

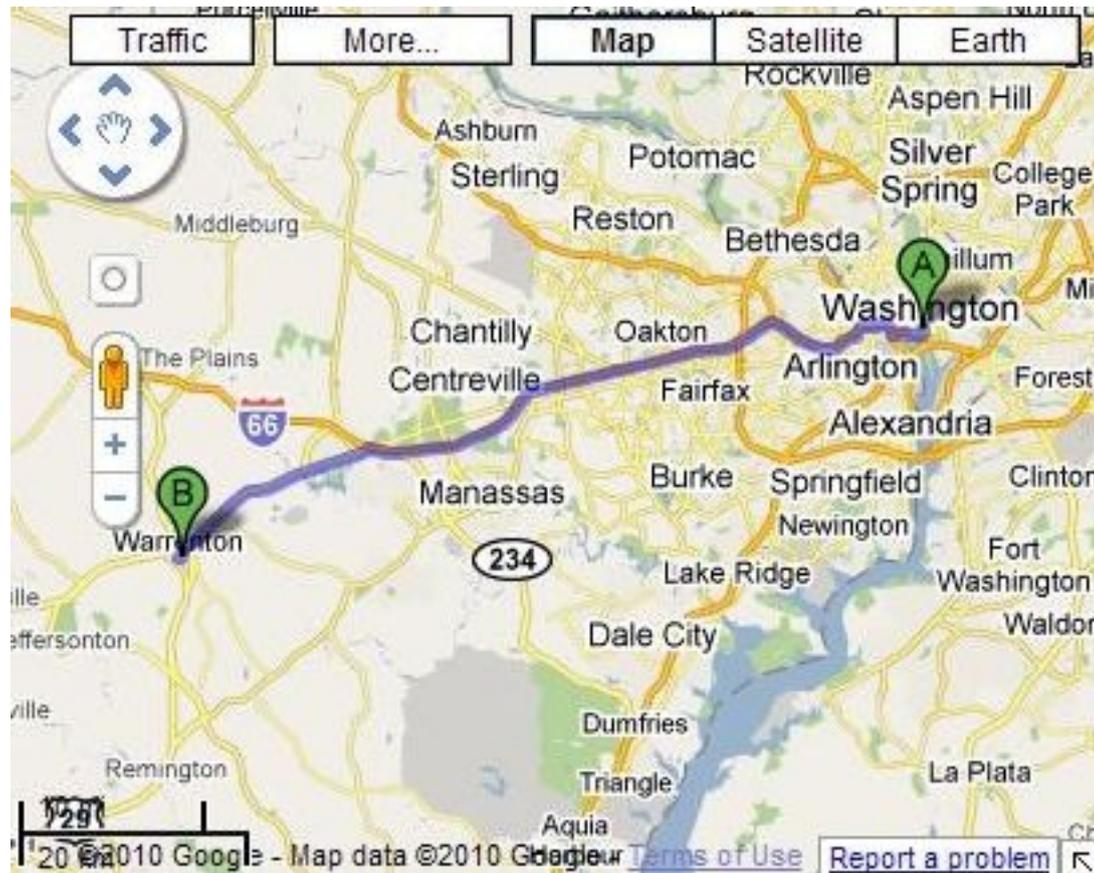
# 6.006 - Introduction to Algorithms

## Lecture 17:

### Heuristics for Faster Graph Search

# Linear time is too slow...

- Google Maps:  $\sim 10^{10}$  locations,  $10^{11}$  edges
- Dijkstra's would take  $\Theta(1 \text{ minute})$

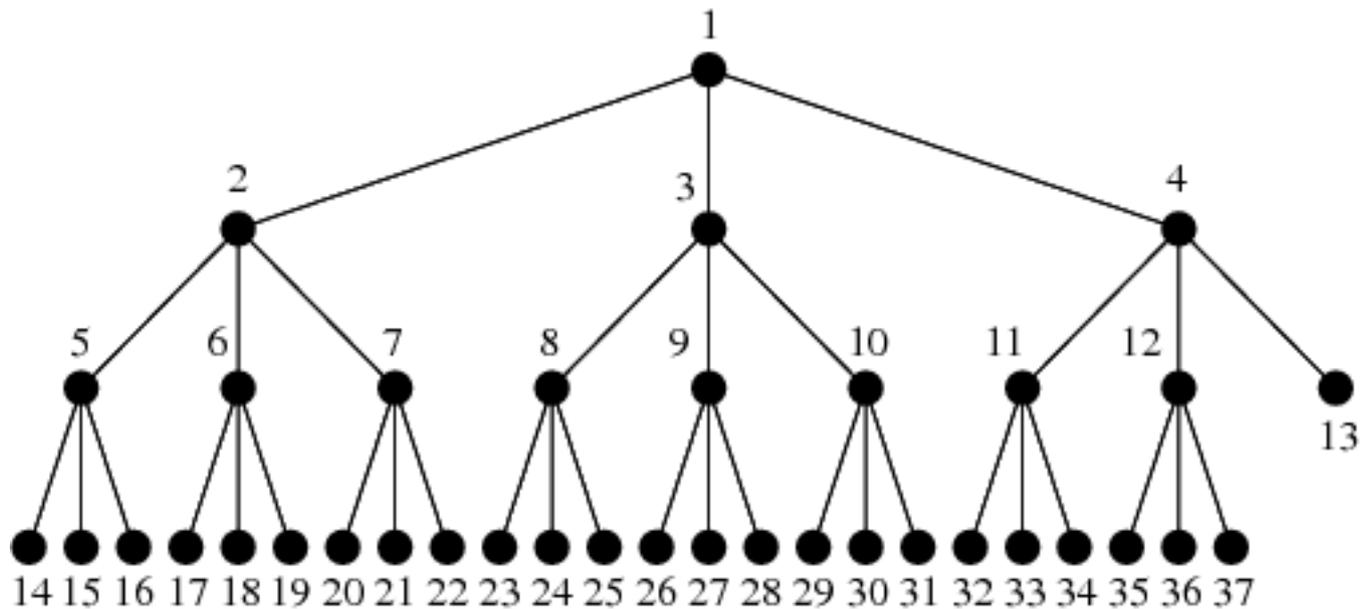


# Today's goals

- Develop heuristics for shortest path searches
  - Preserve correctness
  - Improve runtime in practice, not in theory
- Consider special classes of graphs:
  - Random graphs
  - Planar-weighted graphs

# Part 1: “random” graphs

- Every vertex has  $d$  random neighbors
- Consider the neighborhood of a vertex  $s$ 
  - Number of vertices at distance 1:  $d$
  - Number at distance 2:  $\sim d^2$
  - ...number at distance  $k$ :  $\sim d^k$



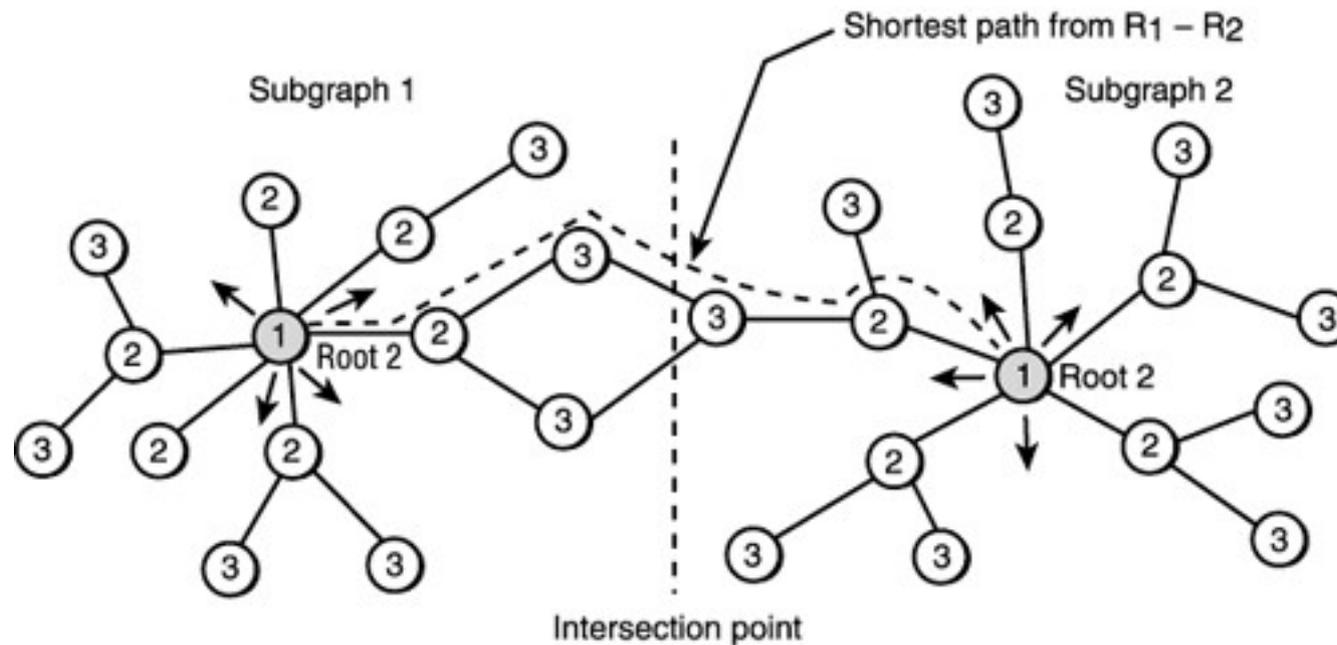
# BFS in random graphs

- $G$  is a random graph ( $n$  vertices, degree  $d$ )
- Suppose we search for a path from  $s$  to  $t$  in  $G$ 
  - Almost all vertices are at levels  $\sim \log_d n$
  - Almost all time spent at the last levels
- How can we improve our runtime?

# Bidirectional BFS

- **Idea:** instead of running a BFS from  $s$  to  $t$ , run BFS from  $s$  to  $t$  and from  $t$  to  $s$  simultaneously
  - For each level  $i$ :
    - Compute vertices at distance  $i$  from  $s$
    - Compute vertices at distance  $i$  from  $t$
  - Stop when a vertex  $v$  has been found from both  $s$  and  $t$
  - Shortest path from  $s$  to  $t$  runs through  $v$

# Example of bidirectional BFS



# Proof of correctness

- If shortest path from  $s$  to  $t$  is of length  $2k$ , then middle vertex  $v_k$  appears in both level  $ks$
- If shortest path is of length  $2k+1$ , then vertex  $v_{k+1}$  appears in  $s$ -level  $k+1$  and  $t$ -level  $k$
- Is this too easy?

# “Analysis” on random graphs

- Bidirectional BFS expands  $(\log_d n) / 2$  levels, instead of  $\log_d n$ 
  - Explores about  $\sqrt{n}$  vertices
  - Graph search in sublinear time!
- Performs well on many non-random graphs

# Bidirectional Dijkstra

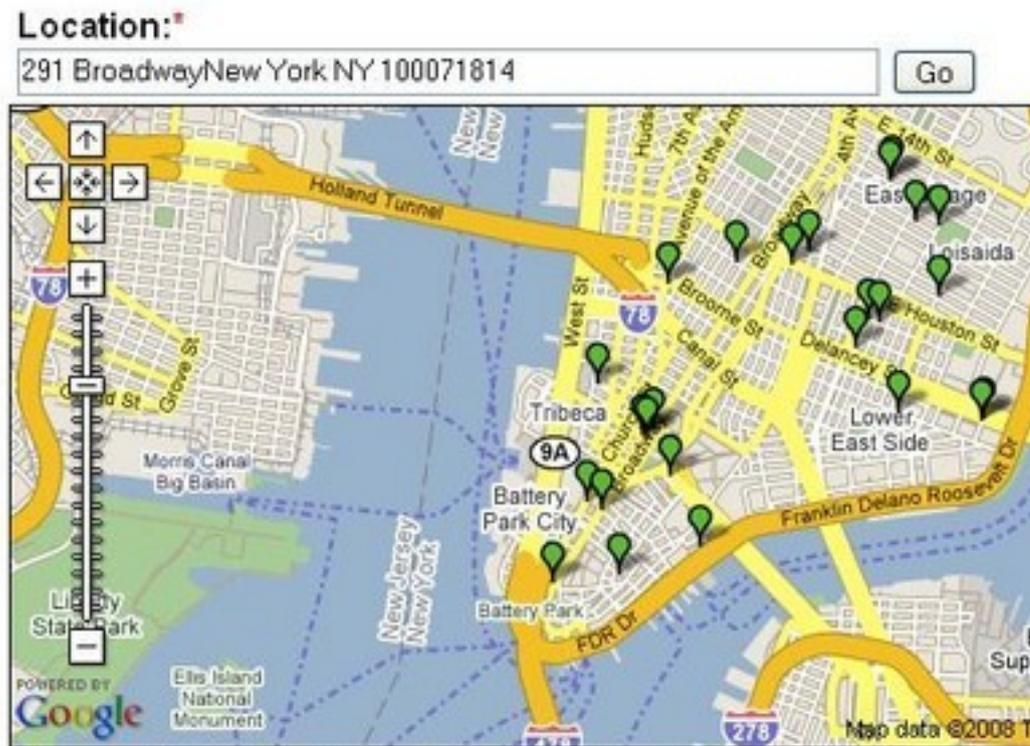
- Run Dijkstra simultaneously forwards from  $s$  and backwards to  $t$
- Keep vertices in two min-heaps:
  - First sorted by distance from  $s$
  - Second sorted by distance to  $t$
- Pop the smaller of the two minimums
  - From  $s$  heap: add it to a set  $S$
  - From  $t$  heap: add it to  $T$
- Repeat till we add a vertex  $v$  to both sets

# Subtleties in bidirectional Dijkstra

- The shortest path from  $s$  to  $t$  does not necessarily run through the vertex  $v$ ...
  - It goes from something in  $S$  to something in  $T$
- Loop over every edge from a vertex  $x$  in  $S$  to a vertex  $y$  in  $T$ 
  - Find paths with lengths  $d(s, x) + l(x, y) + d(y, t)$
  - If any of these paths is shorter than the path through  $v$  ( $d(s, v) + d(v, t)$ ), return it instead

# Part 2: planar-weighted graphs

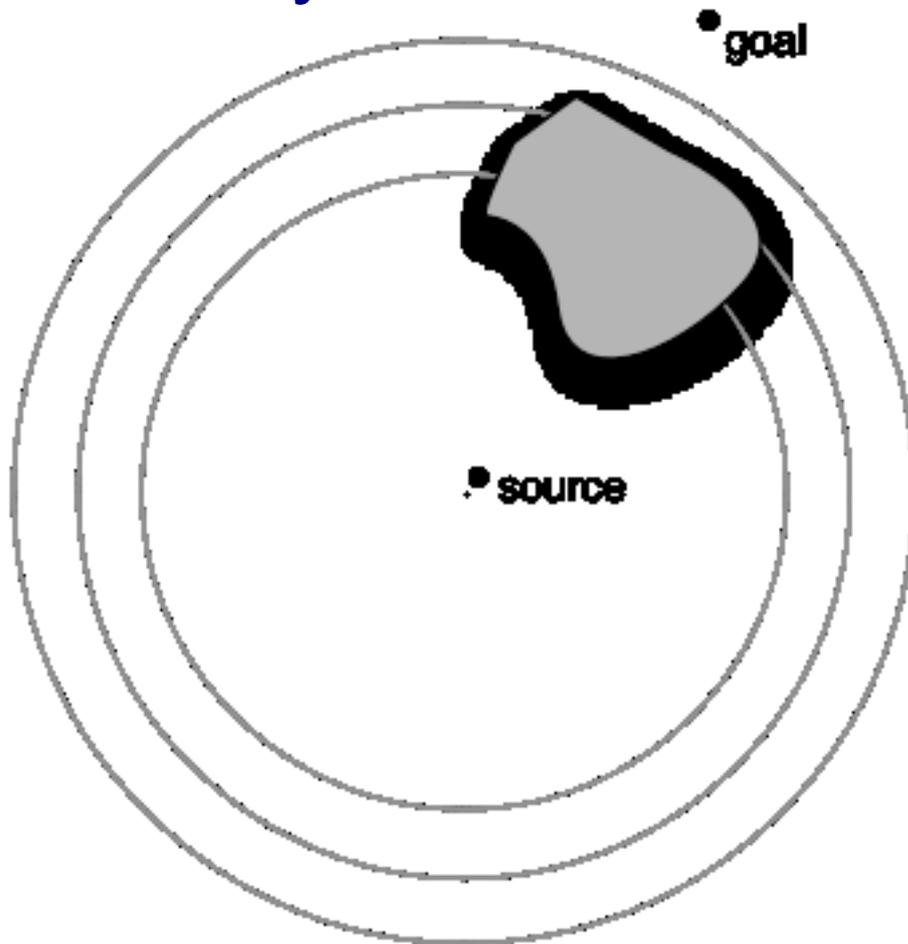
- In a planar-weighted graph, vertices are points
- Edge length  $l(u, v)$  is the distance from  $u$  to  $v$
- We've seen this before:



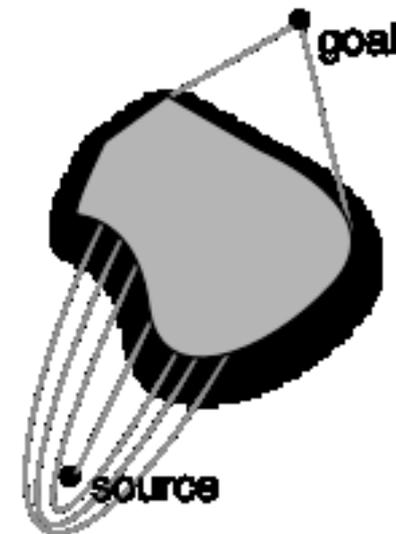
The map is limited to 100 locations for performance reasons.

# Dijkstra on planar-weighted graphs

- In reality:



- In an ideal world:



# Goal-directed search: A\*

- **Idea:** use extra information to guide search from  $s$  to  $t$
- Assign each vertex  $v$  a potential  $\lambda(v)$ 
  - $t$  should have potential  $\lambda(t) = 0$
  - Vertices close to  $t$  should have low potential
- Try to search toward low potential
  - Modify edge costs:  $l'(u, v) = l(u, v) - \lambda(u) + \lambda(v)$
  - Run Dijkstra?

# Edge modification preserves paths

- New edge costs:  $l'(u, v) = l(u, v) - \lambda(u) + \lambda(v)$
- Claim: the shortest path from  $u$  to  $v$  is preserved by edge modification

- Let  $(u, v_1, v_2, \dots, v_k, v)$  be a path from  $u$  to  $v$

- New path length:

$$l'(u, v_1) + l'(v_1, v_2) + \dots + l'(v_k, v)$$

$$= l(u, v_1) - \lambda(u) + \lambda(v_1) + l(v_1, v_2) - \lambda(v_1) + \lambda(v_2) + \dots + l(v_k, v) - \lambda(v_k) + \lambda(v)$$

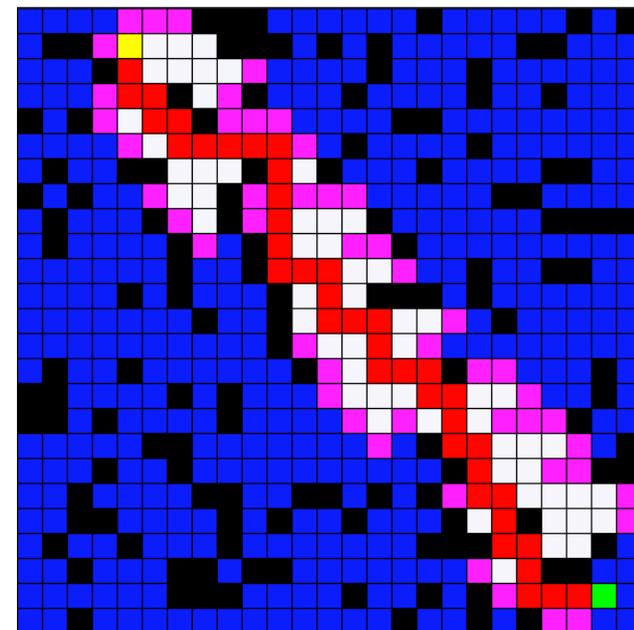
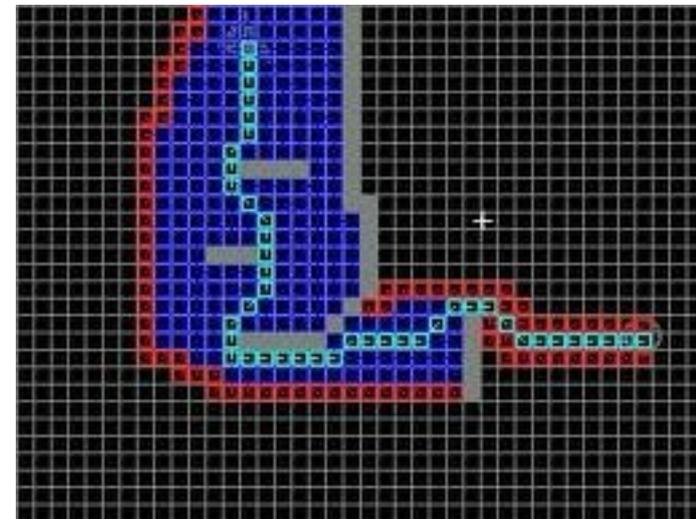
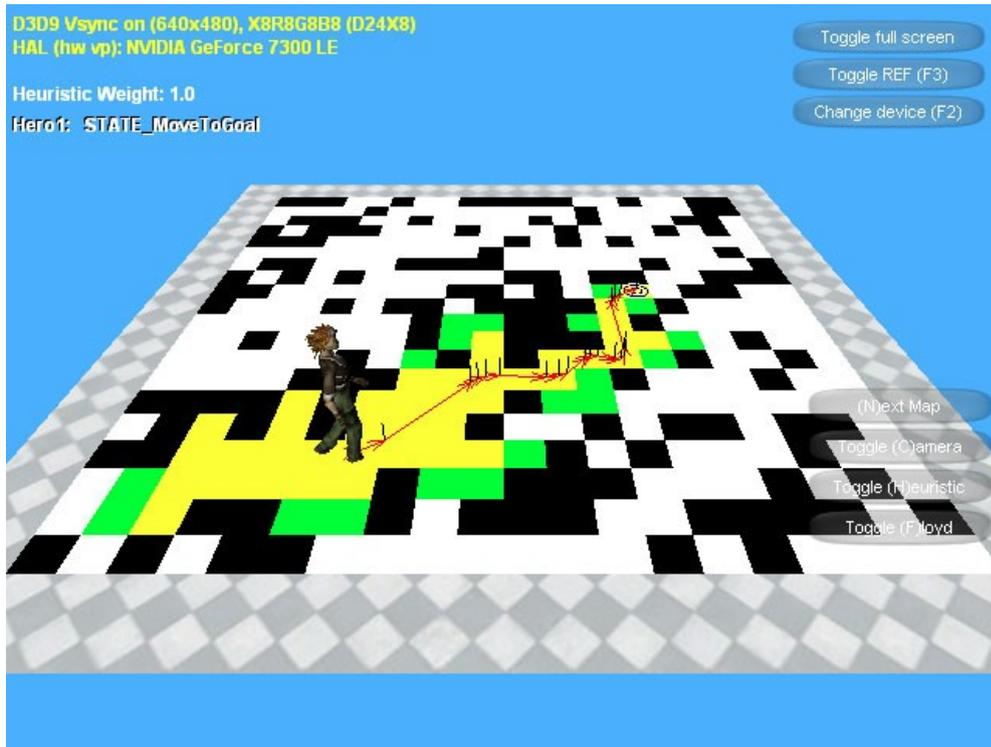
$$= [l(u, v_1) + l(v_1, v_2) + \dots + l(v_k, v)] - \lambda(u) + \lambda(v)$$

- New path length = old path length  $- \lambda(u) + \lambda(v)$

# Consistent heuristics

- Edge modification preserves paths
- We can use Dijkstra if  $l'(u, v) \geq 0$  for all  $u, v$ 
  - As long as  $l(u, v) - \lambda(u) + \lambda(v) \geq 0$
- How to choose  $\lambda(u)$ ?
  - Suppose graph is planar-weighted
  - Use distance to  $t$  as potential:  $\lambda(u) = d(u, t)$
  - Triangle inequality:  $l(u, v) + d(v, t) \geq d(u, t)$
- Other graphs – other potentials

# Results of A\*



A\* has been called one of the top ten algorithms of the last century!

# Other ideas to speed up search...

- Precompute shortest paths for some pairs...
- “Incremental”: use data from prior searches...
- Only return approximate shortest paths...
- ...