Search: Advanced Topics

Computer Science cpsc322, Lecture 9

(Textbook Chpt 3.6)

January, 22, 2010

Course Announcements

Posted on WebCT

- Answers for Second Practice Exercise (uninformed Search)
- Third Practice Exercise : Heuristic Search

Lecture Overview

$f = \underline{c + h}$

- **Recap A* (applications…)**
- Branch & Bound
- A* tricks
- Other Pruning
- Dynamic Programming

Sample A* applications

- **An Efficient A* Search Algorithm For Statistical** Machine Translation. 2001
- **The Generalized A* Architecture**. Journal of Artificial Intelligence Research (2007)
  - Machine Vision … Here we consider a new compositional model for finding salient curves.
- **Factored A*search for models over sequences and trees** International Conference on AI. 2003…. It starts saying… *The primary challenge when using A* search is to find heuristic functions that simultaneously are admissible, close to actual completion costs, and efficient to calculate…* applied to NLP and BioInformatics

(Natural Language Processing)

Lecture Overview

- Recap A* (applications…)
- Branch & Bound → space
- A^* tricks
- Other Pruning
- Dynamic Programming

Branch-and-Bound Search

- What is the biggest advantage of A*?

uses heuristics

- What is the biggest problem with A*?

space

- Possible Solution:

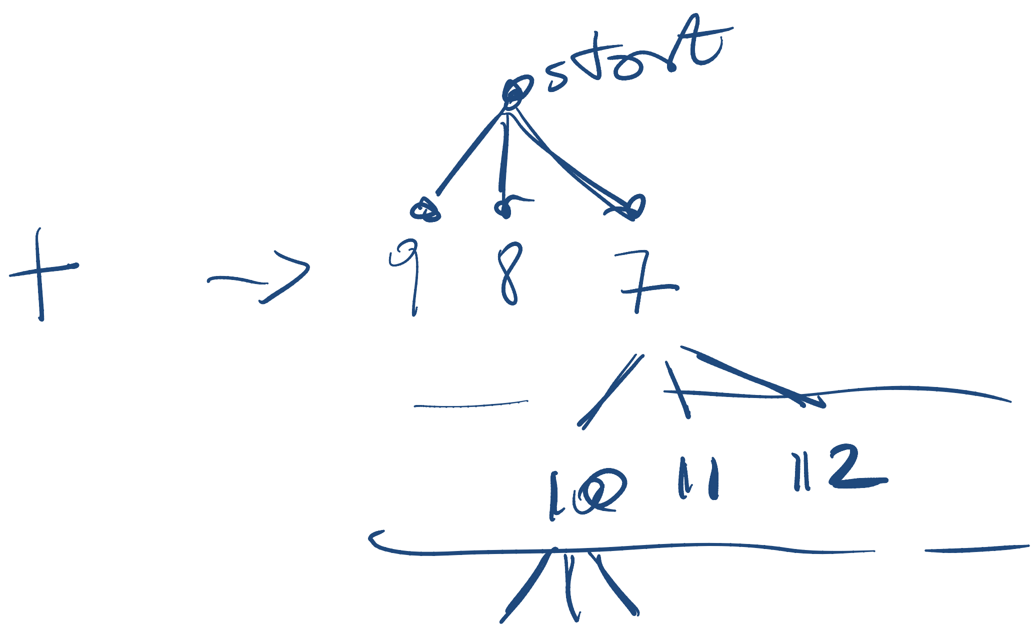
DFS + h

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
  - the order in which neighbors are expanded can be governed by some arbitrary node-ordering heuristic

we can use

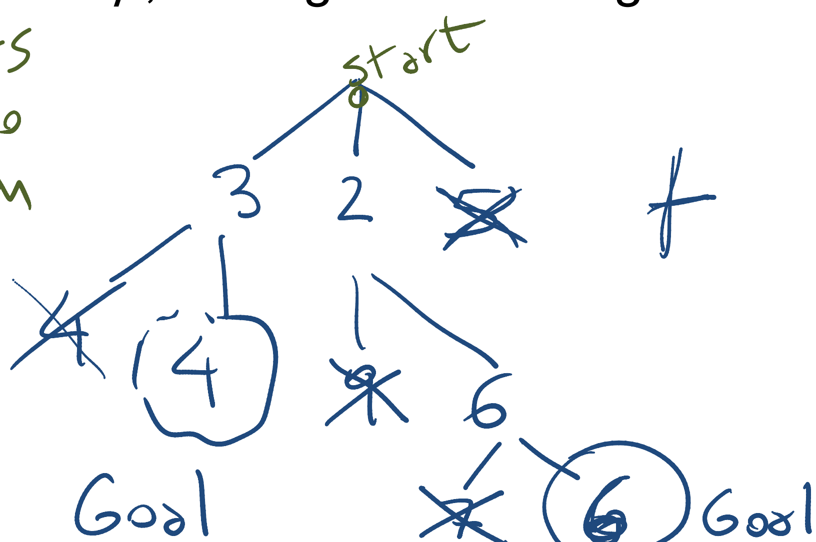
$f = c + h$



Branch-and-Bound Search Algorithm

- Keep track of a lower bound and upper bound on solution cost at each path
  - lower bound: $LB(p) = \boxed{f(p)} = cost(p) + h(p)$
  - upper bound: $UB =$ cost of the best solution found so far.
    - ✓ if no solution has been found yet, set the upper bound to ∞.
- 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

The numbers correspond to f for the path from start to that node



f

$UB = \infty$

6

4

same for all paths at any given time

Branch-and-Bound Analysis

- Completeness: no, for the same reasons that DFS isn't complete
  - however, for many problems of interest there are no infinite paths and no cycles
  - hence, for many problems B&B is complete
- Time complexity: $O(b^m)$
- Space complexity: .... $O(bm)$
  - Branch & Bound has the same space complexity as.... DFS
  - this is a big improvement over ....! A*
- Optimality: .... yes

Lecture Overview

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

Other A^* Enhancements

The main problem with A^* is that 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
  - if you don't find a solution, increase the depth tolerance and try again
  - depth is measured in.....f.............

start node $f(\text{start}) = h(\text{start})$

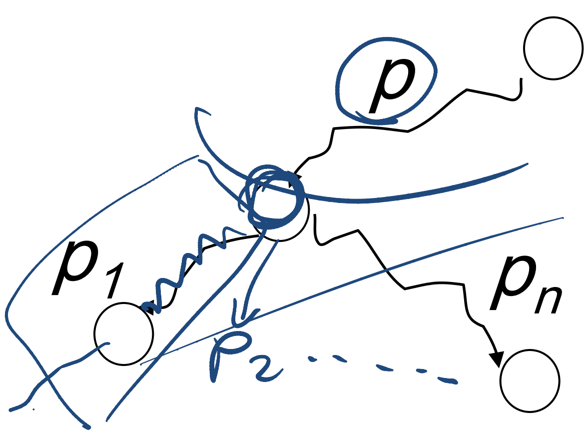
then update with the lowest f that passed the previous bound

- Counter-intuitively, the asymptotic complexity is not changed, even though we visit paths multiple times (*go back to slides on uninformed ID*)

$\left(\frac{b}{b-1}\right)^2$

Memory-bounded *A**

- Iterative deepening A* and B & B use a tiny amount of memory
- **what if we've got more memory to use?**
- keep as much of the fringe in memory as we can
- if we have to delete something:
  - delete the worst paths (with ....highest........f.........)
  - ``back them up'' to a common ancestor

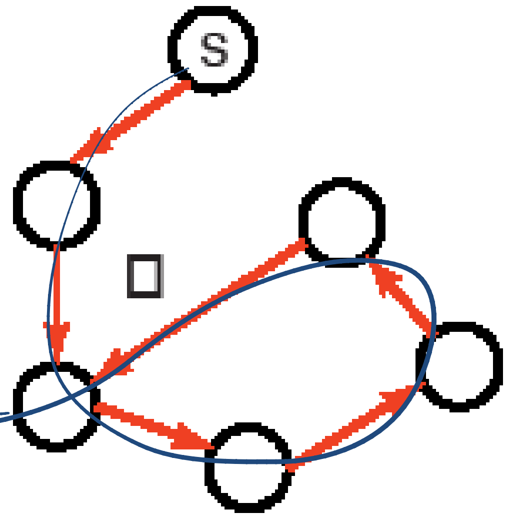


$$h(p) = \max\left(\min\left[cost(p_i) - cost(p)\right] + h(p_i),\ \text{original } h(p)\right)$$

Lecture Overview

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

Cycle Checking



OK
Good

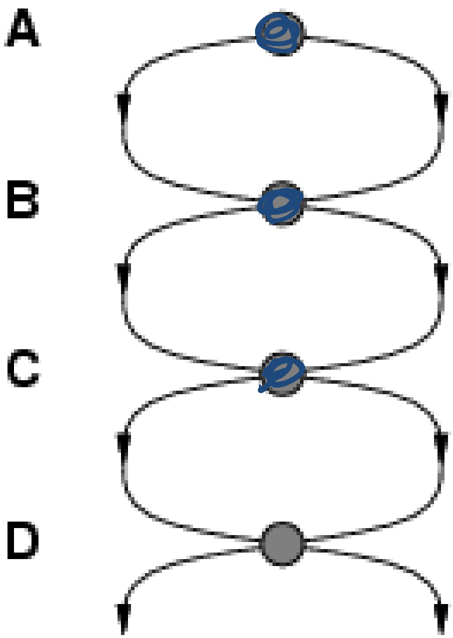
You can prune a path that ends in a node already on the path. This pruning cannot remove an optimal solution.

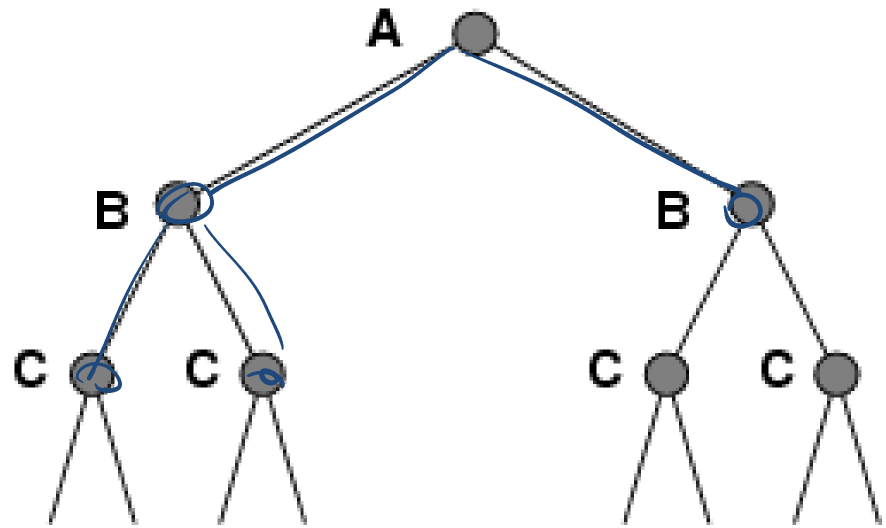
- The time is …linear…… in path length.

$\langle n_0, n_1, n_2 \ldots n_k \rangle$ n_2 n_i n_j

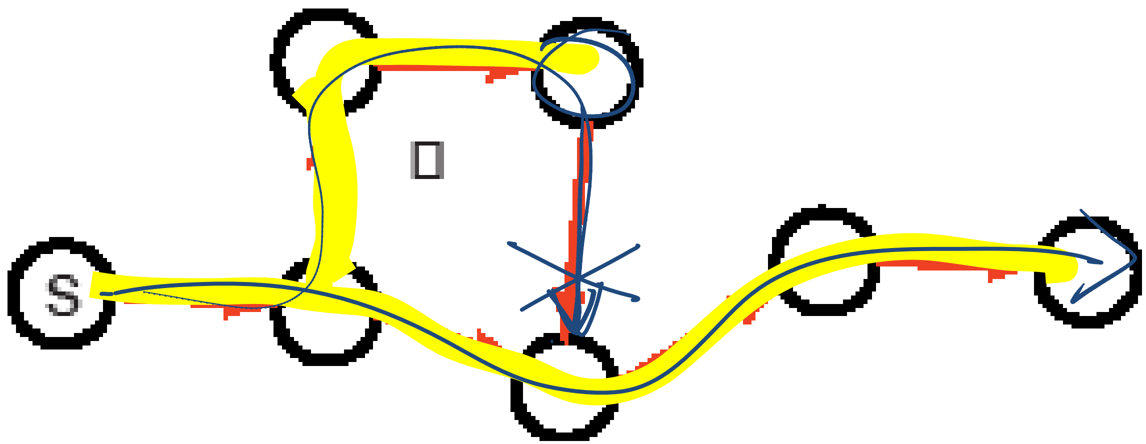
Repeated States / Multiple Paths

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





Multiple-Path Pruning

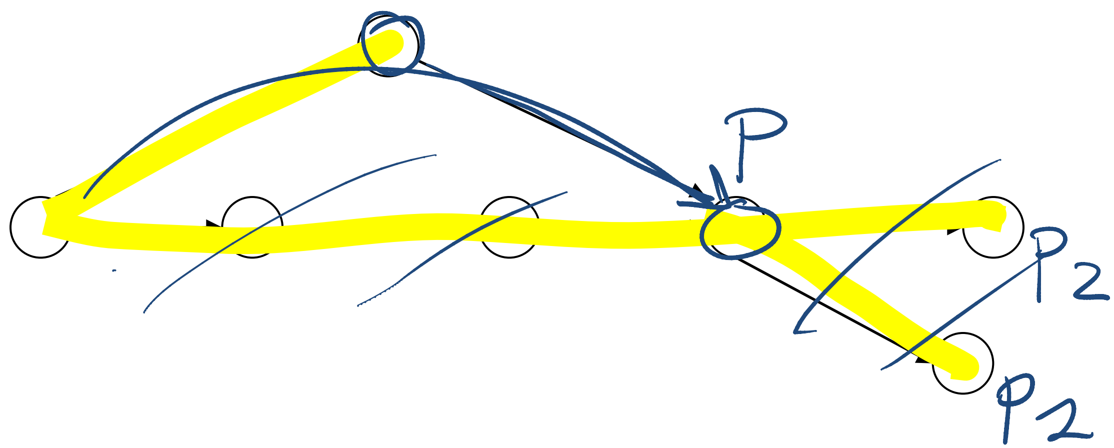


- You can prune a path to node ***n*** that you have already found a path to
- (if the new path is longer – more costly).

Multiple-Path Pruning & Optimal Solutions

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

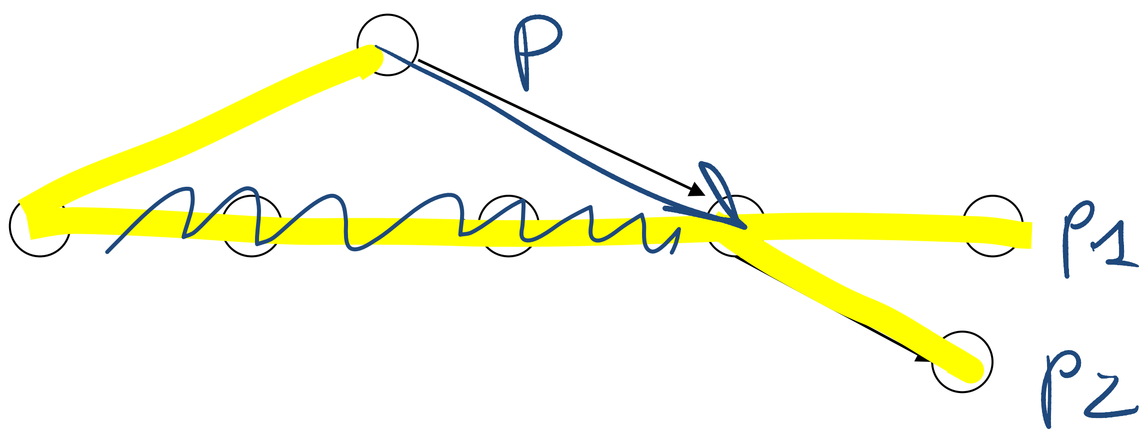
- You can remove all paths from the frontier that use the longer path. (as these can't be optimal)



Multiple-Path Pruning & Optimal Solutions

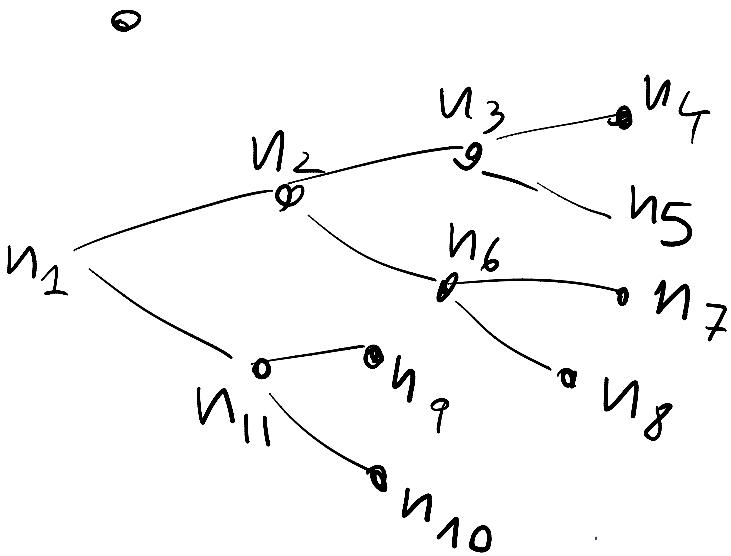
Problem: what if a subsequent path to *n* is shorter than the first path to *n* ?

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



Example

Pruning Cycles



neighbors of $n_4 = \{n_2, n_{11}, n_{14}, n_{15}\}$

Repeated States

neighbors of $n_{10} =$
$\{n_{15}, n_{16}\}$

NEXT CLASS

Lecture Overview

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

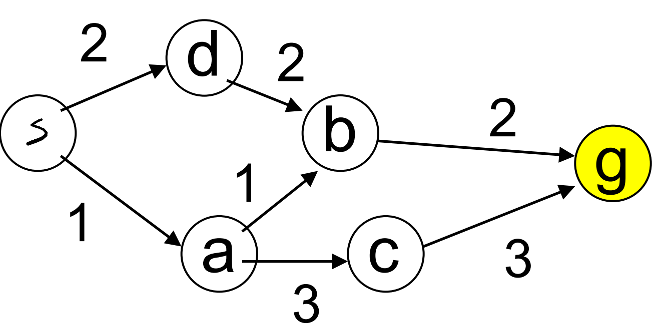
Dynamic Programming

Idea: for statically stored graphs, build a table of *dist(n)* the actual distance of the shortest path from node *n* to a goal.

This is the perfect……..

This can be built backwards from the goal:

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



g

b

c

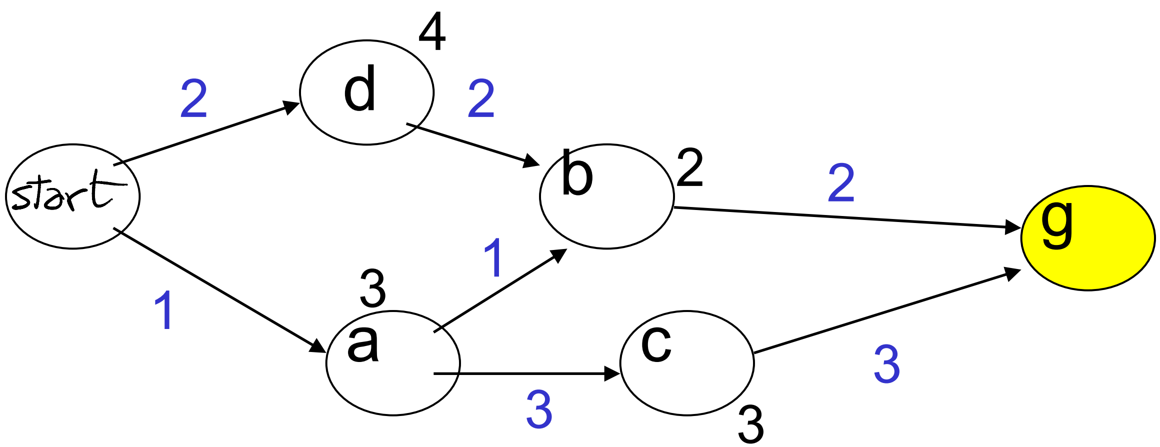
a

Dynamic Programming

This can be used locally to determine what to do.

From each node n go to its neighbor which minimizes

$$\underline{cost(n,m)} + dist(m)$$



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

Learning Goals for today's class

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

- Pruning cycles and Repeated States

Next class

Recap Search

Start Constraint Satisfaction Problems (CSP)

Chp 4.