1 Artificial Intelligence

- An **agent** is an entity that can perceive and act. This course is about designing rational agents.
- Rational behavior: doing the right thing.
- Environment Types: Fully observable; Deterministic; Episodic; Static, Discrete; Single-agent. The counter part: partially observable; stochastic; sequential; dynamic; continuous; multi-agent.
- An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.
- Being rational means maximizing your **expected utility**. And a better title for this course would be **Computational Rationality**.
- **Rational:** maximally achieving pre-defined goals.
- **Rationality:** only concerns what decisions are made (not the thought process behind them)
- **Utility:** Goals are expressed in the terms of the utility of outcomes. And CS188 thinks that being rational means maximizing your expected utility.
- **Automation:** Applied AI involves many kinds of automation. Scheduling, route planning, medical diagnosis, web search engines, spam classifiers, automated help desks, fraud detection, product recommendations. "Did any one of these remind you of subtopics and projects in Machine Learning?"
- **Agent:** An agent is an entity that perceives and acts. A rational agent selects actions that maximize its *expected* utility.
- Making decision, reasoning under Uncertainty, and their applications.

2 Problem Solving

- A search problem consists of
  - a state space
  - a successor function (namely **update function** in data mining algorithm series, more namely **recursion** in bullshit technology.)
  - a start state (**initial value**), goal test (**terminating value**) and path cost function (**we say weights in Graph Theory**)
  - Does any one of the above reminds you of **recursion**?
- Problems are often modelled as a state space, a set of states that a problem can be in. The set of states forms a graph where two states are **connected** if there is an operation that can be performed to transform the first state into the second.
- A solution is a sequence of actions (a plan) which transforms the start state to a goal state.
- **State space graph**: A mathematical representation of a search problem.
- **Search Trees**
  - This is a "what-if" tree of plans and outcomes
  - For most problems, we can never actually build the whole tree
- **General Tree Search** Frontier; Expansion; Exploration Strategy.
- **States vs. Nodes** Nodes in state space graphs are problem states; Nodes in search trees are plans. The same problem state may be achieved by multiple search tree nodes.
- **Graph Search** Graph Search still produces a search tree; Graph search is almost always better than tree search.
- DFS graph search needs to store "explored set", which is $O(b^m)$. However, **DFS is optimal** when the search tree is finite, all action costs are identical and all solutions have the same length. However limitating this may sound, there is an important class of problems that satisfies these conditions: the CSPs (constraint satisfaction problems). Maybe all the examples you thought about fall in this (rather common) category.
- The breadth first search and iterative deepening are conceptually and computationally the same. The only difference is the "space" (we call them **memory**) would be partially saved by iterative deepening search.
- **Heuristics** estimate of how close a node is to a goal; Designed for a particular search problem.
- **A star search** Uniform-cost orders by **path cost**, or backward cost $g(n)$; Best-first orders by **distance** to goal, or forward cost $h(n)$. A* Search orders by the sum: $f(n) = g(n) + h(n)$. The distance is an estimated one.
- When A* terminates its search, it has, by definition, found a path whose actual cost is lower than the estimated cost of any path through any node on the frontier. But since those estimates are optimistic, A* can safely ignore those nodes.
- In general, most natural admissible heuristics tend to be consistent, especially if from relaxed problems.
- Types of agents: reflex agents, planning agents (optimal vs. complete planning).

2.1 Uninformed Search

- **State Space:** A state space is also an abstraction of the world. A successor function **models** how your world works (namely, evolves and response to your actions).
- **Search Problems:** They are just models, aka, no more than models in the mathematical sense.
- **World State:** Includes every last detail of the environment.

- **Search State:** Keeps only the details needed for planning (namely, abstraction). Because only with abstraction can we solve problems smoothly.
- **Search Trees:** For most problems, we can never actually build the whole tree. (So we **ignore** most of the tree.)
- **Complete:** Guaranteed to find a solution if one exists? **Optimal:** Guaranteed to find the least cost path?
- **DFS vs BFS:** When will one outperform the other?
- **Uniform Cost Search:** Expand a cheapest node first. Thus fringe is a **priority queue**. (priority, cumulative cost, namely, add them up!) Therefore it's complete and optimal! But it explores options in every "direction". And this algorithm shows **no information** about goal location.
- Search operates over **models** (namely, abstractions) of the world. Planning is all "in simulation", therefore your search is only as good as your model is.

2.2 Informed Search

- **Informed Search:** Inject information of where the goal might be. Key idea: Heuristics.
- **Successor Function:** If I do this, what will happen, in my model.
- **Search Heuristics:** Something tells you that whether you are getting close to the goal, or not. It's a function that *estimates* how close a state is to a goal. It's designed for a particular search. Examples: Manhattan distance, euclidean distance. (They are not perfect, but they are *at least* something.)
- **Greedy Search:** A common case: best-first takes you straight to the (wrong) goal.
- **A* Search:** Revised. Combine both UCS and Greedy, namely, tortoise and rabbit. Uniform-cost orders by path cost, or backward cost. Greedy orders by goal proximity, or forward cost.
- **A* Search:** Stop when you **dequeue** a goal from the fringe. Lesson: We need estimates to be less than actual costs.
- **Admissibility:** Admissible (optimistic) heuristics slow down bad plans but never out-weight true costs. Inadmissible is just a fancy name for, **pessimistic**, it traps good plans on the fringe.
- A heuristic **h** is **admissible** (optimistic) if:

  $$0 \leq h(n) \leq h^*(n) \tag{1}$$

  where $h^*(n)$ is a true cost to a nearest goal. Thus coming up with admissible heuristics is most of what's involved in using A^* in practice. A^* is not problem specific, but your heuristic is.
- **Crating Admissible Heuristics:** Most of the work in solving hard search problems optimally is in coming up with admissible heuristics. Often, admissible heuristics are solutions to **relaxed problems**, where new actions are available.
- **Graph Search:** For tree search, if it fails to detect repeated states can cause exponentially more work. Idea: **never expand** a state twice.
- Important: (in python's idea) store the closed set as a set, not a list. In Lisp's concept, make it a hash table (it is verified, just use hash table in Lisp).
- **Consistency of Heuristics:** real cost should be larger or equal than cost implied by heuristic. (Namely, please be **Conservative**, aka guess "smally" rather than "biggerly".) Implication: f value along a path never decreases.
- **Optimality:** For tree search, requires heuristic admissible; for graph search, requires consistent. And consistency implies admissibility.
- **Heuristics:** The design of this number (function) is key, often use **relaxed problems**.

2.3 Constraint Satisfaction Problems

- **Search:** a single agent, deterministic actions, fully observed state, discrete state space.
- **Planning:** a sequence of actions. The **path** to the goal is the important thing.
- **Identification:** assignments to variables. The goal itself is important, not the path.
- **CSP:** a special subset of search problems. State is defined by **variable** X_i with values from a domain D (sometimes D depends on i). Goal test is a set of constraints specifying allowable combinations of values for subsets of variables.
- CSP allows useful general-purpose algorithms with more power than standard search algorithms. (Namely, add more "rules", walk through (traverse) less paths.)
- **CSP Varieties:** Discrete variables; continuous variables.
- **Varieties of Constraints:** Unary; Binary; Higher-order constraints. Or Preferences (soft constraints).
- **Backtrack Search:** The basic uninformed algorithm for solving CSPs. Namely, recursion. One variable at a time; check constraints as you go (Online shit? Incremental goal test). So backtracking is equal to DFS add variable ordering and add fail-on-violation.
- **Improve Backtracking:** Ordering; Filtering; Structure.
- **Filtering:** Keep track of domains for unassigned variables and cross off bad options. Namely, build a mathematical filter. Namely, ask (cond, else) when doing forward checking.
- **Forward Checking:** Enforcing consistency of arcs pointing to each new assignment.
- **Arc Consistency:** It still runs inside a backtrack search.
- **Ordering:** Minimum Remaining Values. Variable ordering, **always** choose the variable with the **fewest** legal left values in its domain, given a choice of variables.

- What the hell is CSP? Variables; Domains; Constraints—Implicit, Explicit, Unary/Binary/N-ary. Goals: find any solution; find all; find best, etc.
- **K-Consistency:** For each k nodes, any consistent assignment to $k-1$ can be extended to the k^{th} node.
- Suppose a graph of n variables can be broken into subproblems of only c variables. Example: $n = 80, d = 2, c = 20$. But this "crap" is somehow impractical.
- **Tree-Structured CSPS:** Theorem, if the constraint graph has no loops, the CSP can be solved in $O(nd^2)$ time. For general CSPs, worst case is $O(d^n)$. This also applies to probabilistic reasoning: an example of the relation between syntactic restrictions and the complexity of reasoning.
- **Nearly Tree-Structured CSPs:** Cutset conditioning: instantiate (in all ways) a set of variables such that the remaining constraint graph is a **tree**.
- *Sorry this is in ai class, everything is hard.—CS188*
- **Tree Decomposition:** Create a tree-structured graph of mega-variables. Each mega-variables encodes part of the original CSP.
- CSPs are a special kind of search problems where states are partial assignments and goal test is defined by constraints. The basic solution is backtrack search.
- **Local Search:** (yet another fancy name of EM algorithm.) It improves a single option until you can't make it better. (**No fringe!**)
- Generally local search is much faster and more memory efficient. But it is also **incomplete and suboptimal**.
- **Hill Climbing:** Simple general idea—Start wherever, repeat: move to the best neighboring state; if no neighbors better than current, quit.
- **Simulated Annealing:** Idea, escape local maxima by allowing downhill moves.
- The more downhill steps you need to escape a local optimum, the less likely you are to ever make them all in a row. Therefore people think hard about ridge operators which let you jump around the space in better ways.
- **Genetic Algorithms:** It uses a natural selection metaphor—keep best N hypotheses at each step based on a fitness function; Also have pairwise crossover operators, with optional mutation to give variety.

2.4 Adversarial Search

- **Meaning:** How to decide how to act, when there is an adversary in "your world (model, abstraction, etc.)".
- Monte Carlo methods are just a fancy name for **randomized** methods.
- **Pacman:** Behavior from **Computation**.
- **Axes:** Deterministic or stochastic? One, two or more players? Zero sum? Perfect information (can you see the state)?
- For this course, we want algorithms for calculating a **strategy (policy)** which recommends a **move** from each state.
- Different from search: we do not **control** our opponent. We need to give out **policies**.
- One possible formalization is: States, Players, Actions, Transition Function $S \times A \to S$, Terminal Test $S \to \{t, f\}$, Terminal Utilities $S \times P \to R$.
- Players usually take turns; Actions may depend on player/state; Terminal utilities tells us how much it's worth to **each of the players**.
- **Zero-Sum Games:** Let us think of a single value that one maximizes and the other minimizes.
- **General Games:** Agents have independent utilities. Cooperation, indifference, competition, and more are all possible.
- **Value** of a state: The **best** achievable outcome (utility) from that state.
- **Minimax Values:** States Under Opponent's Control: $V(s') = min\ V(s)$ States Under Agent's Control: **Maximize** out of all possible "worst" results your **opponent** offers. In choosing universities and advisors, pick out the "tallest" guy from the "small man".
- In other words, life is much much worse when there is an (or more than one) adversary. I want the "global maximum", but the adversary just **won't let it happen**.
- **Minimax Search:** A state-space search tree; players alternate turns; compute each node's **minimax value**, namely the best achievable utility against a rational (optimal) adversary.
- Ask this question to yourself: do we **really** need to explore the whole tree?
- **Resource Limits:** In realistic games, cannot search to leaves. Solution: **Depth-Limited Search**. Search only to a limited depth in the tree, and replace terminal utilities with an evaluation function for non-terminal positions.
- **Depth Matters:** An important example of the tradeoff between complexity of features and complexity of computation.
- **Evaluation Functions:** In practice, typically weighted linear sum of features.
- **Game Tree Pruning:** Look at the trees that do not have to be minimized.
- **Alpha-Beta Pruning:** Key idea it symmetric. To sum up, it's already **bad enough** that it won't be played. α MAX's best option on the path to root. β MIN's best option on path to root. Tip: You have to be right for the children of the route. Therefore good child ordering improves effectiveness of pruning.

2.5 Expectimax and Utilities

-