A* optimality proof, cycle checking

Alan Mackworth

UBC CS 322 – Search 5

January 18, 2013

Textbook § 3.6 and 3.7.1

Lecture Overview

- Recap
- Admissibility of A*
- Cycle checking and multiple path pruning

Search heuristics

Def.: A search heuristic *h(n)* is an estimate of the cost of the optimal (cheapest) path from node *n* to a goal node.

- Think of *h(n)* as only using readily obtainable (easy to compute) information about a node.
- h can be extended to paths:

$$h(\langle n_0,\dots,n_k \rangle)=h(n_k)$$

Def.: A search heuristic *h(n)* is admissible if it never overestimates the actual cost of the cheapest path from a node to the goal

How to Construct a Heuristic

Identify relaxed version of the problem:

- where one or more constraints have been dropped
- problem with fewer restrictions on the actions

Result:

The cost of an optimal solution to the relaxed problem is an admissible heuristic for the original problem.

Because it is always weakly less costly to solve a less constrained problem!

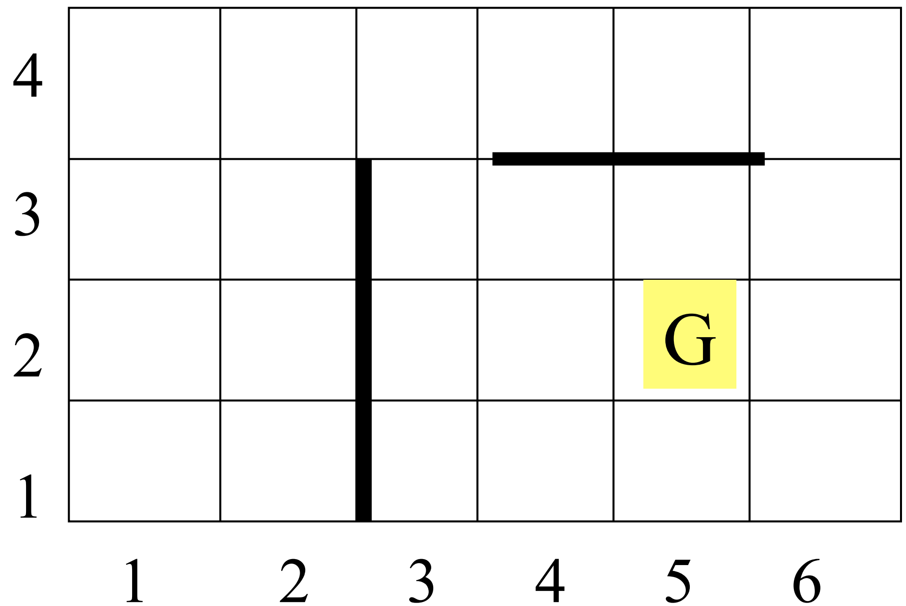
Example 2

Search problem: robot has to find a route from start to goal location on a grid with obstacles

Actions: move *up, down, left, right* from tile to tile

Cost : number of moves

Possible h(n)? *Manhattan distance (L_1 norm) between two points = sum of the (absolute) difference of their coordinates =* $|x_2 - x_1| + |y_2 - y_1|$



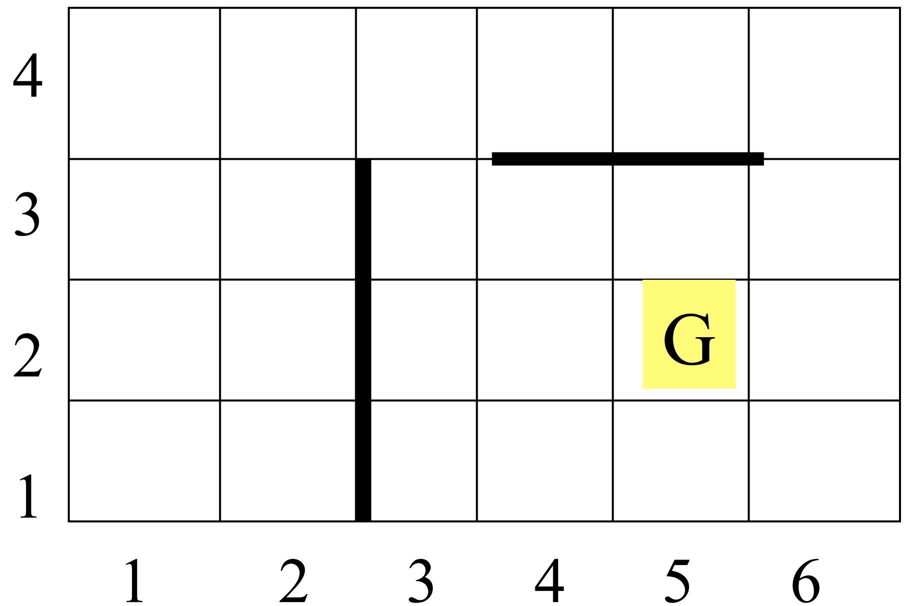
Example 2

Search problem: robot has to find a route from start to goal location on a grid with obstacles

Actions: move *up, down, left, right* from tile to tile

Cost : number of moves

Possible h(n)? *Would the Euclidean distance (straight line distance, L_2 norm) be an admissible heuristic?*



Would the Euclidean distance (straight line distance) be an admissible heuristic for the robot grid problem?

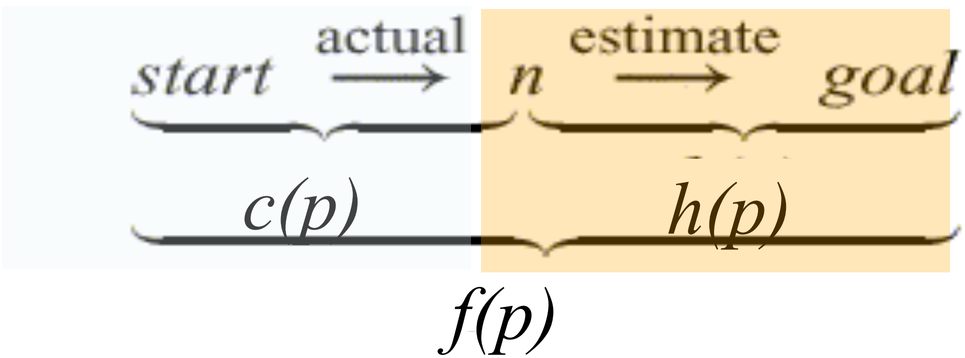
It is an admissible search heuristic

It is a search heuristic, but it is not admissible

It is not a suitable search heuristic for this problem

A* Search

- A* search takes into account both
  - the cost of the path to a node *c(p)*
  - the heuristic value of that path *h(p).*
- Let $f(p) = c(p) + h(p)$.
  - estimate of the cost of a path from the start to a goal via *p*.



- A* always chooses the path on the frontier with the lowest ***estimated*** distance from the start to a goal node constrained to go via that path.

Lecture Overview

- Recap of Lecture 8
- Admissibility of A*
- Cycle checking and multiple path pruning

Admissibility of A*

- A* is complete (finds a solution, if one exists) and optimal (finds the optimal path to a goal) if:

  - *the branching factor is finite*
  - *arc costs are* $> \varepsilon > 0$
  - *h(n) is admissible -> an underestimate of the length of the shortest path from n to a goal node.*

- This property of A* is called admissibility of A*

Why is A* admissible: complete

- It halts (does not get caught in cycles) because:
  - Let f_{min} be the cost of the (an) optimal solution path s (unknown but finite if there exists a solution)
  - Each sub-path p of s has cost $f(p) \leq f_{min}$
    - Due to admissibility (exercise: prove this at home)
  - Let $c_{min} = \varepsilon > 0$ be the minimal cost of any arc
    - All paths with length > f_{min} / c_{min} have cost > f_{min}
  - A* expands path on the frontier with minimal f(n)
    - Always a prefix of s on the frontier
    - Only expands paths p with $f(p) \leq f_{min}$
    - Terminates when expanding s

See how it works on the "misleading heuristic" problem in AIspace:

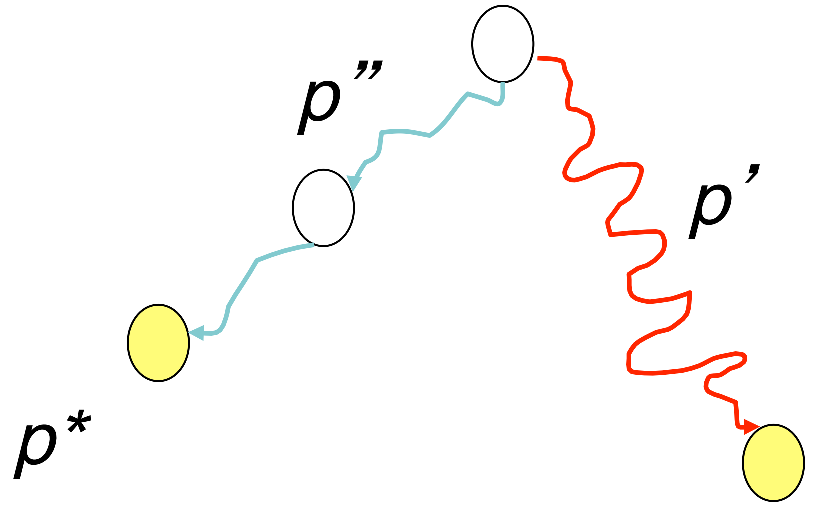
Compare A* with best-first.

Why is A* admissible: optimal

- Let p^* be the optimal solution path, with cost c^*.
- Let p' be a suboptimal solution path. That is $c(p') > c^*$.

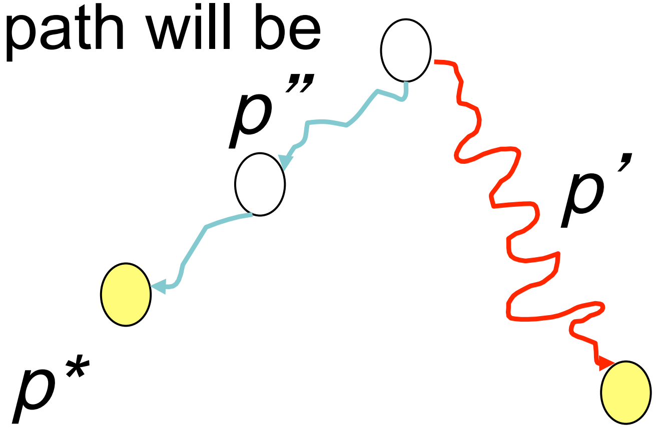
We are going to show that any sub-path p'' of p^* on the frontier will be expanded before p'.

Therefore, A* will find p^* before p'



Why is A* admissible: optimal

- Let *p** be the optimal solution path, with cost *f(p*)*.
- Let *p'* be a suboptimal solution path. That is
  c(p') > f(p).*
- Let *p''* be a sub-path of p* on the frontier.
- We know that f(p*) < f(p') because at a goal node
  f(goal) = c(goal)
- And f(p'') <= f(p*) because *h(.)* is admissible
- Thus f(p'') < f(p')
- Any sub-path of the optimal solution path will be expanded before p'



Analysis of A*

- In fact, we can prove something even stronger about A* (when it is admissible)

- A* is optimally efficient among the algorithms that extend the search path from the initial state.

- It finds the goal with the minimum # of path expansions

Why A* is Optimally Efficient

- No other optimal algorithm is guaranteed to expand fewer paths than A*

- This is because any algorithm that does not expand every node with $f(n) < f^*$ risks missing the optimal solution.

Effect of Search Heuristic

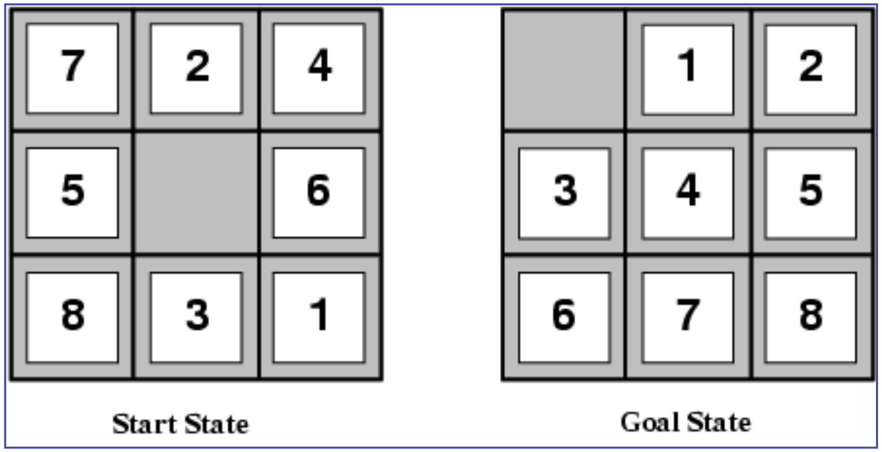
- A search heuristic that is a better approximation to the actual cost reduces the number of nodes expanded by A*
- Example: 8-puzzle
  - tiles can move (jump) anywhere:

    $h_1(n)$: number of tiles that are out of place
  - tiles can move to any adjacent square

    $h_2(n)$: sum of number of squares that separate each tile from its correct position
- average number of paths expanded:

  (d = depth of the solution)
- *d=12* BFS: 3,644,035 paths
  $A^*(h_1)$: 227 paths expanded
  $A^*(h_2)$: 73 paths expanded
- *d=24* BFS = too many paths
  $A^*(h_1)$: 39,135 paths expanded

  $A^*(h_2)$: 1,641 paths expanded



Time Space Complexity of A^*

- Time complexity is $\tilde{O}(b^m)$ the heuristic could be completely uninformative and the edge costs could all be the same, meaning that A^* does the same thing as BFS.

- Space complexity is $O(b^m)$ like BFS, A^* maintains a frontier which grows with the size of the tree.

Learning Goals for today's class

- Formally prove A* optimality
- Define optimally efficient
- Construct admissible heuristics for specific problems.

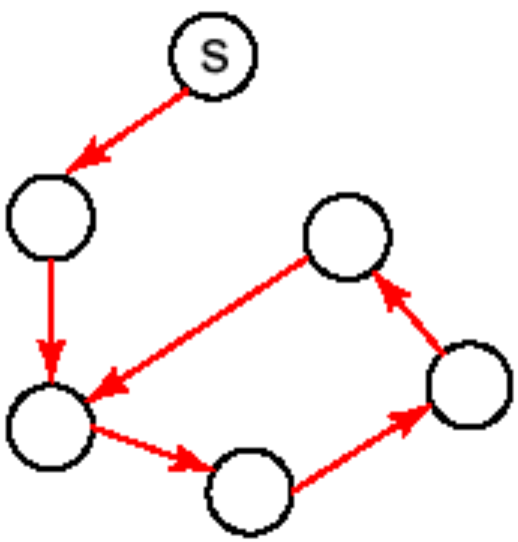
Lecture Overview

- Recap of Lecture 8
- Admissibility of A*
- Cycle checking and multiple path pruning

Cycle Checking

- You can prune a node *n* that is on the path from the start node to n.
- This pruning cannot remove an optimal solution => cycle check

- What is the computational cost of cycle checking?



Computational Cost of Cycle Checking?

Constant time: set a bit to 1 when a node is selected for expansion, and never expand a node with a bit set to 1

Linear time in the path length: before adding a new node to the currently selected path, check that the node is not already part of the path

It depends on the algorithm

None of the above

See P&M text, Section 3.7.1, p.93