The $s - t$ shortest path problem

Combinatorial optimization

Giovanni Righini

UNIVERSITÀ DEGLI STUDI DI MILANO

The $s - t$ shortest path problem

Data:

- a digraph $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ with $|\mathcal{N}| = n$ nodes and $|\mathcal{A}| = m$ arcs;
- a source node $s \in \mathcal{N}$ and a target node $t \in \mathcal{N}$;
- a cost function $c : \mathcal{A} \mapsto \Re_+$.

The (s, t) Shortest Path Problem.
Find a minimum cost (i.e. shortest) path from s to t.

Owing to the non-negativity of arc costs, we do not need to explicitly forbid cycles and we can use Dijkstra algorithm.

SPP: primal formulation

$$\text{minimize } z = \sum_{(i,j)\in\mathcal{A}} c_{ij}x_{ij}$$

$$\text{s.t.} \sum_{(j,i)\in\delta_i^-} x_{ji} - \sum_{(i,j)\in\delta_i^+} x_{ij} = \begin{cases} -1 & i = s \\ 0 & \forall i \in \mathcal{N}\backslash\{s,t\} \\ 1 & i = t \end{cases}$$

$$x_{ij} \in \mathcal{Z}_+ \;\; \forall (i,j) \in \mathcal{A}.$$

Observation 1. The constraint matrix is totally unimodular.

Observation 2. The right-hand-sides of the constraints are integers.

Hence, every base solution of the continuous relaxation has integer coordinates.

A primal-dual pair

$$\text{minimize } z = \sum_{(i,j)\in\mathcal{A}} c_{ij}x_{ij}$$

$$\text{s.t.} \sum_{(j,i)\in\delta_i^-} x_{ji} - \sum_{(i,j)\in\delta_i^+} x_{ij} = \begin{cases} -1 & i = s \\ 0 & \forall i \in \mathcal{N}\setminus\{s,t\} \\ 1 & i = t \end{cases}$$

$$x_{ij} \geq 0 \;\; \forall (i,j) \in \mathcal{A}.$$

$$\text{maximize } w = y_t - y_s$$

$$\text{s.t. } y_j - y_i \leq c_{ij} \qquad \forall (i,j) \in \mathcal{A}$$

$$y_i \text{ free} \qquad \forall i \in \mathcal{N}.$$

The dual variable y_s can be et to 0; its corresponding primal constraint is redundant.

Complementary slackness conditions (CSC)

$$\text{minimize } z = \sum_{(i,j)\in\mathcal{A}} c_{ij}x_{ij}$$

$$\text{s.t.} \sum_{(j,i)\in\delta_i^-} x_{ji} - \sum_{(i,j)\in\delta_i^+} x_{ij} = \begin{cases} 0 & \forall i \in \mathcal{N}\setminus\{s,t\} \\ 1 & i = t \end{cases}$$

$$x_{ij} \geq 0 \;\; \forall (i,j) \in \mathcal{A}.$$

$$\text{maximize } w = y_t$$

$$\text{s.t. } y_j - y_i \leq c_{ij} \qquad \forall (i,j) \in \mathcal{A}$$

$$y_i \text{ free} \qquad \forall i \in \mathcal{N}\setminus\{s\}.$$

Primal CSCs: $x_{ij}(c_{ij} + y_i - y_j) = 0$.

Basic primal variables correspond to active dual constraints.

Only arcs (i,j) for which $y_i + c_{ij} = y_j$ can carry flow x_{ij}.

Bi-directional algorithm

By symmetry, instead of cost labels $d(i)$ representing shortest distances from s to i, one can use cost labels representing shortest distances from i to t.

The same algorithm is executed from t backwards, using reversed arcs.

Correctness and complexity remain unchanged.

The idea of the bi-directional algorithm is to do both things simultaneously.

Intuitively, this allows to decrease the number of extensions needed to find a shortest $s - t$ path.

Data-structures

Two labels are associated with each node, a forward cost label d'_i and a backward cost label d''_i, meaning the current shortest distance from s to i and from i to t, respectively.

Correspondingly, a forward predecessor label π'_i and a backward predecessor label d''_i indicate the best predecessor and the best successor along the shortest path from s to i and from i to t, respectively.

Initially, $d'_s = d''_t = 0$ and all the other labels are set to ∞.

Open (non-permanent) cost labels are kept in two heaps H' and H''.

Upper bounds

For each node i in the digraph, the sum of its two labels, $d'_i + d''_i$, represents the cost of an $s - t$ path visiting i.

Therefore it is an upper bound U_i to the optimal value.

We record the best incumbent upper bound: $U = \min_{i \in \mathcal{N}}\{d'_i + d''_i\}$.

When both labels d'_i and d''_i are permanent, then their sum is the cost of the shortest $s - t$ path visiting i.

Lower bounds

When a label is not permanent, it can still decrease down to the value of the smallest non-permanent label in its direction, i.e. the label at the root of the corresponding heap.

We indicate these minimum non-permanent labels by $top(H)$ for each heap H.

So, $top(H')$ and $top(H'')$ are lower bounds for the values of non-permanent forward and backward labels, respectively.

Therefore $L_i = \min\{d'_i, top(H')\} + \min\{d''_i, top(H'')\}$ is a lower bound for the cost of any $s-t$ path visiting i.

Therefore $L = \min_{i \in \mathcal{N}}\{L_i\}$ is a lower bound for the optimal value.

A stronger lower bound

However, we can stop the algorithm when $U \leq top(H') + top(H'')$.

By contradiction, assume there is a path P with cost $c(P) < U$.

Indicate the shortest distance from s to any $i \in \mathcal{N}$ with $dist'(i)$ and the shortest distance from any $i \in \mathcal{N}$ to t with $dist''(i)$.

For all nodes $i \in \mathcal{N}$ along P,
$dist'(i) + dist''(i) = c(P) < U \leq top(H') + top(H'')$.

Then, for all nodes along P, $dist'(i) < top(H') \ \vee \ dist''(i) < top(H'')$.

Then, all nodes along P have been already permanently labelled in at least one direction and hence P should have been already discovered.

Bi-directional Dijkstra algorithm

```
Initialization
while (top(H') + top(H'') < U) do
    if (top(H') ≤ top(H'')) then
        PropagateFw
    else
        PropagateBw
```

Initialization

for $i \in \mathcal{N} \setminus \{s\}$ **do**
  $d'(i) \leftarrow \infty$
$d'(s) \leftarrow 0$
for $i \in \mathcal{N} \setminus \{t\}$ **do**
  $d''(i) \leftarrow \infty$
$d''(t) \leftarrow 0$
for $i \in \mathcal{N}$ **do**
  Insert$(i, d'(i), H')$
  Insert$(i, d''(i), H'')$
  $\pi'(i) \leftarrow nil$
  $\pi''(i) \leftarrow nil$
$U \leftarrow \infty$

PropagateFw

$k \leftarrow \text{ExtractMin}(H')$
for $j \in \delta^+(k)$ **do**
  if $d'(j) > d'(k) + c(k,j)$ **then**
    $d'(j) \leftarrow d'(k) + c(k,j)$
    $\pi'(j) \leftarrow k$
    if $d'(j) + d''(j) < U$ **then**
      $U \leftarrow d'(j) + d''(j)$

PropagateBw

$k \leftarrow \text{ExtractMin}(H'')$
for $j \in \delta^-(k)$ **do**
  if $d''(j) > d''(k) + c(j,k)$ **then**
    $d''(j) \leftarrow d''(k) + c(k,j)$
    $\pi''(j) \leftarrow k$
    if $d'(j) + d''(j) < U$ **then**
      $U \leftarrow d'(j) + d''(j)$

The A^* algorithm (Hart, Nilsson, Raphael, 1968)

We define a *bounding function* $h : \mathcal{N} \mapsto \Re$ such that:

- $h(t) = 0$
- $h(i) - h(j) \leq c(i,j) \ \ \forall (i,j) \in \mathcal{A}$.

It represents a *lower bound* for the minimum distance from each node to node t, i.e. $dist(i,t)$.

A trivial bounding function is $h(i) = 0 \ \ \forall i \in \mathcal{N}$, which yields Dijkstra algorithm.

Running A^* on the original graph is equivalent to running Dijkstra algorithm on a digraph with modified costs

$$\tilde{c}(i,j) = c(i,j) + h(j) - h(i) \ \ \forall (i,j) \in \mathcal{A}.$$

Dual constraints

Dual constraints:

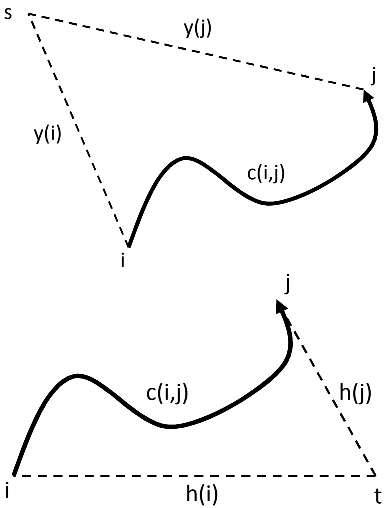
$$y_j - y_i \leq c_{ij} \quad \forall (i,j) \in \mathcal{A}$$

Lower bounding function:

$$\begin{cases} h(t) = 0 \\ h(i) - h(j) \leq c_{ij} \ \ \forall (i,j) \in \mathcal{A} \end{cases}$$

Setting $y(i) = 0 \ \ \forall i \in \mathcal{N}$, yields a feasible dual solution.
Setting $y(i) = -h(i) \quad \forall i \in \mathcal{N}$, too.

The primal-dual algorithm corresponding to Dijkstra algorithm can be slightly modified to represent A^* algorithm.



Primal-dual algorithm (A^*)

$O \leftarrow \{s\}$; $E \leftarrow \emptyset$; $\Phi \leftarrow 0$; $y(s) \leftarrow$ **-h(s)**; $\pi(s) \leftarrow s$
while $(O \neq \emptyset) \wedge (t \notin E)$ **do**
  $j \leftarrow \text{argmin}_{v \in O}\{c(\pi(v), v) - y(v) + y(\pi(v))\}$
  $\theta \leftarrow c(\pi(j), j) - y(j) + y(\pi(j))$
  $O \leftarrow O \setminus \{j\}$; $E \leftarrow E \cup \{j\}$; $\Phi \leftarrow \Phi + \theta$; $y(j) \leftarrow$ **-h(j)** $+ \Phi$
  for $k \in O$ **do**
    $y(k) \leftarrow$ **-h(k)** $+ \Phi$
  for $(j, k) \in \delta^+(j) : k \notin E$ **do**
    if $k \in O$ **then**
      if $y(j) + c(j, k) < y(\pi(k)) + c(\pi(k), k)$ **then**
        $\pi(k) \leftarrow j$
    else
      $O \leftarrow O \cup \{k\}$; $y(k) \leftarrow$ **-h(k)** $+ \Phi$; $\pi(k) \leftarrow j$

The primal-dual algorithm (A^*)

At each iteration θ indicates the minimum slack of the constraints corresponding to arcs crossing the (E, O) cut.

The variable Φ indicates the cumulative amount of slack, from the beginning of the algorithm.

The dual variable $y(s)$ remains fixed at $-h(s)$.
When the algorithm terminates $\Phi = y(t)$.
Then, at the end, $\Phi - y(s)$ gives the optimal value:
$\Phi - y(s) = y(t) - y(s) = w$.

For each node in E, $y(i) - y(s) = dist(s, i)$.
For each node in O, $y(i) = -h(i) + \Phi$.
For each node in O, $y(i) - y(s) = h(s) - h(i) + \Phi \leq dist(s, i)$.

Primal-dual algorithm (A^*)

We now exploit three facts:

- $y(i) = -h(i) + \Phi \;\; \forall i \in O$;
- the predecessor $\pi(i) \;\; \forall i \in O$ always exists and is unique;
- predecessors of nodes in O must be in E.

Therefore we rewrite the algorithm, by replacing $y(i)$ with $y(\pi(i)) + c(\pi(i), i)$ for all nodes $i \in O$, with no need to explicitly update the values of non-permanent dual variables.

Now $y(i)$ appears only for nodes in E.

Primal-dual algorithm (A^*) (revised)

$O \leftarrow \{s\}$; $E \leftarrow \emptyset$; $\Phi \leftarrow 0$; $y(s) \leftarrow$ **-h(s)**; $\pi(s) \leftarrow s$
while $(O \neq \emptyset) \wedge (t \notin E)$ **do**
  $j \leftarrow \text{argmin}_{v \in O}\{c(\pi(v), v) + h(v) - \Phi + y(\pi(v))\}$
  $\theta \leftarrow c(\pi(j), j) + h(j) - \Phi + y(\pi(j))$
  $O \leftarrow O \backslash \{j\}$; $E \leftarrow E \cup \{j\}$; $\Phi \leftarrow \Phi + \theta$; $y(j) \leftarrow$ **-h(j)** $+ \Phi$
  for $(j, k) \in \delta^+(j) : k \notin E$ **do**
    if $k \in O$ **then**
      if $y(j) + c(j, k) < y(\pi(k)) + c(\pi(k), k)$ **then**
        $\pi(k) \leftarrow j$
    else
      $O \leftarrow O \cup \{k\}$; $\pi(k) \leftarrow j$

The label d

Let introduce $d(j)$ such that:

$$d(j) = \begin{cases} dist(s, j) & \forall j \in E \\ d(\pi(j)) + c(\pi(j), j) & \forall j \in O \end{cases}$$

The label $d(j)$ is defined only for nodes in $E \cup O$, i.e. for nodes with a predecessor. Their predecessor is guaranteed to be in E.

The selection test

We now exploit the relation $y(i) - y(s) = dist(s, i) \quad \forall i \in E$ to rewrite the selection criterion

$$j \leftarrow \text{argmin}_{v \in O}\{c(\pi(v), v) - y(v) + y(\pi(v))\}$$

in an equivalent way:

$$\begin{aligned}
&c(\pi(v), v) - y(v) + y(\pi(v)) = \\
&c(\pi(v), v) - (\Phi - h(v)) + y(\pi(v)) = \\
&c(\pi(v), v) + y(\pi(v)) + h(v) - \Phi = \\
&c(\pi(v), v) + (y(\pi(v)) - y(s)) + h(v) - \Phi + y(s) = \\
&c(\pi(v), v) + dist(s, \pi(v)) + h(v) - \Phi + y(s) = \\
&(c(\pi(v), v) + d(\pi(v))) + h(v) - \Phi + y(s) = \\
&d(v) + h(v) - (\Phi - y(s)).
\end{aligned}$$

Since $\Phi - y(s)$ does not depend on the nodes, the selection criterion can rewritten as

$$j \leftarrow \text{argmin}_{v \in O}\{d(v) + h(v)\}$$

The A^* algorithm

$O \leftarrow \{s\};\ \ E \leftarrow \emptyset;\ \ d(s) \leftarrow 0$
while $(O \neq \emptyset) \wedge (t \notin E)$ **do**
  $j \leftarrow \text{argmin}_{v \in O}\{d(v) + h(v)\}$
  $O \leftarrow O \setminus \{j\};\ \ E \leftarrow E \cup \{j\}$
  for $k \in \delta^+(j) : k \notin E$ **do**
    if $k \in O$ **then**
      if $d(k) > d(j) + c(j,k)$ **then**
        $d(k) \leftarrow d(j) + c(j,k);\ \ \pi(k) \leftarrow j$
    else
      $O \leftarrow O \cup \{k\};\ \ d(k) \leftarrow d(j) + c(j,k);\ \ \pi(k) \leftarrow j$

Selection rule

After defining $f(i) = d(i) + h(i)$, the nodes are scanned in non-decreasing order of f.

In Dijkstra algorithm, they are scanned in non-decreasing order of d.

If i enters E before j, then $f(i) \leq f(j)$.

Then, for each $i \in E$ we have $f(i) \leq dist(s,t)$, because $f(j) \geq f(i) \;\; \forall i \in E, j \notin E$ and $dist(s,t) \geq \max_{i \in \mathcal{N}}\{f(i)\}$.

The “most promising” node is selected, instead of the closest to s.

The properties of h guarantee that its label selected in this way is permanent.

Dominance

Given two bounding functions h_1 and h_2, if $h_1(i) > h_2(i)$ for each $i \in \mathcal{N}$, then $E_1 \subseteq E_2$ when t is closed and the algorithm stops.

This means that h_1 dominates h_2.

The larger is h, the more efficient A^* is: it needs considering fewer nodes.

The trivial bounding function $h = 0$ is dominated by any other.

The ideal bounding function is such that $h(i) = dist(i, t)$.
In such an ideal case, only the nodes in P^* are inserted in E.

Finding a bounding function

A bounding function h can be obtained from an associated function H defined for all pairs of nodes, although they are not connected by arcs.

Properties of $H : (\mathcal{N} \times \mathcal{N}) \mapsto \Re_+$:

- $H(i,j) \geq 0 \;\; \forall i,j \in \mathcal{N}$
- $H(i,i) = 0 \;\; \forall i \in \mathcal{N}$
- $c(i,j) + H(j,k) \geq H(i,k) \;\; \forall (i,j) \in \mathcal{A}, k \in \mathcal{N}$

This yields $h(i) = H(i,t) \;\; \forall i \in \mathcal{N}$.

A typical example is the Euclidean distance, when we compute shortest paths on street networks.

Strengthening the bounding function

Assume to run Dijkstra algorithm from t backwards and to stop it at a generic iteration, before making the label of s permanent.

The selected basic arcs form an arborescence T rooted in t, including nodes with a permanent label (set E^T) and nodes with a non-permanent label (O^T).

The following function provides a valid lower bound:

$$h^{HT}(i) = \begin{cases} dist(i,t) & \forall i \in E^T \\ \min_{j \in O^T} \{H(i,j) + dist(j,t)\} & \forall i \notin E^T \end{cases}$$

Therefore $h(i) = H(i,t) \leq h^{HT}(i) \leq dist(i,t)$.
First inequality from the triangle inequality.
Second inequality from the definition above and Bellman's principle.
So, h^{HT} gives a stronger lower bound than h^T, but it takes more time to evaluate.

Bi-directional A^*

We can define a forward lower bounding function $h' : \mathcal{N} \mapsto \Re_+$ and a backward lower bounding function $h'' : \mathcal{N} \mapsto \Re_+$ such that:

- $h'(i), h''(i) \geq 0 \;\; \forall i \in N$
- $h'(t) = h''(s) = 0$
- $c(i,j) + h'(j) \geq h'(i) \;\; \forall (i,j) \in \mathcal{A}$
- $c(i,j) + h''(i) \geq h''(j) \;\; \forall (i,j) \in \mathcal{A}$

Setting $y = h''$ yields another dual feasible solution, suitable for bi-directional search.

We need sets O', O'', E' and E''.
We also need dual variables y' and y'' and primal variables π' and π''.

Bi-directional A^*

When a node is reached in both directions, i.e. $\exists i \in O' \cap O''$, then a feasible $s - t$ path is found, visiting i.

Its cost is
$U_i = c(\pi'(i), i) + y'(\pi'(i)) - y'(s) + c(i, \pi''(i)) + y''(\pi''(i)) - y''(t)$ and it is a valid upper bound.

We record the best incumbent upper bound U.

$$y'(t) - y'(s) \leq dist(s,t) \leq U$$

$$y''(s) - y''(t) \leq dist(s,t) \leq U$$

The search stops when

$$\max\{y'(t) - y'(s), y''(s) - y''(t)\} = U.$$

Heuristic A^*

Using a bounding function $\tilde{h} = \epsilon h$, with $\epsilon > 1$, we lose the optimality guarantee, because $\tilde{h}$ is not guaranteed to be a valid lower bounding function.

However, the resulting algorithm gurantees to provide a (heuristic) solution whose value is not larger than ϵ times the optimum.

In this way, we may design a constant-factor approximation algorithm, by suitably tuning the trade-off between solution quality and computing time.