Lec09-A*-search

Thursday, September 12, 2024 3:19 PM

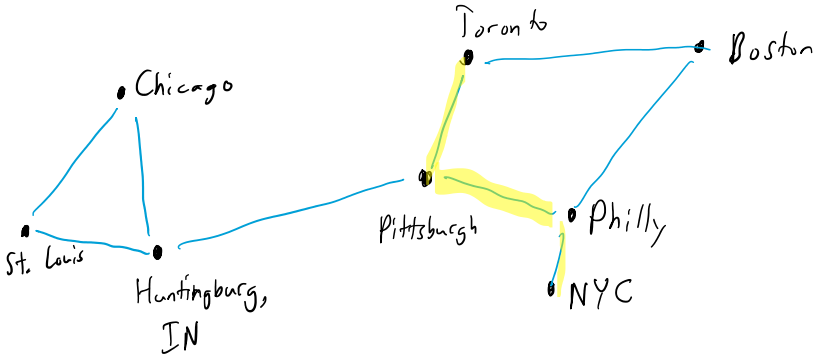
Can we do better if we specifically want the $s \to t$ shortest path?

Dijkstra gives all shortest paths from source, and we can stop after finishing t (not first reaching), but that's overkill.

A* algorithm Faster heuristic for finding shortest $s \to t$ path in graph with positive edge weights.

Dijkstra chooses next Edge by tentative distance from s $d[u] + \text{length}(u,v)$.

↳ could be exploring in opposite direction from t.



Pittsburgh to St Louis trip, but Dijkstra has to compute all shortest distances in the East first, which feels inefficient.

Problem: Dijkstra knows nothing about unreached nodes

Heuristic: Give an estimate $h(u)$ of distance from target as part of minimization

For railroad paths, perhaps Euclidean distance

A* alg:

- Maintain $d[u]$ = current best distance from s to u found so far
- Need function $h(u)$ = estimate of u to t distance
- Let $f(u) = d[u] + h(u)$ → estimate of best $s \to t$ distance through u so far

Modify Dijkstra to use $f(u)$ as key instead of $d[u]$ as key

Pseudocode:

tentative $s \to u$ distance — tentative parent — frontier — tentative $u \to t$ distance

for $u \in V$, $d[u] = \infty$, $p[u] = \emptyset$, $F = V$, $h[u]$, $f[u] = d[u] + h[u]$

$d[s] = 0$

~~$F =$ makeheap (V, d)~~

$F =$ makeheap (V, f)

while $F \neq \emptyset$:

  $u \leftarrow$ vertex in F with min ~~$d[u]$~~ $f[u]$ — always explore ~~next nearest node~~ node we think is a best path to t

  remove u from F

  for each neighbor v of u in F:

    if $d[u] + \text{length}(u, v) < d[v]$:

      $p[v] = u$

      $d[v] = d[u] + \text{length}(u, v)$

      $f[v] = d[v] + h[v]$

  } reduce tentative distance (reducekey of heap)

  return $d[u], p[u]$

Choice of $h(u)$:

Def. Let $h^*(u)$ be the real shortest dist from u to t.

A heuristic $h(u)$ is admissible if $h(u) \leq h^*(u)$ $\forall u$.

actual $u \to t$ distance on graph

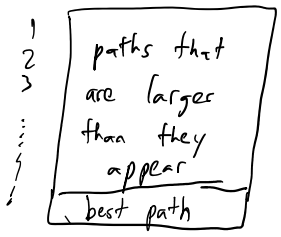
- When $h(u) = 0$ $\forall u$, $A^* =$ Dijkstra

Obviously admissible.

Theorem If $h(u)$ is admissible, then A^* is guaranteed to find an optimal route.

- Want $h(u)$ to be admissible
- Want to minimize $h^*(u) - h(u)$

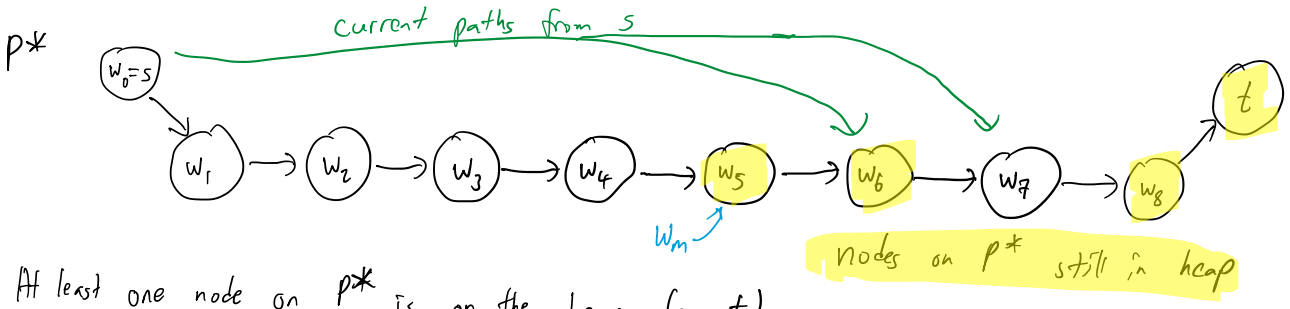
Intuition Guessed distances are always underestimates, so order of exploration is:



We will never search a path that is better than our guess, so once we have found an optimal path, all other paths will look (and be) worse.

proof. Suppose not, and let p^* be an optimal path.

Consider the state of nodes along p^* when t is the next node removed from heap.



At least one node on p^* is on the heap (e.g. t)

Let w_m be the first such node

Lemma: When each node $w_0, \ldots, w_{m-1}$ was last removed from heap, $d(w_i) = d^*(w_i)$

(proof by induction Base case: $d(w_0) = d(s) = 0 = d^*(s)$ ($d^*(u)$ is actual $s \to u$ dist)

Inductive step: assume true for $i-1$.

(i.e. when w_{i-1} was removed, $d(w_{i-1}) = d^*(w_{i-1})$)

After w_{i-1}'s removal, we updated neighbors, including w_i:

$$d(w) \leftarrow d(w_{i-1}) + \text{length}(w_{i-1}, w_i)$$
$$= d^*(w_{i-1}) + \text{length}(w_{i-1}, w_i)$$
$$= d^*(w_i) \quad \text{because } p^* \text{ is optimal.}$$

□

Corollary As we remove t, $d(w_m) = d^*(w_m)$.

(proof. When w_{m-1} was removed, it updated

$$d(w_m) \leftarrow d(w_{m-1}) + \text{length}(w_{m-1}, w_m)$$
$$= d^*(w_{m-1}) + \text{length}(w_{m-1}, w_m)$$
$$= d^*(w_m) \quad \text{because } p^* \text{ is optimal.}$$

Since optimal, $d(w_m)$ never gets smaller later, so done □

back to main proof: Let $w = w_m$. Since $f(t) = d(t) + h(t)$,

$f(t) = d(t) + h(t)$ ($h(t) \to 0$)

$$f(t) = d(t) + h(t)$$

$\leq f(w)$ b/c f is min of paths through frontiers

$= d(w) + h(w)$

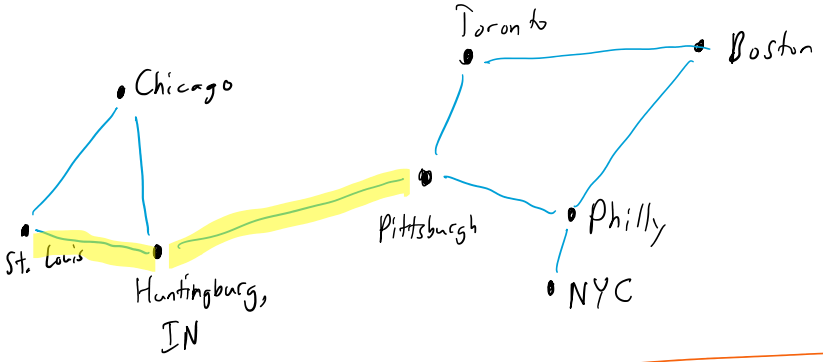
$= d^*(w) + h(w)$ since p^* is optimal

$\leq d^*(w) + h^*(w)$ by admissibility

$= \text{length}(p^*)$

$= \text{optimal}$.

$\Rightarrow d(t)$ is optimal when t is removed from heap



Sometimes, even when when start with a graph, need to transform to a more complicated graph to apply an alg.

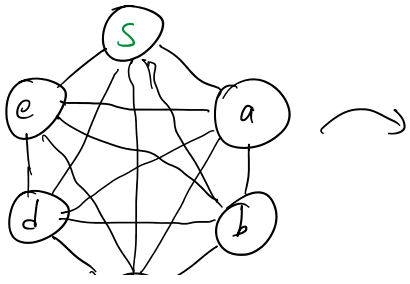
Traveling Saleesman Problem (non-Euclidean + non-symmetric) (returning back to starting city after visiting every place)

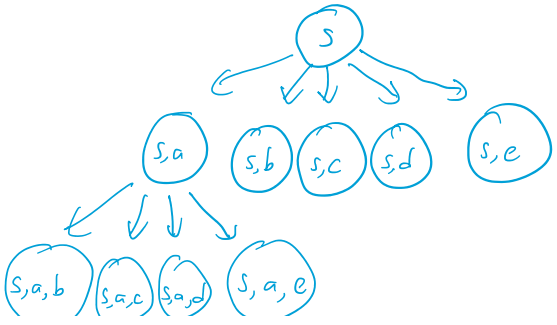
$\exists$ dist b/t any two cities but $dist(i,j) \neq d(j,i)$ (one-way train routes)

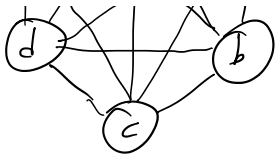
No triangle inequality. (so can't go back through visited city)

Convert graph to state graph that encodes partial tours

Graph with n nodes

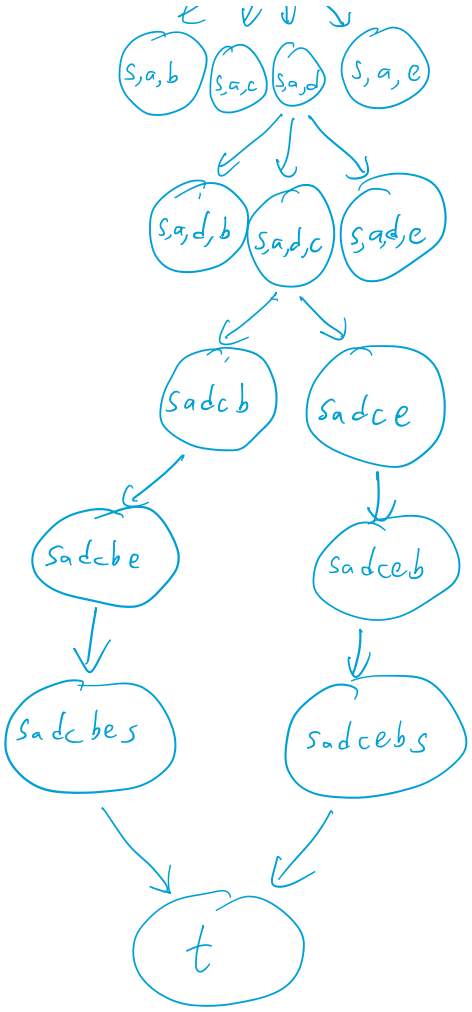






Convert traveling salesman problem to $s \to t$ shortest path problem

Can now use Dijkstra



Graph with $\Omega(n!)$ nodes

$O(n^n)$ nodes

To use A*, need good admissible heuristic for $h(a_1 \to \cdots \to a_k)$

Ex. 0

Ex. smallest unused out-edge of a_k

Ex. smallest out-edge of a_k + smallest in-edge of a_1.

Ex. length of shortest path $a_k \to a_1$ that avoids $a_2, \ldots, a_{k-1}$

Ex. cost (MST) on all nodes except $a_2, \ldots, a_{k-1}$

- We can use shortest paths for combinatorial problems by constructing state graph.
- A* incorporates heuristics to sometimes speed things up.
- Can guarantee optimality using heuristics w/ admissibility

- Can guarantee optimality using heuristics w/ admissibility

Summary of shortest paths

Alg	Runtime	Application
BFS	$O(\|V\|+\|E\|)$	unweighted edges
Dijkstra	$O(\|E\| \log \|V\|)$	positive edge weights
A*	possibly large	need heuristic $h(u)$
Bellman-Ford	$O(\|E\|\|V\|)$	arbitrary edge weights (incl. negative)

Algorithm design techniques

- Based on BFS/DFS (bipartite testing, topo sort)
- Greedy tree growing (Prim's, Dijkstra's)
- A* (design admissible heuristic; state graphs)