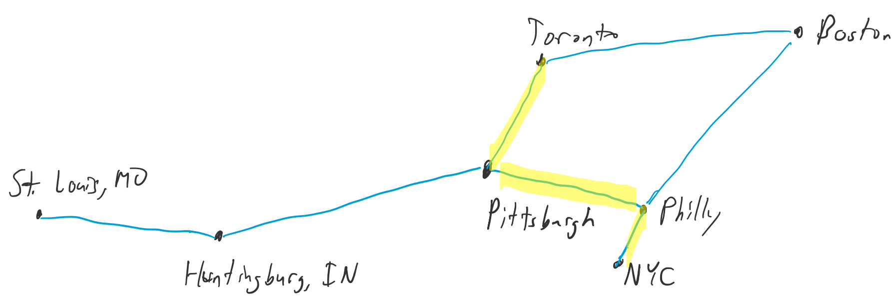
Lec10-A*-search

Tuesday, September 19, 2023 9:06 PM

Can we do better if we specifically want $s \to t$ shortest path?
Dijkstra gives all shortest paths from source, and we can stop after finishing t (not first reaching), but that's still overkill.

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

Dijkstra chooses nextEdge by tentative distance from s $d[u] + \text{length}(u,v)$,
↳ could be exploring in the opposite direction from t.



Pittsburg → 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 the minimization.

For driving 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.

Dijkstra uses $f'(u) = d(v) + \text{length}(v,u)$

Pseudocode: tentative distance tentative (parent) (frontier) $h[u]$ tentative dist to t $f[u] = d[u] + h[u]$

Pseudocode:

for $u \in V$, $d[u] = \infty$ (tentative distance from s), $p[u] = \emptyset$ (tentative (parent)), $F = V$ (frontier), $h[u]$ (tentative cost to t), $f[u] = d[u] + h[u]$

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

$F = makeHeap(V, f)$

$d[s] = 0$,

while $F \neq \emptyset$:

  $u \leftarrow$ vertex in F with min. ~~$d[u]$~~ $f[u]$ } deletemin(u) in heap

  remove u from F } always explore ~~next nearest node~~ node we think is on best path to t.

  if $u = t$, BREAK.

  for each neighbor v of u in F: } reduce tentative distances (reducekey of heap)

    if $d[u] + length(u,v) < d[v]$:

      $p[v] = u$

      $d[v] = d[u] + length(u,v)$

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

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

Choice of $h(u)$:

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

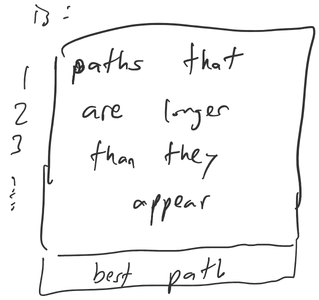
A heuristic $h(u)$ is **admissible** if $h(u) \leq h^*(u)$ $\forall u$.

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

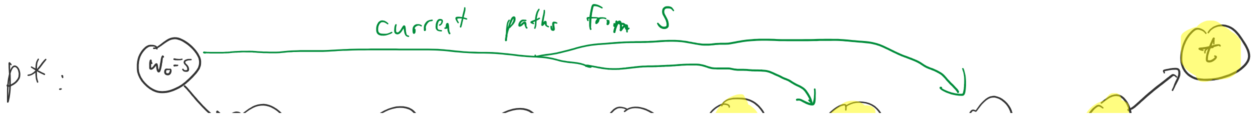


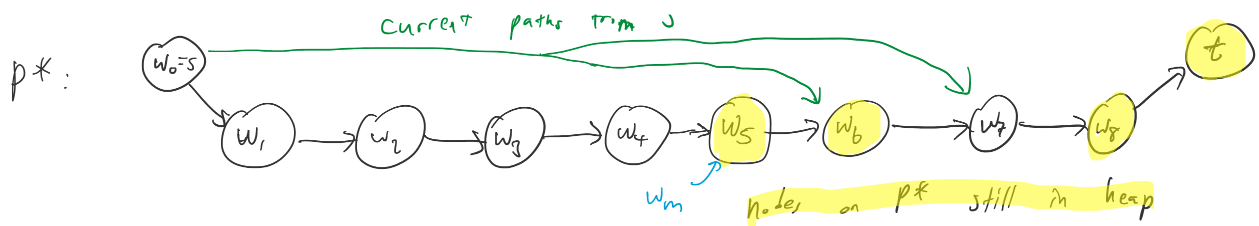
We will never search a path that is better than our guess, so once we have found an optimal path, all other paths will look (and be) worse.

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.

current paths from S





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 $w_0, \dots, w_{m-1}$ was last removed from the heap,
$d(w_i) = d^*(w_i)$

proof by induction: Base case: $d(w_0) = d(s) = 0 = d^*(s)$.
Inductive step: assume true for $i-1$.
(i.e. when w_{i-1} was removed, $d(w_{i-1}) = d^*(w_{i-1})$.)

($d^*[u]$ is the actual shortest dist $s \to u$)

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

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

Corollary: As we remove t, $d[w_m] = d^*[w_m]$

proof. By lemma, $d(w_{m-1}) = d^*(w_{m-1})$.
When w_{m-1} was removed, it updated

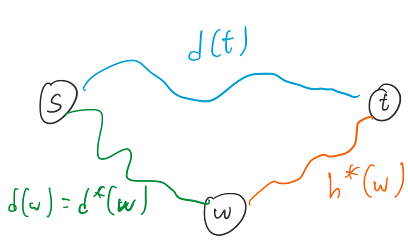
$$d(w_m) \leftarrow d(w_{m-1}) + \text{length}(w_{m-1}, w_m)$$
$$= d^*(w_{m-1}) + \text{length}(w_{m-1}, w_m)$$
$$= d^*(w_m) \quad \text{because } p^* \text{ is optimal.}$$

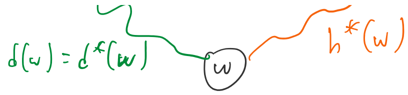
Since optimal, it never gets smaller, so done. $\square$

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

$$f(t) = d(t) + \cancel{h(t)}^{\,0} = d(t)$$
$$\leq f(w) \quad \text{b/c } f \text{ is min. of paths thru frontier}$$
$$= g(w) + h(w)$$
$$= g^*(w) + h(w) \quad \text{since } p^* \text{ is optimal}$$

by admissibility





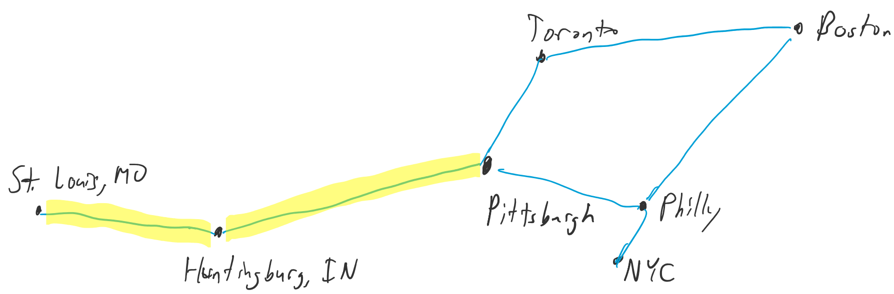
$= g^*(w) + h(w)$ since p^* is optimal

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

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

$=$ optimal.

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



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

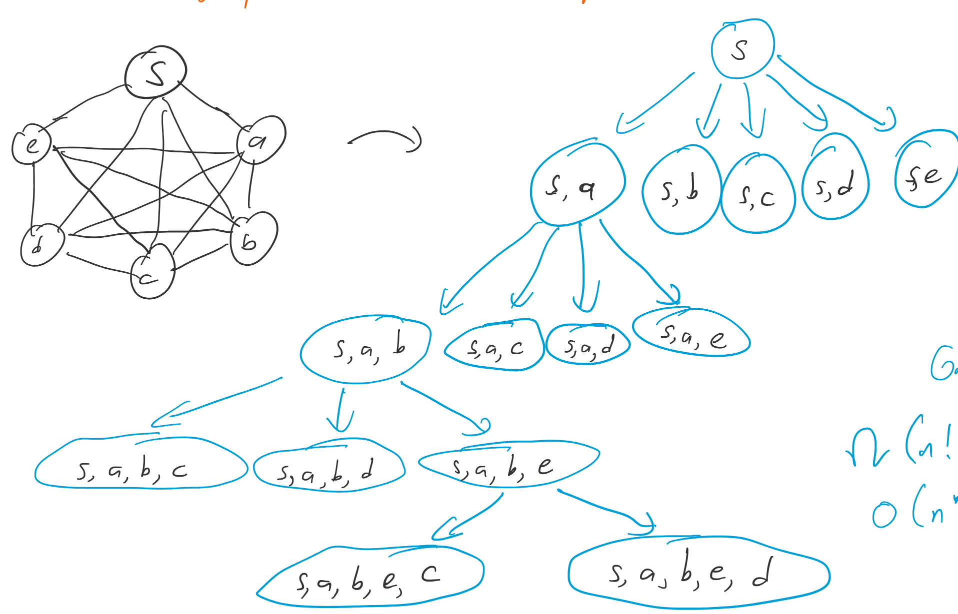
Traveling Salesman Problem (non-Euclidean & non-symmetric) (returning back to starting city)

$\exists$ dist b/t any two cities but $\text{dist}(i,j) \neq d(j,i)$.

Also no triangle inequality.

Convert graph to state graph, that encodes partial tours

Graph with n nodes



Graph with $\Omega(n!)$ nodes

$O(n^n)$ nodes.

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 $a_2, \dots, a_{k-1}$.

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

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

Summary of Shortest Paths

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

Algorithm design techniques

- Based on BFS/DFS (bipartite testing, top. sort)
- Greedy tree growing (Prim's, Dijkstra's)
- A* (design admissible heuristic: TSP)