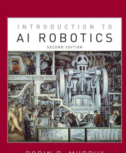
14 Metric Path Planning

What is the difference between topological navigation and metric navigation/path planning?

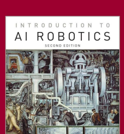
What is commonly used or works good enough?

How much path planning do you need?

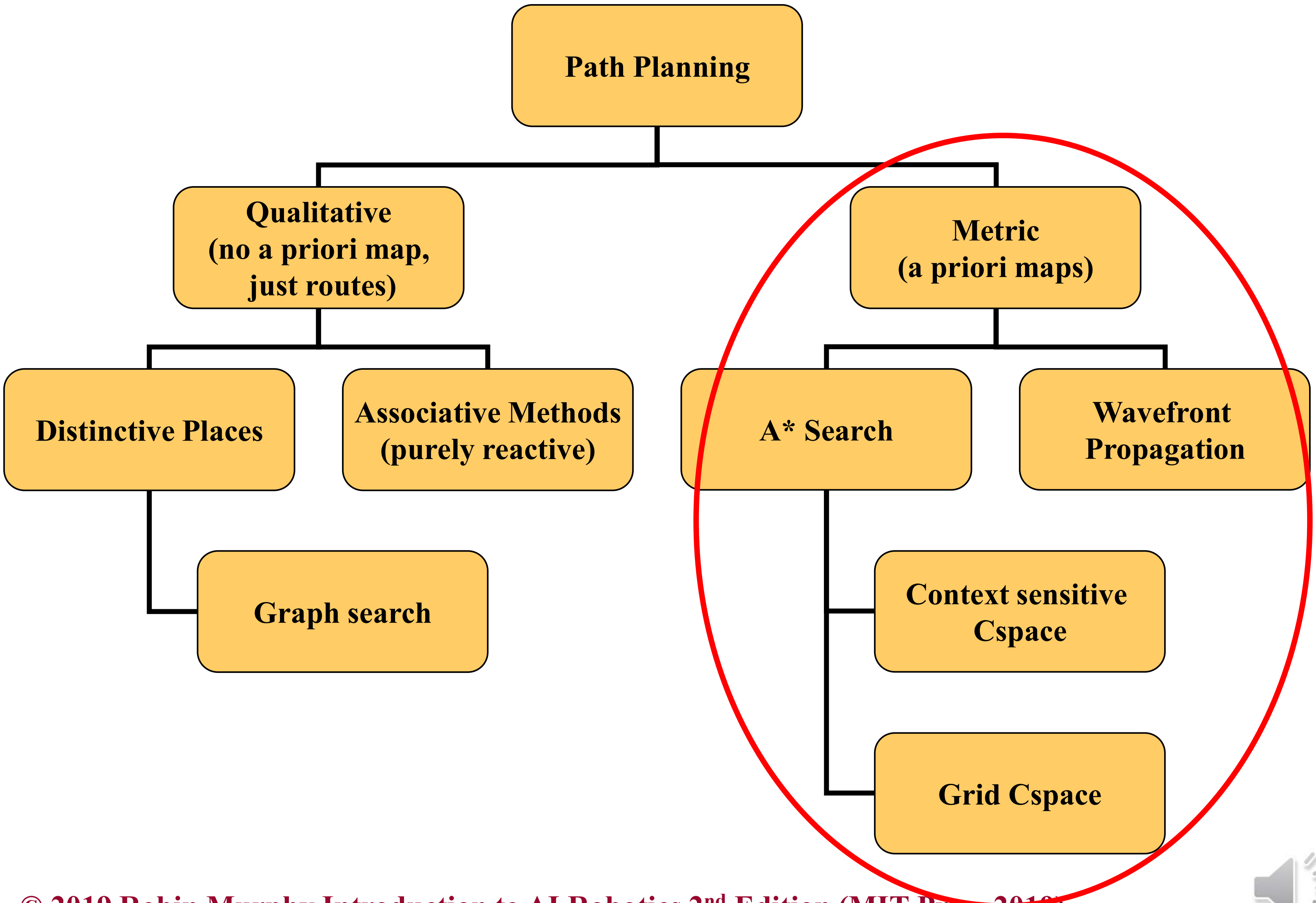


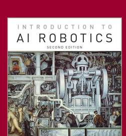
14 Specific Learning Objectives

- Define Cspace, path relaxation, digitization bias, subgoal obsession, and termination condition.
- Represent an indoor environment with a generalized Voronoi graph, a regular grid, or a quadtree, and create a graph suitable for path planning.
- Apply the A* search algorithm to a graph to find the optimal path between two locations.
- Explain the differences between continuous and event-driven replanning.
- Explain how the D* search algorithm accomplishes continuous replanning.



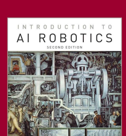
Path Planning Taxonomy





Situations where topological navigation is not sufficient

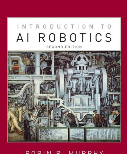
- the space between the starting point and the destination is not easily abstracted into labeled or perceptually distinct regions
  - an unmanned aerial vehicle in the sky may have very few perceptual landmarks.
- the choice of route impacts the control or energy costs of the vehicle.
- the purpose of the path is to allow sensor coverage of an area
  - Search in an area, require locations in a coordinate frame
- the pose of the robot is important, either while reaching a destination or during coverage of an area
  - real robots are not holonomic



14 Two Parts of Metric Path Planning

- Representations:
  - Many different ways to represent an area or volume of space
    - But all look like a “bird’s eye” view, position & viewpoint independent
  - Configuration Space (or Cspace)

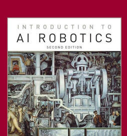
- Algorithms
  - Graph or network algorithms
  - Wavefront or graphics-derived algorithms



14

Metric Maps

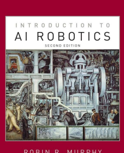
- Motivation for having a metric map is often *path planning* (others include reasoning about space…)
- Determine a path from one point to goal
  - Generally interested in "best" or "optimal"
  - What are measures of best/optimal?
  - Relevant: occupied or empty
- Path planning assumes an *a priori* map of relevant aspects
  - Only as good as last time map was updated



14

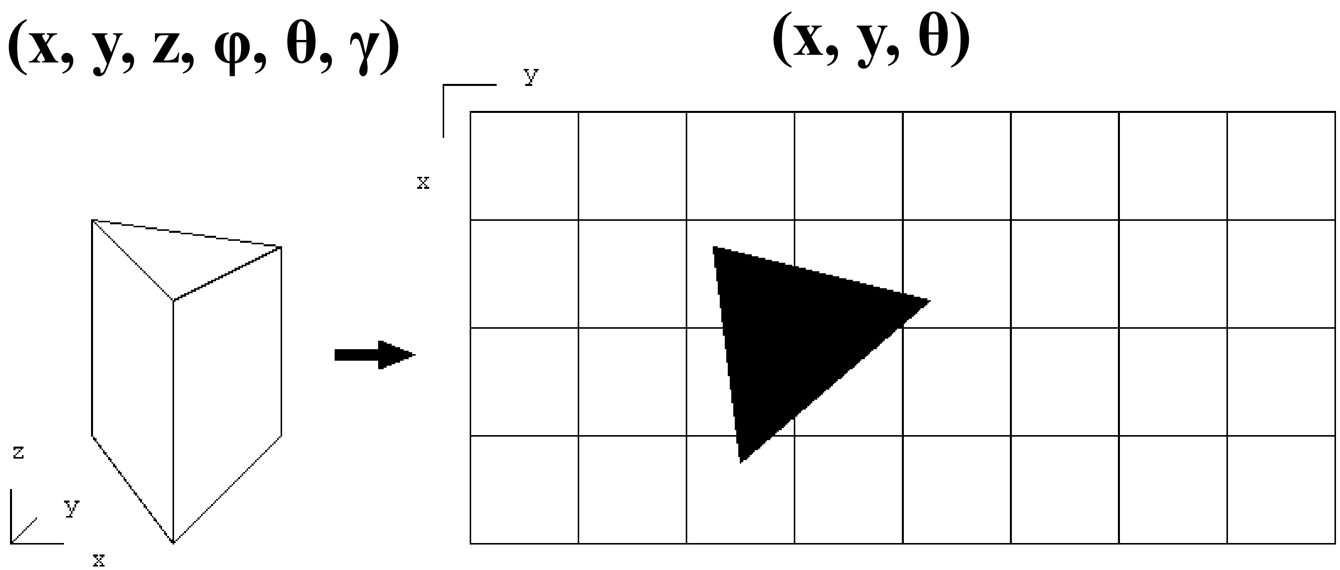
Metric Maps use Cspace

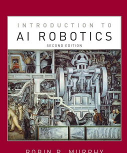
- Physical space: Any rigid 3D object has 6 DOF
  - 3 coordinates: x, y, z
  - 3 Euler angles: φ, θ, γ, also known as pitch, yaw, and roll
- Six degrees of freedom are more than needed for a mobile ground robot for planning a path
  - In general, metric path planning algorithms for mobile robots have assumed only two DOF, translation and rotation
  - The robot can only move forward or backward, and turn within the (x, y) plan
- Configuration Space (Cspace)
  - Transform physical space into a representation suitable for robots, simplifying assumptions



14 Configuration Space

- The configuration space, or Cspace for short, is a data structure which allows the robot to specify the position (location and orientation) of itself and any other objects and the robot

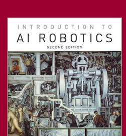




14

Major Cspace Representations

- Idea: reduce physical space to a Cspace representation which is more amenable for storage in computers and for rapid execution of algorithms
- Major types
  - – Meadow Maps
  - – Generalized Voronoi Graphs (GVG)
  - – Regular grids, quadtrees

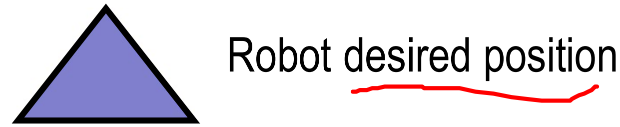


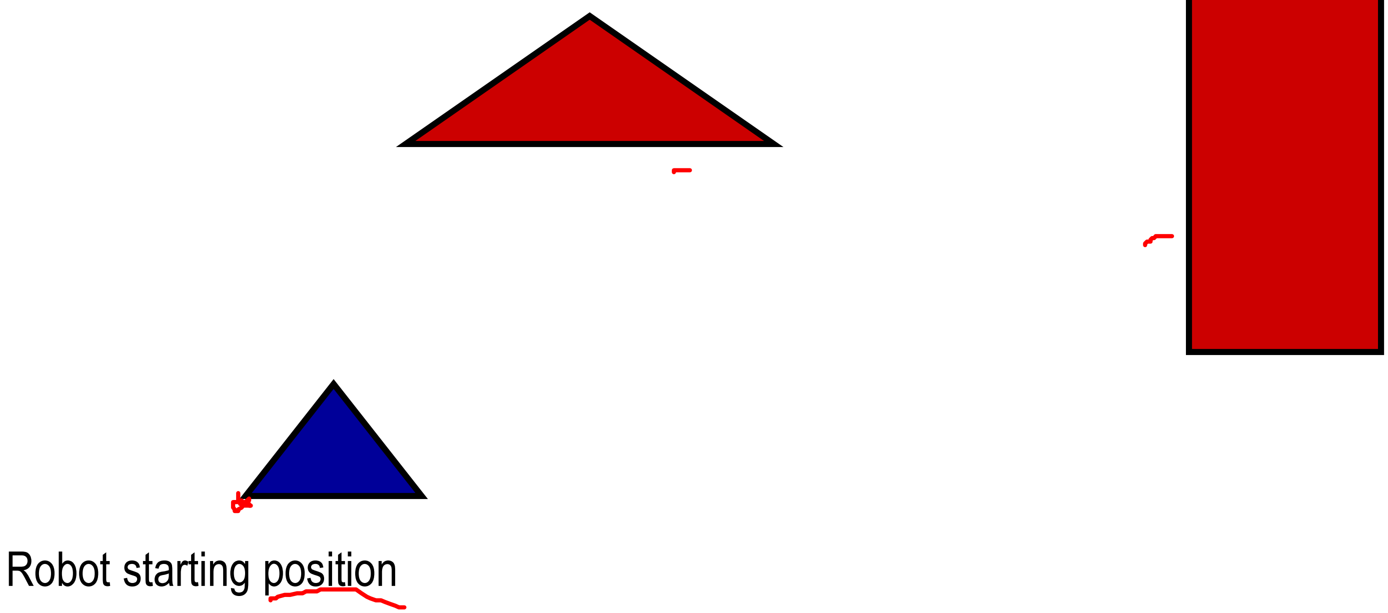
Object Growing

- Since we assume robot is round, we can "grow" objects by the width of the robot and then consider the robot to be a point
- Greatly simplifies path planning
- New representation of objects typically called "configuration space object"

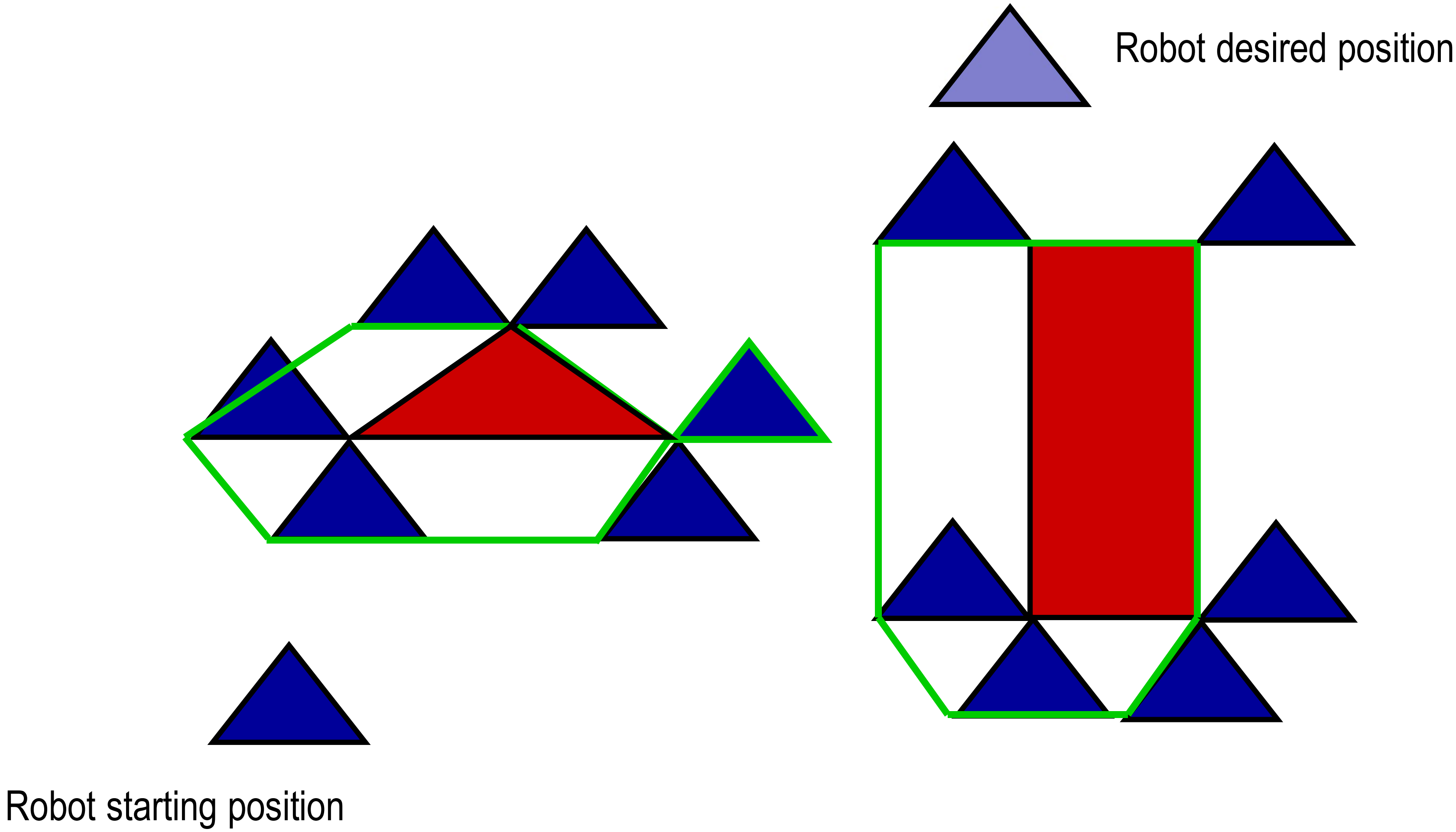
Method for Object Growing

- In this example: Triangular robot
- Configuration growing: based on robot's bottom left corner
- Method: conceptually move robot around obstacles without collision, marking path of robot's bottom left corner



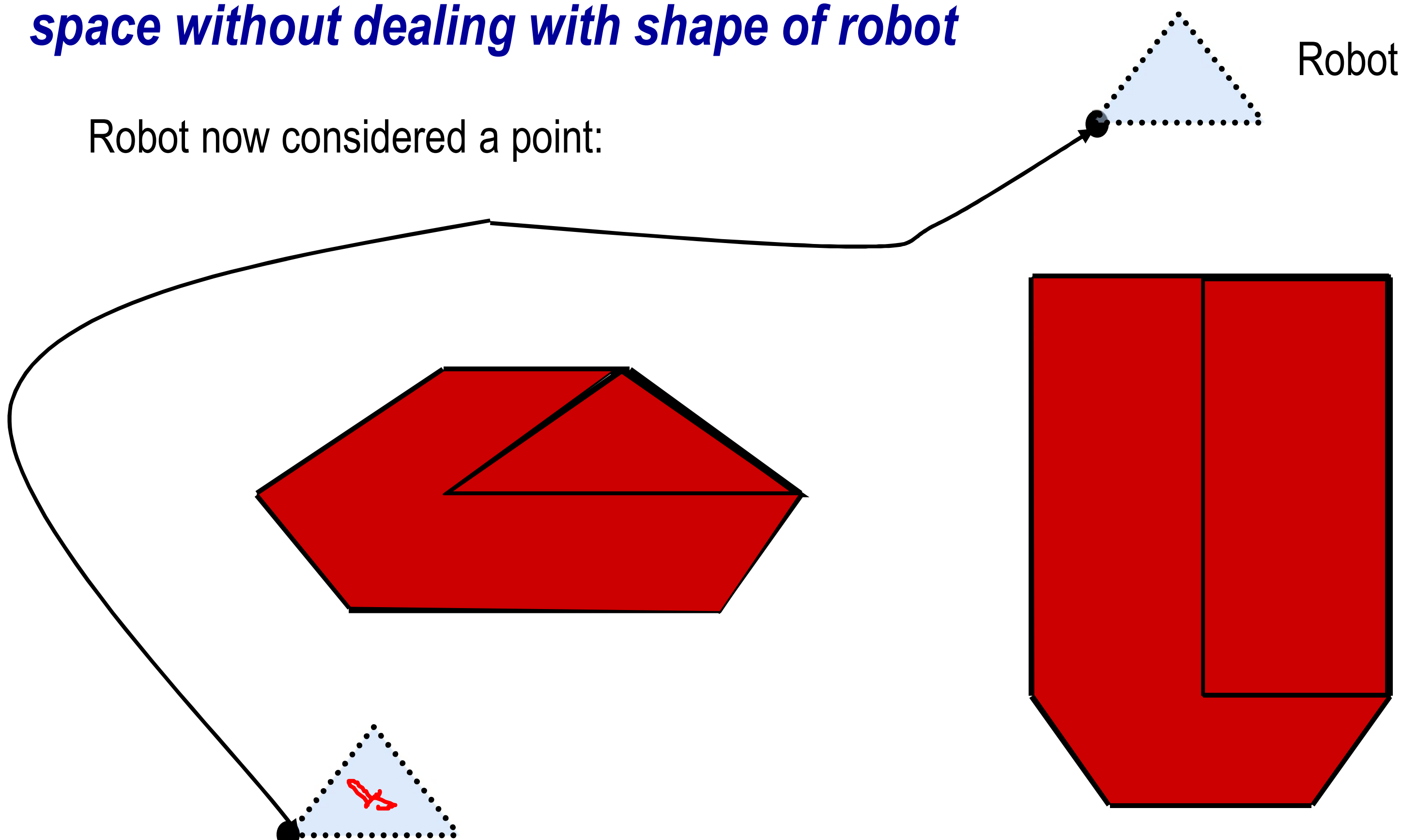


Method for Object Growing



Result of Object Growing: New Configuration Space

Can now plan path of point through this space without dealing with shape of robot



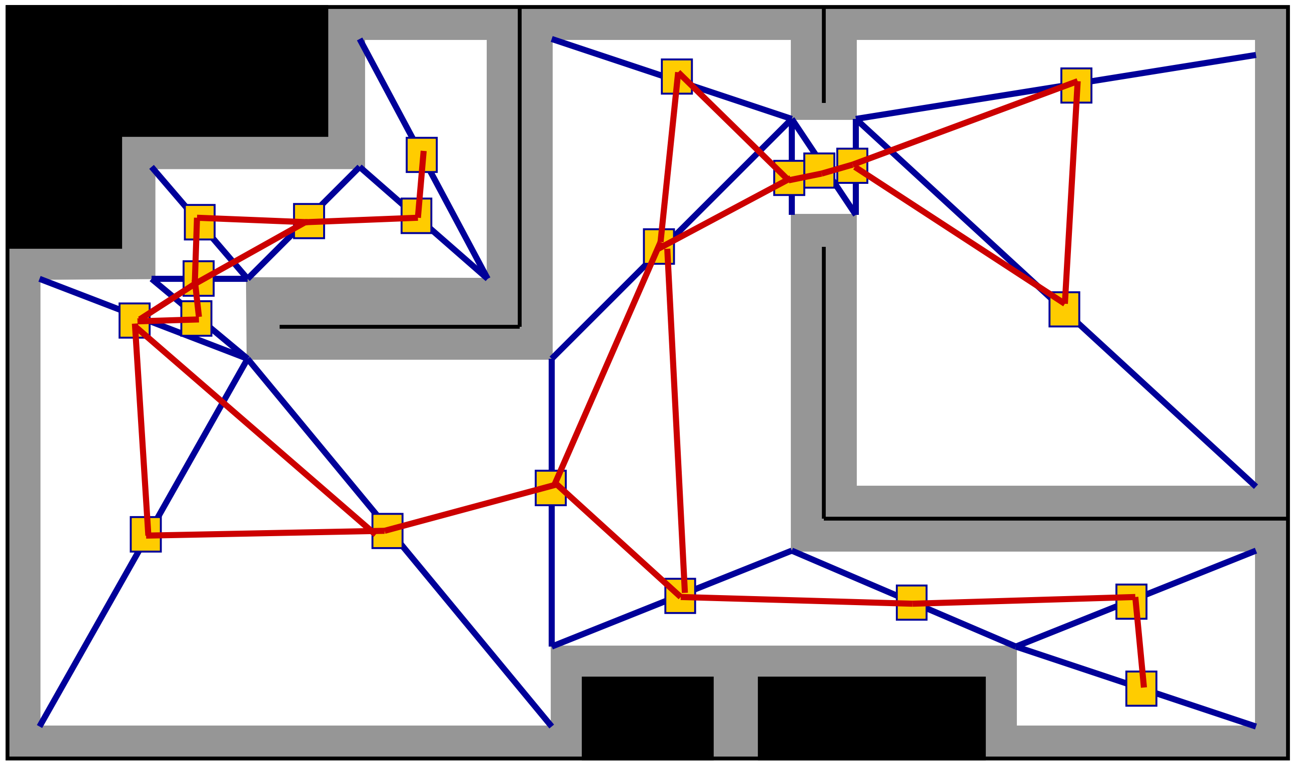
IMPORTANT NOTE: Must make multiple configurations spaces corresponding to various degrees of rotations for moving objects. Then, generalize search to move from space to space

Meadow Maps (Hybrid Vertex-graph Free-space)

- Transform space into convex polygons
  - Polygons represent safe regions for robot to traverse
- Important property of convex polygons:
  - If robot starts on perimeter and goes in a straight line to any other point on the perimeter, it will not go outside the polygon
- Path planning:
  - Involves selecting the best series of polygons to transit through

Example Meadow Map

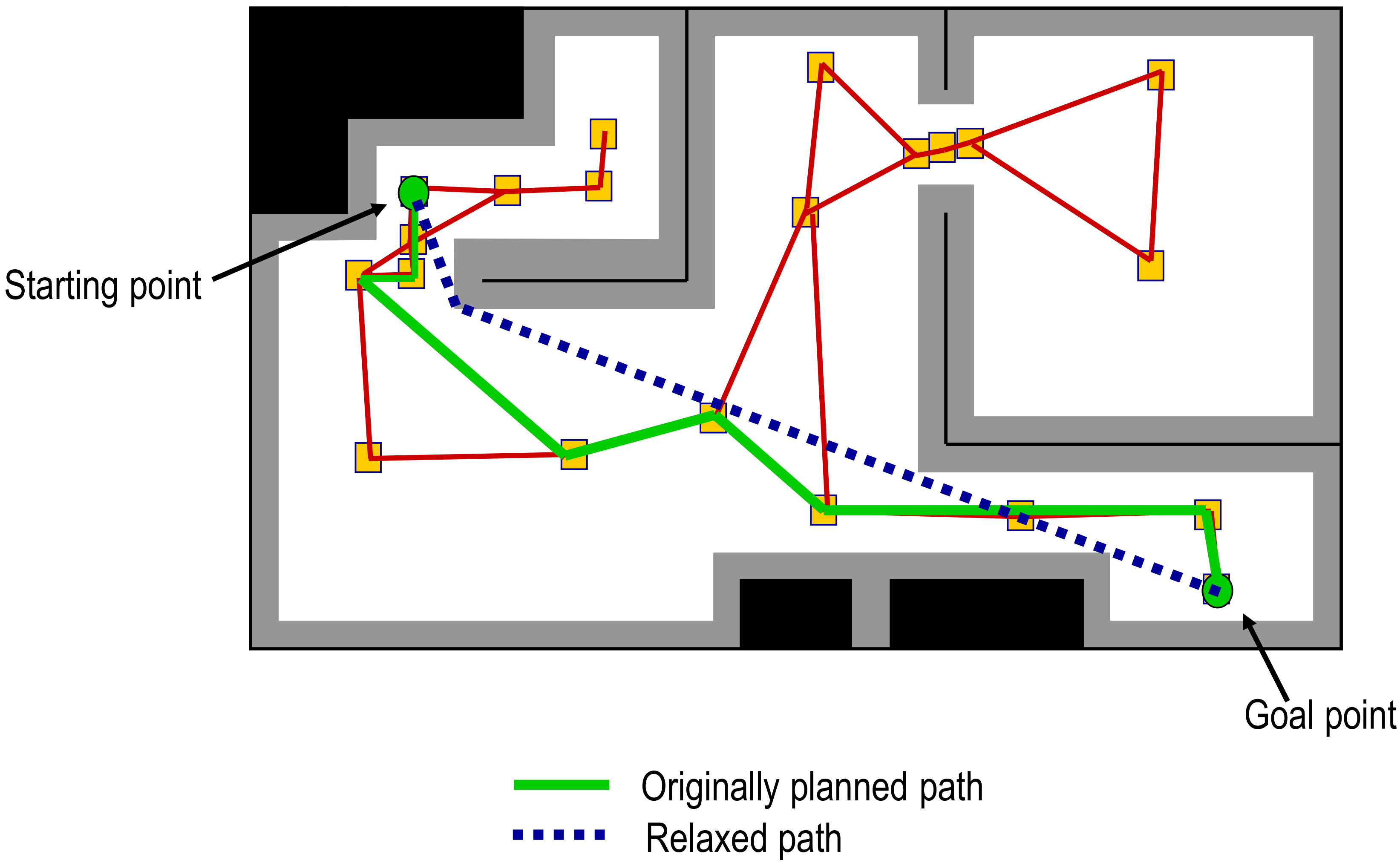
1. Grow objects
2. Construct convex polygons
3. Mark midpoints; these become graph nodes for path planner
4. Path planner plans path based upon new graph



Path Relaxation

- Disadvantage of Meadow Map:
  - Resulting path is jagged
- Solution: path relaxation
  - Technique for smoothing jagged paths resulting from any discretization of space
- Approach:
  - Imagine path is a string
  - Imagine pulling on both ends of the string to tighten it
  - This removes most of “kinks” in path

Example of Path Relaxation



Limited Usefulness of Meadow Maps

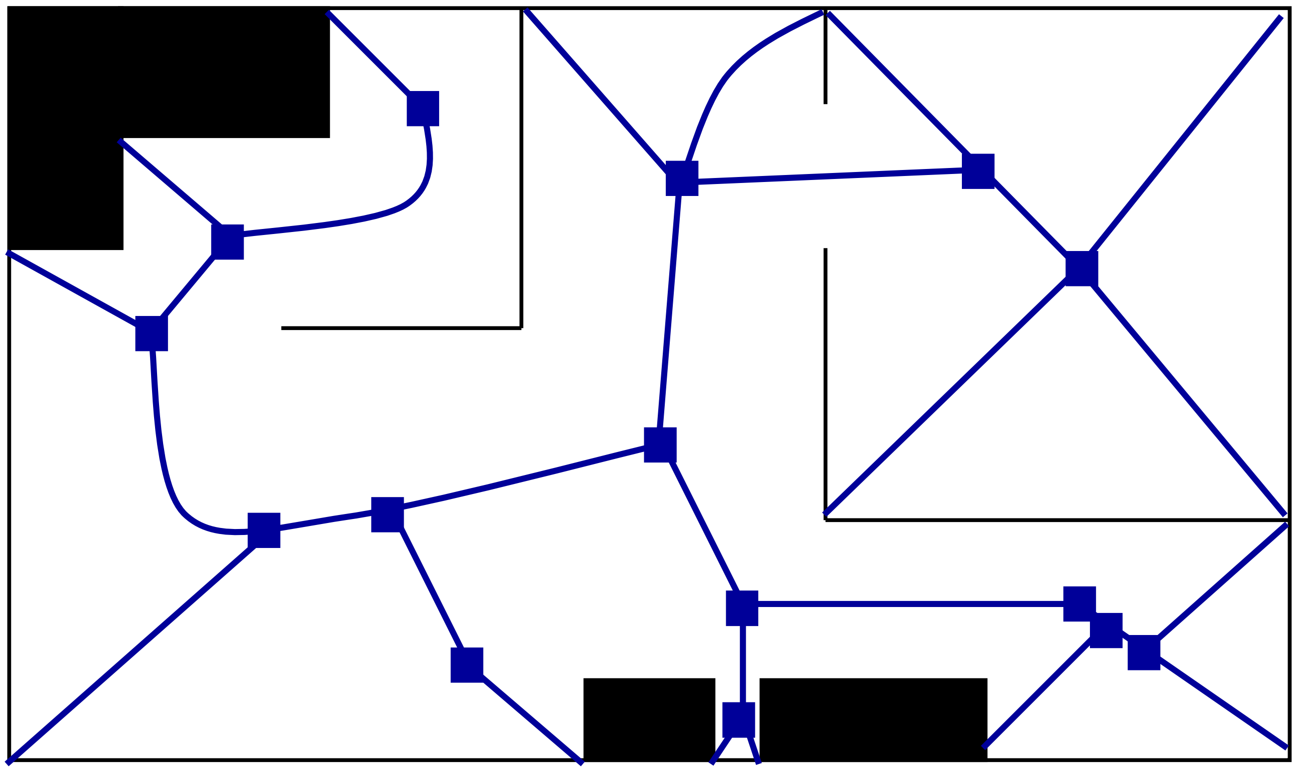
- Three problems with meadow maps:
  - Technique to generate polygons is computationally complex
  - Uses artifacts of the map to determine polygon boundaries, rather than things that can be sensed
  - Unclear how to update or repair diagrams as robot discovers differences between *a priori* map and the real world

Generalized Voronoi Diagrams (GVGs)

- GVGs:
  - Popular mechanism for representing Cspace and generating a graph
  - Can be constructed as robot enters new environment
- Basic GVG approach:
  - Generate a Voronoi edge, which is equidistant from all points
  - Point where Voronoi edge meets is called a Voronoi vertex
  - Note: vertices often have physical correspondence to aspects of environment that can be sensed
  - If robot follows Voronoi edge, it won't collide with any modeled obstacles → don't need to grow obstacle boundaries
- GVG problems
  - Sensitive to sensor noise
  - Path execution: requires robot to be able to sense boundaries

Example Generalized Voronoi Graph

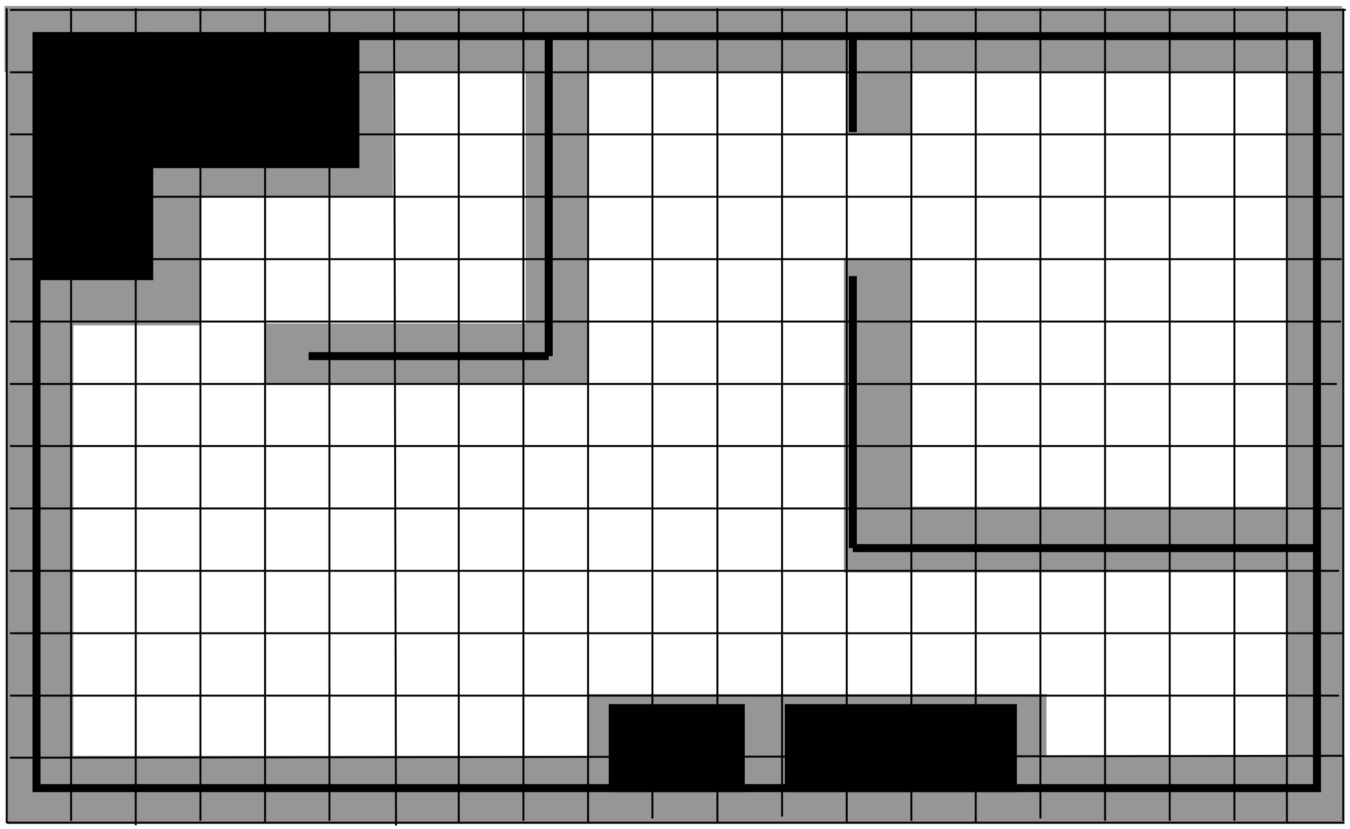
- (NOTE: This is only an approximate, hand-drawn graph to give the basic idea)



Regular Grids / Occupancy Grids

- Superimposes a 2D Cartesian grid on the world space
- If there is any object in the area contained by a grid element, that element is marked as occupied
- Center of each element in grid becomes a node, leading to highly connected graph
- Grids are either considered 4-connected or 8-connected

Example of Regular Grid / Occupancy Grid



Disadvantages of Regular Grids

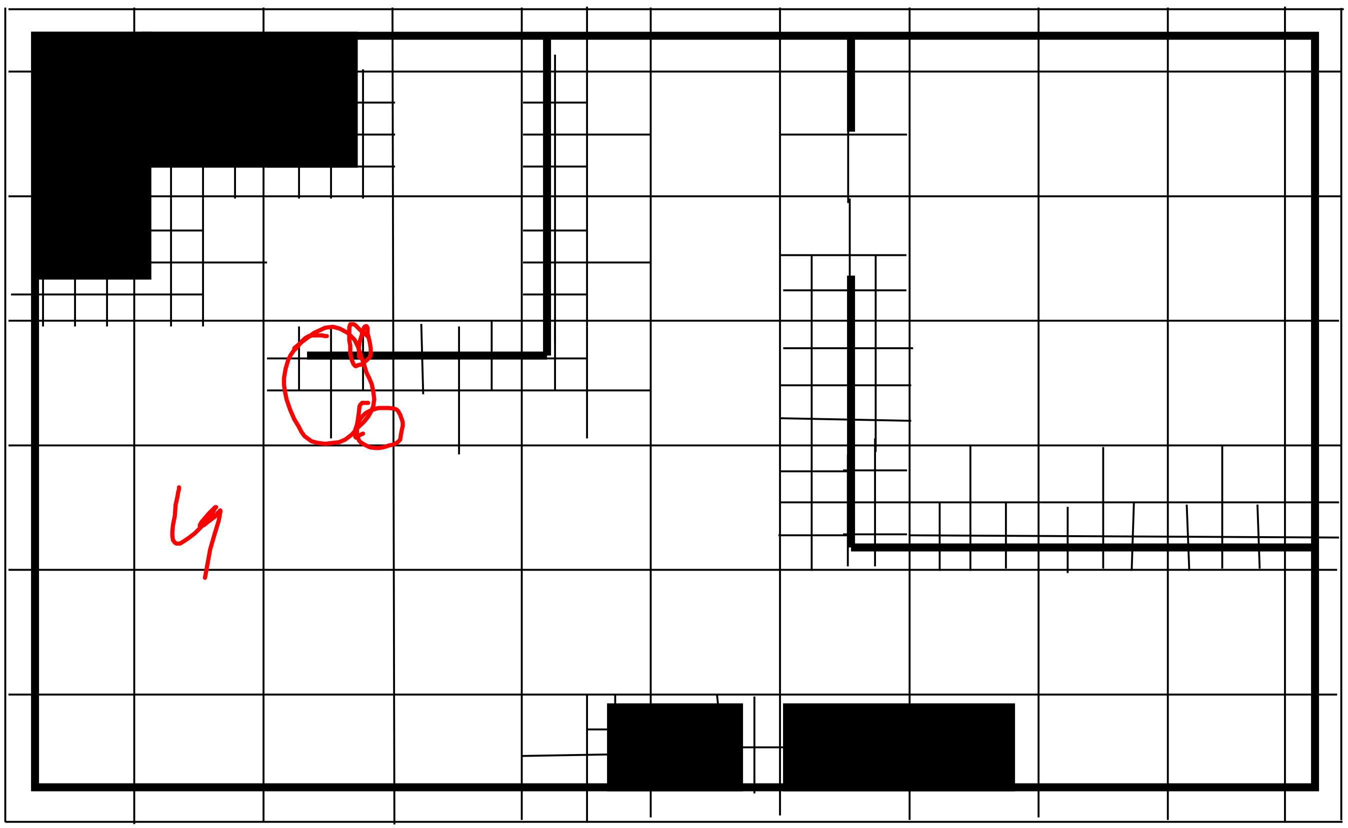
- Digitization bias:
  - If object falls into even small portion of grid element, the whole element is marked as occupied
  - Leads to wasted space
    - Solution: use fine-grained grids (4-6 inches)
    - But, this leads to high storage cost and high # nodes for path planner to consider
- Partial solution to wasted space: Quadtrees

Quadtrees

- Representation starts with large area (e.g., 8x8 inches)
- If object falls into part of grid, but not all of grid, space is subdivided into for smaller grids
- If object doesn’t fit into sub-element, continue recursive subdivision
- 3D version of Quadtree – called an Octree.

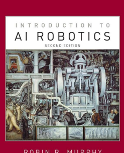
Example Quadtree Representation

(Not all cells are subdivided as in an actual quadtree representation (too much work for a drawing by hand!, but this gives basic idea)

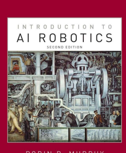


14 Summary of Representations

- Metric path planning requires
  - Representation of world space, usually try to simplify to cspace
  - Algorithms which can operate over representation to produce best/optimal path
- Representation
  - Usually try to end up with relational graph
  - Regular grids are currently most popular in practice, GVGs are interesting
  - Tricks of the trade
    - Grow obstacles to size of robot to be able to treat *holonomic* robots as point
    - Relaxation (string tightening)
- Metric methods often ignore issue of
  - how to execute a planned path
  - Impact of sensor noise or uncertainty, localization



14 Metric Path Planning as Search

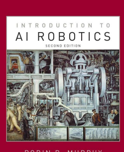


- In AI “search” means that the answer is in the search space, often just finding the path to the answer (goal)
- Types of AI search
  - Blind, brute-force, uninformed
    - Breadth-first
    - Depth-first
    - Uniform-cost
  - *Heuristic*
    - *Greedy*
    - *A**
  - *Local*
    - *Hill Climbing*
    - *Simulated annealing*
    - *Genetic algorithms*

Algorithms

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

- For interleaving planning and execution



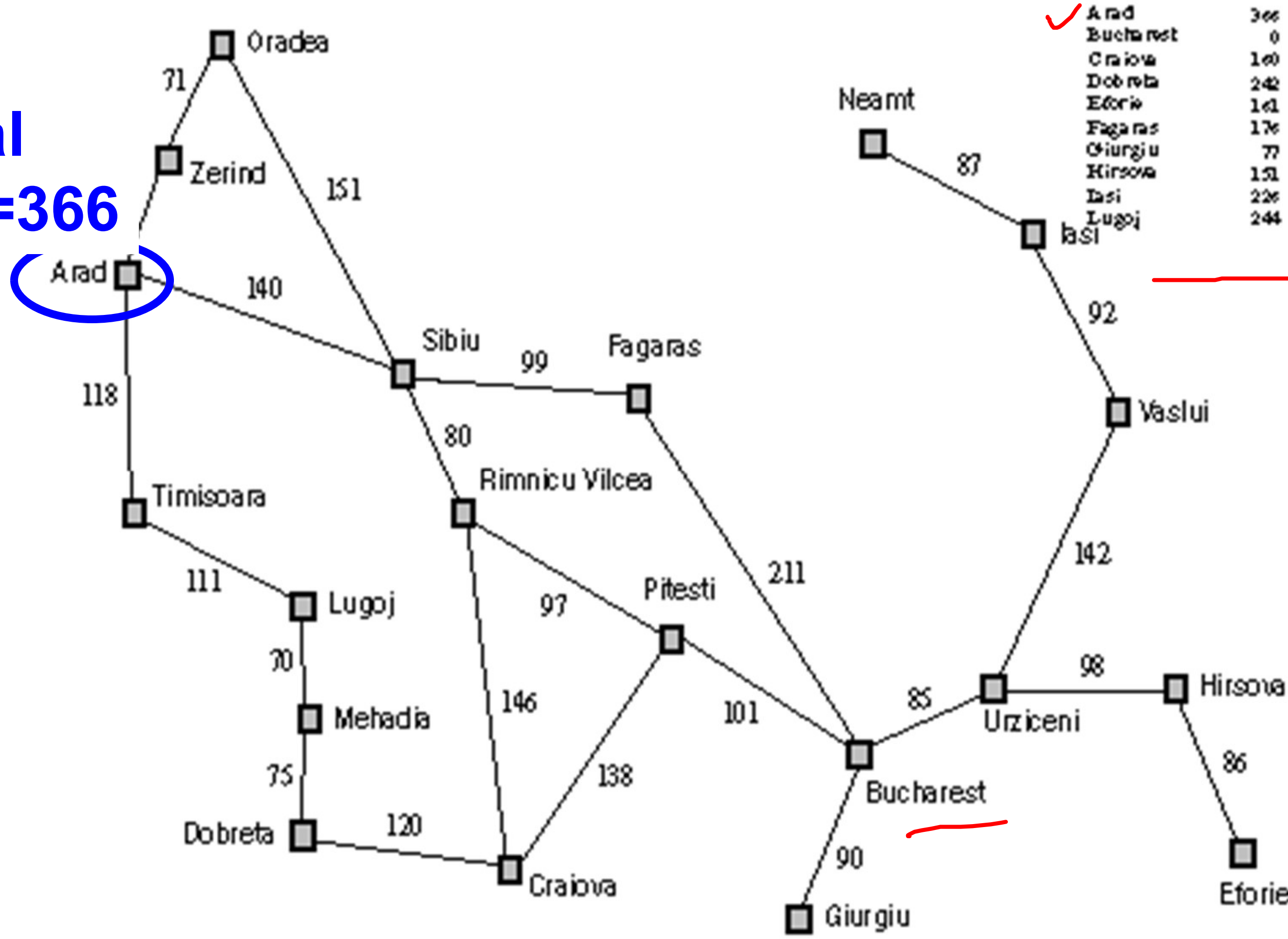
14

A Greedy Method

- Assume we know the straight-line (Euclidean) distance from every note n to the goal node
  - $f(n)=h(n)$
- Pick the starting node
- Repeat
  - Pick a neighboring node (not picked before) that is closest to the goal
- Until the node is goal note

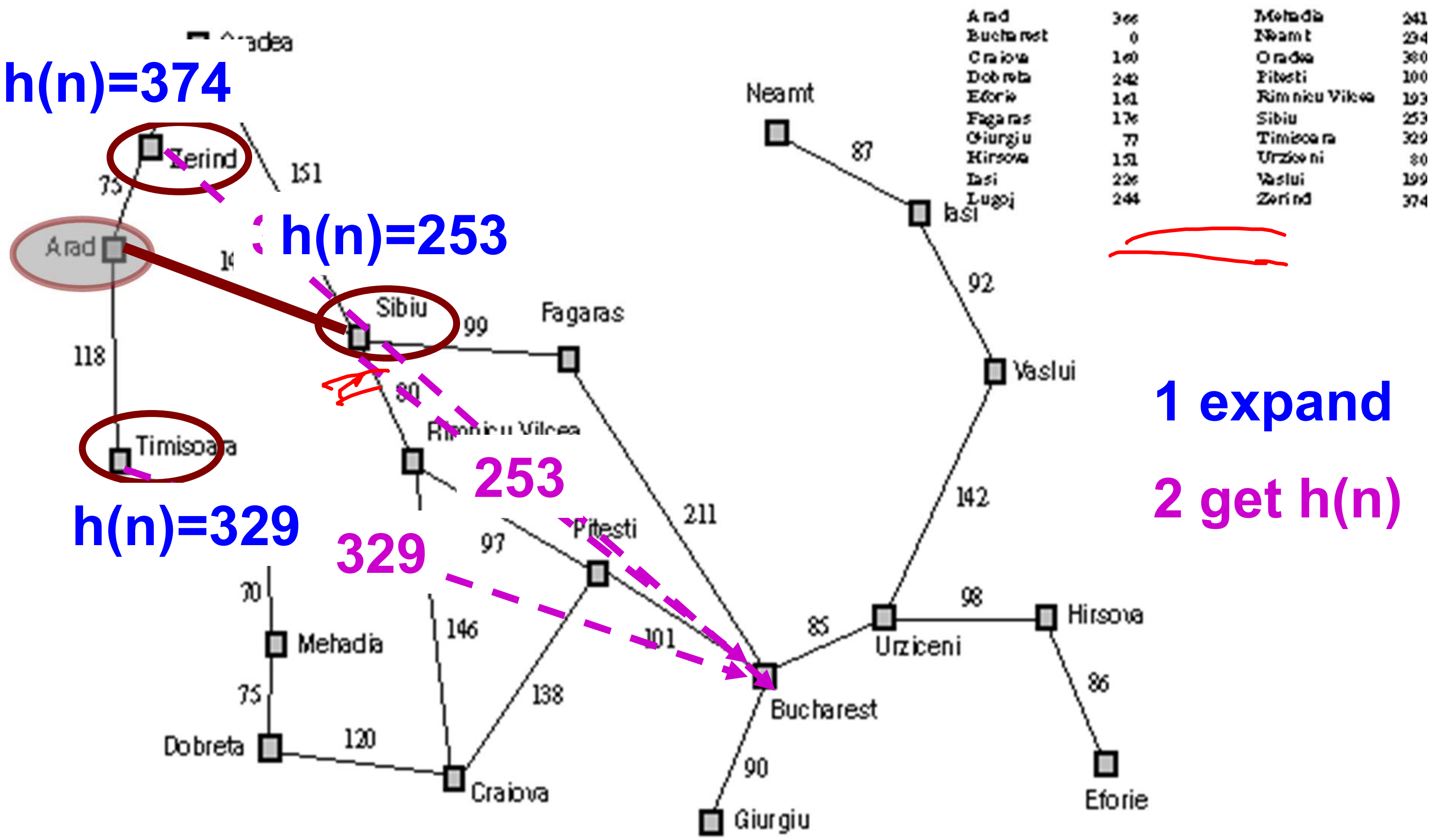
Greedy Ex.: Arad to Bucharest

Initial
h(n)=366

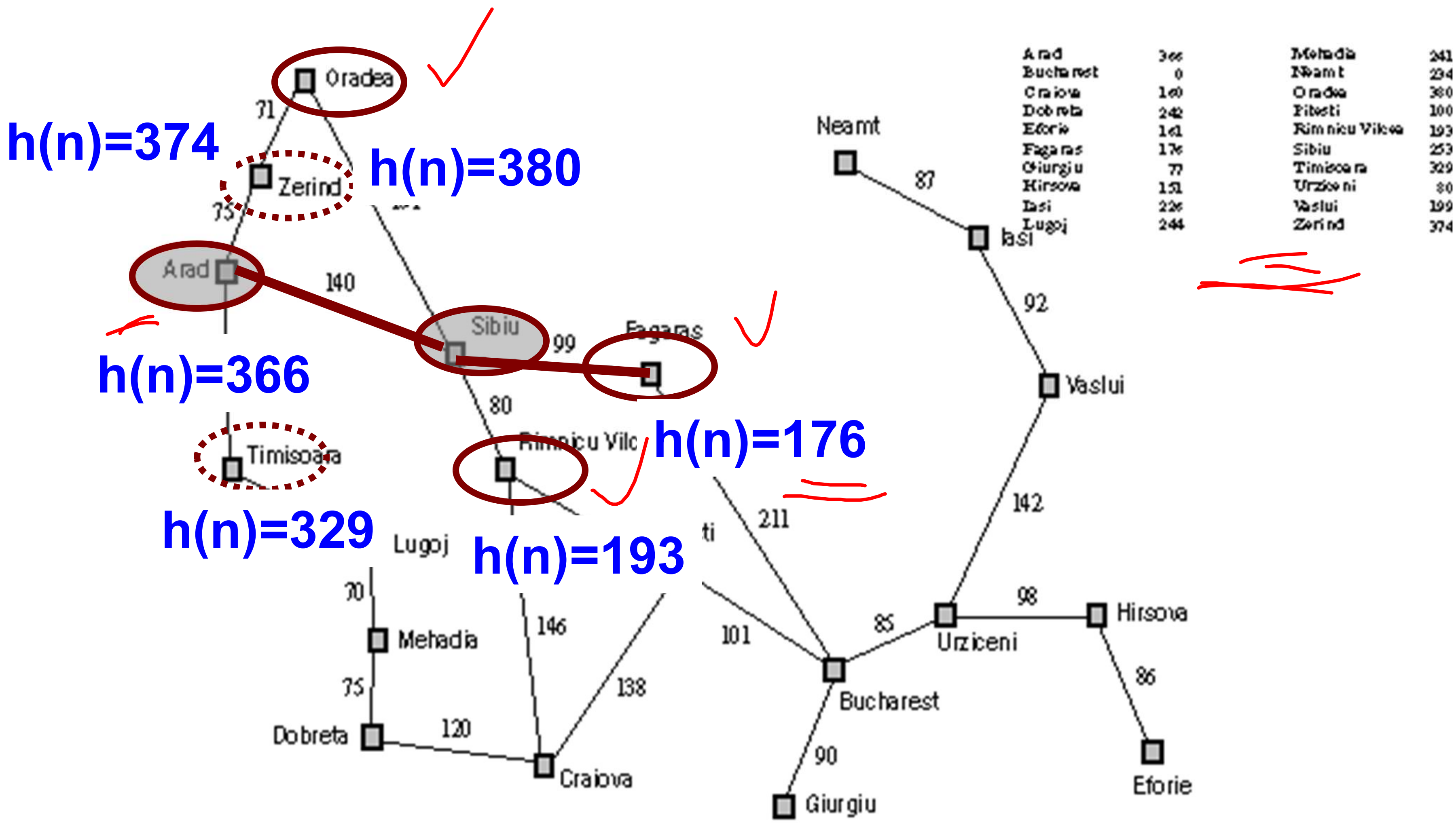


Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

Greedy Ex.: Arad to Bucharest

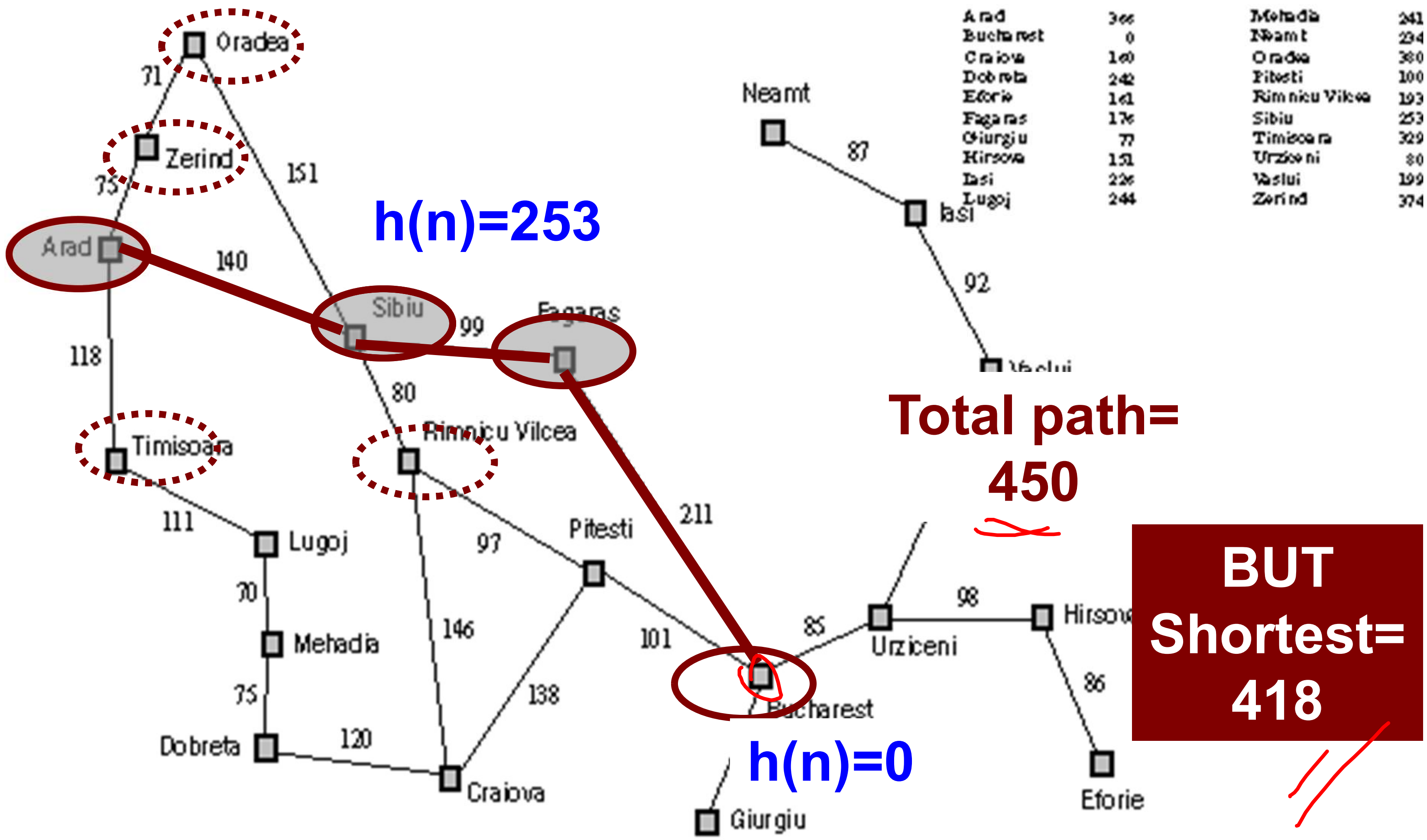


Greedy Ex.: Arad to Bucharest



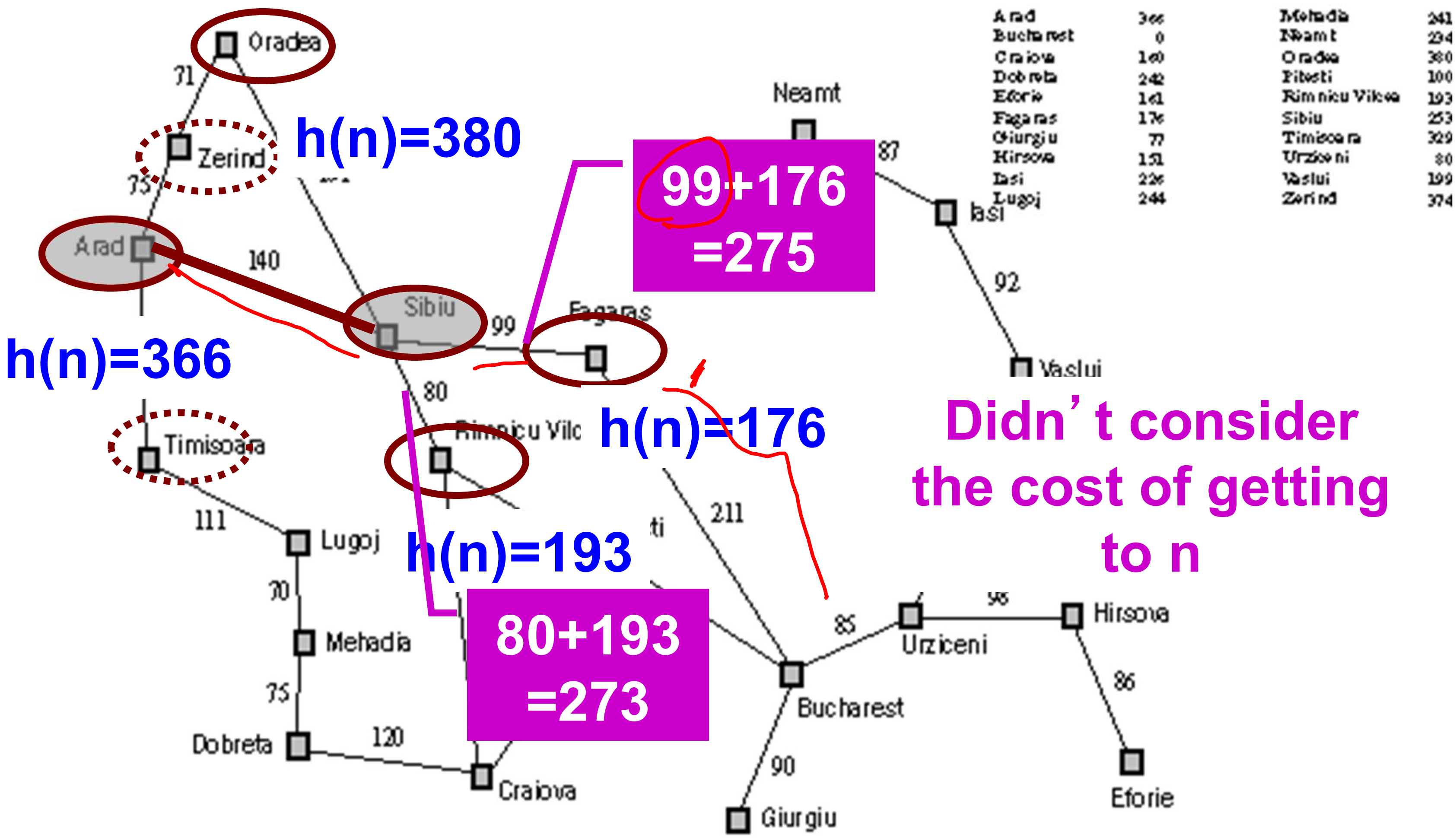
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

Greedy Ex.: Arad to Bucharest



Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

What Went Wrong?



Rimcicu Vilc is almost as close but shorter to get to

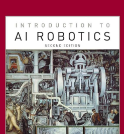
A*

- *Recall greedy f(n)=h(n) and wasn't optimal because it didn't consider "past"*

- *So, add g(n) to consider past:*
  - $f(n)=g(n)+h(n)$

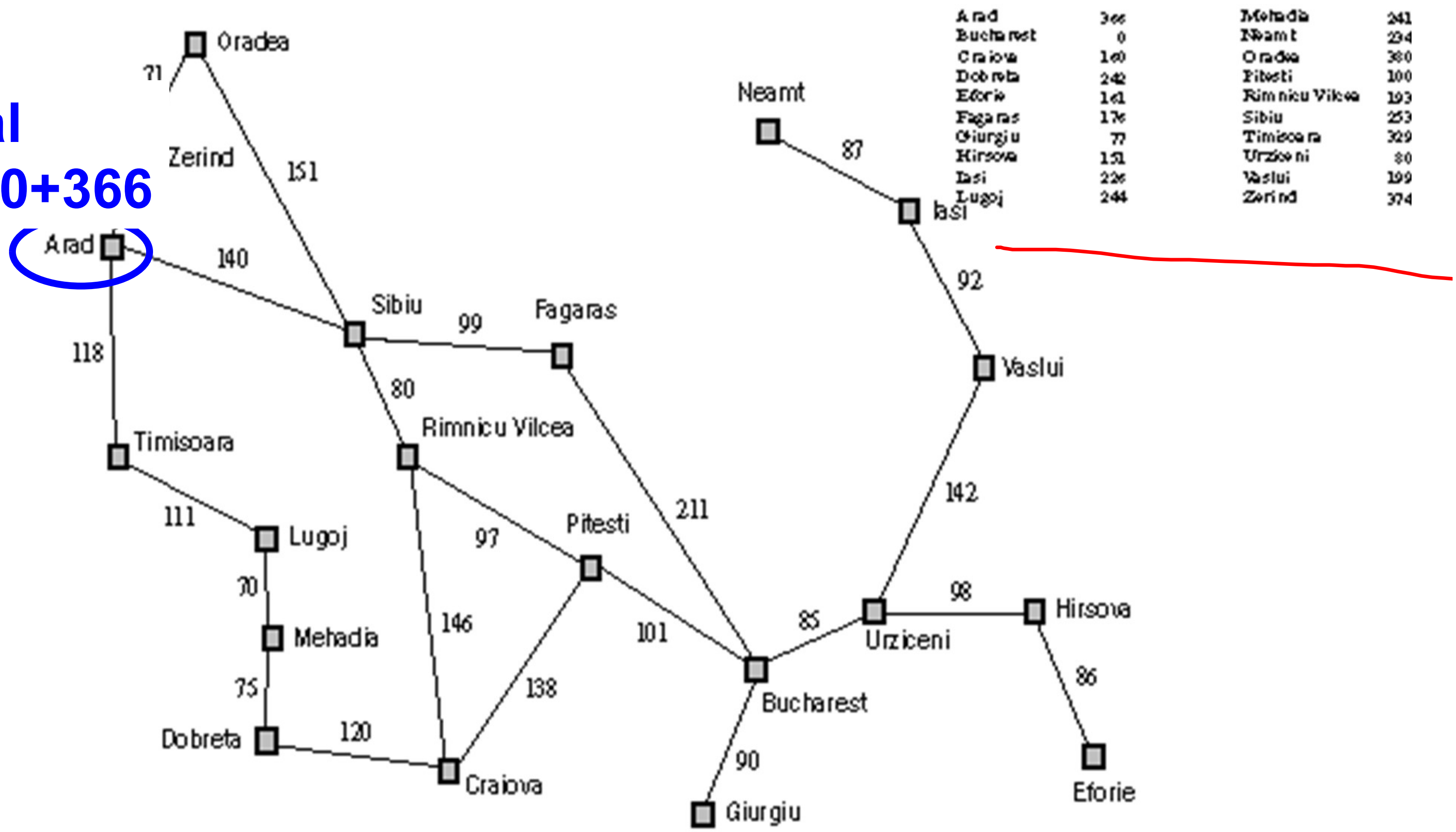
$h^*(n) \leq h(n)$

- Will be *optimal* if $h^*(n)$ is admissible
  - Admissible means $h^*(n)$ will never overestimate true cost
  - In path planning, this is generally Euclidean distance

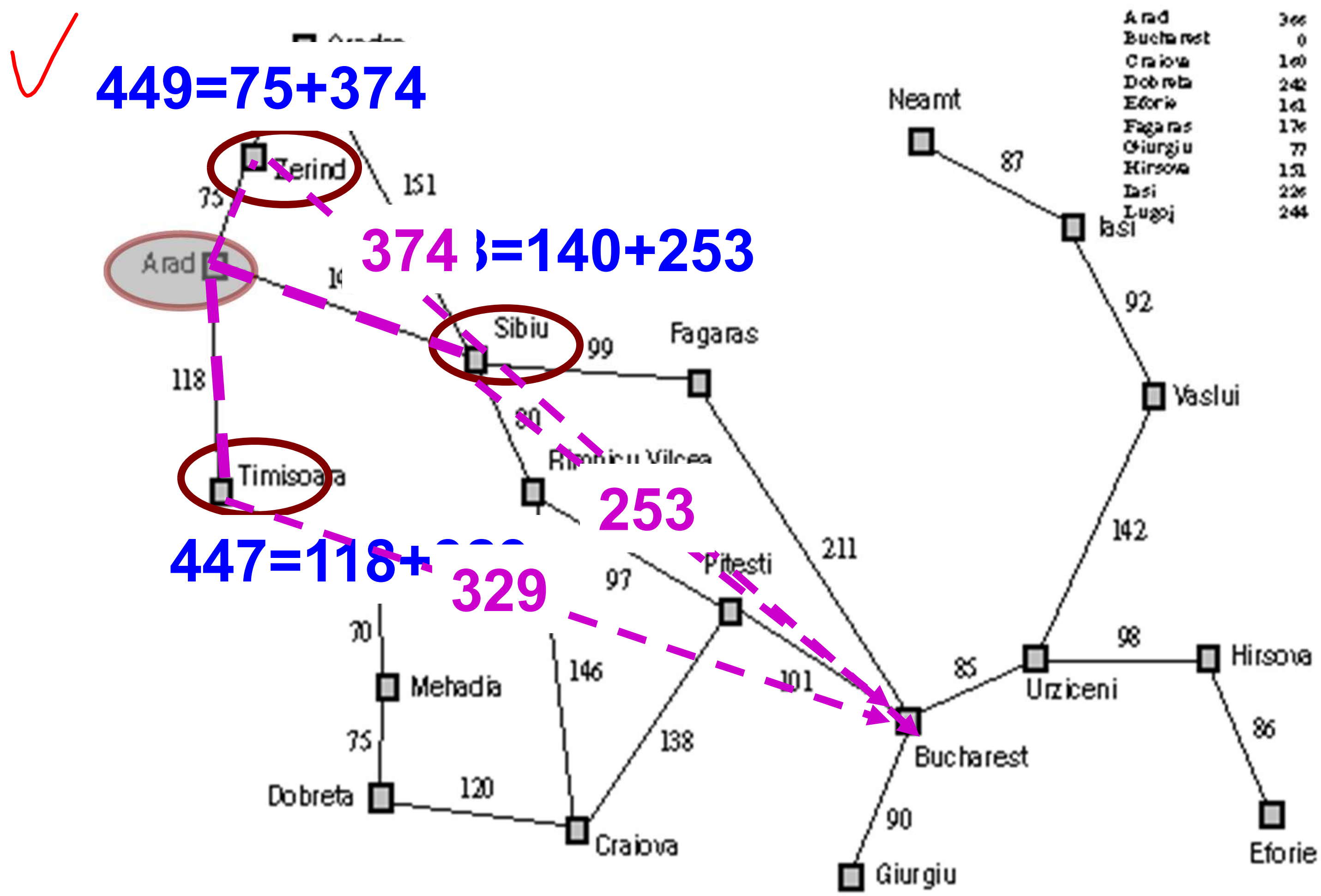


A* Ex.: Arad to Bucharest

Initial
f(n)=0+366

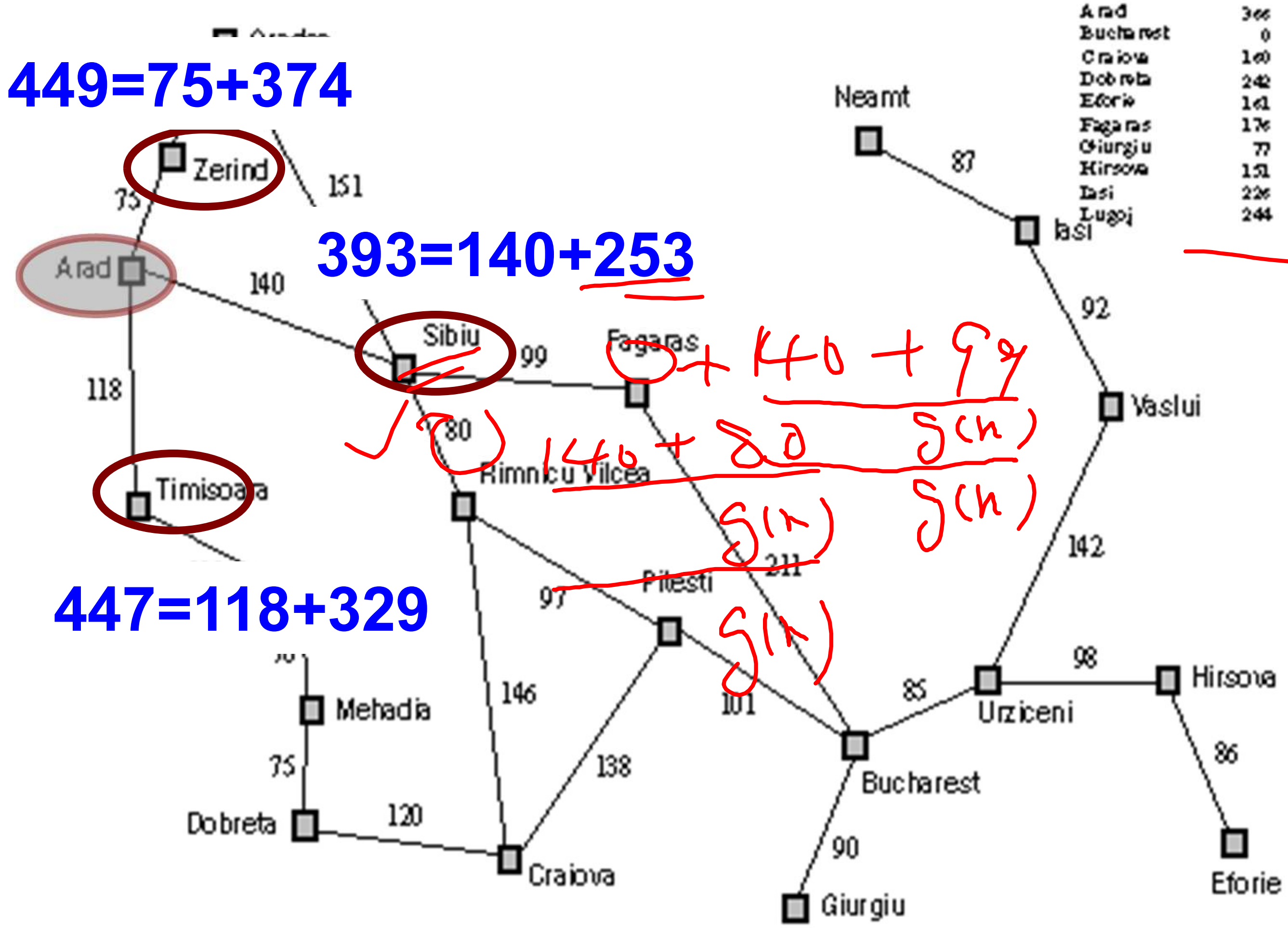


A* Ex.: Arad to Bucharest



Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

A* Ex.: Arad to Bucharest



Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

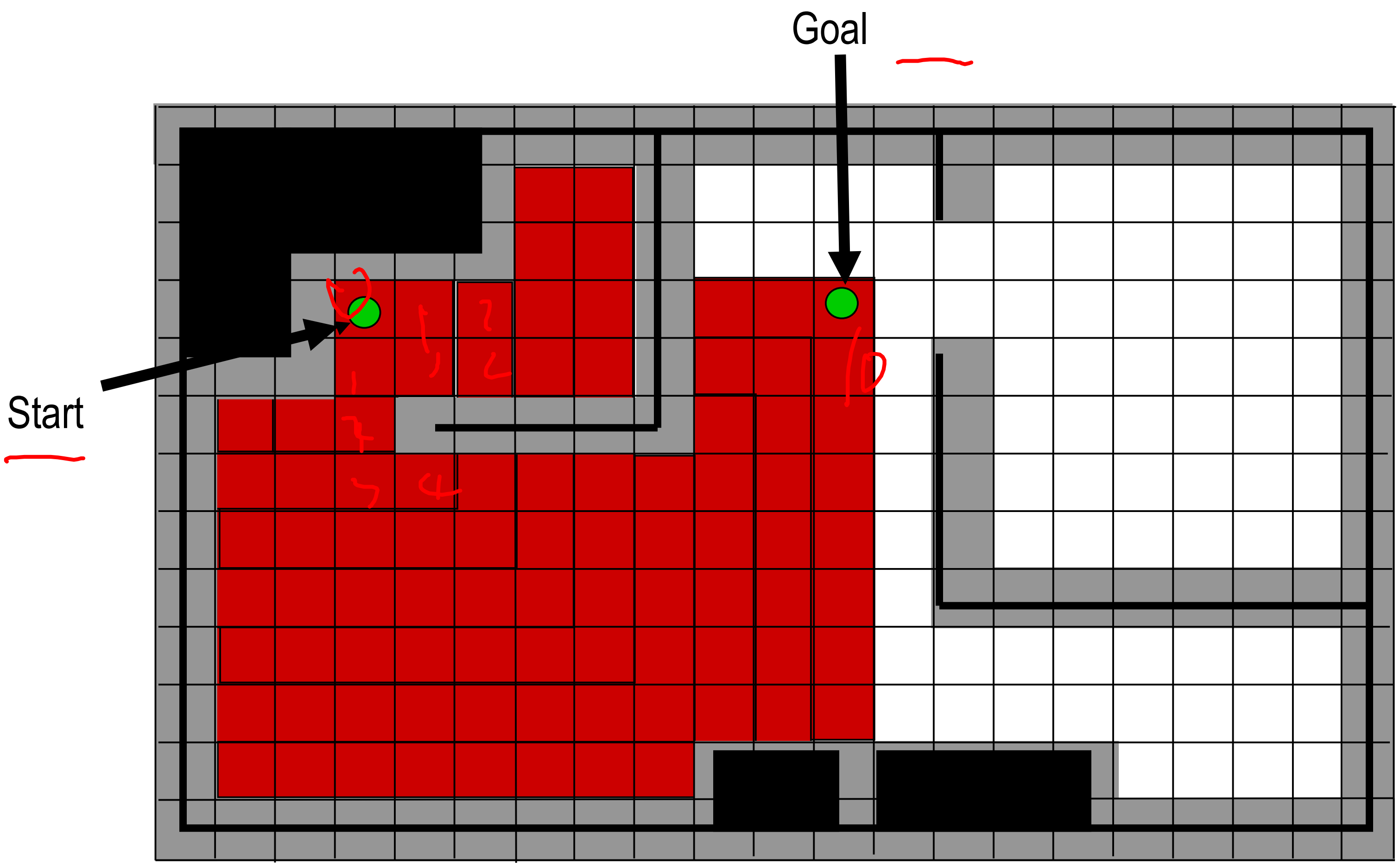
Pros and Cons of A* Search/Path Planner

- 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.)

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

Example of Wavefront Planning



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

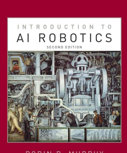
- Iteratively eliminate waypoint i if path from waypoint i-1 to i+1 does not cross through obstacle
- Repeat until no other waypoints can be eliminated
- Return waypoints as path for robot to follow

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

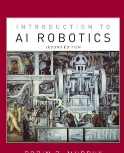
14 Path Planning and Path Executing

- Graph-based planners (like A*) generate a *path* and *subpaths* or subsegments
- Recall NHC
  - Pilot looks at current subpath, instantiates behaviors to get from current location to subgoal
- Subgoal obsession
  - the robot spends too much time and energy trying to reach the exact subgoal position
- Termination condition
  - When does the robot think it has reached subgoal?
  - What about encoder error?
- What happens if blocked? What happens if avoid an obstacle and actually is now closer to the next subgoal?



14 Two Approaches to Path Replanning

- Continuous replanning
  - Essentially imposing a hierarchical Sense, Plan, Act cycle.
  - Example: D* algorithm
    - An extension to A* algorithm
- Event-driven replanning
  - Replan when there is some event, exception, or indication that the plan execution is not working.
  - Can be used in a hybrid Plan, then Sense-Act architecture but it requires the addition of deliberative monitoring.
  - Example: an extension to the Trulla algorithm

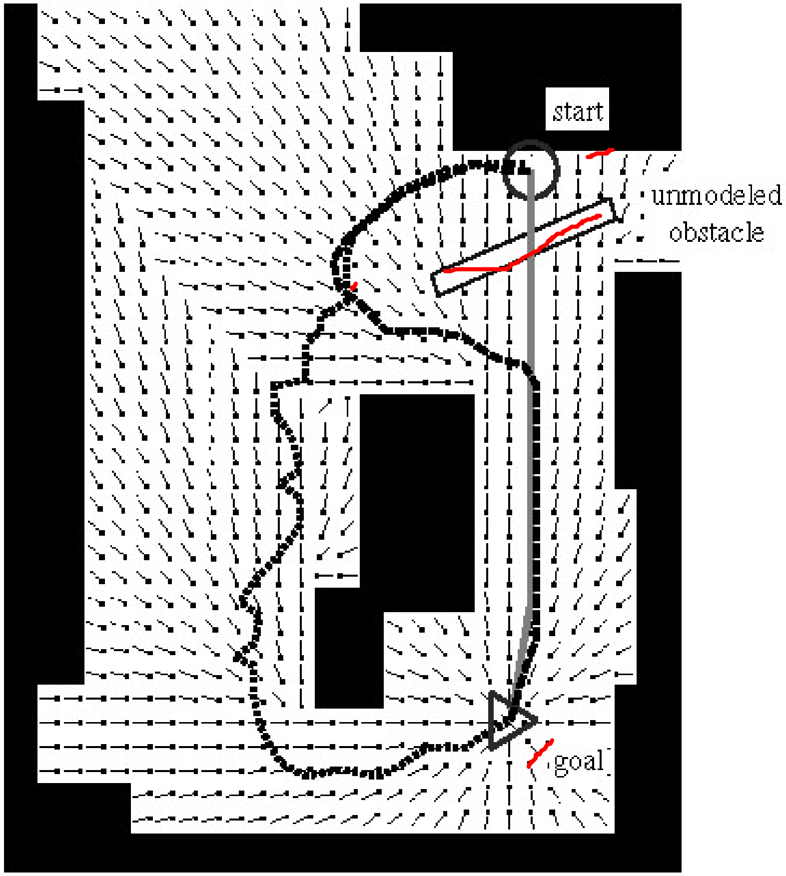


D* Algorithm: Extension to A*

- 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)
- Then, D* continuously replans, by updating map with newly sensed information
  - Approach: "repair" pre-planned paths based on new information
- Advantage: this approach eliminates sub-goal obsession
  - sub-goal obsession is when the robot spends too much time and energy trying to reach the exact sub-goal position
- Disadvantages:
  - Too computationally expensive
  - Highly dependent on the sensing quality

14 Trulla Algorithm and Example

- Trulla uses the dot-product of the intended path vector and the actual path vector.
- When the actual path deviates by 90° or more, the dot-product becomes 0 or negative. This trigs replanning
- Therefore the dot product acts as an affordance for triggering replanning:



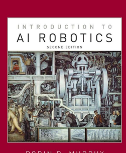
14

Summary

- Metric path planners
  - graph-based (A*, D* is best known)
  - Wavefront

- Graph-based generate paths and subgoals.
  - Good for NHC styles of control
  - In practice leads to:
    - Subgoal obsession
    - Termination conditions

- Planning all possible paths helps with subgoal obsession
  - What happens when the map is wrong, things change, missed opportunities? How can you tell when the map is wrong?



14 Returning to Questions…

- What is the difference between topological navigation and metric navigation/path planning?
  - One focuses on sensed routes, the other on maps
- What is commonly used or works good enough?
  - A* or D* variant which considers the actual controls aspect (i.e., non-holonomic characteristics of a robot and its velocity and trajectory)
- How much path planning do you need?
  - It depends. Moore's Law has led to where it is possible to replan at each step but interleaving planning and execution may be more elegant & free up CPU