CSP Introduction

CPSC 322 – CSPs 1

Textbook §4.0 – 4.2

Lecture Overview

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 node first
  - the order in which neighbors are expanded can be governed by some arbitrary node-ordering heuristic
- Keep track of a lower bound and upper bound on solution cost at each node
  - lower bound: $LB(n) = cost(n) + h(n)$
  - upper bound: $UB = cost(n')$, where n' is the best solution found so far.
    - if no solution has been found yet, set the upper bound to ∞.
- When a node n is selected for expansion:
  - if $LB(n) \geq UB$, remove n from frontier without expanding it
    - this is called "pruning the search tree" (really!)
  - else expand n, adding all of its neighbours to the frontier

Other Search Ideas

The main problem with A^* is that it uses exponential space.
Branch and bound was one way around this problem.
Two others are:

- Iterative deepening
- Memory-bounded $A*$

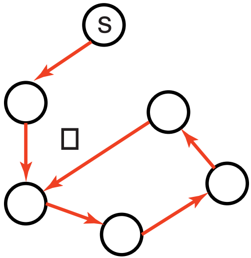
Lecture Overview

Non-heuristic pruning

What can we prune besides nodes that are ruled out by our heuristic?

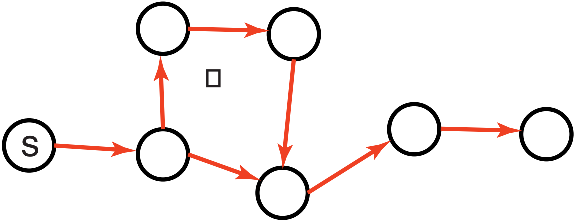
- Cycles
- Multiple paths to the same node

Cycle Checking



- You can prune a path that ends in a node already on the path. This pruning cannot remove an optimal solution.
- Using depth-first methods, with the graph explicitly stored, this can be done in constant time.
- For other methods, the cost is linear in path length.

Multiple-Path Pruning



- You can prune a path to node n that you have already found a path to.
- Multiple-path pruning subsumes a cycle check.
- This entails storing all nodes you have found paths to.

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.
- You can change the initial segment of the paths on the frontier to use the shorter path.
- You can ensure this doesn't happen. You make sure that the shortest path to a node is found first.
    - Heuristic function h satisfies the monotone restriction if $|h(m) - h(n)| \leq d(m, n)$ for every arc $\langle m, n \rangle$.
    - If h satisfies the monotone restriction, A^* with multiple path pruning always finds the shortest path to every node
        - otherwise, we have this guarantee only for goals

Lecture Overview

Direction of Search

- The definition of searching is symmetric: find path from start nodes to goal node or from goal node to start nodes.
  - Of course, this presumes an explicit goal node, not a goal test.
  - Also, when the graph is dynamically constructed, it can sometimes be impossible to construct the backwards graph
- Forward branching factor: number of arcs out of a node.
- Backward branching factor: number of arcs into a node.
- Search complexity is b^n. Should use forward search if forward branching factor is less than backward branching factor, and vice versa.

Bidirectional Search

- You can search backward from the goal and forward from the start simultaneously.
- This wins because $2b^{k/2} \ll b^k$. This can result in an exponential saving in time and space.
  - The main problem is making sure the frontiers meet.
  - This is often used with one breadth-first method that builds a set of locations that can lead to the goal. In the other direction another method can be used to find a path to these interesting locations.

Lecture Overview

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.

Initialize $dist(n) = \infty$ for each node n

Then repeatedly, until no $dist(n)$ value changes, set each $dist(n)$ value to the smallest (neighboring $dist(n')$ value + cost of reaching n' from n):

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

Dynamic Programming

The main problem is that you need enough space to store the graph.

Complexity: polynomial in the size of the graph.

- but so is DFS (in fact, it's linear)
- the gain is when there are lots of nested cycles

Lecture Overview

Variables

- Recall that we defined the state of the world as an assignment of values to a set of (one or more) variables
  - variable: a synonym for feature
  - we denote variables using capital letters
  - each variable V has a domain $dom(V)$ of possible values
- Variables can be of several main kinds:
  - Boolean: $|dom(V)| = 2$
  - Finite: the domain contains a finite number of values
  - Infinite but Discrete: the domain is countably infinite
  - Continuous: e.g., real numbers between 0 and 1
- We'll call the set of states that are induced by a set of variables the set of possible worlds

Examples

- Crossword Puzzle:
  - variables are words that have to be filled in
  - domains are English words of the correct length
  - possible worlds: all ways of assigning words

Examples

- Crossword Puzzle:
  - variables are words that have to be filled in
  - domains are English words of the correct length
  - possible worlds: all ways of assigning words
- Crossword 2:
  - variables are cells (individual squares)
  - domains are letters of the alphabet
  - possible worlds: all ways of assigning letters to cells

Examples

- Crossword Puzzle:
  - variables are words that have to be filled in
  - domains are English words of the correct length
  - possible worlds: all ways of assigning words
- Crossword 2:
  - variables are cells (individual squares)
  - domains are letters of the alphabet
  - possible worlds: all ways of assigning letters to cells
- Sudoku
  - variables are cells
  - domains are numbers between 1 and 9
  - possible worlds: all ways of assigning numbers to cells

More Examples

- Scheduling Problem:
  - variables are different tasks that need to be scheduled (e.g., course in a university; job in a machine shop)
  - domains are the different combinations of times and locations for each task (e.g., time/room for course; time/machine for job)
  - possible worlds: time/location assignments for each task

More Examples

- Scheduling Problem:
  - variables are different tasks that need to be scheduled (e.g., course in a university; job in a machine shop)
  - domains are the different combinations of times and locations for each task (e.g., time/room for course; time/machine for job)
  - possible worlds: time/location assignments for each task
- n-Queens problem
  - variable: location of a queen on a chess board
    - there are n of them in total, hence the name
  - domains: grid coordinates
  - possible worlds: locations of all queens

Lecture Overview

Constraints

Constraints are restrictions on the values that one or more variables can take

- Unary constraint: restriction involving a single variable
  - of course, we could also achieve the same thing by using a smaller domain in the first place
- k-ary constraint: restriction involving the domains of k different variables
  - it turns out that k-ary constraints can always be represented as binary constraints, so we'll often talk about this case
- Constraints can be specified by
  - giving a list of valid domain values for each variable participating in the constraint
  - giving a function that returns true when given values for each variable which satisfy the constraint
- A possible world satisfies a set of constraints if the set of variables involved in each constraint take values that are consistent with that constraint

Examples

- Crossword Puzzle:
  - variables are words that have to be filled in
  - domains are valid English words
  - constraints: words have the same letters at points where they intersect
- Crossword 2:
  - variables are cells (individual squares)
  - domains are letters of the alphabet
  - constraints: sequences of letters form valid English words
- Sudoku
  - variables are cells
  - domains are numbers between 1 and 9
  - constraints: rows, columns, boxes contain all different numbers

More Examples

- Scheduling Problem:
  - variables are different tasks that need to be scheduled (e.g., course in a university; job in a machine shop)
  - domains are the different combinations of times and locations for each task (e.g., time/room for course; time/machine for job)
  - constraints: tasks can't be scheduled in the same location at the same time; certain tasks can't be scheduled in different locations at the same time; some tasks must come earlier than others; etc.
- n-Queens problem
  - variable: location of a queen on a chess board
  - domains: grid coordinates
  - constraints: no queen can attack another