25.8 A*-Algorithm

Disclaimer

These slides contain the most important formalities around the A*-algorithm and its correctness. We motivate the algorithm in the lectures and give more examples there.

Another nice motivation of the algorithm can found here:

`https://www.youtube.com/watch?v=bRvs8rOQU-Q`

A*-Algorithm

Prerequisites

- Positively weighted graph $G = (V, E, c)$
- G finite or δ-Graph: $\exists\, \delta > 0 : c(e) \geq \delta$ for all $e \in E$
- $s \in V$, $t \in V$
- Distance estimate $\widehat{h}_t(v) \leq h_t(v) := \delta(v, t)\ \forall\, v \in V$.
- Wanted: shortest path $p : s \rightsquigarrow t$

A*-Algorithm($G, s, t, \hat{h}$)

Input: Positively weighted Graph $G = (V, E, c)$, starting point $s \in V$, end point $t \in V$, estimate $\widehat{h}(v) \leq \delta(v, t)$

Output: Existence and value of a shortest path from s to t

foreach $u \in V$ **do**

  $d[u] \leftarrow \infty$; $\widehat{f}[u] \leftarrow \infty$; $\pi[u] \leftarrow$ null

$d[s] \leftarrow 0$; $\widehat{f}[s] \leftarrow \widehat{h}(s)$; $R \leftarrow \{s\}$; $M \leftarrow \{\}$

while $R \neq \emptyset$ **do**

  $u \leftarrow$ ExtractMin$_{\widehat{f}}(R)$; $M \leftarrow M \cup \{u\}$

  if $u = t$ **then return** success

  foreach $v \in N^+(u)$ with $d[v] > d[u] + c(u, v)$ **do**

    $d[v] \leftarrow d[u] + c(u, v)$; $\widehat{f}[v] \leftarrow d[v] + \widehat{h}(v)$; $\pi[v] \leftarrow u$

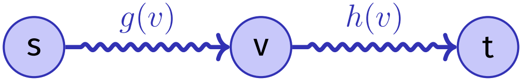
    $R \leftarrow R \cup \{v\}$; $M \leftarrow M - \{v\}$

return failure

Notation

Let $f(v)$ be the distance of a shortest path from s to t via v, thus

$$f(v) := \underbrace{\delta(s,v)}_{g(v)} + \underbrace{\delta(v,t)}_{h(v)}$$



let p be a shortest path from s to t.
It holds that $f(s) = \delta(s,t)$ and $f(v) = f(s)$ for all $v \in p$.
Let $\widehat{g}(v) := d[v]$ be an estimate of $g(v)$ in the algorithm above. It holds that $\hat{g}(v) \geq g(v)$.
$\widehat{h}(v)$ is an estimate of $h(v)$ with $\widehat{h}(v) \leq h(v)$.

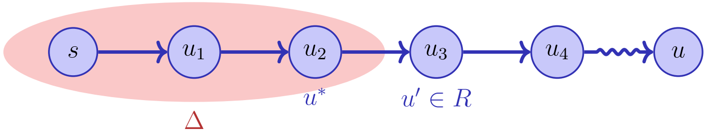
Why the Algorithm Works

Lemma 24

Let $u \in V$ and, at a time during the execution of the algorithm, $u \notin M$. Let p be a shortest path from s to u. Then there is a $u' \in p$ with $\widehat{g}(u') = g(u')$ and $u' \in R$.

The lemma states that there is always a node in the open set R with the minimal distance from s already computed and that belongs to a shortest path (if existing).

Illustration and Proof



Proof: If $s \in R$, then $\widehat{g}(s) = g(s) = 0$. Therefore, let $s \notin R$.

Let $p = \langle s = u_0, u_1, \ldots, u_k = u \rangle$ and $\Delta = \{u_i \in p, u_i \in M, \widehat{g}(u_i) = g(u_i)\}$.

$\Delta \neq \emptyset$, because $s \in \Delta$.

Let $m = \max\{i : u_i \in \Delta\}$, $u^* = u_m$. Then $u^* \neq u$, since $u \notin M$. Let $u' = u_{m+1}$.

1. $\widehat{g}(u') \leq \widehat{g}(u^*) + c(u^*, u')$ because u' has already been relaxed
2. $\widehat{g}(u^*) = g(u^*)$ (because $u^* \in \Delta$)
3. $\hat{g}(u') \geq g(u')$ (construction of $\hat{g}$)
4. $g(u') = g(u^*) + c(u^*, u')$ (because p optimal)

Therefore: $\widehat{g}(u') = g(u')$ and thus also $u' \in R$ because $u' \notin \Delta$. ■

Corollary

Corollary 25

If $\widehat{h}(u) \leq h(u)$ *for all* $u \in V$ *and A**- *Algorithmus has not yet terminated. The for each shortest path* p *from* s *t* t *there is some node* $u' \in p$ *with* $\hat{f}(u') \leq \delta(s,t) = f(t)$.

If there is a shortest path p from s to t, then there is always a node in the open set R that underestimates the overal distance and that is on the shortest path.

Proof of the Corollary

Proof:
From the lemma: $\exists u' \in p$ with $\widehat{g}(u') = g(u')$.
Therefore:

$$\begin{aligned}\widehat{f}(u') &= \widehat{g}(u') + \widehat{h}(u') \\ &= g(u') + \widehat{h}(u') \\ &\leq g(u') + h(u') = f(u')\end{aligned}$$

Because p is shortest path: $f(u') = \delta(s, t)$. ∎

Admissibility

Theorem 26

If there is a shortest path from s to t and $\hat{h}(u) \leq h(u) \; \forall \; u \in V$ then A terminates with $\hat{g}(t) = \delta(s,t)$*

Proof: If the algorithm terminates, then it termines with t with $f(t) = \widehat{g}(t) + 0 = g(t)$. That is because $\widehat{g}$ overestimates g at most and by the corollary above that algorithm always finds an element $v \in R$ with $f(v) \leq \delta(s,t)$.

The algorithm terminates in finitely many steps. For finite graphs the maximal number of relaxing steps is bounded.[42] ■

[42] For a δ-graph the maximum number of relaxing steps before R contains only nodes with $\hat{f}(s) > \delta(s,t)$ is limited as well. The exact argument can be found in the seminal article Hart, P. E.; Nilsson, N. J.; Raphael, B. (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths".

Revisiting nodes

- The A*-algorithm can re-insert nodes that had been extracted from R before.
- This can lead to suboptimal behavior (w.r.t. running time of the algorithm).
- If $\widehat{h}$, in addition to being admissible ($\widehat{h}(v) \leq h(v)$ for all $v \in V$), fulfils monotonicity, i.e. if for all $(u, u') \in E$:

$$\widehat{h}(u') \leq \widehat{h}(u) + c(u', u)$$

  then the A*-Algorithm is equivalent to the Dijsktra-algorithm with edge weights $\tilde{c}(u, v) = c(u, v) + \widehat{h}(u) - \widehat{h}(v)$, and no node is re-inserted into R.
- It is not always possible to find monotone heuristics.