CIS 192: Artificial Intelligence

Search and Constraint Satisfaction

Alex Frias

Nov. 30th
What is AI?

• Designing computer programs to complete tasks that are thought to require ‘intelligence’

• 4 categories
  – Think like a human
  – Act like a human
  – Think rationally
  – Act rationally
Creating ‘Agents’ that Act Rationally

• What is an agent?
  – Anything that perceives its environment and acts upon it

• Want the program to maximize some goal
  – i.e. win a game, solve a puzzle, drive a car

• Agent perceives its environment, and a function acts upon precepts it receives
DARPA Grand Challenge

• Race by autonomous cars across 150-mile desert

• Held in 2004, 2005, and 2012 Urban Challenge
  – In 2004, farthest car got 7.32 miles
  – In 2005, 5 cars completed the course

• How do the cars ‘see’ the road and know where to go?
Search Problems

• AI problems involving searching a search space for solution

• Examples:
  – Shortest path
  – Driving a car (computer vision)
  – Parsing grammar
  – Scheduling (Hubble telescope)
Finding the fastest route

• Google Maps:
  – Want to find the fastest route from start to end point

• Challenges:
  – Need to formulate search space
  – Need to represent routes
  – Need to find the optimal path
How to solve search problems

• Formal math definitions:
  – States, actions, transitions, path costs, goal tests
• Generally: Formulate problem, set goal, search for solution
• Need to find optimal path, not just any path
• Solution: Graph Search
Graph Search

• Why do we have an explored set?
  – Greatly reduce the search space since we don’t want to go backwards
  – Increases memory usage, however

• Type of search is dependent on how we remove nodes from the frontier
BFS vs. DFS vs. Iterative Deepening

• BFS:
  – Complete, optimal, $O(b^d)$ time and space

• DFS:
  – Not complete, not optimal, only $O(bm)$ space

• Iterative Deepening DFS:
  – Counter-intuitive
  – Complete, optimal, only $O(bd)$ space

• $d$ is depth of solution, $b$ is branching factor, and $m$ is maximum possible depth
Uniform Cost Search

• So far, we’ve assumed uniform path costs
• Uniform Cost Search (UCS) is used when paths do not have a uniform cost (it’s a bad name)
• Similar to graph search, but use priority queue to order nodes by path cost
  – Remove the node with the shortest path cost next
• Python has a PriorityQueue implementation in the queue class
  – Need to store a tuple of (path cost, (current state, path)) in the priority queue
Informed Search

• Why should we search the entire search space if we know some parts are bad?
  – E.g. Why would we search backwards paths unless we have to?

• Use heuristics to only search the promising paths

• Leads us to the A* algorithm
A*

• Best known informed search algorithm
• Simple idea:
  – \( F(n) = g(n) + h(n) \)
  – We order priority queue on \( F(n) \)
  – \( g(n) \) is the path cost so far
  – \( h(n) \) is the heuristic of future cost
• \( h(n) \) must never overestimate the distance to the goal
• Code is basically the same as UCS
• [http://www.algomation.com/player?algorithm=545ec29d01f03a02007b101d](http://www.algomation.com/player?algorithm=545ec29d01f03a02007b101d)
Constraint Satisfaction

• Idea: Why bother searching when we can eliminate bad paths from the start?
  – i.e. If we know that placing a 3 in Sudoku would be wrong, why bother searching those paths with a 3 for a solution?

• Constraint Satisfaction Problems consist of:
  – A set of variables
  – A domain for each variable
  – A set of constraints
Examples: Sudoku and Map Coloring
AC-3

• Idea: Constraint Propagation/Inference
• Arc – a constraint between two variables
• ‘Arc-Consistency’
  – Idea: If a value for a variable gives another variable no possible values, this cannot possibly be a solution, so we can stop there
  – We can now propagate these constraints as we pick more values
AC-3: Steps

• Formulate all of the constraints, and add them to a queue
• Get a two variables from the queue
• Remove any inconsistent values between those variables
  – An inconsistent value is one that leaves another variable with no possible values
• Add all of the neighboring constraints to the queue to recheck
• Repeat until solution
Advanced Topics

• Searching advanced spaces
  – Hill-climbing and Simulated Annealing
  – Genetic Algorithms

• Games
  – Designing AI to play against an opponent
  – Alpha-Beta Pruning

• Natural Language Processing
  – Naïve Bayes
  – Hidden Markov Models
If you’re interested in more, check out CIS 421/521 in the Fall!