### Topics covered in the midterm

- Solving problems by searching (Chap. 3)
  - How to formulate a search problem?
  - How to measure a search strategy?
  - What are popular uninformed search strategies?
- Informed search and exploration (Chap. 4)
  - What are the greedy best-first search and A*?
  - How to design and use a heuristic function?
  - What is local search? Hill-climbing? Simulated annealing?
  - What are Genetic Algorithms?

### Readings: Chapters 3-8 of Russell & Norvig.

### Topics covered in the midterm

- Propositional Satisfiability (SAT) (Chap. 7.6)
  - How to formulate an instance of SAT?
  - What's a backtracking search?
  - What's DIMACS CNF format?
  - How to use local search for SAT?
- First-order Logic (Chap. 8)
  - What are its syntax and semantics?
  - What are quantifiers and their meaning?
  - What's a model of a FOL sentence?
  - How to use FOL to describe assertions and queries?

### Topics covered in the midterm

- Adversarial Search
  - What's the minimax algorithm?
  - What's Alpha-Beta pruning?
- Propositional Logic (Chap. 7.1-7.5)
  - What are its syntax and semantics?
  - Validity vs. satisfiability?
  - How to obtain CNF?
  - What's the truth-table inference system?
  - What's the inference rule system?
  - What's resolution and its property?
Exercise 3.1

Define in your own words the following items: state, state space, search tree, search node, action, successor function, and branching factor.

Exercise 3.3

Suppose that \textsc{Legal-Action}(s) denotes the set of actions that are legal in state \(s\), and \textsc{Result}(a, s) denotes the state that results from performing a legal action \(a\) in state \(s\). Define \textsc{Successor-Fn} in terms of \textsc{Legal-Actions} and \textsc{Result}, and \textit{vice versa}.

Exercise Problems

- Chap. 3: 1, 3, 6, 7, 9, 11, 13, 17.
- Chap. 4: 2, 3, 4, 9, 11, 12, 15.
- Chap. 6: 2, 3, 15.
- Chap. 7: 3, 4, 5, 6, 8, 12.
- Chap. 8: 2, 3, 4, 6, 7, 8, 10, 11, 15

Exercise 3.1

- State: A condition of (a problem or an agent) being in a stage or form
- State space: A collection of all states
- Action: An act which causes a state to change to another state, called \textit{successor}.
- Successor function: returns all the successors of a state
- Branching factor: The maximum number of successors of any state
- Search tree: A graphical representation of the order of successors explored from the initial state.
- Search node: A node in the search tree which contains the information of a state and its location in the tree.
Exercise 3.3

Suppose that \textsc{Legal-Action}(s) denotes the set of actions that are legal in state \(s\), and \textsc{Result}(a, s) denotes the state that results from performing a legal action \(a\) in state \(s\).

Define \textsc{Successor-Fn} in terms of \textsc{Legal-Actions} and \textsc{Result}, and \textit{vice versa}.

\[
\textsc{Successor-Fn}(s) = \{ \textsc{Result}(a, s) \mid a \in \textsc{Legal-Actions}(s) \}
\]

\[
\textsc{Legal-Actions}(s) = \{ a \mid \textsc{Result}(a, s) \in \textsc{Successor-Fn}(s) \}
\]

Exercise 3.6

Does a finite state space always lead to a finite search tree? How about a finite state space that is a tree? Can you be more precise about what types of state spaces always lead to finite search trees?

Does a finite state space always lead to a finite search tree? No. Without checking duplicates in a path, a loop may occur.

How about a finite state space that is a tree? Yes.

Can you be more precise about what types of state spaces always lead to finite search trees? Acyclic graphs of successor relations will always lead to finite search trees. Finite state space with duplicate checking in a path will always lead to finite search trees.

Exercise 3.7b

A 3-foot-tall monkey is in a room where some bananas are suspended from the 8-foot ceiling. Hw would like to get the bananas. The room contains two stackable, movable, climbing 3-foot-high crates.

- initial state
- goal test
- successor function
- cost function
Exercise 3.7b

initial state: Initially, four objects, \( m \) (monkey), \( b \) (bananas), \( c_1 \) (crate1), \( c_2 \) (crate2), are at four different locations \( l_1, l_2, l_3, \) and \( l_4 \), respectively. We use \( (l_1, l_2, l_3, l_4) \) as the initial state.

goal test: To get bananas, crate2 must be at location \( l_2 \); crate1 must be on top of crate2 (\( c_2 \)); monkey must be on top of crate1 (\( c_1 \)). That is, the goal state is \( (c_1, l_2, c_2, l_2) \). Switching crate1 and crate2, we have another goal state: \( (c_2, l_2, l_2, c_1) \).

successor function: There are many.

\[
\begin{align*}
1. \quad & s((l_1, l_2, l_3, l_4)) = \{(l_2, l_2, l_3, l_4), (l_3, l_2, l_3, l_4), (l_4, l_2, l_3, l_4)\} \\
2. \quad & s((l_3, l_2, l_3, l_4)) = \{(l_2, l_2, l_3, l_4)\} \\
3. \quad & s((l_2, l_2, l_2, l_4)) = \{(l_4, l_2, l_2, l_4)\} \\
4. \quad & s((l_4, l_2, l_2, l_4)) = \{(l_2, l_2, l_2, l_4)\} \\
5. \quad & s((l_2, l_2, l_2, l_2)) = \{(l_2, l_2, c_2, l_2)\} \\
6. \quad & \ldots
\end{align*}
\]

cost function: 1 for each action.

Exercise 3.9

The missionaries and cannibals problem is usually stated as follows. Three missionaries and three cannibals are on one side of a river, along with a boat that can hold one or two people. Find a way to get everyone to the other side, without ever leaving a group of missionaries in one place outnumbered by the cannibals in that place.

a. Formulate the problem precisely, making only those distinctions necessary to ensure a valid solution.

Exercise 3.11

Iterative lengthening search

a. Show that this algorithm is optimal for general path costs.
b. How many iterations for solution depth \( d \) and unit step costs?
c. If the minimum possible cost is \( \epsilon \), how many iterations are required in the worst case?
d. ...

Exercise 3.9a

states: \( \{(b, m, c) \mid b \in \{0, 1\}, 0 \leq m \leq 3, 0 \leq c \leq 3, (m = c \lor m \in \{0, 1\})\} \)

initial state: \( \langle 1, 3, 3 \rangle \)
goal test: \( \langle 0, 0, 0 \rangle \)
successor function: There are not many.
a solution: \( \langle 1, 3, 3 \rangle \rightarrow \langle 0, 3, 1 \rangle \rightarrow \langle 1, 3, 2 \rangle \rightarrow \langle 0, 3, 0 \rangle \rightarrow \langle 1, 3, 1 \rangle \rightarrow \langle 0, 1, 1 \rangle \rightarrow \langle 1, 2, 2 \rangle \rightarrow \langle 0, 0, 2 \rangle \rightarrow \langle 1, 0, 3 \rangle \rightarrow \langle 0, 0, 1 \rangle \rightarrow \langle 1, 0, 2 \rangle \rightarrow \langle 0, 0, 0 \rangle \)
Exercise 3.13

Describe a state space in which iterative deepening search performs much worse than depth-first search (for example, $O(n^2)$ vs. $O(n)$).

Exercise 3.17

Let us consider that actions can have negative costs.

a If the negative costs are arbitrarily large, explain why this will force any optimal algorithms to explore the entire state space.

b Does it help if we know the negative costs are not smaller than a negative constant $c$?

c ...

d ...

e ...

Exercise 4.2

The heuristic path algorithm is a best-first search in which the objective function is $f(n) = (2 - w)g(n) + wh(n)$. For what values of $w$ is this algorithm guaranteed to be optimal? What kind of search does this perform when $w = 0$? When $w = 1$? When $w = 2$?

Exercise 4.3

Prove each of the following statements:

a Breadth-first search is a special case of uniform-cost search.

b Breadth-first search, depth-first search, and uniform-cost search are special cases of best-first search.

c Uniform-cost search is a special case of A* search.
Exercise 4.4

Devise a state space in which A* using GraPh-Search returns a suboptimal solution with an $h(n)$ function that is admissible but inconsistent.

Exercise 4.9

One of the relaxations of the 8-puzzle in which a tile can move from square A to square B if B is blank. Explain why this heuristic is at least as accurate as $h_1$ (misplaced tiles), and show cases where it is more accurate than both $h_1$ and $h_2$ (Manhattan distance).

Exercise 4.11

Give the name of the algorithm that results from each of the following special cases:

a Local beam search with $k = 1$.

b Local beam search with $k = \infty$.

c Simulated annealing with $T = 0$ (supposing no termination) at all times.

c Genetic algorithm with population size $N = 1$.

Exercise 4.12

Sometimes there is no good evaluation function for a problem, but there is a good comparison method: a way to tell whether one node is better than another, without assigning numerical values to either. Show that this is enough to do a best-first search. Is there an analog of A*?
Exercise 6.3

Draw the complete game tree:

- Write each state as \((s_A, s_B)\) where \(s_A\) and \(s_B\) denote the token locations.
- Put each terminal state in square boxes and write its game value in a circle.
- Put loop states (states that already appear on the path to the root) in double square boxes.

Exercise 4.15a

Devise a hill-climbing approach to solve TSP.

Exercise 6.15

Describe how the minimax and alpha-beta algorithms change for two-player, non-zero-sum games in which each player has his or her own utility function. You may assume that each player knows the other’s utility function. Is it possible for any node to be pruned by alpha-beta?