Advanced algorithms for the $s - t$ shortest path problem: node and arc reach

Giovanni Righini

University of Milan

UNIVERSITÀ DEGLI STUDI DI MILANO

References

Main references:

- Reach: R. Gutman, *Reach-based routing: a new approach to shortest path algorithms optimized for road networks*, Workshop on Algorithm Engineering and Experiments (ALENEX), 2004.
- Reach: A.V. Goldberg, H. Kaplan, R.F. Werneck, *Reach for A* : efficient point-to-point shortest path algorithms*, Workshop on Algorithm Engineering and Experiments (ALENEX), 2006.

Reach

Given a shortest path $P^*(u, v)$ from $u \in \mathcal{N}$ to $v \in \mathcal{N}$ and given a node $i \in P^*(u, v)$,

$$r(i, P^*(u, v)) = \min\{dist(u, i), dist(i, v)\},$$

where *dist* indicates the shortest path distance.

On the whole graph

$$r(i) = \max_{u \in \mathcal{N}, v \in N : u \neq v} \{r(i, P^*(u, v))\}.$$

Intuitively, the reach r of a node is a measure of how likely the node is to belong to long shortest paths.

The use of reach

Let $\overline{r}(i)$ be an upper bound: $\overline{r}(i) \geq r(i)$.

Let $\underline{d}(i,j)$ be a lower bound: $\underline{d}(i,j) \leq dist(i,j)$.

By definition

$$i \in P^*(s,t) \Longrightarrow r(i) \geq r(i, P^*(s,t)) = \min\{dist(s,i), dist(i,t)\}.$$

Therefore

$$\overline{r}(i) < \min\{\underline{d}(s,i), \underline{d}(i,t)\} \Longrightarrow i \notin P^*(s,t).$$

This allows to neglect many nodes (with small reach value) while running Dijkstra algorithm or A^*.

The use of reach

When we consider j as a successor of i in a labeling algorithm (Dijkstra, A^*), we already know $dist(s, i)$.

The following test is done before possibly updating the label of node j (*early pruning*):

$$\overline{r}(j) < \min\{dist(s, i) + c(i, j), \underline{d}(j, t)\} \text{ implies } (i, j) \notin P^*(s, t).$$

A lower bound $\underline{d}(j, t)$ can be provided

- by the Euclidean distance between j and t, if the nodes are embedded in a plane;
- by the largest permanent label in the reverse direction, if bi-directional search is used.

Bi-directional bounding

In bi-directional search, let γ^{bw} the minimum cost of non-permanent backward labels (labels in O^{bw}).

Consider the iteration in which node $i \in O^{fw}$ is selected for being permanently labelled. If

$$\overline{r}(i) < \min\{dist(s, i), \gamma^{bw}\},$$

then we can prune the search at i (*bi-directional bound*).

Self-bounding

Alternatively (*self-bounding*) we can prune node i checking whether

$$\overline{r}(i) < dist(s, i)$$

and we stop the search in a direction when

- O in that direction is empty,
- or the minimum distance label in O is at least half of μ,

where μ is an upper bound (best incumbent $s - t$ path).

It is advisable to scan the minimum label among the forward and the backward candidates.

Each node i can be inserted in E^{fw} (E^{bw}) only if $dist(s, i) \leq (\geq) dist(i, t)$.

Arc sorting

Arc sorting: sort the outstars (in-stars) by non-increasing value of (estimated) *reach* of the head (tail) node.

If $\overline{r}(j) < \min\{dist(s, i), \gamma\}$, all the arcs after (i, j) in the out-star of i can be safely skipped.

Hence, arc sorting may allow to neglect some arcs.

Computing *reach* exactly

To compute the *reach* values exactly:

- Set all reaches to ∞.
- Compute all-pairs shortest paths.
- For each $s - t$ shortest path:
  - Compute the reach of all nodes along the path.
  - Possibly update the reach of each node with the new value, if it is larger than the incumbent one.

Complexity: $O(nm)$, impractical for large graphs (even if sparse).

Computing *reach* approximately

Three main ideas are combined:

- partial trees
- iterative node deletion
- shortcuts

Preprocessing works in two phases:

- Main phase:
  - shortcut arcs are added;
  - partial trees are grown and low reach nodes are deleted;
- Refinement phase: upper bounds on reaches are re-evaluated and possibly strengthened.

Main phase

Main phase:

- Add shortcuts
- For each iteration k
  - Select a threshold value ϵ_k
  - Grow partial trees depending on ϵ_k
  - Eliminate nodes with reach less than ϵ_k
  - Add shortcuts

The threshold values are computed as $\epsilon_k = \alpha\epsilon_{k-1}$ for some $\alpha > 1$.

Canonical paths

Gutman (2004) observed that if more than one shortest path exists from s to t, only one is included in the partial trees. Therefore all nodes along alternative shortest paths may not appear in the partial tree. Therefore they can be misclassified as “low reach nodes” even if they are “high reach nodes” and they can receive an incorrect upper bound $\overline{r}(i)$.

However, this incorrect upper bounding does not prevent a shortest path algorithm like Dijkstra or A^* to find *at least one* shortest $s - t$ path.

Canonical paths

Goldberg et al. (2006) introduced the notion of *canonical path*, i.e. a shortest path with the additional property of being unique for each $s - t$ pair.
A small random perturbation is computed for each arc cost.
The perturbation of a path cost is the sum of the perturbations of its arcs.
When two or more shortest paths exist between s and t, the canonical one is the path with minimum perturbed cost.

Partial trees

For each node i compute a partial shortest path arborescence $T^{\epsilon}(i)$ rooted at i (with Dijkstra algorithm).
At a generic iteration the arborescence T of the labelled nodes includes an arborescence $\overline{T}$ of nodes with permanent label.

Stop criterion: for all leaves j of $\overline{T}$

- either j is a leaf of T,
- or $dist(i', j) \geq 2\epsilon$,

where i' is the node next to i in the (i, j) shortest path.

Let $T^{\epsilon}(i)$ be the partial tree $\overline{T}$ when the algorithm stops.
Nodes with reach larger than ϵ in $T^{\epsilon}(i)$ are marked as "high reach nodes".
Repeating this procedure for all roots $i \in \mathcal{N}$ allows to partition nodes with reach larger than ϵ from nodes with reach smaller than ϵ.

Proof

Thesis 1. Nodes with reach less than ϵ cannot be marked from any root i.
Proof. Their reach in $T^{\epsilon}(i)$ cannot be larger than their actual reach in the digraph.

Thesis 2. All nodes k with $r(k) \geq \epsilon$ are guaranteed to be identified as "high reach nodes" in at least one partial arborescence $T^{\epsilon}(i)$ for some $i \in \mathcal{N}$.
Proof. If $r(k) \geq \epsilon$, then $\exists$ a path P in which k has reach at least ϵ.
Then, $\exists$ a minimal canonical path P' in P, in which k has reach at least ϵ.
Let x and y be the first and the last node of P'.

Proof

Consider $T^{\epsilon}(x)$, which contains $\overline{T}^{\epsilon}(x)$.
Owing to the uniqueness of (perturbed) shortest paths, two cases can occur:

- Case 1: P' is completely contained in $\overline{T}^{\epsilon}(x)$;
- Case 2: P' is partially contained in $\overline{T}^{\epsilon}(x)$.

Case 1.
In this case k is identified as a “high reach node” in $\overline{T}^{\epsilon}(x)$.

Proof

Case 2.
$\overline{T}^{\epsilon}(x)$ contains a subpath of P', starting at x and ending at a leaf z.

By definition of reach, $r(k) \geq \epsilon \Longrightarrow dist(x,k) \geq \epsilon$.

Let x' be the node next to x along P'.

Since P' is minimal, $dist(x',k) < \epsilon$.

Node z cannot be a leaf of $T^{\epsilon}(x)$, because

- it belongs to $\overline{T}^{\epsilon}(x)$ (it has a permanent label) and
- it has got at least one successor (the next node along P').

Hence z being a leaf of $\overline{T}^{\epsilon}(x)$ implies $dist(x',z) \geq 2\epsilon$.

$$dist(k,z) = dist(x',z) - dist(x',k) > 2\epsilon - \epsilon = \epsilon.$$

Therefore $\min\{dist(x,k), dist(k,z)\} \geq \epsilon$ and k is marked as a "high reach node" in $\overline{T}^{\epsilon}(x)$.

Long arcs

Assume all arc costs are integer.

Consider the case when an arc (x, y) adjacent to the root x has a cost equal to $M\epsilon$ for some large M.

Then $\overline{T}^{\epsilon}(x)$ will extend up to the successors of y, at a distance at least 2ϵ from y, i.e. at a distance at least $(M+2)\epsilon$ from x.

Therefore the algorithm cannot stop until all nodes within a distance $(M+2)\epsilon$ have been permanently labelled.

Solution: smaller trees are built, with the drawback that some low reach nodes can be misclassified as high reach nodes.

This produces weaker upper bounds, but does not affect the correctness of the $s-t$ shortest path algorithm.

Smaller trees

Let

- x be the root of the shortest path arborescence $T^\epsilon(x)$;
- $k \neq x$ a node in $T^\epsilon(x)$;
- $f(k)$ the successor of x along the shortest path from x to k.

The set of *inner nodes* of $T^\epsilon(x)$ is

$$I^\epsilon(x) = \{x\} \cup \{k \in T^\epsilon(x) : (k \neq x) \wedge (dist(f(k), k) \leq \epsilon)\}.$$

The set of *outer nodes* of $T^\epsilon(x)$ is its complement.

$$O^\epsilon(x) = T^\epsilon(x) \backslash I^\epsilon(x).$$

Smaller trees

For any outer node $w \in O^\epsilon(x)$, its distance from $I^\epsilon(x)$ is defined as

$$\min_{v \in I^\epsilon(x)} \{dist(v, w)\}.$$

The algorithm stops growing the shortest paths arborescence when

- all nodes with non-permanent labels are outer nodes, and
- they have distance at least ϵ from $I^\epsilon(x)$.

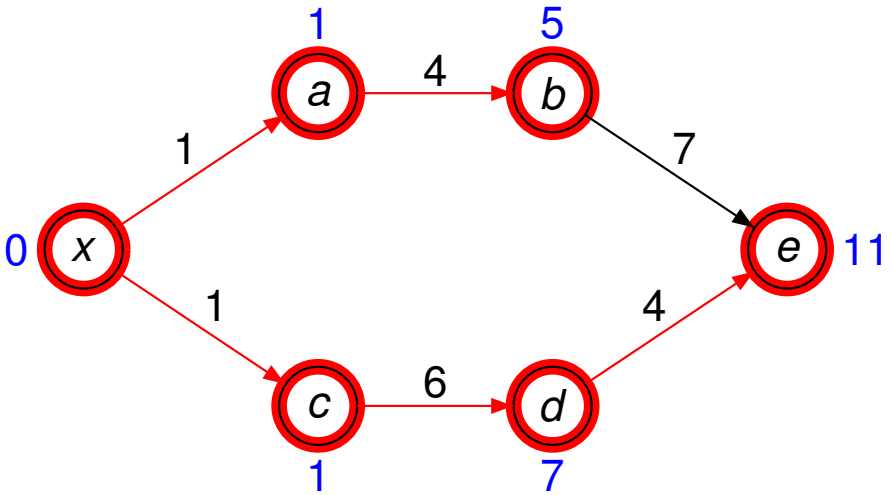


Figure: Shortest path tree T^ϵ for $\epsilon = 5$. Red: permanent labels. $r(b) = 0$.

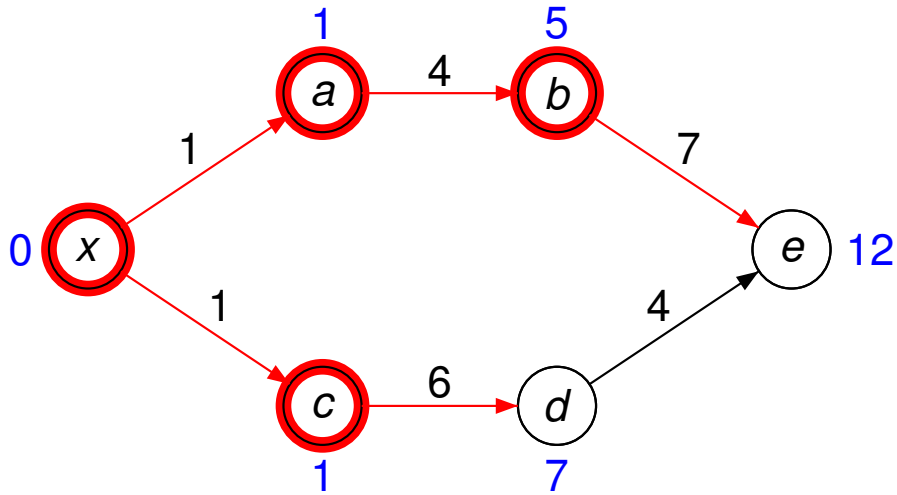


Figure: Smaller tree T^ϵ for $\epsilon = 5$. Red: permanent labels. $r(b) = 5$.

Arc reaches

Let (u, v) be an arc along an $s - t$ shortest path $P^*(s, t)$.

Then $r(u, v, P^*(s, t)) = \min\{dist(s, v), dist(u, t)\}$.

On the whole graph $r(u, v) = \max_{s \in \mathcal{N}, t \in N}\{r(u, v, P^*(s, t))\}$.

Node reaches can be computed from arc reaches:
$r(i) = \max\{\max_{(i,j) \in \mathcal{A}}\{r(i, j)\}, \max_{(j,i) \in \mathcal{A}}\{r(j, i)\}\}$.

The reach of an arc can be smaller than the reaches of its endpoints.

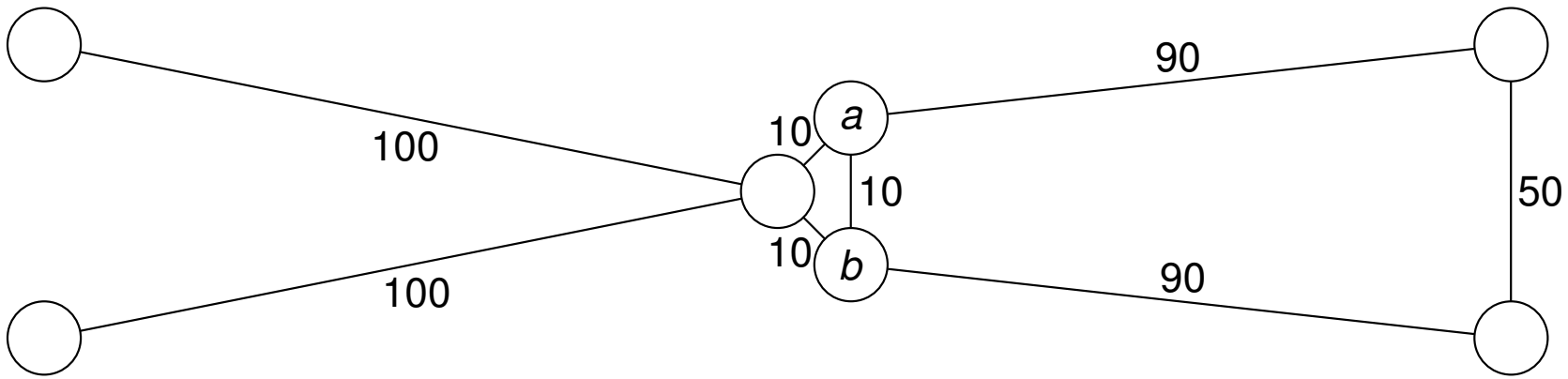


Figure: $r(a) = r(b) = 90$. $r(a, b) = 10$.

Penalties

When an arc is identified as a “low reach arc” and its reach is bounded above by the current ϵ, in the next iteration the arc is deleted and replaced by a penalty, representing upper bounds on the effect of the deleted arc on the reach of its endpoints.

Let A_k be the set of arcs remaining (not yet upper bounded) at iteration k.

In-penalties π^- and out-penalties π^+ are defined as follows for all nodes that are endpoints of deleted (low-reach) arcs:

$$\pi^-(i) = \max_{(j,i)\in\mathcal{A}^+:(j,i)\notin A_k} \{\overline{r}(j,i)\}$$

$$\pi^+(i) = \max_{(i,j)\in\mathcal{A}^+:(i,j)\notin A_k} \{\overline{r}(i,j)\},$$

where $\mathcal{A}^+$ includes both $\mathcal{A}$ and the shortcut arcs added in previous iterations. The definition of reach is generalized as follows:

$$r(u,v,P^*(s,t)) = \min\{dist(s,v)+\pi^+(v), dist(u,t)+\pi^-(u)\}.$$

Penalties

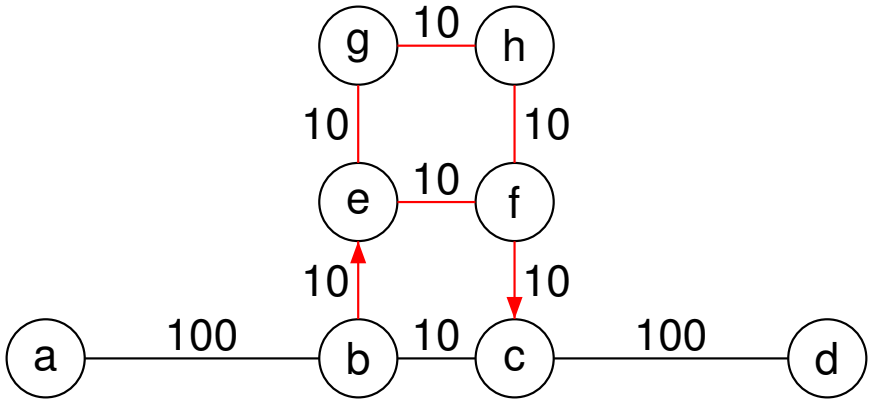


Figure: Red arcs have (low) reach ≤ 30.

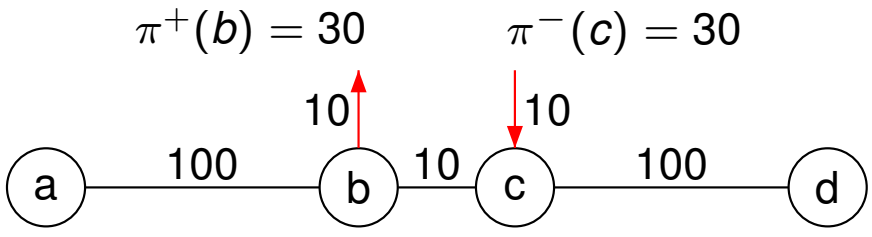


Figure: Low reach arcs replaced by penalties when growing partial trees.

Partial trees keep the same definition as before.

Shortcuts

A node j is *by-passable* if

- it has only one incoming arc (i, j) and one outgoing arc (j, k) (one-way by-passable), or
- it has only two incoming arcs (i, j) and (k, j) and only two outgoing arcs (j, i) and (j, k) (two-ways by-passable).

A *line* is a path of at least three nodes, where all nodes different from the endpoints are by-passable.

Lines can be one-way or two-ways.

A by-pass is an arc directly connecting the endpoints of a line.

By-passes can be one-way or two-ways.

Cost and perturbation of shortcut arcs are given by the sum of costs and perturbation of the by-passed arcs.

Shortcuts

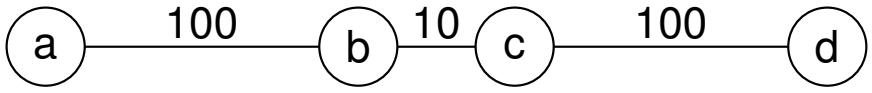
By-passed nodes are no longer visited in shortest paths containing their by-passed line.
Therefore shortcuts reduce the reaches of by-passed nodes.

If a line (s, t) has more than two arcs,

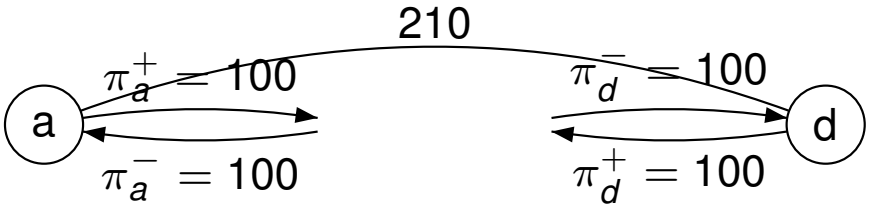
- find the node k in it, that minimizes $|dist(s, k) - dist(k, t)|$ (median node);
- add a shortcut (s, t) (if it is not in the current arc set $\mathcal{A}^+$);
- recursively do the same on each subpath (s, k) and (k, t).

Shortcuts

When a node is by-passed, it is deleted and replaced by a penalty assigned to its neighbors.



A two-ways line with three arcs.



The by-passed line replaced by π.

To avoid long shortcuts that would imply large partial trees, the maximum length of shortcuts is limited to $\frac{\epsilon_{k+1}}{2}$ at each iteration k.

Shortcuts

Given a one-way line (u, v, w), when a shortcut (u, w) is added, arc (u, v) will never be used on any shortest path that goes through u and w anymore.

Any shortest path traversing (u, v) will end either in v or in some low-reach area neighboring v.

Therefore, a valid upper bound for the reach of (u, v) is $\overline{r}(u, v) = c_{uv} + \pi^+(v)$ (and the same holds for (v, w) symmetrically).

Owing to these upper bounds, one can immediately remove v, (u, v) and (v, w) from the graph and update the appropriate penalties.

A similar procedure can be adopted for two-way lines.

Refinement phase

The use of penalties makes the upper bounds looser and looser as the algorithm progresses.

This is more evident on nodes with larger reaches.

Therefore the reaches are re-computed in a more accurate way for the δ nodes with highest reaches, where $\delta = \lceil 10\sqrt{n} \rceil$.

Let V_δ the set of such nodes and G_δ the subgraph induced by V_δ.

A complete shortest path arborescence is computed from each node in G_δ, using penalties to account for missing nodes.

Parameter tuning

Select $k = \min\{500, \lfloor \lceil \sqrt{n} \rceil / 3 \rfloor\}$ nodes at random.

Grow a partial shortest path arborescence until $\lfloor n/k \rfloor$ nodes are permanently labeled.

For each root consider the radius, i.e. the distance of the last label.

Set ϵ_1 to twice the minimum among the k radii.

Parameter tuning

We also have to choose a multiplier α to compute $\epsilon_i = \alpha^{i-1}\epsilon_1$ at each iteration.

- **Running time:** the smaller α is, the more iterations will be done; but if α is large, iterations will take longer (since vertices are eliminated less frequently).
- **Number of shortcuts:** if α is relatively small, the algorithm has a better chance of shortcutting vertices before they are eliminated.
- **Upper bounds:** the error in an arc reach estimate at iteration *i* depends on the penalties, which in turn depend on the maximum reaches of arcs eliminated in previous iterations; the larger α is, the smaller the sum $\sum_{j<i} \epsilon_j$ compared to ϵ_i.

Heuristic rule: keep $\alpha = 3$ while the number of nodes remains larger than δ. Then reduce it to $\alpha = 1.5$.

Combining reaches with A^*

In A^* each node i has a label $k(i) = d(i) + \pi(i)$, where $d(i)$ is the distance from s and $\pi(i)$ is a lower bound on the distance to t.

In bi-directional A^* each node has two labels, $f(i) = d^s(i) + \pi^t(i)$ (forward) and $b(i) = d^t(i) + \pi^s(i)$ (backward).

When A^* is about to make the forward label of node i permanent, it checks the reach of i: if

$$r(i) < \min\{d_s(i), \pi^t(i)\},$$

then i is pruned.

The stop criterion (lower bound = upper bound) is not affected.