The $s - t$ shortest path problem

Giovanni Righini

Partially based on: Ye, Han, Lin, *A Note on the Connection between the Primal-Dual and the A* Algorithm*

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 assumption that arc costs are non-negative, we do not need to explicitly forbid cycles.

SPP: primal formulation

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

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

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

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

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

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

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

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

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

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

The dual variable y_s can be set 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}.

Mono-directional $s-t$ Dijkstra algorithm (dual ascent)

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

Mono-directional $s - t$ Dijkstra algorithm (dual ascent) simplified

We exploit $y(k) = w \;\; \forall k \in O$ at any iteration.

$O \leftarrow \{s\}$; $E \leftarrow \emptyset$; $\Phi \leftarrow 0$; $y(s) \leftarrow 0$; $\pi(s) \leftarrow s$
while $(O \neq \emptyset) \wedge (t \notin E)$ **do**
  $j \leftarrow \text{argmin}_{v \in O}\{c(\pi(v), v) - (\Phi - y(\pi(v)))\}$
  $\theta \leftarrow c(\pi(j), j) - (\Phi - y(\pi(j)))$
  $\Phi \leftarrow \Phi + \theta$
  $O \leftarrow O \backslash \{j\}$; $E \leftarrow E \cup \{j\}$; $y(j) \leftarrow \Phi$
  for $(j, k) \in \delta^+(j) : k \notin E$ **do**
    if $k \in O$ **then**
      if $\Phi + c(j, k) < y(\pi(k)) + c(\pi(k), k)$ **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 $d(i)$ is the shortest current distance from s to i.

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

The predecessor is guaranteed to be in E: $d(\pi(j)) = dist(s, \pi(j))$.

The selection criterion

Since Φ does not depend on any specific node,

$\text{argmin}_{v \in O}\{c(\pi(v), v) - \Phi + y(\pi(v))\} = \text{argmin}_{v \in O}\{c(\pi(v), v) + y(\pi(v)))\}$.

Since $\pi(v) \in E$, by definition $y(\pi(v)) = dist(s, \pi(v)) = d(\pi(v))$.

$\text{argmin}_{v \in O}\{c(\pi(v), v) + y(\pi(v)))\} = \text{argmin}_{v \in O}\{c(\pi(v), v) + d(\pi(v)))\}$.

For each $v \in O$, by definition $d(v) = d(\pi(v)) + c(\pi(v), v)$.

$$\text{argmin}_{v \in O}\{c(\pi(v), v) + d(\pi(v)))\} = \text{argmin}_{v \in O}\{d(v)\}.$$

Mono-directional $s - t$ Dijkstra algorithm (labels d)

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

The set O can be implemented with a heap H.

Mono-directional $s - t$ Dijkstra algorithm: implementation

for $i \in \mathcal{N}$ **do**
  $\pi(i) \leftarrow nil$
$d(s) \leftarrow 0$; $\pi(s) \leftarrow s$; $H \leftarrow nil$
$Insert(s, d(s))$
repeat
  $(j, d(j)) \leftarrow ExtractMin$
  if $j \neq t$ **then**
    for $(j, k) \in \delta^+(j)$ **do**
      if $\pi(k) \neq nil$ **then**
        if $d(k) > d(j) + c(j, k)$ **then**
          $DecreaseKey(k, d(j) + c(j, k))$
          $\pi(k) \leftarrow j$
      else
        $Insert(k, d(j) + c(j, k))$
        $\pi(k) \leftarrow j$
until $(H = nil) \vee (j = t)$

Bi-directional Dijkstra algorithm

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

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

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.

Correctness and complexity proofs remain unchanged.

Data-structures

Two labels and two predecessor are associated with each node:

- a forward cost label $d'(i)$: current shortest distance from s to i;
- a backward cost label $d''(i)$: current shortest distance from i to t;
- a forward predecessor $\pi'(i)$: predecessor along the current shortest path from s to i;
- a backward predecessor $\pi''(i)$: successor along the current shortest path from i to t.

Initially, $d'(s) = d''(t) = 0$ and $\pi'(s) = s$ and $\pi''(t) = t$.

Only 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 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 and termination test

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.

Termination test: $U \leq Top(H') + Top(H'')$.

Termination test

Label extensions from $i \in O'$ to $j \in E''$ (forward) or from $j \in O''$ to $i \in E'$ (backward) can be disregarded, because they would produce a same path twice.

Hence, no arc is used forward *and* backward.

New paths can be found only by traversing arcs (i, j) with $i \in O'$ and $j \in O''$, in either direction.

Labels d' and d'' generated traversing these arcs are not smaller than $Top(H')$ and $Top(H'')$, respectively.

Hence, new paths cannot be shorter than $Top(H') + Top(H'')$.

Bi-directional $s - t$ Dijkstra algorithm (labels d)

$O' \leftarrow \{s\}$; $E' \leftarrow \emptyset$; $d'(s) \leftarrow 0$; $\pi'(s) \leftarrow s$
$O'' \leftarrow \{t\}$; $E'' \leftarrow \emptyset$; $d''(t) \leftarrow 0$; $\pi''(t) \leftarrow t$
$L' \leftarrow 0$; $L'' \leftarrow 0$; $U \leftarrow \infty$
while $(O' \neq \emptyset) \wedge (O'' \neq \emptyset) \wedge (L' + L'' < U)$ **do**
  /* Select direction */
  if *direction* $=$ *forward* **then**
    PropagateFw
  else
    PropagateBw

Label propagation (forward)

$j \leftarrow \text{argmin}_{v \in O'} \{d'(v)\}$
$L' \leftarrow d'(j)$
$O' \leftarrow O' \setminus \{j\}; \quad E' \leftarrow E' \cup \{j\}$
for $(j, k) \in \delta^+(j) : k \notin E' \cup E''$ **do**
  if $k \in O'$ **then**
    if $d'(k) > d'(j) + c(j, k)$ **then**
      $d'(k) \leftarrow d'(j) + c(j, k); \quad \pi'(k) \leftarrow j$
  else
    $O' \leftarrow O' \cup \{k\}; \quad d'(k) \leftarrow d'(j) + c(j, k); \quad \pi'(k) \leftarrow j$
  if $(k \in O'') \wedge (d'(k) + d''(k) < U)$ **then**
    $U \leftarrow d'(k) + d''(k)$

Backward propagation is symmetric.

Bi-directional Dijkstra algorithm (implementation)

$d'(s) \leftarrow 0; \quad \pi'(s) \leftarrow s$
$d''(t) \leftarrow 0; \quad \pi''(t) \leftarrow t$
$Insert(s, d'(s), H')$
$Insert(t, d''(t), H'')$
$U \leftarrow \infty$
while $(Top(H') + Top(H'') < U)$ **do**
  if $(Top(H') \leq Top(H''))$ **then**
    $PropagateFw$
  else
    $PropagateBw$

PropagateFw

$(j, d'(j)) \leftarrow \text{ExtractMin}(H')$
for $(j,k) \in \delta^+(j)$ **do**
  if $\pi'(k) \neq nil$ **then**
    if $d'(k) > d'(j) + c(j,k)$ **then**
      $d'(k) \leftarrow d'(j) + c(j,k); \quad \pi'(k) \leftarrow j$
      $DecreaseKey(k, d'(k), H')$
  else
    $d'(k) \leftarrow d'(j) + c(j,k); \quad \pi'(k) \leftarrow j$
    $Insert(k, d'(k), H')$
  if $(\pi''(k) \neq nil) \wedge (d'(k) + d''(k) < U)$ **then**
    $U \leftarrow d'(k) + d''(k)$

PropagateBw is symmetric.

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:

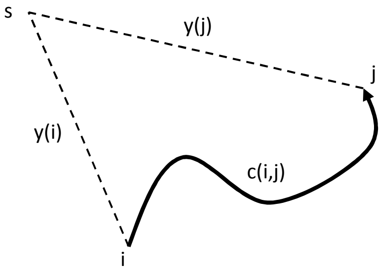
$$y_j - y_i \leq c_{ij} \quad \forall (i,j) \in \mathcal{A}$$

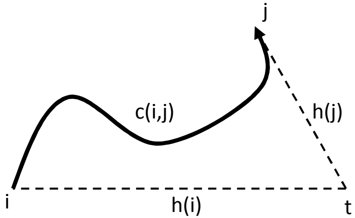
Lower bounding function:

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

Feasible dual solutions:
$y(i) = 0 \;\; \forall i \in \mathcal{N}$;
$y(i) = -h(i) \quad \forall i \in \mathcal{N}$.

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





Primal-dual algorithm (A^*)

$O \leftarrow \{s\};\quad E \leftarrow \emptyset;\quad \Phi \leftarrow 0;\quad y(s) \leftarrow \mathbf{-h(s)};\quad \pi(s) \leftarrow s$
while $(O \neq \emptyset) \wedge (t \notin E)$ **do**
  $j \leftarrow \mathrm{argmin}_{v \in O}\{c(\pi(v), v) - y(v) + y(\pi(v))\}$
  $\theta \leftarrow c(\pi(j), j) - y(j) + y(\pi(j))$
  $\Phi \leftarrow \Phi + \theta$
  for $k \in O$ **do**
    $y(k) \leftarrow \mathbf{-h(k)} + \Phi$
  $O \leftarrow O \setminus \{j\};\quad E \leftarrow E \cup \{j\}$
  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
      $O \leftarrow O \cup \{k\};\quad y(k) \leftarrow \mathbf{-h(k)} + \Phi;\quad \pi(k) \leftarrow j$

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:

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

Therefore we rewrite the algorithm, by replacing $y(i)$ with $-h(i) + \Phi$ 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**
  $j \leftarrow \text{argmin}_{v \in O}\{c(\pi(v), v) + h(v) - \Phi + y(\pi(v))\}$
  $\theta \leftarrow c(\pi(j), j) + h(j) - \Phi + y(\pi(j))$
  $O \leftarrow O \setminus \{j\}$; $E \leftarrow E \cup \{j\}$; $\Phi \leftarrow \Phi + \theta$; $y(j) \leftarrow$ **-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 $y(i) - y(s) = dist(s, i) \ \ \forall i \in E$ 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,

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

The A^* algorithm (with labels d)

$O \leftarrow \{s\}$; $E \leftarrow \emptyset$; $d(s) \leftarrow 0$
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:

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

Therefore $h(i) = H(i,t) \leq h^{HT}(i) \leq dist(i,t)$.

Then, h^{HT} gives a stronger lower bound than h^T, but it takes more time to evaluate.

Proof (1).

By definition of distance: $H(i,t) \leq H(i,j) + H(j,t) \;\; \forall j \in N$.

Since $H(j,t) \leq dist(j,t) \;\; \forall j \in N$, then

$H(i,t) \leq H(i,j) + dist(j,t) \;\; \forall j \in N$.

Hence,

$H(i,t) \leq \min_{j \in N}\{H(i,j) + dist(j,t)\} \leq \min_{j \in E^T}\{H(i,j) + dist(j,t)\} = h_i^{HT}$.

Proof (2).

Let $\bar{j}$ be a node in E^T along the shortest path from i to t. Then

$h_i^{HT} \leq H(i,\bar{j}) + dist(\bar{j},t) \leq dist(i,\bar{j}) + dist(\bar{j},t) = dist(i,t)$.

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.

Bi-directional A^*

To make A^* bi-directional, we 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''.

Forward and backward labels

Forward labels $d'(i)$ represent best incumbent distances from s to i.
Backward labels $d''(i)$ represent best incumbent distances from i to t.

The node set $\mathcal{N}$ is subdivided into subsets

- E': nodes with a permanent forward label $d'(i) = dist(s, i)$;
- E'': nodes with a permanent backward label $d''(i) = dist(i, t)$;
- O': nodes with a temporary forward label $d'(i) \geq dist(s, i)$;
- O'': nodes with a temporary backward label $d''(i) \geq dist(i, t)$;
- other nodes, not yet reached in either direction.

Only O' and O'' can intersect.

Labels in O' are sorted according to $f(i) = d'(i) + h'(i)$ in a heap F.
Labels in O'' are sorted according to $b(i) = d''(i) + h''(i)$ in a heap B.

A best incumbent upper bound U is possibly updated every time a new $s - t$ path is found.

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

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

Potential functions and lower bounds

A potential function $h : \mathcal{N} \mapsto \Re$ is used to define the reduced costs:

$$c^h(i,j) = c(i,j) - h(i) + h(j).$$

A potential function h is feasible iff

$$c^h(i,j) \geq 0 \ \forall (i,j) \in \mathcal{A}.$$

Property. A shortest $s-t$ path with respect to c^h is a shortest $s-t$ path with respect to c.

If $h(t) \leq 0$ and h is feasible, then h is a lower bounding function i.e.

$$h(i) \leq dist(i,t) \ \forall i \in \mathcal{N},$$

where $dist(i,t)$ is the shortest path cost from i to t.

Property 1

Property 1. If h is a feasible potential function, then $p = h + K$ is also a feasible potential function for any constant K.

Proof.
Reduced costs do not change by adding K: $c^h(i,j) \geq 0$ implies $c^p(i,j) \geq 0 \ \forall (i,j) \in \mathcal{A}$.

Property 2

Property 2. If h_1 and h_2 are feasible potential (lower bounding) functions, then $p = \max\{h_1, h_2\}$ is a feasible potential (lower bounding) function.

Proof.
(i) Feasibility.
If $(h_1(i) \geq h_2(i)) \wedge (h_1(j) \geq h_2(j))$, then $p = h_1$ which is feasible.
If $(h_1(i) \leq h_2(i)) \wedge (h_1(j) \leq h_2(j))$, then $p = h_2$ which is feasible.
If $(h_1(i) \geq h_2(i)) \wedge (h_1(j) \leq h_2(j))$, then

$$c^p(i,j) = c(i,j) - h_1(i) + h_2(j) \geq c(i,j) - h_1(i) + h_1(j) = c^{h_1}(i,j) \geq 0.$$

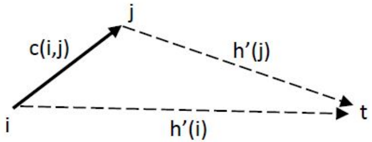
If $(h_1(i) \leq h_2(i)) \wedge (h_1(j) \geq h_2(j))$, then

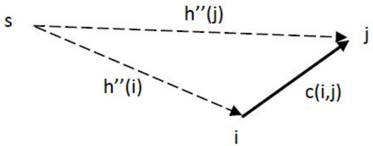
$$c^p(i,j) = c(i,j) - h_2(i) + h_1(j) \geq c(i,j) - h_2(i) + h_2(j) = c^{h_2}(i,j) \geq 0.$$

(ii) Lower bounding.
$h_1(t) \leq 0$ and $h_2(t) \leq 0$ imply $p(t) \leq 0$.

Symmetric and consistent bi-directional A^*

In bi-directional A^* let h' and h'' be the two lower bounding functions used in the forward and backward search, respectively.





The two lower bounding functions are consistent iff $\forall (i,j) \in \mathcal{A}$ the forward and backward reduced costs are the same:

$$c^{h'}(i,j) = c(i,j) - h'(i) + h'(j) = c(i,j) - h''(j) + h''(i) = c^{h''}(i,j).$$

This is equivalent to $h'(i) + h''(i) = K$ for some constant K.

One can use the same lower bounding algorithm (the best available one) to compute h' and h'' separately. In this case the two lower bounding functions are symmetric.

Symmetric and consistent potentials

Let h' and h'' be two lower bounding functions:

$$h'(i) \leq dist(i,t), \quad h''(i) \leq dist(s,i) \quad \forall i \in \mathcal{N}.$$

Two options for bi-directional A*:

- Symmetric algorithm. Use h' and h'' independenty in forward search and backward search.

  Pro: one can use the tightest available lower bounds in each direction.

- Consistent algorithm. Combine h' and h'' to obtain two consistent potentials.

  Pro: optimality is guaranteed as soon as the two searches meet.

Symmetric bi-directional A*

Pohl (1971), Kwa (1989). Using two independent lower bounds h' and h'', run the forward and backward searches, alternating in some way.

Each time a forward search scans an arc (j, k) s.t. $k \in O''$, if $d'(j) + c(j, k) + d''(k) < U$, then update U.

Each time a forward search selects a node $j \in O'$ and moves it to E', if $j \in O''$, remove it from O''.

Do the same symmetrically during backward search.

Termination. Stop as soon as one of these three conditions hold:

- forward search scans a node $i \in O'$ with $f(i) \geq U$;
- backward search scans a node $i \in O'$ with $b(i) \geq U$;
- one of the two searches has no nodes with temporary labels: $(O' = \emptyset) \vee (O'' = \emptyset)$.

Consistent bi-directional A*

T. Ikeda, M.-Y. Hsu, H. Imai, S. Nishimura, H. Shimoura, T. Hashimoto, K. Tenmoku, K. Mitoh, *A Fast Algorithm for Finding Better Routes by AI Search Techniques*. In Proc. Vehicle Navigation and Information Systems Conference. IEEE, 1994.

Let h' and h'' be valid (fw and bw, resp.) bounding functions:

$$c(i,j) - h'(i) + h'(j) \geq 0 \;\; \forall (i,j) \in A \quad h'(t) = 0.$$

$$c(i,j) - h''(j) + h''(i) \geq 0 \;\; \forall (i,j) \in A \quad h''(s) = 0.$$

Then, p' and p'' are valid (fw and bw, resp.) bounding functions:

$$p'(i) = (h'(i) - h''(i) + h''(t))/2$$

$$p''(i) = (h''(i) - h'(i) + h'(s))/2.$$

They are consistent: $p'(i) + p''(i) = (h''(t) + h'(s))/2 \; \forall i \in \mathcal{N}$.

Termination condition for the bi-directional consistent A* algorithm:

$$Top(F) + Top(B) \geq U + (h''(t) + h'(s))/2.$$