Course Review – First Half

CZ3005: Artificial Intelligence

Shen Zhiqi

So far we have covered

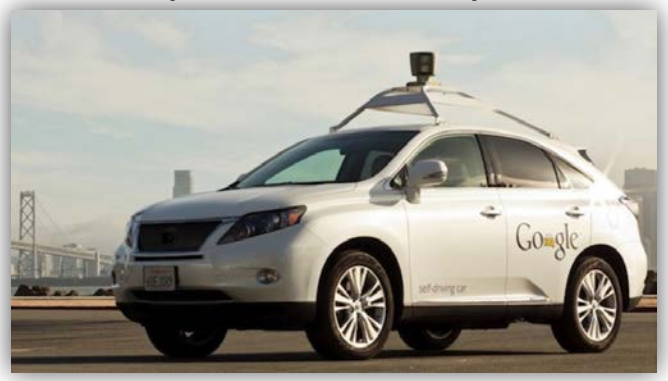
- How to describe an agent
  - Percepts, Actions, Goals, Environment
- Environment description
  - Accessible, Deterministic, Episodic, Static, Discrete
- Search problem formulation
- Uninformed search methods
  - Breadth-First, Uniform-Cost, Depth-First, Depth-Limited, and Iterative Deepening
- Informed search: greedy and A*
- Constraint satisfaction problem
- Game playing

How to describe an agent?

- Intelligent Agent
  - Rational agents
  - Autonomous agents
- The Types of Agents
  - Reflex agents
  - Reflex agents with state
  - Goal based agents
  - Utility based agents
- Describe an Agent
  - Percepts
  - Goals
  - Actions
  - Environment

Agent Description: Driverless Taxi

- **Percepts:** video, speed, acceleration, engine status, GPS, radar, …
- **Actions:** steer, accelerate, brake, horn, display, …
- **Goals:** safety, reach destination, maximize profits, obey laws, passenger comfort,…
- **Environment:** Singapore urban streets, highways, traffic, pedestrians, weather, customers, …



Environment Description: Driverless Taxi

- **Accessible?**
  - No. Some traffic information on road is missing
- **Deterministic?**
  - No. Some cars in front may turn right suddenly
- **Episodic?**
  - No. The current action is based on previous driving actions
- **Static?**
  - No. When the taxi moves, Other cars are moving as well
- **Discrete?**
  - No. Speed, Distance, Fuel consumption are in real domains

Search Problem Formulation

- Types of Problem
  - Single-state
  - Multiple-states
  - Contingency
- Well-defined Formulation
  - Problem definition
  - Specifications
    - Initial state, action set, state space, goal test
  - Performance measurement
    - Cost functions
  - Solution

Well-Defined Formulation

- Definition of a problem:
  - The information used by an agent to decide what to do
- Specification:
  - Initial state
  - Action set, i.e. available actions (successor functions)
  - State space, i.e. states reachable from the initial state
    - Solution path: sequence of actions from one state to another
  - Goal test predicate
    - Single state, enumerated list of states, abstract properties
  - Cost function
    - Path cost *g(n),* sum of all (action) step costs along the path
- Solution:
  - A path (a sequence of operators leading) from the Initial-State to a state that satisfies the Goal-Test

Search Strategies

- Uninformed Search Strategies
  - Breadth-First Search (BFS) – Depth Function *Depth(n)*
  - Uniform-Cost Search (UCS) – Path Cost Function *g(n)*
  - Depth-First Search (DFS) – Variables *b, d, m*
  - Depth-Limited Search (DLS) – Variable *l*
  - Iterative Deepening Search (IDS)
- Informed Search Strategies
  - Greedy Search – *h(n)*
  - A* Search – *f(n) = h(n) + g(n)*

Uninformed Search Strategies

Criterion	Breadth-first	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Time	b^d	b^d	b^m	b^l	b^d
Space	b^d	b^d	bm	bl	bd
Optimal	Yes	Yes	No	No	Yes
Complete	Yes	Yes	No	Yes, if $l \geq d$	Yes

Informed Search Strategies

- Greedy search
  - *h(n):* cost from n to goal (Future Prediction)
  - neither optimal nor complete, but cuts search space considerably
- Uniform-cost search (Uninformed search strategy)
  - *g(n):* cost to reach n (Past Experience)
  - optimal and complete, but can be very inefficient
- A* Search

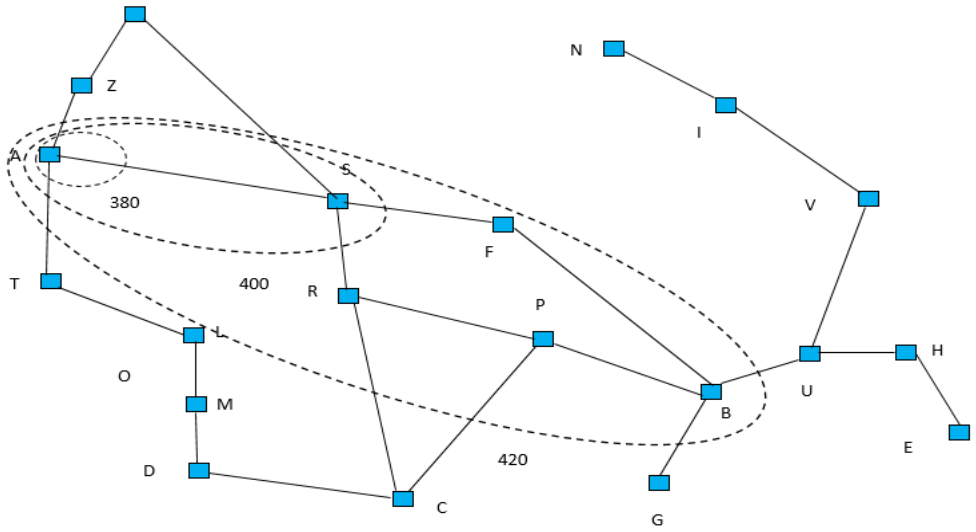
$$f(n) = g(n) + h(n)$$

  - *f (n):* estimated total cost of path through *n* to goal (Whole Life)
  - combine greedy search with uniform-cost search
  - If g = 0 → greedy search; If h = 0 → uniform-cost search
  - optimal and complete

Greedy Search

- m: maximum depth of the search space
- Time: $O(b^m)$
- Space: $O(b^m)$(keeps all nodes in memory)
- Optimal: No
- Complete: No

Complexity of A*



Time

- exponential in length of solution

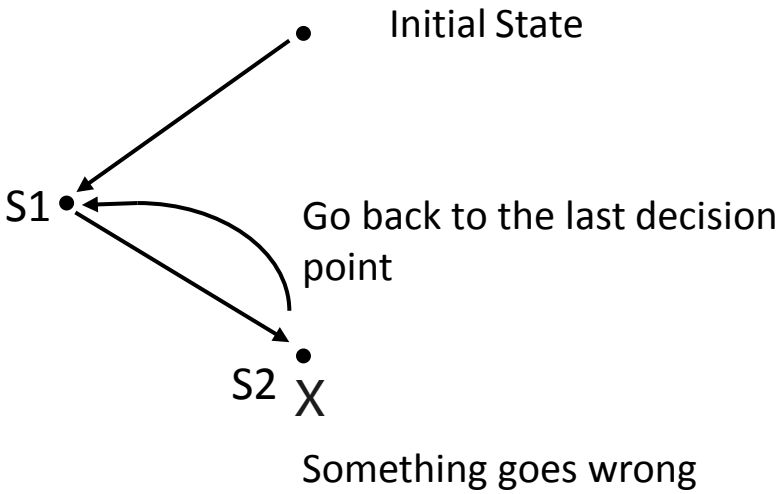
Space: (all generated nodes are kept in memory)

- exponential in length of solution

With a good heuristic, significant savings are still possible compared to uninformed search methods

(CSP) Backtracking Search

Backtracking search: Do not waste time searching when constraints have already been violated



- Before generating successors, check for constraint violations
- If yes, backtrack to try something else

Heuristics for CSPs

Plain backtracking is an uninformed algorithm!!

More intelligent search that takes into consideration

- Which variable to assign next
- What order of the values to try for each variable
- Implications of current variable assignments for the other unassigned variables
  - forward checking and **constraint propagation**

Constraint propagation: propagating the implications of a constraint on one variable onto other variables

Game Playing: Minimax Search Strategy

Search strategy

- Find a sequence of moves that leads to a terminal state (goal)

Minimax search strategy

- Maximize one's own utility and minimize the opponent's
  - Assumption is that the opponent does the same

- 3-step process:
  1. Generate the entire game tree down to terminal states
  2. Calculate utility
     1. Assess the utility of each terminal state
     2. Determine the best utility of the parents of the terminal state
     3. Repeat the process for their parents until the root is reached
  3. Select the best move (i.e. the move with the highest utility value)

Perfect Decisions by Minimax Algorithm

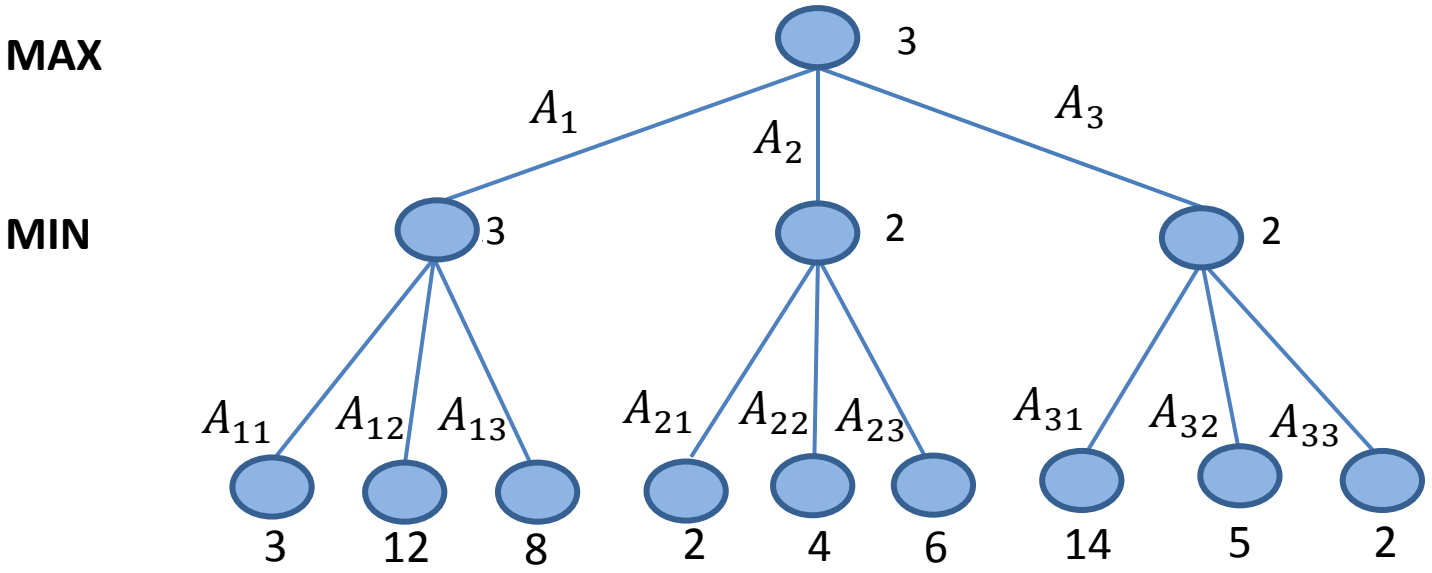
Perfect decisions: no time limit is imposed

- generate the complete search tree

Two players: MAX and MIN

- Choose move with best achievable payoff against best play
- MAX tries to max the utility, assuming that MIN will try to min it

Example



Imperfect Decisions

For chess, branching factor ≈ 35, each player typically makes 50 moves → for the complete game tree, need to examine 35^{100} positions

Time/space requirements → complete game tree search is intractable → impractical to make perfect decisions

Modifications to minimax algorithm

1. replace utility function by an estimated desirability of the position
   - Evaluation function
2. partial tree search
   - E.g., depth limit
   - Replace terminal test by a cut-off test

Evaluation Functions

Requirements

- Computation is efficient
- Agrees with utility function on terminal states
- Accurately reflects the chances of winning

Trade off between accuracy and efficiency

Example (Chess)

Define features

- e.g. , (number of white queens) – (number of black queens)

Use a weighted sum of these features

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots w_n f_n(s)$$

- Need to learn the weight

Exam Preparation

- Lecture Notes
- Tutorial Questions and Reference Answers
- Past Exam Papers
- Text Books
- Discussions