Search: Advanced Topics

CPSC 322 Lecture 9

Learning Goals for this class

- Define/read/write/trace/debug different search algorithms
    - With / Without cost
    - Informed / Uninformed

- Justify and describe methods for pruning cycles and repeated states (multiple paths)

Lecture Overview

- **Branch & Bound**
- A^* tricks
- Pruning Cycles and Repeated States
- Dynamic Programming

Branch-and-Bound Search

- Biggest advantages of A*
  - informed
  - optimal
  - optimally efficient
- What is the biggest problem with A*?
  - space
- Possible, preliminary solution:

Branch-and-Bound Search Algorithm

- Follow exactly the same search path as **depth-first search**
  - **treat the frontier as a stack**: expand the most-recently added path first

Once this strategy has found a solution….

What should it do next ?

A. Keep searching, looking for deeper solutions
B. Stop and return that solution
C. Keep searching, but only for shorter solutions
D. None of the above
E. Create a startup that it can sell to Google for billions of dollars

Branch-and-Bound Search Algorithm

Keep track of a lower bound and upper bound on solution cost at each path

- lower bound: $LB(p) = f(p) = cost(p) + h(p)$
- upper bound: UB = cost of the best solution found so far.
  - Initialize UB to ∞ (or some finite **overestimate** of the solution cost).

When a path *p* is selected for expansion:

- if $LB(p) \geq UB$, remove *p* from frontier without expanding it (pruning)
- else expand *p*, adding all of its neighbors to the frontier

Branch-and-Bound Analysis

iclicker.

- Complete?

yes | no | it depends

- Optimal?

yes | no | it depends

- Space complexity?

$O(b^m)$ | $O(m^b)$ | $O(bm)$ | $O(b+m)$

- Time complexity?

$O(b^m)$ | $O(m^b)$ | $O(bm)$ | $O(b+m)$

Branch-and-Bound Analysis

- Completeness: **not in general**, for the same reasons that DFS isn't complete
  - however, for many problems of interest there are no infinite paths and no cycles
  - also, you may be able to initialize the upper bound to some large finite number that is an overestimate of the solution cost
  - hence, for many problems B&B is complete
- Time complexity: **$O(b^m)$**
- Space complexity: **O(mb)** (like DFS!)
  - Big improvement over A*
- Optimality: **YES, but not optimally efficient**

A note on B&B and AIspace

The AIspace search applet performs B&B slightly differently than is covered here in the lectures

- sometimes it expands a goal node even if that goal node shouldn't have been expanded next (according to how we've set up the algorithm)
- So be careful if using the applet to check your B&B tracethroughs

Lecture Overview

- Branch & Bound
- **A^* tricks**
- Pruning Cycles and Repeated States
- Dynamic Programming

Other A^* Enhancements

The main problem with A^* is that (in the worst case) it uses exponential space. Branch and bound was one way around this problem. Are there others?

- Iterative Deepening A* (IDA*)
- Memory-bounded A^*

(Heuristic) Iterative Deepening – IDA*

B & B can still get stuck in infinite (extremely long) paths

- Search **depth-first**, but to a **fixed depth/bound**
  - depth is measured in **f-values**
  - if you don't find a solution, **update the bound** with the **lowest f** that passed the previous bound, and try again

Analysis of Iterative Deepening A* (IDA*)

- Complete and optimal:

yes | no | it depends

- Space complexity:

$O(b^m)$ | $O(m^b)$ | $O(bm)$ | $O(b+m)$

- Time complexity:

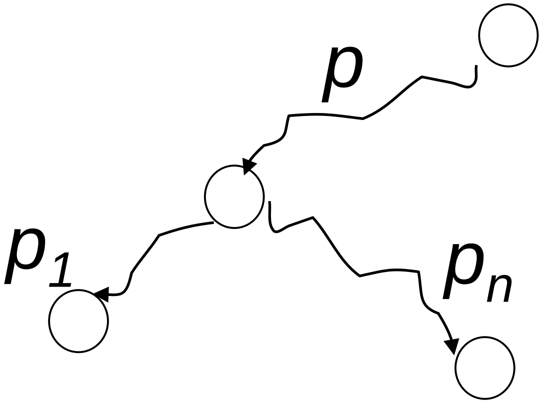
$O(b^m)$ | $O(m^b)$ | $O(bm)$ | $O(b+m)$

(Heuristic) Iterative Deepening – IDA*

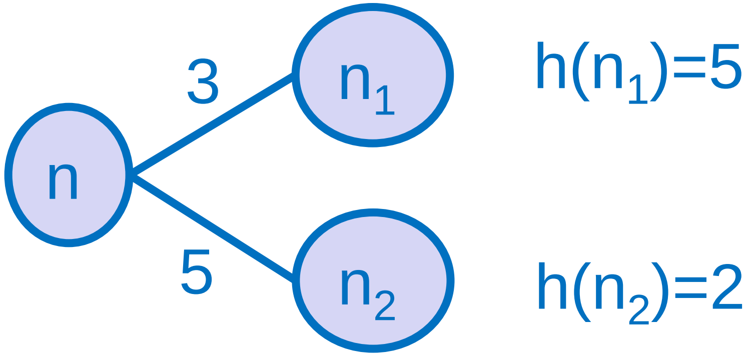
- Counter-intuitively, the asymptotic time complexity is not changed, even though we visit paths multiple times (***as we saw in previous slides on IDS***)

Memory-bounded A^*

- IDA* and B&B use a tiny amount of memory
- **what if we have more memory available?**
- keep as much of the frontier in memory as we can
- if we have to delete something:
  - delete the "worst" paths (with highest f-values.)
  - "back them up" to a common ancestor
  - **Update the heuristic value of the ancestor if possible**



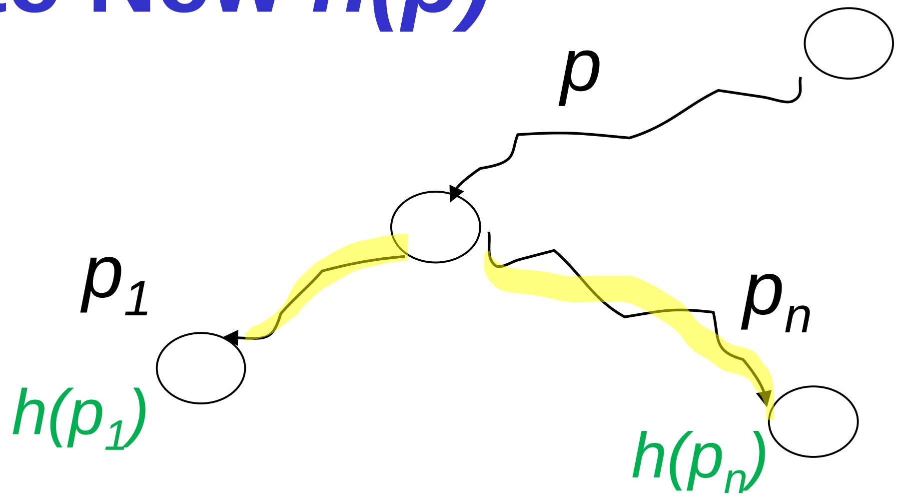
Heuristic value by look ahead



What is the most accurate admissible heuristic value for n, given only this info ?

A. 7
B. 5
C. 2
D. 8
E. 42

MBA*: Compute New *h(p)*



A $$\text{New } h(p) = \min\left(\max_i [(cost(p_i) - cost(p)) + h(p_i)], \text{Old } h(p)\right)$$

B $$\text{New } h(p) = \max\left(\min_i [(cost(p_i) - cost(p)) + h(p_i)], \text{Old } h(p)\right)$$

C $$\text{New } h(p) = \max\left(\max_i [(cost(p_i) - cost(p)) + h(p_i)], \text{Old } h(p)\right)$$

Search Summary Table

	complete?	optimal?	time O( )	space O( )
DFS	No	No	b^m	mb
BFS	Yes	Yes*	b^m	b^m
IDS	Yes	Yes*	b^m	mb
LCFS	Yes^	Yes^	b^m	b^m
BestFS	No	No	b^m	b^m
A*	Yes^	Yes$^{\wedge +}$	b^m	b^m
B&B	No	Yes^{+}	b^m	mb
IDA*	Yes^	Yes$^{\wedge +}$	b^m	mb
MBA*	Yes$^{\wedge \#}$	Yes$^{\wedge + \#}$	b^m	b^m

* arc costs are equal   # enough memory to store a solution

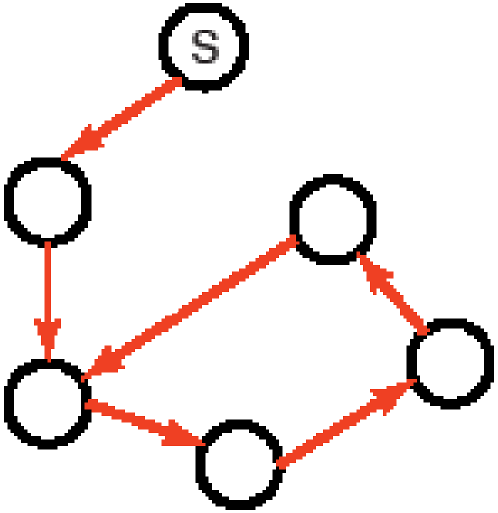
^ arc costs are positive

$^{+}$ h(n) is admissible and non-negative

Lecture Overview

- Branch & Bound
- A^* tricks
- Pruning Cycles and Repeated States
- Dynamic Programming

Cycle Checking

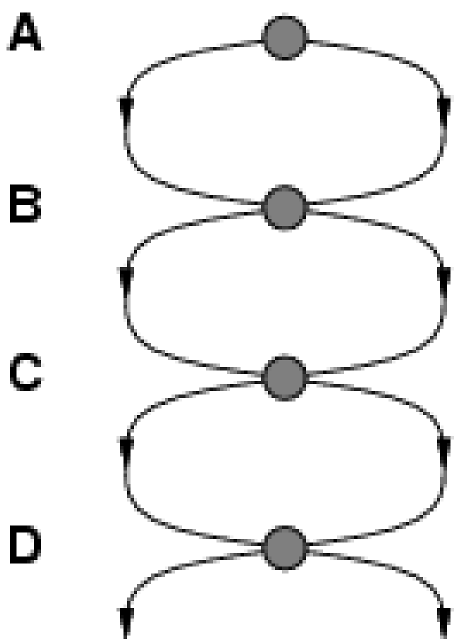


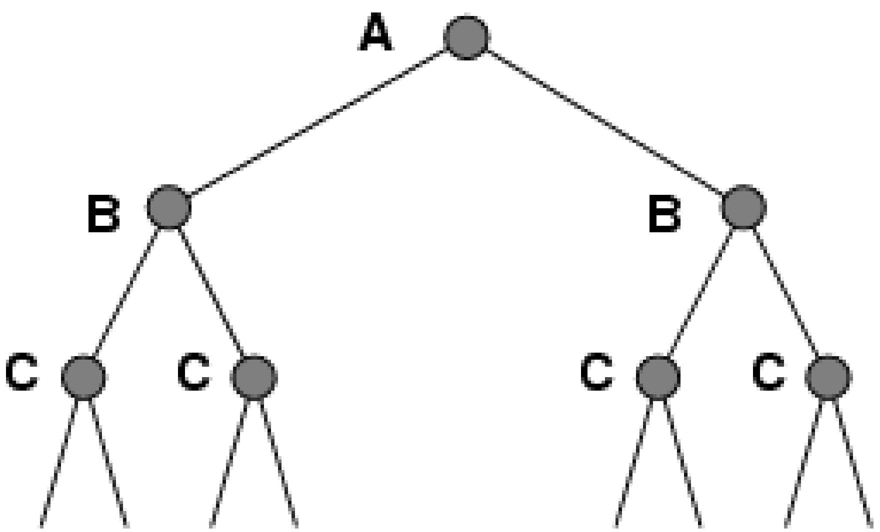
You can prune a path that ends in a node already on the path. This pruning cannot remove an optimal solution.

- In general, the time is **linear** in path length.

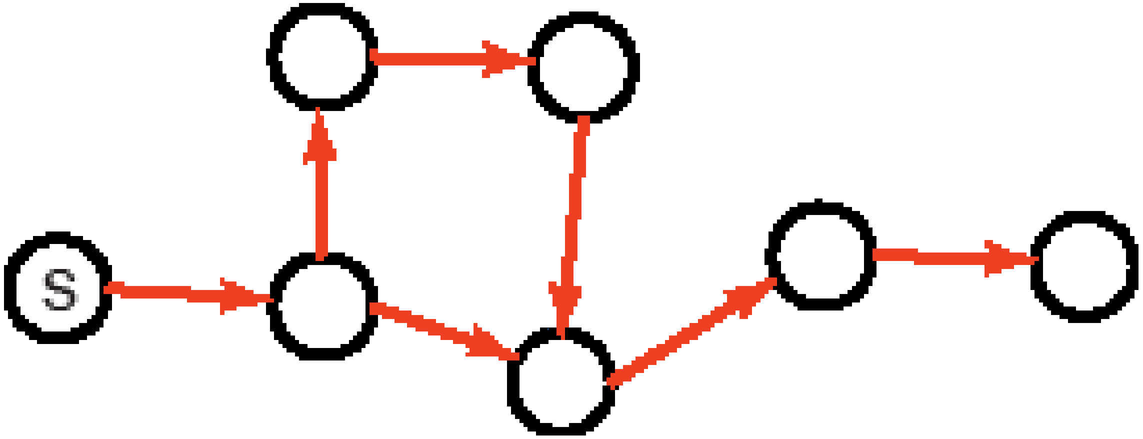
Repeated States / Multiple Paths

Failure to detect repeated states can turn a linear problem into an exponential one!





Multiple-Path Pruning



- You can prune a path to node ***n*** that you have already found a path to
- (if the new path is longer – more costly).

Multiple-Path Pruning & Optimal Solutions

Problem: **what if a subsequent path to *n* is shorter than the first path to *n* ?**

- You can remove all paths from the frontier that use the longer path. (as these can't be optimal)

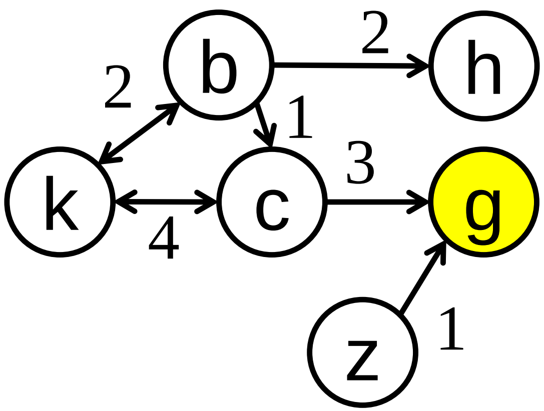
- You can also change the initial segment of the paths on the frontier to use the shorter path

Lecture Overview

- Branch & Bound
- A^* tricks
- Pruning Cycles and Repeated States
- **Dynamic Programming**

Dynamic Programming

- Idea: for statically stored graphs, build a table of dist(n):
  - The actual distance of the shortest path from node n to a goal g
  - This is the perfect **f function** **heuristic** cost



- dist(g) = 0
- dist(z) = 1
- dist(c) = 3
- dist(b) = 4
- dist(k) = ? **6** **7** ∞
- dist(h) = ? **6** **7** ∞

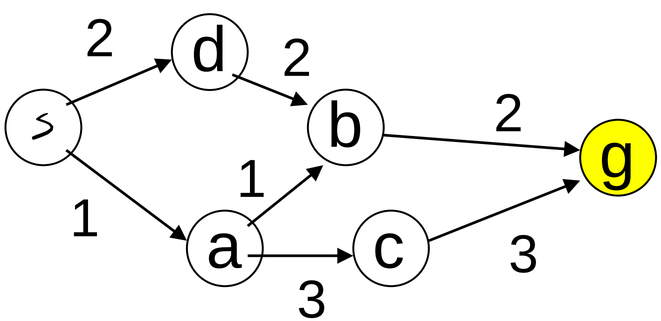
- How could we implement that?

Dynamic Programming

This can be built backwards from the goal:

$$dist(n) = \begin{cases} 0 & if \quad is\_goal(n), \\ \min_{\langle n,m \rangle \in A} \left(\text{cost}(n,m) + dist(m) \right) & otherwise \end{cases}$$

n	dist(n)
g	0
b	min[(2+0)] = 2
c	min[(3+0)] = 3
a	min[(3+3),(1+2)] = 3

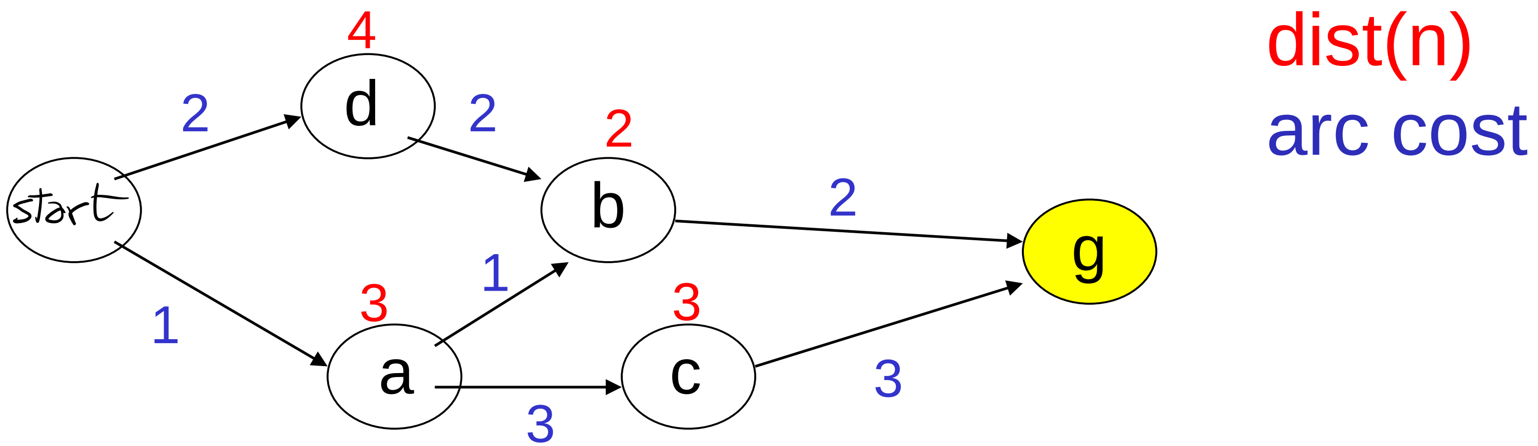


Dynamic Programming

This can be used locally to determine what to do.

From each node ***n*** go to its neighbor which minimizes

$$\left(\mathrm{cost}(\mathrm{n},\mathrm{m}) + dist(m)\right)$$



But there are at least two main problems:

- You need enough space to store the graph.
- The *dist* function needs to be recomputed for each goal

Next class

- Start Constraint Satisfaction Problems (CSP)
  - Chp 4.

- Keep working on Assignment 1!