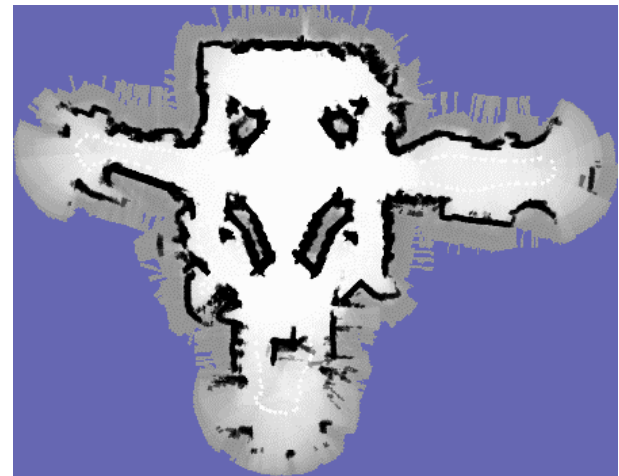


# ***Navigation and Metric Path Planning Part II***

October 7, 2008



Minerva tour guide robot (CMU):  
Gave tours in Smithsonian's National Museum of History



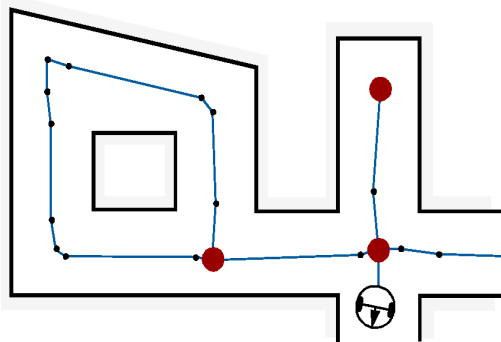
Example of Minerva's occupancy  
map used for navigation

# Path Planning Summary (so far)

1. Construct configuration space (by growing obstacles)
2. Select representation: either graph (sometimes called “roadmap”) or cell decomposition (or potential fields -- we’ll study these next)

- Graph (“roadmap”) construction

- Identify a set of routes within the free space

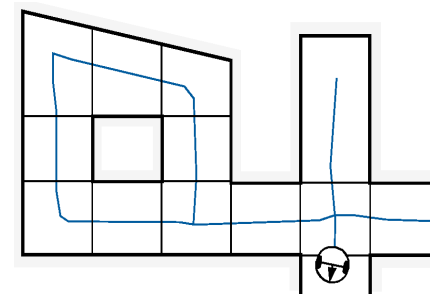


- Alternative approaches:

- Meadow maps
- Generalized Voronoi graphs
- Visibility graphs

- Cell decomposition

- Discriminate between free and occupied cells



- Variants:

- Regular grids
- Quadtree grids (also called “adaptive” and “variable cell” decomposition in your text)
- Exact cell decomposition

# Path Planning Summary (so far)

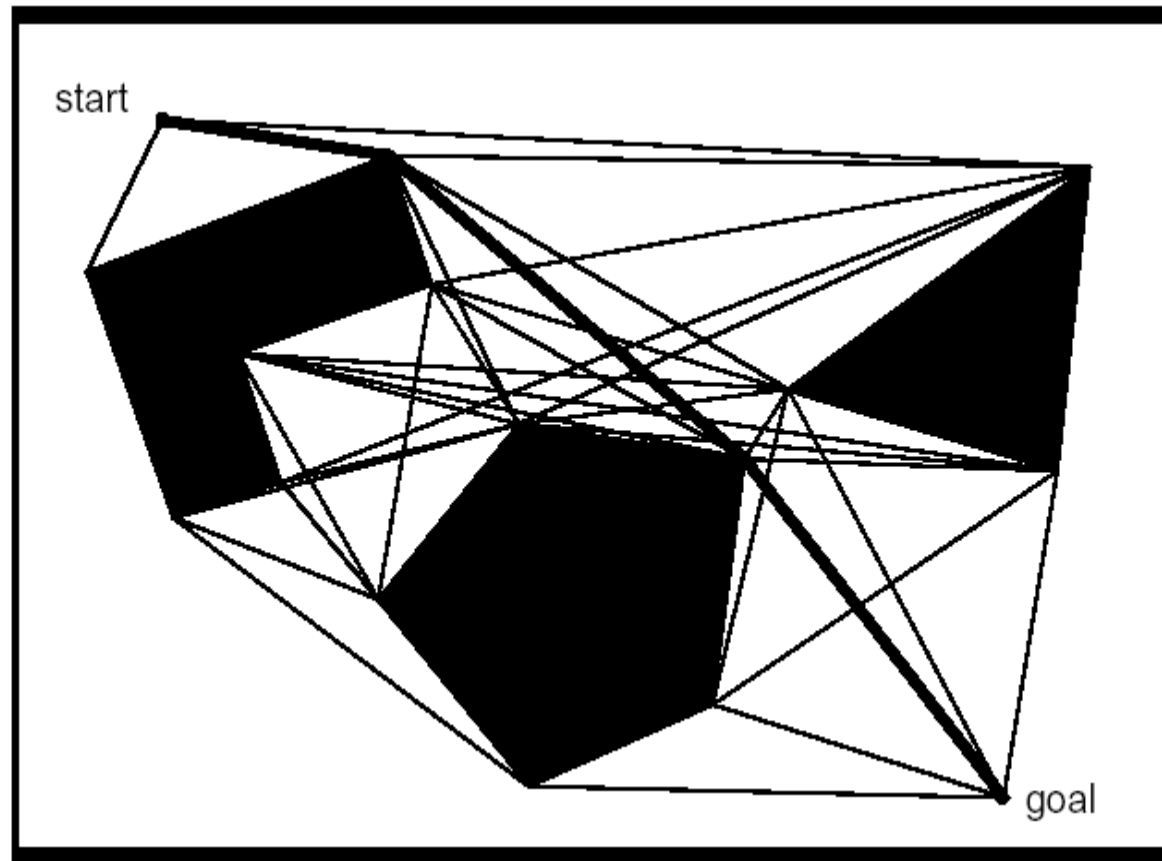
---

## 3. Plan path

- Common algorithms:
  - A\*: Typically used for graph-based methods
  - Wavefront path planning (which your text also calls “NF1” and “grassfire”):  
Typically used in cellular decomposition methods

# Additional Graph-Based Representation: Visibility Graph

---

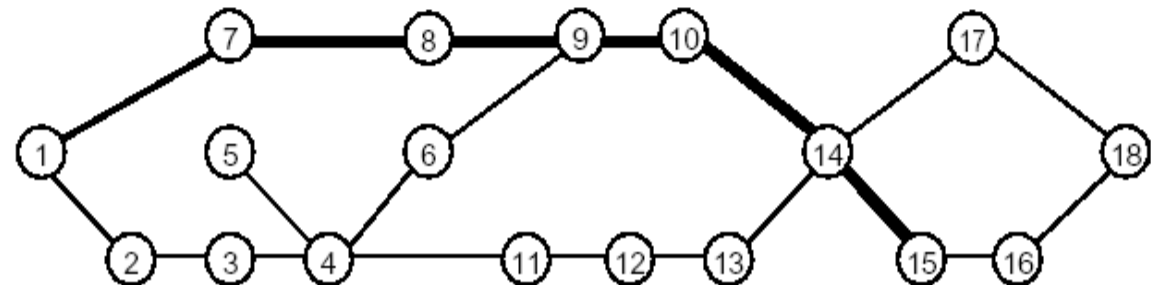
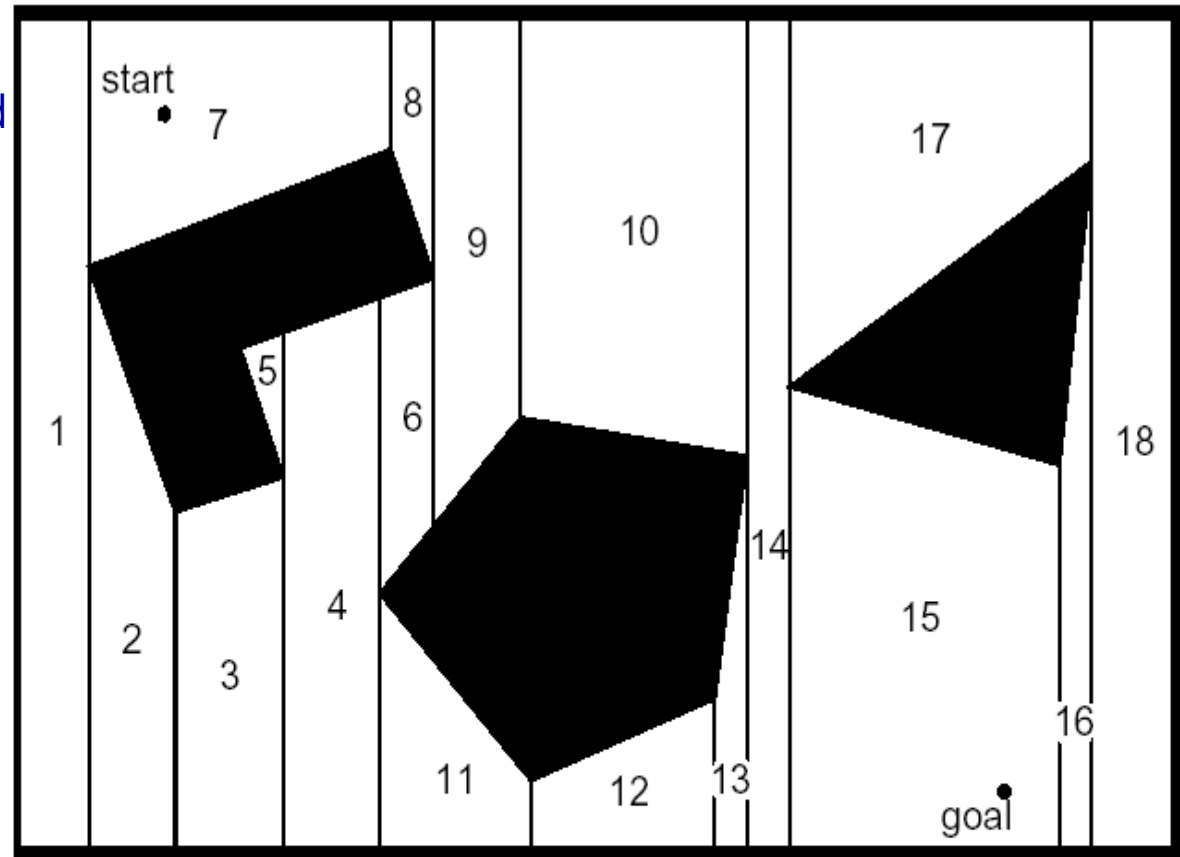


- **Approach:** Connect all vertices that are “visible” to each other
  - Advantage: Can generate optimal shortest paths (based on path length)
  - Disadvantage: Can cause robot to move too closely to obstacles (solution: grow obstacles even more, to give more open space between robot and obstacle)

# Additional Cell Decomposition: Exact Cell Decomposition (Similar to Meadow Map)

Approach:

- Divide space into simple, connected regions called **cells**
- Determine which open cells are adjacent and construct a **connectivity graph**
- Find cells in which the initial and goal configuration (state) lie and search for a path in the connectivity graph to join them.
- From the sequence of cells found with an appropriate search algorithm, compute a path within each cell.
  - e.g. passing through the midpoints of cell boundaries or by sequence of wall following movements.



# Algorithms

---

- For Path planning
  - A\* for relational graphs
  - Wavefront for operating directly on regular grids

# A\* Search Algorithm

---

- Similar to breadth-first: at each point in the time the planner can only “see” its node and 1 set of nodes “in front”
- Idea is to rate the choices, choose the best one first, throw away any choices whenever you can:

$$f^*(n) = g^*(n) + h^*(n) \quad // \text{ '*' means these are estimates}$$

where:

- $f^*(n)$  is the “goodness” of the path from Start to  $n$
- $g^*(n)$  is the “cost” of going from the Start to node  $n$
- $h^*(n)$  is the cost of going from  $n$  to the Goal
  - $h$  is for “heuristic function”, because must have a way of guessing the cost of  $n$  to Goal since can’t see the path between  $n$  and the Goal

# A\* Heuristic Function

---

$$f^*(n) = g^*(n) + h^*(n)$$

- $g^*(n)$  is easy: just sum up the path costs to  $n$
- $h^*(n)$  is tricky
  - But path planning requires an *a priori* map
  - Metric path planning requires a METRIC *a priori* map
  - Therefore, know the distance between Initial and Goal nodes, just not the optimal way to get there
  - $h^*(n)$  = distance between  $n$  and Goal

# Estimating $h(n)$

---

- Must ensure that  $h^*(n)$  is never greater than  $h(n)$
- Admissibility condition:
  - Must always underestimate remaining cost to reach goal
- Easy way to estimate:
  - Use Euclidian (straight line) distance
  - Straight line will always be shortest path
  - Actual path may be longer, but admissibility condition still holds

# Class Exercise

---

Compute optimal path from A-city to B-city

Straight-line distance to B-city from:

A-city: 366

B-city: 0

F-city: 176

O-city: 380

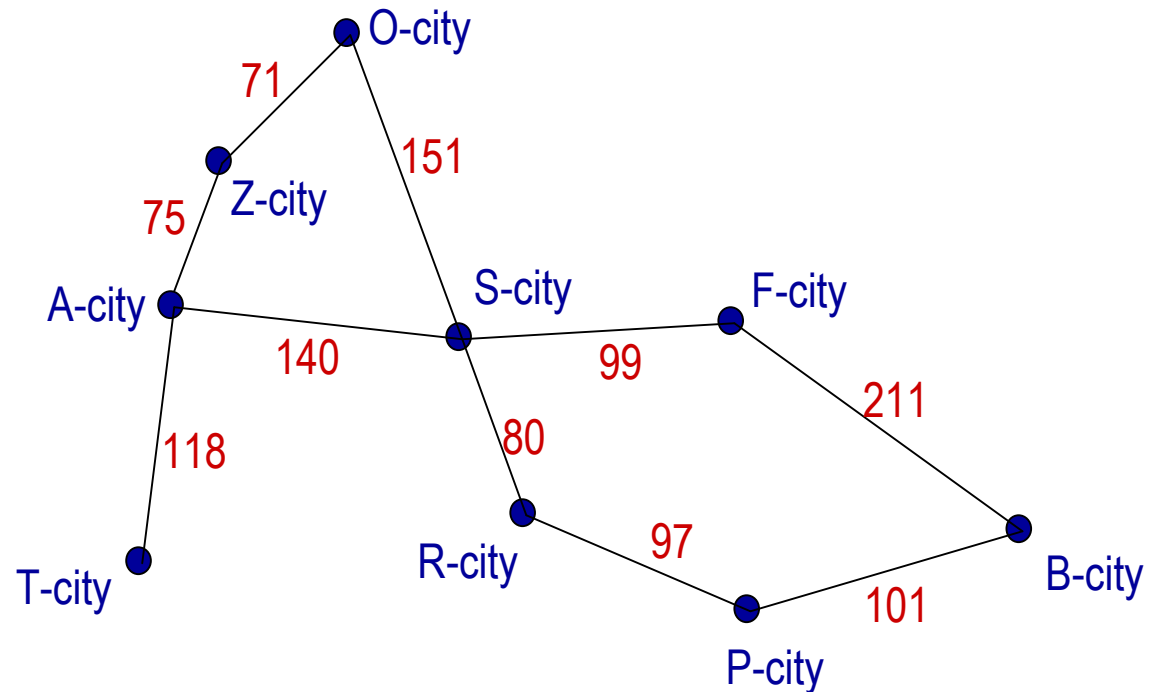
P-city: 98

R-city: 193

S-city: 253

T-city: 329

Z-city: 374



# Pros and Cons of A\* Search/Path Planner

---

- Advantage:

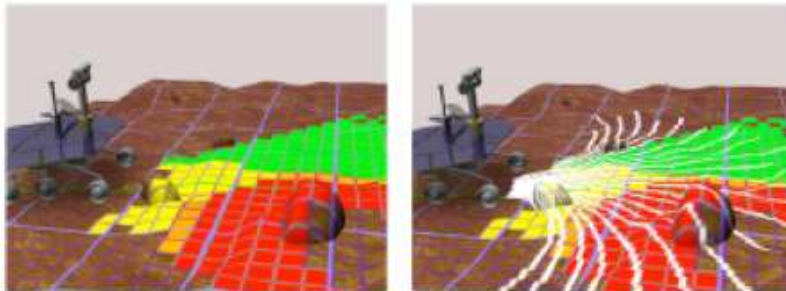
- Can be used with any Cspace representation that can be transformed into a graph

- Limitation:

- Hard to use for path planning when there are factors to consider other than distance (e.g., rocky terrain, sand, etc.)

# Extension to $A^* = D^*$

- $D^*$ : initially plans path to goal just like  $A^*$ , but plans a path from every position to the goal in advance
  - I.e., rather than “single source shortest path” (Dijkstra’s algorithm),
    - Solve “all pairs shortest path” (e.g., Floyd-Warshall algorithm)
- In  $D^*$ , the estimated distance,  $h^*$ , is based on **traversability**



Calculate traversability using stereo cameras;  
can also manually mark maps

- Then,  $D^*$  continuously replans, by updating map with newly sensed information
  - Approach: “repair” pre-planned paths based on new information



# Wavefront-Based Path Planners

---

- Well-suited for grid representations
- **General idea:** consider Cspace to be conductive material with heat radiating out from initial node to goal node
- **If there is a path,** heat will eventually reach goal node
- **Nice side effect:** optimal path from all grid elements to the goal can be computed
- **Result:** map that looks like a potential field



# Algorithmic approach for Wavefront Planning

---

## Part I: Propagate wave from goal to start

- Start with binary grid; 0's represent free space, 1's represent obstacles
- Label goal cell with "2"
- Label all 0-valued grid cells adjacent to the "2" cell with "3"
- Label all 0-valued grid cells adjacent to the "3" cells with "4"
- Continue until wave front reaches the start cell.

## Part II: Extract path using gradient descent

- Given label of start cell as "x", find neighboring grid cell labeled "x-1"; mark this cell as a waypoint
- Then, find neighboring grid cell labeled "x-2"; mark this cell as a waypoint
- Continue, until reach cell with value "2" (this is the goal cell)

## Part III: Smooth path

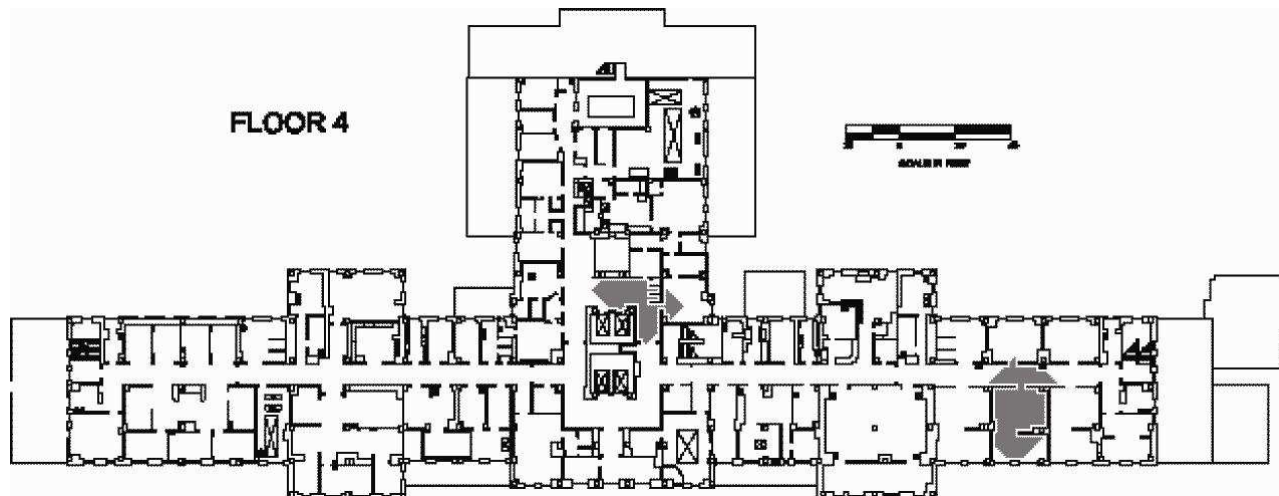
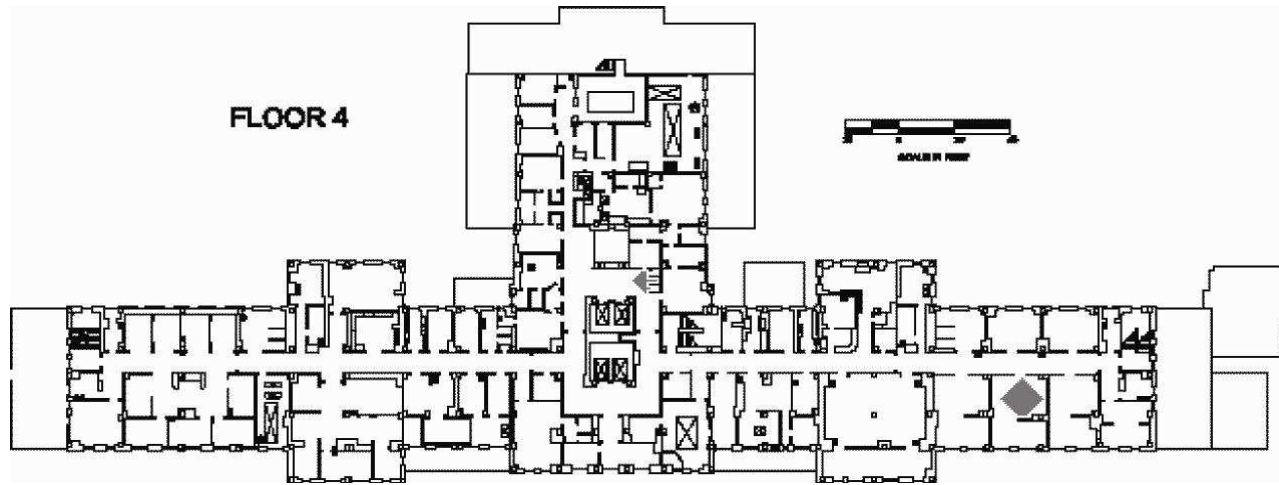
- Iteratively eliminate waypoint  $i$  if path from waypoint  $i-1$  to  $i+1$  does not cross through obstacle
- Repeat until no other waypoints can be eliminated
- Return waypoints as path for robot to follow

# Wavefront Propagation Can Handle Different Terrains

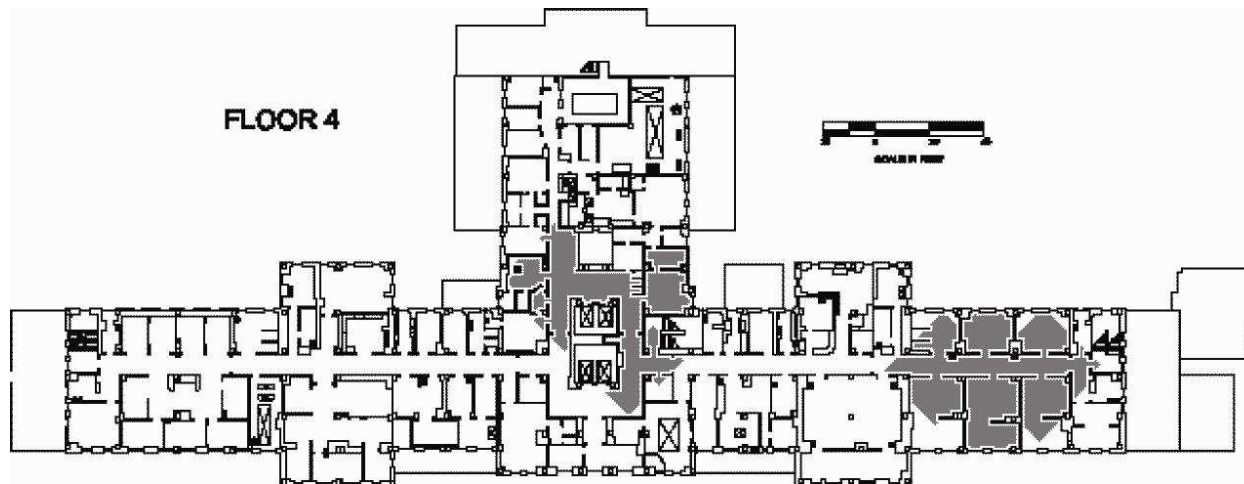
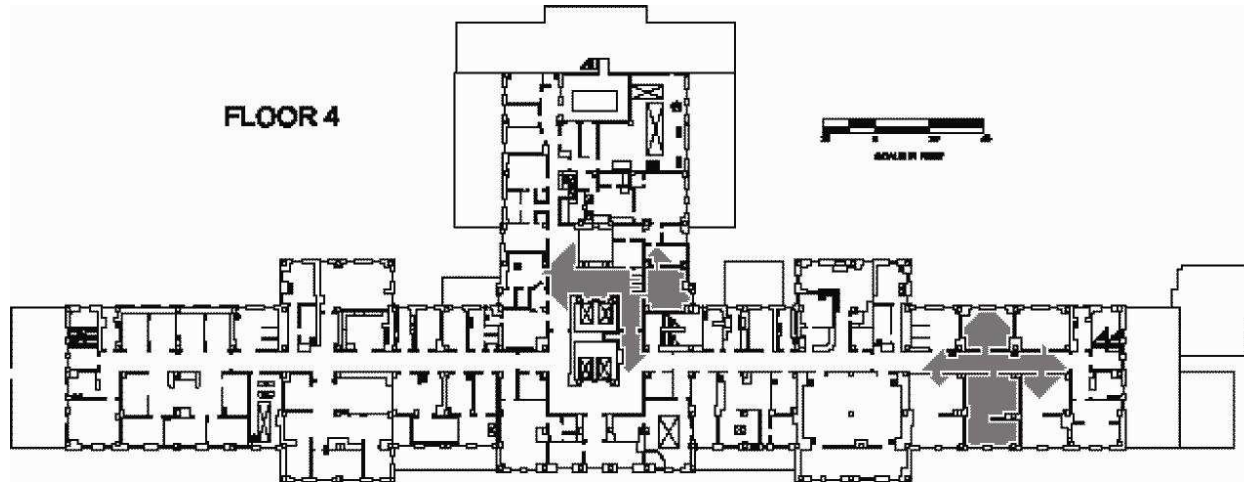
---

- Obstacle: zero conductivity
- Open space: infinite conductivity
- Undesirable terrains (e.g., rocky areas): low conductivity, having effect of a high-cost path
- Also: To save processing time, can use *dual* wavefront propagation, where you propagate from both start and goal locations

# Example Using Dual Wavefront Propagation

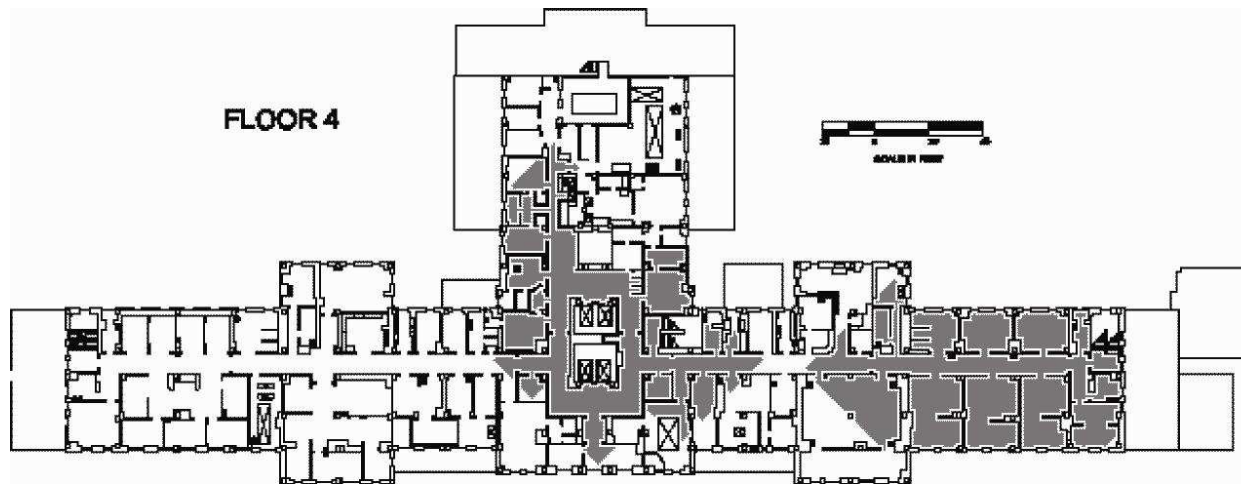
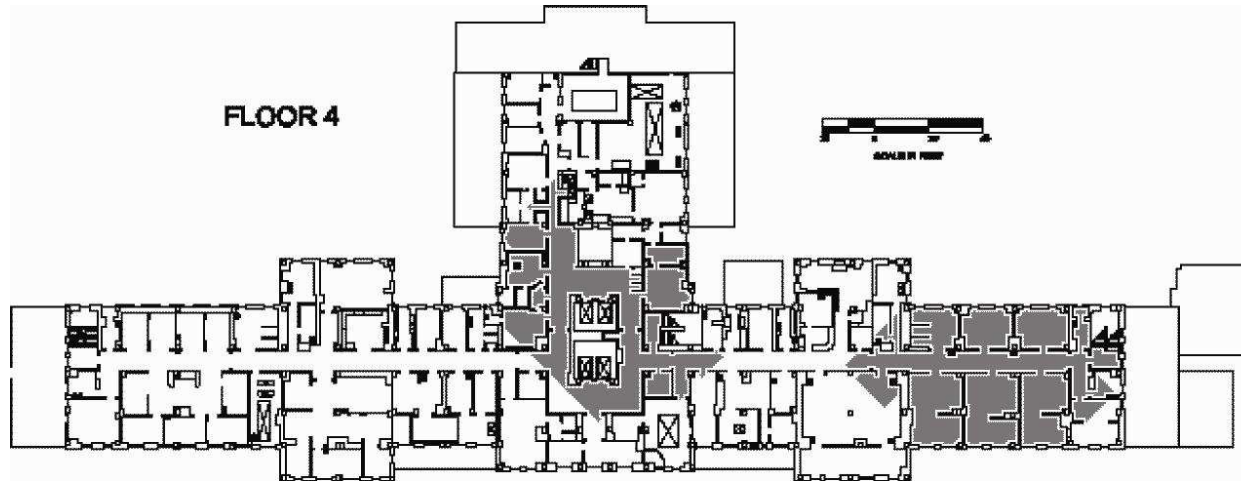


# Dual Wavefront Propagation in Progress



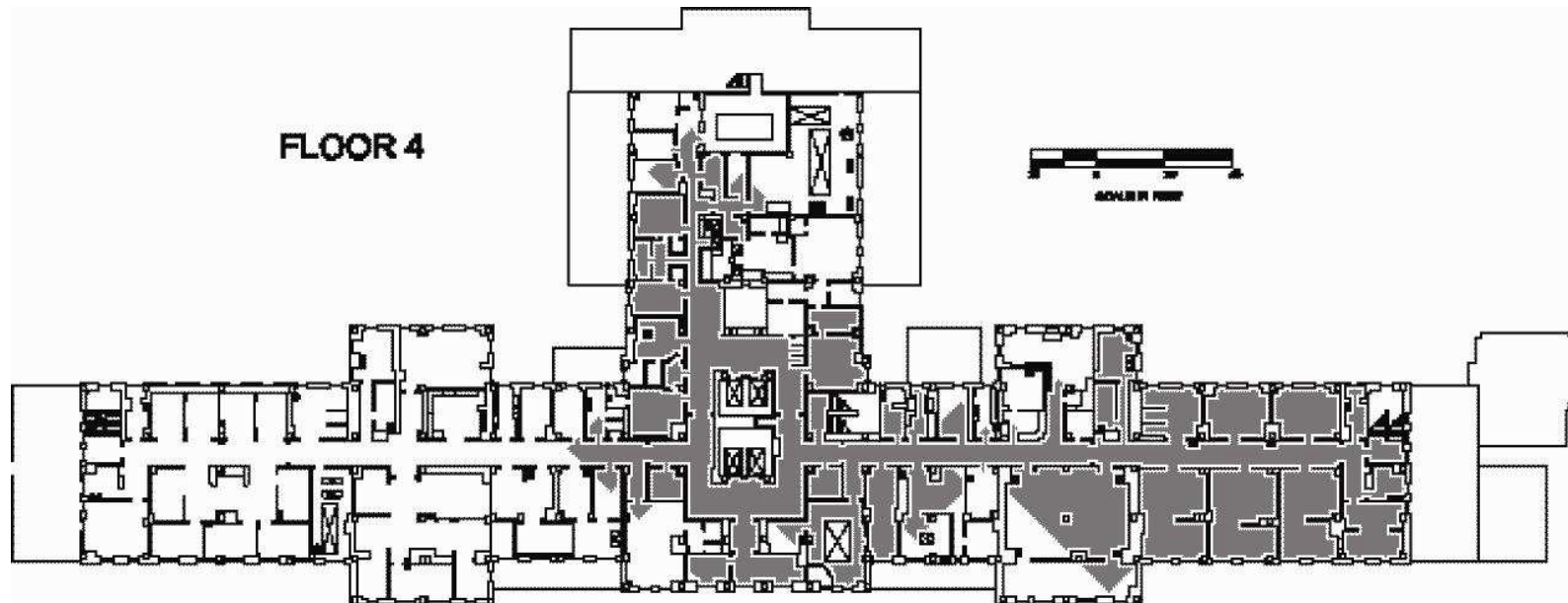
# Dual Wavefront Propagation in Progress (con't.)

---



# Dual Wavefront Propagation in Progress (con't.)

---



Propagation complete.

# Extracted Path

---

