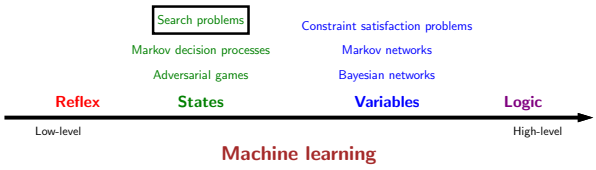
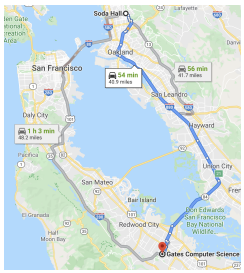
Search: overview

Course plan



Application: route finding



Objective: shortest? fastest? most scenic?

Actions: go straight, turn left, turn right

- Route finding is perhaps the most canonical example of a search problem. We are given as the input a map, a source point and a destination point. The goal is to output a sequence of actions (e.g., go straight, turn left, or turn right) that will take us from the source to the destination.
- We might evaluate action sequences based on an objective (distance, time, or pleasantness).

Application: robot motion planning

Objective: fastest path

Actions: acceleration and throttle

- In robot motion planning, the goal is get a robot to move from one position/pose to another. Some of the most popular search algorithms like A star are developed for some of the first intelligent robots (Shakey 1983)

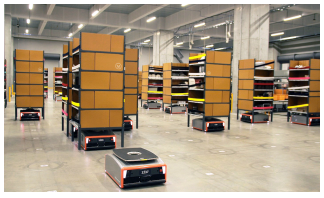
Application: robot motion planning

Objective: fastest? most energy efficient? safest? most expressive?

Actions: translate and rotate joints

- In robot motion planning, the goal is get a robot to move from one position/pose to another. The desired output trajectory consists of individual actions, each action corresponding to moving or rotating the joints by a small amount.
- Again, we might evaluate action sequences based on various resources like time, energy, safety, or expressiveness.

Application: multi-robot systems

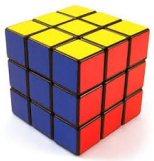


Objective: fastest? most energy efficient?

Actions: acceleration and steering of all robots

- Instead of planning for one agent, we can plan for a fleet of agents. For example, a group of robots need to coordinate in a warehouse to move objects from one shelf to another.

Application: solving puzzles



Objective: reach a certain configuration

Actions: move pieces (e.g., `Move12Down`)

- In solving various puzzles, the output solution can be represented by a sequence of individual actions. In the Rubik's cube, an action is rotating one slice of the cube. In the 15-puzzle, an action is moving one square to an adjacent free square.
- In puzzles, even finding one solution might be an accomplishment. The more ambitious might want to find the best solution (say, minimize the number of moves).

Application: machine translation

la maison bleue

↓

the blue house

Objective: fluent English and preserves meaning

Actions: append single words (e.g., `the`)

- In machine translation, the goal is to output a sentence that's the translation of the given input sentence. The output sentence can be built out of actions, each action appending a word or a phrase to the current output.

Beyond reflex

Classifier (reflex-based models):

$x \rightarrow \boxed{f} \rightarrow$ single action $y \in \{-1, +1\}$

Search problem (state-based models):

$x \rightarrow \boxed{f} \rightarrow$ **action sequence** $(a_1, a_2, a_3, a_4, \dots)$

Key: need to consider future consequences of an action!

- Last week, we finished our tour of machine learning of **reflex-based models** (e.g., linear predictors and neural networks) that output either a $+1$ or -1 (for binary classification) or a real number (for regression).
- While reflex-based models were appropriate for some applications such as sentiment classification or spam filtering, the applications we will look at today, such as solving puzzles, demand more.
- To tackle these new problems, we will introduce **search problems**, our first instance of a **state-based model**.
- In a search problem, in a sense, we are still building a predictor f which takes an input x, but f will now return an entire **action sequence**, not just a single action. Of course you should object: can't I just apply a reflex model iteratively to generate a sequence? While that is true, the search problems that we're trying to solve importantly require reasoning about the consequences of the entire action sequence, and cannot be tackled by myopically predicting one action at a time.
- Tangent: Of course, saying "cannot" is a bit strong, since sometimes a search problem can be solved by a reflex-based model. You could have a massive lookup table that told you what the best action was for any given situation. It is interesting to think of this as a time/memory tradeoff where reflex-based models are performing an implicit kind of caching. Going on a further tangent, one can even imagine **compiling** a state-based model into a reflex-based model; if you're walking around Stanford for the first time, you might have to really plan things out, but eventually it kind of becomes reflex.
- We have looked at many real-world examples of this paradigm. For each example, the key is to decompose the output solution into a sequence of primitive actions. In addition, we need to think about how to evaluate different possible outputs.

Paradigm

Modeling

Inference **Learning**

- Recall the modeling-inference-learning paradigm. For reflex-based classifiers, modeling consisted of choosing the features and the neural network architecture; inference was trivial forward computation of the output given the input; and learning involved using stochastic gradient descent on the gradient of the loss function, which might involve backpropagation.
- Today, we will focus on the modeling and inference part of search problems. The next lecture will cover learning.

Roadmap

Modeling

Modeling Search Problems

Learning

Structured Perceptron

Algorithms

Tree Search

Dynamic Programming

Uniform Cost Search

Programming and Correctness of UCS

A*

A* Relaxations

- Here are the rest of the modules under the search unit.
- We will start by talking about how we can define search problems, We will define states, successor, and cost functions.
- Next, we will introduce different types of search problems, starting with backtracking search and following with some of the other search algofithms such as DFS, BFS, and DFS-ID.
- Then we will continue discussing dynamic programming and uniform cost search. We will prove that uniform cost search is correct. Then we introduce the A* algorithm which tries to speed up UCS. Through this we introduce the concept of heuristics and their properties such as consistency, efficiency, and admissibility. Finally, we go over the idea of relaxations and how that allows us to come up with heuristics.
- We also have an optional module focusing on learning cost functions and covering the structured perceptron algorithm.