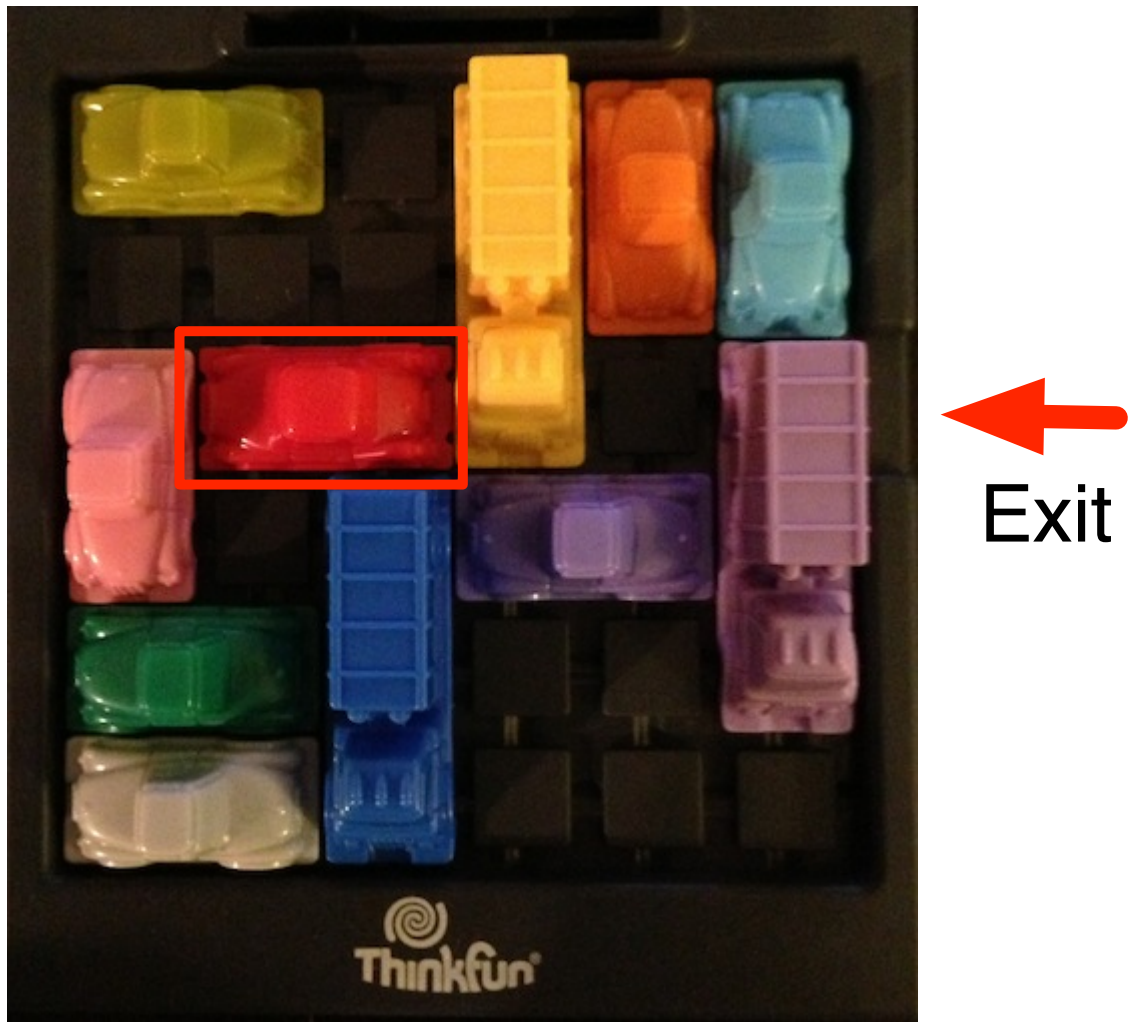
Applications of Shortest Paths and A*

Slides by Carl Kingsford

Feb. 7, 2014

Rush Hour Puzzle

Puzzle game by ThinkFun. Object: drive the red car out of the exit with as few moves as possible.




Thinkfun


Thinkfun


Thinkfun


Thinkfun


Thinkfun


Thinkfun


Thinkfun

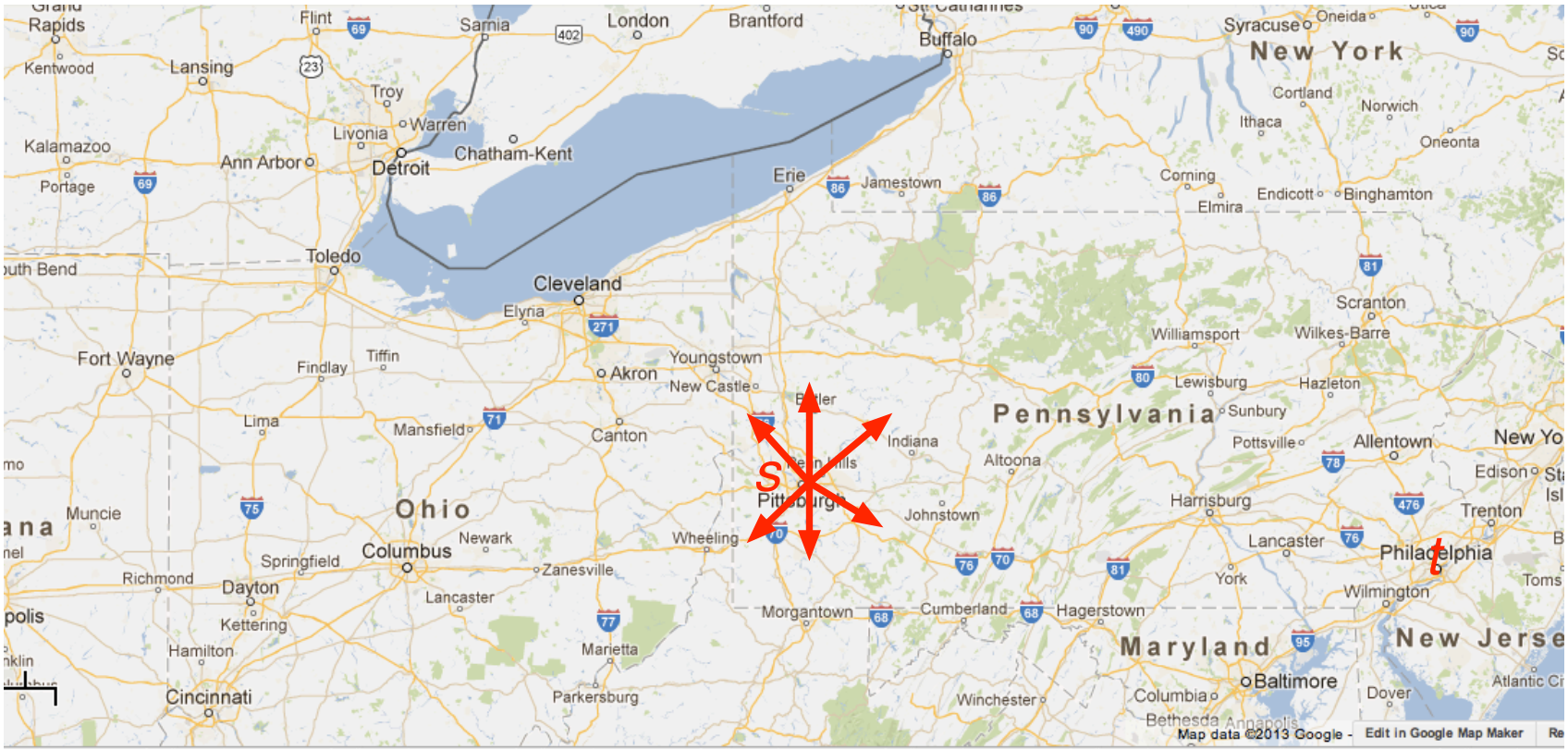
Let's make it more interesting

- Real game has cost 1 for every move, no matter which car or how far the car moves.
- **Extension:** Cost $c(\mathrm{car})$ for moving a particular car 1 space. E.g. $c(\mathrm{yellow}) = 3$.
- Find lowest cost solution.

How can we find a low cost solution?

Shortest Paths in the Rush Hour Graph

More efficient shortest paths



When looking for the shortest path from Pittsburgh to Philadelphia, it would be strange to start out going west.

Can we use this to find the path faster?

Using a heuristic for shortest paths

Dijkstra's algorithm assumes it knows nothing about nodes it hasn't reached during the algorithm.

Suppose instead we have $h(u)$ which is an estimate of the distance from node u to t.

What's a plausible choice for $h(u)$ if we were implementing a driving direction application?

Using a heuristic for shortest paths

Dijkstra's algorithm assumes it knows nothing about nodes it hasn't reached during the algorithm.

Suppose instead we have $h(u)$ which is an estimate of the distance from node u to t.

What's a plausible choice for $h(u)$ if we were implementing a driving direction application?

$h(u) =$ *the distance from* u *to* t *"as the crow flies."*

Using a heuristic for shortest paths

Dijkstra's algorithm assumes it knows nothing about nodes it hasn't reached during the algorithm.

Suppose instead we have $h(u)$ which is an estimate of the distance from node u to t.

What's a plausible choice for $h(u)$ if we were implementing a driving direction application?

$h(u) =$ *the distance from* u *to* t *"as the crow flies."*

How can we use $h(u)$ to speed up our search for a shortest path to some destination t?

A* algorithm

Maintain two values for every visited node:

- $g(u) =$ best distance from s to u found so far.
- $f(u) = g(u) + h(u) =$ estimate of the length of the best path from s to t through u.

Idea: Run a Dijkstra-like algorithm using $f(u)$ as the key

A*

```
 1: g[s] ← 0
 2: f[s] ← g[s] + h[s]
 3: Heap ← MakeHeap((s,f[s])) # heap key is f[s]
 4: repeat
 5:     u ← DeleteMin(Heap) # Expand node with minimum f[u]
 6:     if u = goal then return
 7:     for v ∈ Neighbors(u) do
 8:         if g[u] + d(u,v) < g[v] then
 9:             parent[v] ← u
10:             g[v] ← g[u] + d(u,v)
11:             f[v] ← g[v] + h(v)
12:             if v ∉ Heap then
13:                 Insert(Heap, v, f[v])
14:             else
15:                 ReduceKey(Heap, v, f[v]) # Sift up for new key
16: until Heap is empty
```

Choice of $h(u)$

Definition (Admissible). Let $h^*(u)$ be the real shortest distance from u to t. A heuristic $h(u)$ is *admissible* if $h(u) \leq h^*(u)$ for all u.

- When $h(u) = 0$ for all u: A* is equivalent to Dijkstra's algorithm.
- This choice of $h(u)$ is obviously admissible.

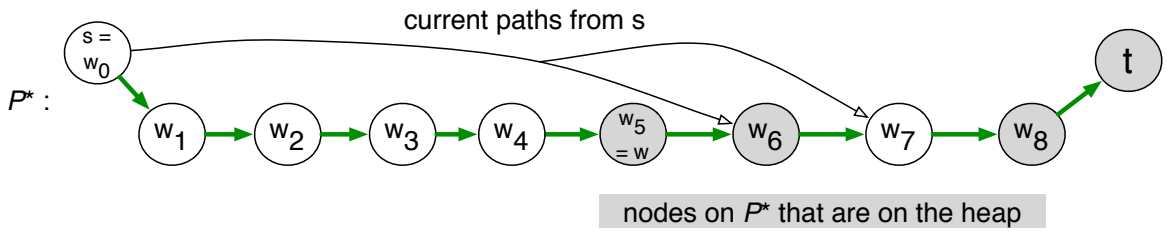
Theorem. *If $h(u)$ is admissible, then A* is guaranteed to find an optimal route to the destination t.*

- Want $h(u)$ to be admissible.
- Want it to be as close to $h^*(u)$ as possible.
- (Price-is-right criteria)

Admissible $\Longrightarrow$ Optimal

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

Proof. Suppose not, and let P^* be an optimal path. Consider the state of the nodes on P^* just before node t is removed from the heap (line 6):

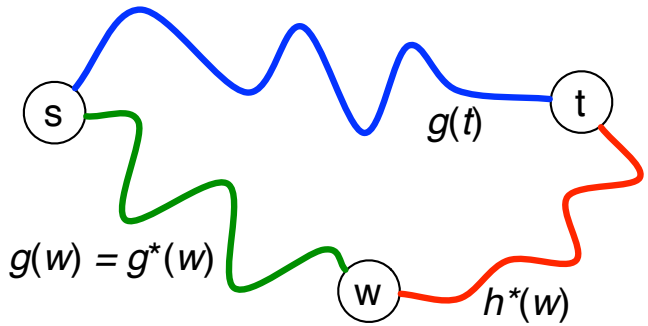


At least one node on P^* must be on the heap (since $t \in P^*$).
Let w be the *first* such node on P^*.

Claim: The current distance $g(w)$ must equal the optimal distance $g^*(w)$:

- s must have been expanded, adding the optimal path to w_1 and adding w_1 to the heap.
- Since w_1 is no longer on the heap, it must have been expanded, adding w_2 to the heap and assigning $g(w_2)$ to the length of the optimum path $s - w_1 - w_2$, and so on.

Then, since $f(t) := g(t) + h(t)$, we have:



$$
\begin{aligned}
g(t) + h(t) &= g(t) \\
&\leq f(w) && (*) \\
&= g(w) + h(w) \\
&= g^*(w) + h(w) && (**) \\
&\leq g^*(w) + h^*(w) && (***) \\
&= \text{length}(P^*) \\
&= \text{optimal}
\end{aligned}
$$

Hence, the distance $g(t)$ we compute for t is optimal. $\square$

$(*)$ b/c $f(t)$ is the minimum of all things on the frontier.

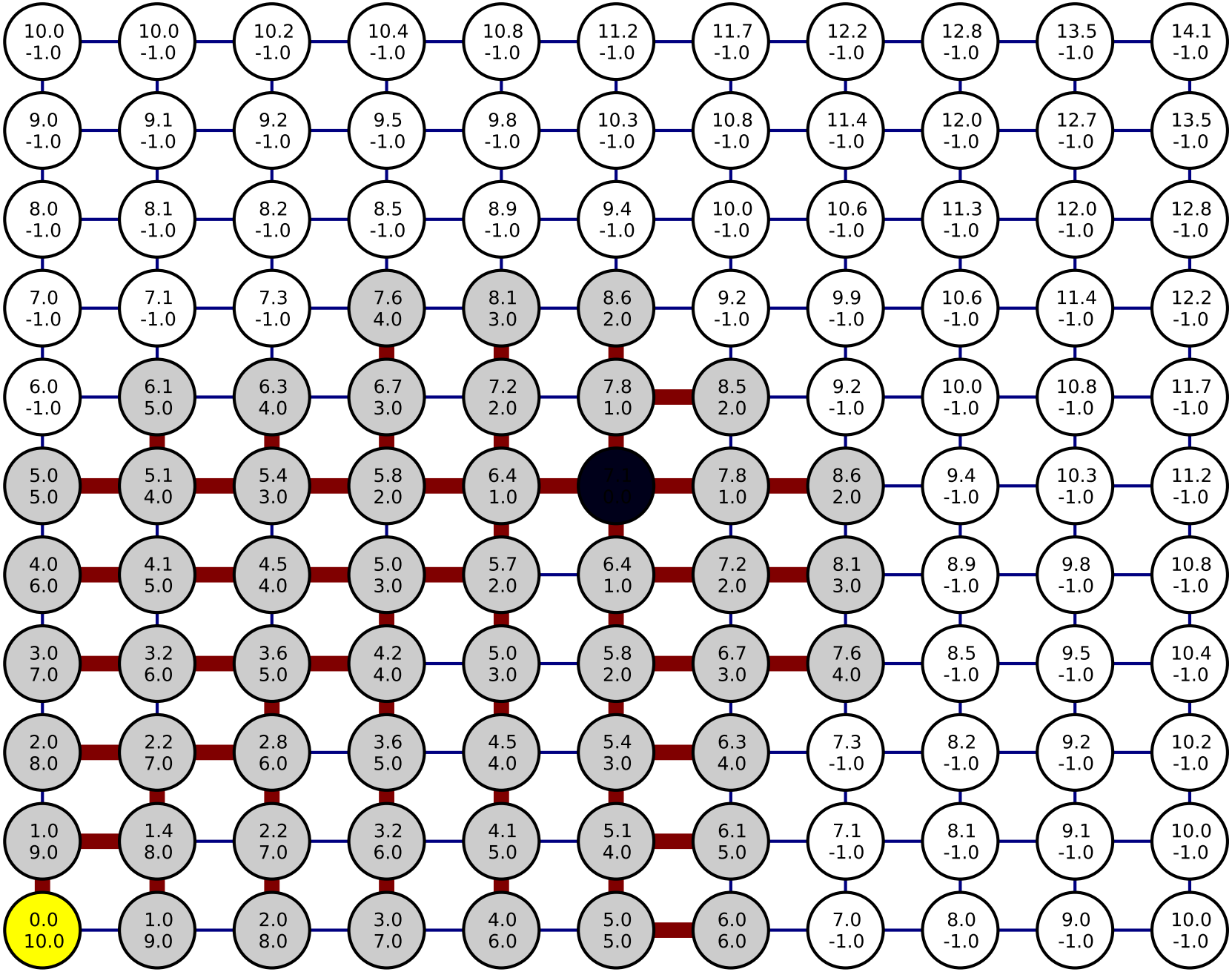
$(**)$ b/c P^* is optimal.

$(***)$ by admissibility.

Example run of A* on grid graph

(See full run in gridastar.pdf)

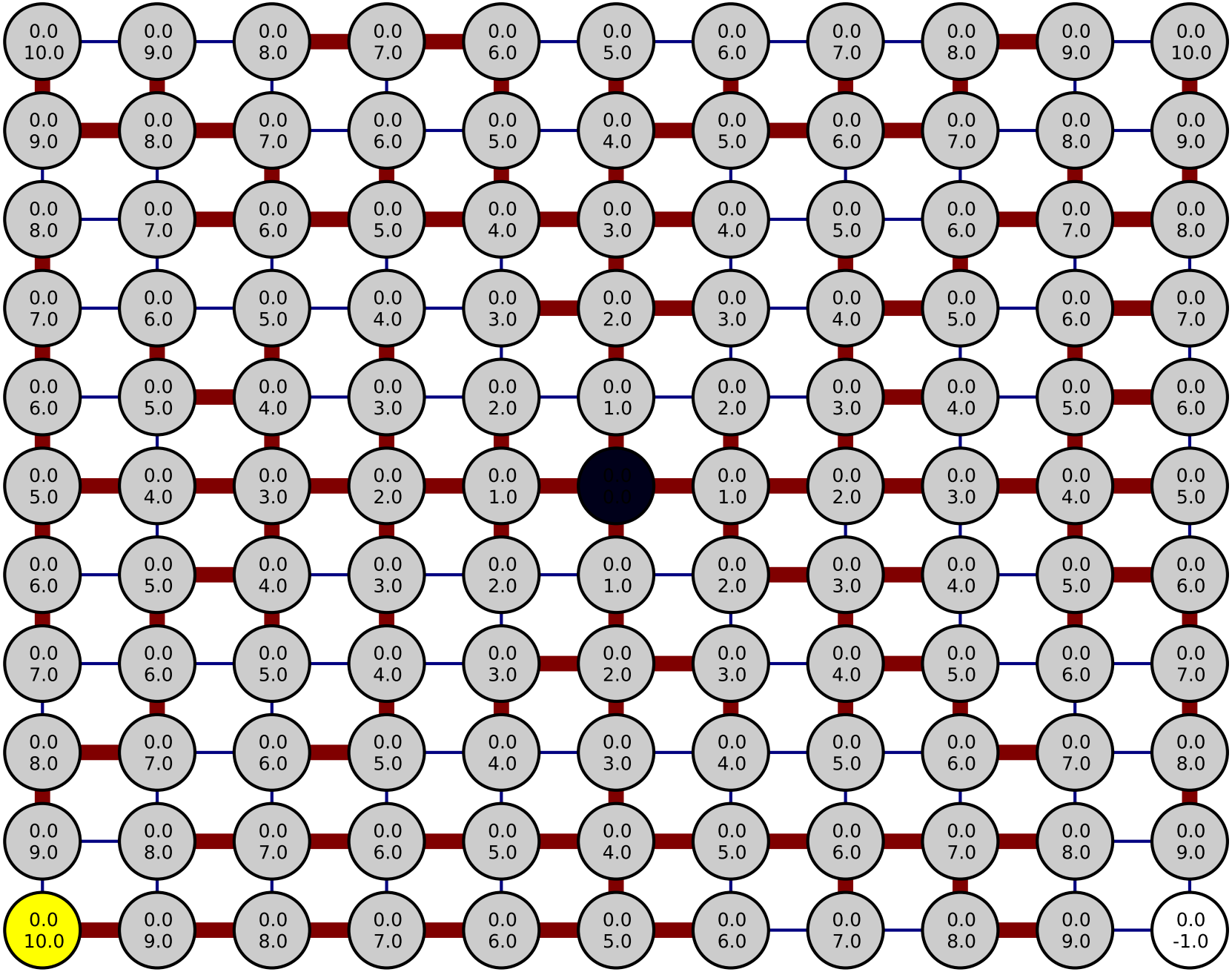
- top number = $h(u)$, which is the straight-line Euclidean distance
- bottom number = $g(u)$, best distance to u so far computed


10.0 -1.0
10.0 -1.0
10.2 -1.0
10.4 -1.0
10.8 -1.0
11.2 -1.0
11.7 -1.0
12.2 -1.0
12.8 -1.0
13.5 -1.0
14.1 -1.0
9.0 -1.0
9.1 -1.0
9.2 -1.0
9.5 -1.0
9.8 -1.0
10.3 -1.0
10.8 -1.0
11.4 -1.0
12.0 -1.0
12.7 -1.0
13.5 -1.0
8.0 -1.0
8.1 -1.0
8.2 -1.0
8.5 -1.0
8.9 -1.0
9.4 -1.0
10.0 -1.0
10.6 -1.0
11.3 -1.0
12.0 -1.0
12.8 -1.0
7.0 -1.0
7.1 -1.0
7.3 -1.0
7.6 4.0
8.1 3.0
8.6 2.0
9.2 -1.0
9.9 -1.0
10.6 -1.0
11.4 -1.0
12.2 -1.0
6.0 -1.0
6.1 5.0
6.3 4.0
6.7 3.0
7.2 2.0
7.8 1.0
8.5 2.0
9.2 -1.0
10.0 -1.0
10.8 -1.0
11.7 -1.0
5.0 5.0
5.1 4.0
5.4 3.0
5.8 2.0
6.4 1.0
7.1 0.0
7.8 1.0
8.6 2.0
9.4 -1.0
10.3 -1.0
11.2 -1.0
4.0 6.0
4.1 5.0
4.5 4.0
5.0 3.0
5.7 2.0
6.4 1.0
7.2 2.0
8.1 3.0
8.9 -1.0
9.8 -1.0
10.8 -1.0
3.0 7.0
3.2 6.0
3.6 5.0
4.2 4.0
5.0 3.0
5.8 2.0
6.7 3.0
7.6 4.0
8.5 -1.0
9.5 -1.0
10.4 -1.0
2.0 8.0
2.2 7.0
2.8 6.0
3.6 5.0
4.5 4.0
5.4 3.0
6.3 4.0
7.3 -1.0
8.2 -1.0
9.2 -1.0
10.2 -1.0
1.0 9.0
1.4 8.0
2.2 7.0
3.2 6.0
4.1 5.0
5.1 4.0
6.1 5.0
7.1 -1.0
8.1 -1.0
9.1 -1.0
10.0 -1.0
0.0 10.0
1.0 9.0
2.0 8.0
3.0 7.0
4.0 6.0
5.0 5.0
6.0 6.0
7.0 -1.0
8.0 -1.0
9.0 -1.0
10.0 -1.0

Example run of Dijkstra on the same grid graph

(See full run in griddijk.pdf)

- top number $= h(u) = 0$
- bottom number $= g(u)$, best distance to u so far computed


0.0 10.0
0.0 9.0
0.0 8.0
0.0 7.0
0.0 6.0
0.0 5.0
0.0 6.0
0.0 7.0
0.0 8.0
0.0 9.0
0.0 10.0
0.0 9.0
0.0 8.0
0.0 7.0
0.0 6.0
0.0 5.0
0.0 4.0
0.0 5.0
0.0 6.0
0.0 7.0
0.0 8.0
0.0 9.0
0.0 8.0
0.0 7.0
0.0 6.0
0.0 5.0
0.0 4.0
0.0 3.0
0.0 4.0
0.0 5.0
0.0 6.0
0.0 7.0
0.0 8.0
0.0 7.0
0.0 6.0
0.0 5.0
0.0 4.0
0.0 3.0
0.0 2.0
0.0 3.0
0.0 4.0
0.0 5.0
0.0 6.0
0.0 7.0
0.0 6.0
0.0 5.0
0.0 4.0
0.0 3.0
0.0 2.0
0.0 1.0
0.0 2.0
0.0 3.0
0.0 4.0
0.0 5.0
0.0 6.0
0.0 5.0
0.0 4.0
0.0 3.0
0.0 2.0
0.0 1.0
0.0 0.0
0.0 1.0
0.0 2.0
0.0 3.0
0.0 4.0
0.0 5.0
0.0 6.0
0.0 5.0
0.0 4.0
0.0 3.0
0.0 2.0
0.0 1.0
0.0 2.0
0.0 3.0
0.0 4.0
0.0 5.0
0.0 6.0
0.0 7.0
0.0 6.0
0.0 5.0
0.0 4.0
0.0 3.0
0.0 2.0
0.0 3.0
0.0 4.0
0.0 5.0
0.0 6.0
0.0 7.0
0.0 8.0
0.0 7.0
0.0 6.0
0.0 5.0
0.0 4.0
0.0 3.0
0.0 4.0
0.0 5.0
0.0 6.0
0.0 7.0
0.0 8.0
0.0 9.0
0.0 8.0
0.0 7.0
0.0 6.0
0.0 5.0
0.0 4.0
0.0 5.0
0.0 6.0
0.0 7.0
0.0 8.0
0.0 9.0
0.0 10.0
0.0 9.0
0.0 8.0
0.0 7.0
0.0 6.0
0.0 5.0
0.0 6.0
0.0 7.0
0.0 8.0
0.0 9.0
0.0 -1.0

Another application of A*

Traveling Salesman Problem

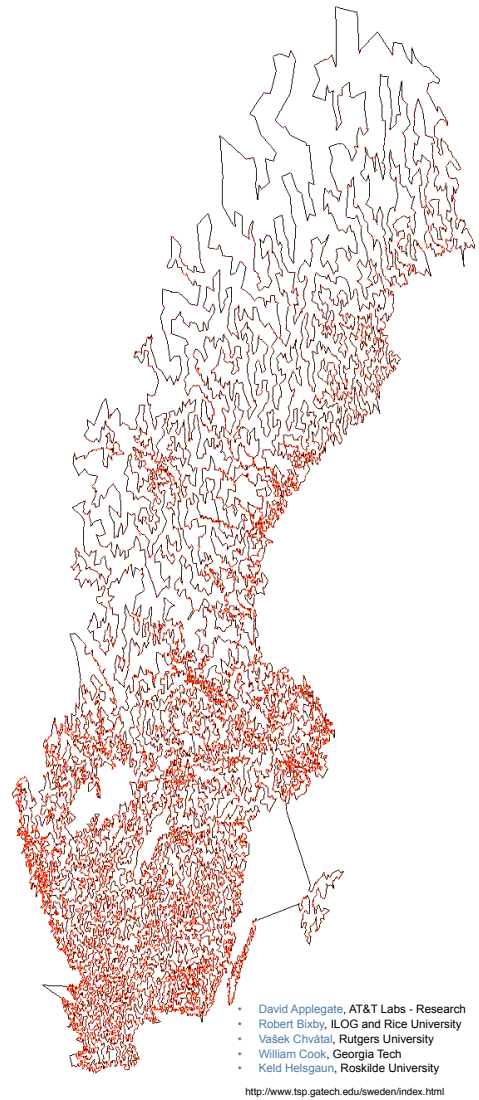
Traveling Salesman Problem. Given n cities, and distances $d(i, j)$ between each pair of cities, find the shortest route that visits each city exactly once.

Notes:

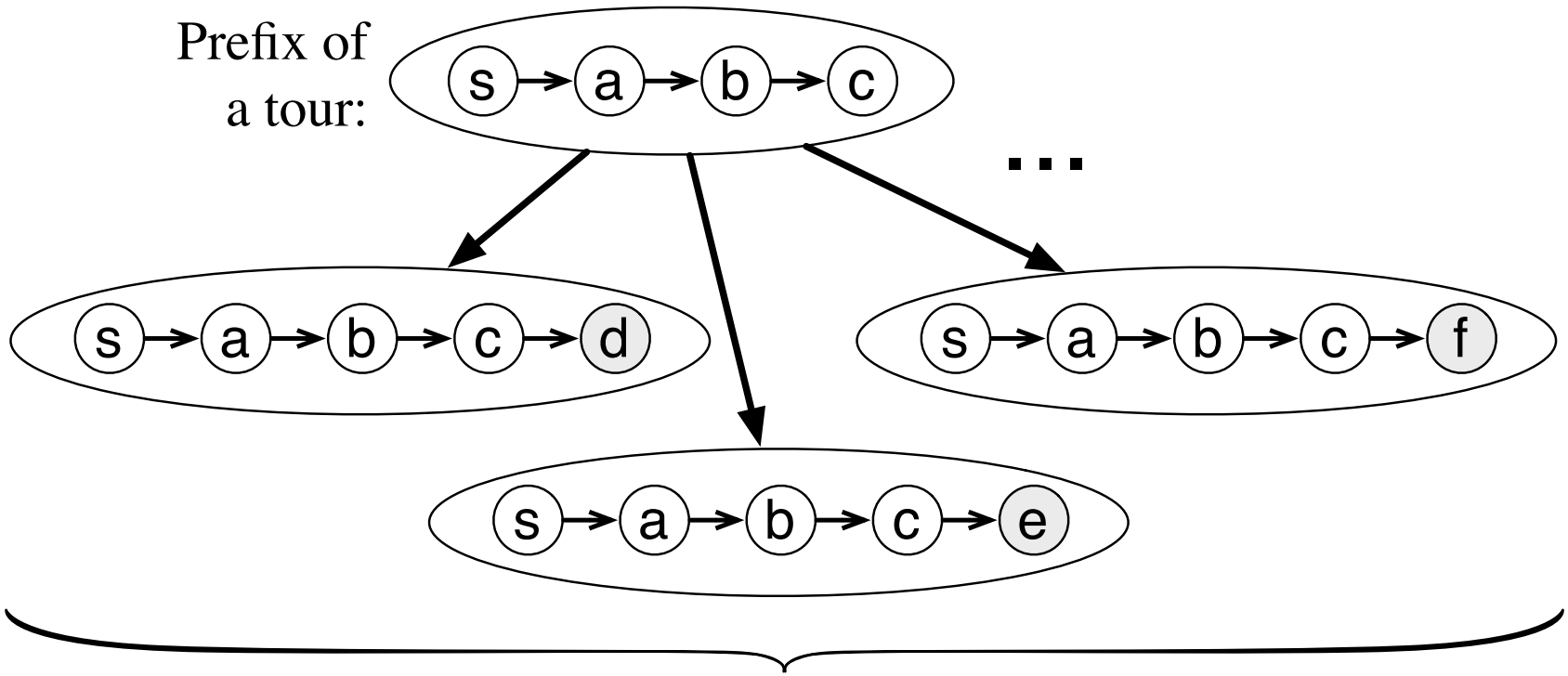
- We have a distance between every pair of cities.
- In this version, $d(i, j)$ doesn't have to equal $d(j, i)$.
- And the distances don't have to obey the triangle inequality ($d(i, j) \leq d(i, k) + d(k, j)$ for all i, j, k).

TSP large instance

- TSP visiting 24,978 (all) cities in Sweden.
- Solved by David Applegate, Robert Bixby, Vašek Chvátal, William Cook, and Keld Helsgaun
- `http://www.tsp.gatech.edu/sweden/index.html`
- Lots more cool TSP at `http://www.tsp.gatech.edu/`



Traveling Salesman State Graph



Admissible heuristics for TSP

What's a good admissible $h(a_1 \to \cdots \to a_k)$?

- 0

Admissible heuristics for TSP

What's a good admissible $h(a_1 \to \cdots \to a_k)$?

- 0
- length of the smallest unused edge leaving a_k.

Admissible heuristics for TSP

What's a good admissible $h(a_1 \to \cdots \to a_k)$?

- 0
- length of the smallest unused edge leaving a_k.
- length of smallest unused edge leaving a_k + length of smallest unused edge entering a_1.

Admissible heuristics for TSP

What's a good admissible $h(a_1 \to \cdots \to a_k)$?

- 0
- length of the smallest unused edge leaving a_k.
- length of smallest unused edge leaving a_k + length of smallest unused edge entering a_1.
- length of the shortest path from a_k to a_1 that doesn't use any nodes in $a_2, \ldots, a_{k-1}$.

Admissible heuristics for TSP

What's a good admissible $h(a_1 \to \cdots \to a_k)$?

- 0
- length of the smallest unused edge leaving a_k.
- length of smallest unused edge leaving a_k + length of smallest unused edge entering a_1.
- length of the shortest path from a_k to a_1 that doesn't use any nodes in $a_2, \ldots, a_{k-1}$.
- length of the minimum spanning tree on all nodes except $a_2, \ldots, a_{k-1}$

Admissible heuristics for TSP

What's a good admissible $h(a_1 \to \cdots \to a_k)$?

- 0
- length of the smallest unused edge leaving a_k.
- length of smallest unused edge leaving a_k + length of smallest unused edge entering a_1.
- length of the shortest path from a_k to a_1 that doesn't use any nodes in $a_2, \ldots, a_{k-1}$.
- length of the minimum spanning tree on all nodes except $a_2, \ldots, a_{k-1}$: a path from a_k to a_1 is a MST, so MST must be of less cost than the TSP completion.

Conclusion

- Shortest path can be used to solve a lot of combinatorial problems.
- A* is a slight extension that let's us incorporate heuristic information.
- With admissible heuristics, we can still guarantee finding the optimal solution.

Summary of Shortest Paths

For a graph $G = (V, E)$:

Algorithm	**Runtime**	**Application**
BFS	$O(\|V\| + \|E\|)$	edge weights all the same
Dijkstra's	$O(\|E\| \log \|V\|)$	positive edge weights
A*	possibly large	have heuristic $h(u)$
Bellman-Ford	$O(\|E\|\|V\|)$	arbitrary edge weights

Algorithmic design techniques

- Based on BFS or DFS (bipartite testing, topological sort)
- Greedy tree growing (Prim's, Dijkstra's)
- A* (design an admissible heuristic: TSP)
- Next: A preview of dynamic programming (Bellman-Ford)