initial value of ψ is not larger than n.
The final value of ψ is 0.
The maximum increase of ψ is $2n^2$.
Therefore the number of stages without *Relabel* operations is at most $2n^2 + n$.

Hence, the total number of stages is $2n^2 + 2n^2 + n$, i.e. $O(n^2)$.

FIFO preflow push algorithm

The number of stages is $O(n^2)$.

Each stage examines every node at most once.

Each node examination performs at most one non-saturating push.

Hence the number of non-saturating pushes is $O(n^3)$.

Since the number of non-saturating pushes is the bottleneck in the generic preflow push algorithm, the time complexity of the FIFO implementation is $O(n^3)$.

Highest label preflow push algorithm

Another specific implementation of the preflow push algorithm selects an active node with the highest distance label. The number of non-saturating pushes in this case can be easily bounded by $O(n^3)$.

Proof. Let d^* be the maximum distance among all active nodes. The algorithm examines nodes with label d^*, pushing flow from them to nodes with distance $d^* - 1$. Then the same is repeated from nodes with label $d^* - 1$ to nodes with label $d^* - 2$ and so on, until either a node is relabeled or no active nodes are left.

No more than n nodes can be examined between two *Relabel* operations.
No more than $2n^2$ *Relabel* operations can occur.
No more than one non-saturating push can occur for each node examination.
Therefore the number of non-saturating pushes is $O(n^3)$.

Highest label preflow push algorithm: complexity

With a more detailed analysis (Cheriyan and Maheshwari, 1989), it is possible to establish a better worst-case bound on the number of non-saturating pushes. For this analysis we need some observations and definitions.

At any point during the execution of the algorithm each node $i \in \mathcal{N}$ may have a successor $j \in \mathcal{N}$. This is the effect of $Relabel(i)$ that sets $d(i)$ to $d(j) + 1$ and makes arc (i, j) admissible. We say that $Succ(i)$ is set to j.

Since each node has no more than one successor, the set of arcs $(i, Succ(i))$ forms a forest $\mathcal{F}$ in the residual digraph $\mathcal{A}_R$. One of its components is rooted in t.

Highest label preflow push algorithm: complexity

We define $D(i)$ as the set of descendants of each node i. It includes i itself and all nodes from which i can be reached following the arcs of $\mathcal{F}$. Therefore $d(j) > d(i) \;\; \forall j \in D(i), j \neq i$.

A maximal active node is an active node with no active descendants.

Let H be the set of maximal active nodes.

Highest label preflow push algorithm: complexity

Let K be a parameter.

We define a potential function

$$\Phi = \sum_{i \in H} \phi(i)$$

where

$$\phi(i) = \max\{0, K + 1 - |D(i)|\}.$$

The following properties hold:

- $\phi(i) \leq K \quad \forall i \in \mathcal{N}$, because $|D(i)| \geq 1$ for all nodes.
- Φ changes when:
  - H changes or
  - $|D(i)|$ changes for some node $i \in H$.

Now we examine the effect of different operations on Φ.

Highest label preflow push algorithm: complexity

Non-saturating pushes.

Push operations can only occur from nodes in H because of the "highest label" selection rule.
Push operations can only occur along admissible arcs from i to $Succ(i)$. Consider a non-saturating push from $i \in H$ along arc (i, j).

- $\mathcal{F}$ does not change;
- node i becomes inactive and leaves H;
- node j can become a maximal active node, entering H;
- j is the successor of i and hence $|D(j)| > |D(i)|$ and hence $\phi(j) \leq \phi(i)$.

Therefore, the potential Φ:

- decreases by at least one unit, if $|D(i)| \leq K$ (because $\phi(i) \geq 1$ and $\phi(j) \leq \phi(i)$)
- remains unchanged, otherwise (because $\phi(i) = 0$ and hence $\phi(j) = 0$ too).

Highest label preflow push algorithm: complexity

Saturating pushes.

Consider a saturating push from $i \in H$ along an admissible arc (i,j):

- arcs (i,j) leaves $\mathcal{F}$, becoming inadmissible;
- node i remains active and then it remains in H;
- node j can become a maximal active node, entering H;

This can increase Φ by at most K units (because $\phi(j) \leq K$).

Highest label preflow push algorithm: complexity

Relabel.

Consider an operation *Relabel*(i) on a node $i \in H$:

- there is no arc $(i,j) \in \mathcal{F}$, then i is one of the roots of $\mathcal{F}$;
- since $i \in H$, no descendants of i are active;
- all arcs $(k,i) \in \mathcal{F}$ become inadmissible when i is relabeled;
  - no new maximal active nodes are created;
  - $|D(i)|$ is decreased to 1; so, $\phi(i)$ can increase by at most K;
- new arcs entering $\mathcal{F}$ do not create new nodes in H;
  - some node can leave H;
  - $|D(k)|$ can increase for some k;

  In both cases Φ does not increase.

Highest label preflow push algorithm: complexity

Summing up, indicating the variation of Φ by $\Delta\Phi$:

- Non-saturating pushes: $\Delta\Phi \leq 0$ in general and $\Delta\Phi \leq -1$ if $|D(i)| \leq K$;
- Saturating pushes: $\Delta\Phi \leq K$;
- Relabelling: $\Delta\Phi \leq K$;

We define $\overline{d} = \max_{i \in I}\{d(i)\}$ the maximum label of an active node.

We define a stage as the sequence of pushes during which $\overline{d}$ remains unchanged.

Since there are $O(n^2)$ *Relabel* operations, there are $O(n^2)$ stages.

A stage is short if it contains no more than $2n/K$ non-saturating pushes, long otherwise.

Highest label preflow push algorithm: complexity

There are $O(n^2)$ stages.

Each short stage has no more than $2n/K$ non-saturating pushes.

Hence, the number of non-saturating pushes in short stages is $O(n^3/K)$.

For the definition of the set $D(i)$, the sets of descendants of different maximal active nodes are disjoint.

Therefore, there are at most n/K maximal active nodes with K descendants or more.

Then at least n/K non-saturating pushes occur from maximal active nodes with less than K descendants.

These pushes decrease Φ by at least one unit.

Highest label preflow push algorithm: complexity

Summing up:

- there are $O(nm)$ pushes;
- saturating pushes produce $\Delta\Phi \leq K$;
- there are $O(n^2)$ relabelings;
- relabelings produce $\Delta\Phi \leq K$.

Then, the total increase of Φ due to saturating pushes and relabelings is $O(nmK)$.

Non-saturating pushes do not increase Φ.

Non-saturating pushes in long stages decrease Φ by at least one unit.

Therefore the number of non-saturating pushes in long stages is $O(nmK)$.

Highest label preflow push algorithm: complexity

The number of non-saturating pushes in short stages is $O(n^3/K)$.
The number of non-saturating pushes in long stages is $O(nmK)$.

Selecting $K = n/\sqrt{m}$ we can bound the total number of non-saturating pushes with $O(n^2\sqrt{m})$, which is the computational complexity of the whole algorithm, since non-saturating pushes are the bottleneck.

Highest label preflow push algorithm: data-structure

To select a node with highest distance label, we keep an array L of linked lists, such that each list $L[k]$ contains the active nodes with distance label k.

A variable λ contains an upper bound on the highest value of k for which $L[k] \neq \emptyset$.

At each iteration the search starts from λ and goes down until a non-empty list is found.
Then any node is taken from it.

When a distance label increases owing to a *Relabel* operation, λ is set to the new value of the distance label.

The total increase is bounded by $2n^2$; so the total decrease is bounded by $n + 2n^2$, i.e $O(n^2)$.

Excess scaling preflow push algorithm

A different way of selecting active nodes in a preflow push algorithm is excess scaling.

The idea is to select an active node with a relatively high value of excess, so that the maximum excess is monotone non-increasing.

The idea is analogous to that of capacity scaling in maxflow algorithms based on augmenting paths.

Excess scaling preflow push algorithm

Let Δ be an upper bound to $e_{max} = \max_{i \in I}\{e_i\}$.

For each node $i \in I$ we say that if $e_i \geq \Delta/2$, then it is a large excess; otherwise it is a small excess.

The excess scaling algorithm

- pushes flow from nodes with large excess.
- does never raise the excess of any node above Δ.

Excess scaling preflow push algorithm

The selection rule is: select an active node with a large excess and a minimum distance.

The amount of flow to push along arc (i, j) is set as follows:

$$\delta = \min\{e_i, r_{ij}, \Delta - e_j\}$$

The execution of the algorithm is divided into phases and Δ is halved from each phase to the next.

Initially $\Delta = 2^{\lceil \log_2 U \rceil}$; this implies $U \leq \Delta \leq 2U$.

When $e_{max} \leq \Delta/2$ a new phase is started.

Excess scaling preflow push algorithm: properties

Property 1. Each non-saturating push sends at least $\Delta/2$ units of flow.

For any non-saturating push along arc (i,j), $d_i = 1 + d_j$.

Since i has a minimum distance among nodes with large excess, $e_i \geq \Delta/2$ and $e_j < \Delta/2$.

Since the push is non-saturating, it sends $\delta = \min\{e_i, \Delta - e_j\}$ units of flow.

Therefore $\delta \geq \Delta/2$.

Property 2. No excess ever exceeds Δ.

The new excess of node j after the push is

$$e_j + \min\{e_i, \Delta - e_j\} \leq e_j + (\Delta - e_j) = \Delta.$$

Excess scaling preflow push algorithm: complexity

To establish the complexity of the algorithm we use a potential function

$$\Theta = \sum_{i \in N} \frac{e_i d_i}{\Delta}.$$

The value of Θ at the beginning of each Δ-scaling phase is bounded by $2n^2$, because $e_i \leq \Delta$ and $d_i \leq 2n$ for each node i.

We study separately what happens in case of *Push* and *Relabel*.

Excess scaling preflow push algorithm: complexity

Case I (Relabel).

When node i is relabeled, its distance label d_i is increased by $\epsilon \geq 1$.

Therefore Θ is increased by $e_i \epsilon / \Delta$ which is bounded by ϵ because $e_i \leq \Delta$.

Since the overall increase of d_i is bounded by $2n$ for each node i, the total increase of Θ is bounded by $2n^2$.

Hence it is also bounded by $2n^2$ in each single Δ-scaling phase.

Excess scaling preflow push algorithm: complexity

Case II (Push).

A non-saturating push along arc (i, j) sends at least $\Delta/2$ units of flow from i to j, with $d_i = d_j + 1$.

Therefore Θ decreases by at least $1/2$ unit.

Since the initial value of Θ at the beginning of a phase is bounded by $2n^2$ and its increase during a phase is also bounded by $2n^2$, the number of non-saturating pushes during a phase is bounded by $8n^2$.

Since the number of scaling phases is $1 + \lceil \log U \rceil$, the algorithm performs $O(n^2 \log_2 U)$ non-saturating pushes.

All the other operations require $O(nm)$ time.

Therefore the complexity of the algorithm is $O(nm + n^2 \log_2 U)$.

Excess scaling preflow push algorithm: data-structure

The search for the active node we use:

- an array L of linked lists s.t. $L[k] = \{i \in N : e_i > \Delta/2 \wedge d_i = k\}$;
- a variable λ, indicating a lower bound for the smallest index k for which $L[k]$ is not empty.

At every node selection, we start from $L[\lambda]$ and we scan L until a non-empty list is found.
The maximum decrease of λ is bounded by the number of pushes, because each push makes λ either to remain unchanged or to decrease by 1 unit.
The maximum increase of λ is n for each phase and there are $O(\log U)$ phases.
Hence the effort needed to scan L is bounded by the number of pushes plus $O(n \log U)$.
Hence it is not a bottleneck.