Search: Advanced Topics

Computer Science cpsc322, Lecture 9

(Textbook Chpt 3.6)

May, 23, 2013
Lecture Overview

- Recap A*
- Branch & Bound
- A* tricks
- Other Pruning
A* advantages

What is a key advantage of A*?

A. Does not need to consider the cost of the paths
B. Has a linear space complexity
C. It is often optimal
D. None of the above
Branch-and-Bound Search

- Biggest advantages of A*:

  uses heuristics + optimal

- What is the biggest problem with A*?

  space

- Possible, preliminary 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

![Diagram of search tree]

- We can use $T = c + h$
Once this strategy has found a solution….

What should it do next?

A. Keep running DFS, looking for deeper solutions?
B. Stop and return that solution
C. Keep searching, but only for shorter solutions
D. None of the above
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.
    ✓ if no solution has been found yet, set the upper bound to $\infty$.

• 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

- 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?
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
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
  - if you don’t find a solution, increase the depth tolerance and try again
  - depth is measured in $f$ (start node) $= h$(start) $+$

- Then update with the lowest $f$ that passed the previous bound
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 complexity is not changed, even though we visit paths multiple times
  (go back to slides on uninformed ID)

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

because

\[
\min_{n} [ \text{cost}(n, n_i) + h(n_i) ]
\]
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

[Diagram showing a tree with nodes $p_1$, $p_2$, ..., $p_n$, and $p$ connecting them]
MBA*: Compute New $h(p)$

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

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

New $h(p) = \max_i \max [(\text{cost}(p_i) - \text{cost}(p)) + h(p_i)], \text{Old } h(p)$
MBA*: Compute New $h(p)$

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

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

C. New $h(p) = \max \left\{ \max_i \left[ (\text{cost}(p_i) - \text{cost}(p)) + h(p_i) \right], \text{Old } h(p) \right\}$
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 $\max$)
  - ``back them up'' to a common ancestor

$$h(p) = \min_{i} \left[ \max \left( \min \left( \text{cost}(p_i) - \text{cost}(p) \right) \right) + h(p_i) \right]$$
Lecture Overview

- Recap A*
- Branch & Bound
- A* tricks
- Pruning Cycles and Repeated States
Cycle Checking

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

- The time is \( \text{linear} \) 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).
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.
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: Thurs

- Dynamic Programming
- Recap Search
- Start Constraint Satisfaction Problems (CSP)
- Chp 4.

- Start working on assignment-1!