Syllabus

Course Homepage: HLT Link

Homework Submission: on eLearning

Piazza: Piazza Link

Optional Textbook: *Artificial Intelligence: A Modern Approach* Russell and Norvig, 3rd edition.

Miscellaneous Notes:

1. Honorlock will be used for exams.
2. Don't email for class-related questions, use Piazza private posts.
3. Textbook is not required, only what is presented in lecture will be tested.

Prerequisites:

1. Python for projects, maybe C/C++ and Java
2. Big-O notation
3. Propositional and first-order logic (will have review lecture)
4. Elementary knowledge of probability theory (will have review lecture)

Collaboration Policy:

1. Assignments can be done in group of two
2. Term projects can be done in group of two
3. Only need to turn in one solution/program per group

Late Policy:

1. 6 free late days ($\leq$ 2 late days can be used per homework)
2. One day late is 10% penalty, two days late is 30%

Day 1

General

The main goal of the course is **problem solving and decision making**. There are two main types of problems:

- Problems do not require *sophisticated knowledge*

One example is the **Path Finding** problem. One way to solve the problem is to search through all possible paths and find the minimum. However, that is often not feasible because there are too many possible paths.

Another example is the **Map Coloring** problem. One way to solve the problem is to perform a complete (or exhaustive) search, which would take a very long time.

The focus for these search problems is to find efficient solutions to solving these difficult problems. Note that hardware has improved in the last 20 or 30 years, we do not care about those improvements because algorithmic improvements almost always outperform hardware improvements.

- Problems that do require *sophisticated knowledge*

One example is asking a computer "what is the Capital of California." Although this is easy for a human who had learned this knowledge before to answer, it is hard for a computer to know even if they are given the Wikipedia article for the state of California. Humans are able to process contextual information based on prior knowledge as well as comprehension level; since computers are not able to do that, humans must turn natural language into some kind of **knowledge representation** so that the resulting information can be easily understood by a machine.

The second challenge is to use **knowledge reasoning** to derive new knowledge from existing knowledge. However, deriving information from some statements that are only partially true requires **Probabilistic Reasoning** to understand the final probability of a different statement.

AI Goal

> **Intelligence**
> "the capacity to learn and solve problems"

Views of AI falls into fall different perspectives:

1. Thinking: being able to think
2. Acting: being able to do something that can be observed
3. Human: emotion might be involved in decision-making
4. Rational: decisions are always rational and ideal

1. Acting Humanly (Turing Test)

Turing proposed the question of "can machines act intelligently?" The Test consists of a test subject who is chatting with either a real human or a machine. After a certain period of time, if the test subject

can not tell if they're talking to the machine or the human, then the machine would have passed by the Turing Test.

Key components from the Turing Test:

1. Natural Language Processing: processing the user's input and respond intelligently
2. Knowledge Representation: store and manipulate information from knowledge base
3. Automated reasoning: used store info to draw conclusion to answer questions
4. Machine Learning: to adapt and extrapolate from existing data

One example of a "human impersonator" is *ELIZA*, which attempts to impersonate a psychotherapist. It was built in the 1960s, and used basic strategies to decide on what response is the most fitting. *ALICE*, which was built 30 years later, is a lot smarter and can intelligently answer questions with additional information from a knowledge base.

2. Thinking humanly: modeling cognitive processes

Cognitive Science studies the internal activities of the brain, and *Cognitive Neuroscience* attempts to mirror the human brain.

3. Thinking rationally: formalizing the "laws of thought"

Using logic to make the right inferences. The classic example is if Socrates is a man, and all men are mortal, then the logical conclusion is that "Socrates is mortal".

However, since not all decisions can be made by direct logical reasoning, logic has limited usage in AI.

4. Acting rationally: rational agent

Rational behavior means doing the right thing, or the best thing to **maximize goal achievement given the available information**, which might mean using a math textbook to score well on a math test.

Building Intelligent Machines

Two approaches

1. One approach is building an exact model of human cognition (main view from psychology and cognitive sciences)
2. Developing methods to match/exceed human performance, not necessary through emulating human behavior (main focus of this course as well as recent AI progress)

AI has many domains, and it also leverages the knowledge from a lot of different disciplines (philosophy, CS theory, mathematics/physics).

Historical Perspective

The goal is to obtain an understanding of the human mind. Major people include:

1. Formalizing the laws of human thought: George Boole, Gottlob Frege, and Alfred Tarski
2. Thinking as computation: Alan Turing, John von Neumann, and Claude Shannon

Direct founders;

1. John McCarthy, Marvin Minsky, Herbert, Simon, and Allen Newell

History of AI:

- 1943: 40-neuron network, model of artificial neurons
- 1950: Turing's "Computing machinery and intelligence"
- 1960s: Dartmouth meeting with founders
- 1950s: Early enthusiasm with great expectations for its future
- 1958: John McCarthy's LISP
- 1965: The **resolution principle**, which was a basis for automated theorem proving
  - At this time, the world is simplified and limited to very small number of states
- 1960s: "dose of reality", *NP-Completeness* (intractability of problems)
- 1970s: Expert systems that require a knowledge base (correlate symptoms with diseases)
- 1980s: Neural nets, LISP
- 2000-: focus on integration of reasoning

Achievements:

1. Microsoft '97, Answer Wizard

2. Deep Blue beating Russian chess grandmaster
3. 1999: *Proverb* Solving crossword puzzles
4. Robocup, Autonomous Vehicle

The recent progress is from the advancement of **machine perception**, which means that machines can see and hear from the outside world. Techniques that were developed by *assuming* outside input are now able to be applied into technology like autonomous driving. There is also advancement in both processing speed as well as memory space, which is getting close to humans' capabilities.

Other progress comes from crowd-sourced human data for training data, engineering teams at big companies, as well as research activity at many institutions. Recent achievements include Watson winning *Jeopardy!*, Google's AlphaGo, as well as Watson automating insurance claim workers' jobs.

Day 2: 1/25

Uninformed Search

Search is very important in AI, and researchers try to use humans' reasoning to design better searching algorithms. Since some problems like map-coloring is NP-complete (no efficient polynomial algorithms, must perform complete search), searching is crucial to finding a solution.

Defining a Search Problem

State space is described by the following:

1. **Initial state**: the first starting state
2. **Actions**: possible actions from each state
3. **Successor function**: given a state *x*, return a set of `<action, successor>` pairs.

A **path** is a sequence of states connected by a sequence of actions. A **goal test** determines if a state is the goal (target) state. **Path cost** is a function that assigns a cost to a path, useful if we need to determine the *shortest path* from many different paths. The assumption for search problems is that the cost of a path is the sum of the costs of each steps of the path (assume to be nonnegative).

Search Description Examples

The 8-Puzzle

Description: there are eight tiles on a 3x3 board, how many moves to get to target state?
State: the tiles on the board
Initial State: the initial board
Goal test: true if the board is the same as the target board, false otherwise
Successor function: move one tile to the free spot
Path cost: N/A

Cryptarithmetic

Description: solve `SEND+MORE=MONEY` if each letter is a digit.
State: digits assigned to each of the 8 letters
Initial State: "null", no values assigned yet
Goal test: true if the sum is correct, false otherwise
Successor function: assign the next unassigned letter from the rest of the available digits.

Generic Search Algorithm

```
L = make-quee(initial-state)
loop
    node = remove-front(L) // save to return as part of path to goal
    if goal-test(node) = true: return (path to goal)
    S = successors(node)
    insert(S,L)
until L is empty
return failure
```

The search algorithm generates a **search tree**, which has branches from each node to each of its successor nodes generated by the **successor function**. The search tree only cares about nodes that are relevant to finding the goal step, and we want the tree to be as small as possible to save memory space since the state space can be huge.

Fun fact: the set of leaves nodes currently in the search tree is the same set of nodes currently in the queue.

Node Data Structure

A **Node** contains more information than a **State**, because we might store additional information such as the parent node to return information about the final steps. This is because the goal step needs to go back to the root state to retrieve the final *goal path*. Other information could include path cost, path depth, and other information that might be useful for the search process.

Evaluating a Search Strategy

Completeness: is the strategy guaranteed to find a solution?

Time Complexity: how long does it take to find the solution?

Space Complexity: how much memory does the strategy require?

Optimality: does the strategy find the best (highest-quality) solution when there are several solutions.

Evaluating BFS
Completeness: Yes, because all nodes are explored
Optimal: Yes
Time Complexity: $O(b^{d+1})$ need to traverse at most $d+1$ levels where d is the target's level
Space Complexity: $O(b^{d+1})$ because all nodes must be kept until target has been found

Evaluating DFS
Completeness: Yes, up till a finite depth
Optimal: No (depth limit)
Time Complexity: $O(b^m)$ where m is the max depth
Space Complexity: $O(bm)$ because each route can be deleted after no goal is found

Advantages of **BFS**: Optimal solution

Advantages of **DFS**: Space efficient

Combining the two, we get **iterative deepening**, which we will explore next lecture.

Day 3: 1/27

Tree Search: does not check if a node has been visited before

Graph Search: checks if a node has been visited before prior to inserting

Even though it looks like graph searches will always be superior, we need to know that graph search needs to keep a hash table/set to remember what nodes have been visited before. This might be be very costly with regards to memory space.

Uniform-Cost Search

Uniform-Cost Search is similar to BFS, but we always expand the lowest-cost node from the initial state. This cost is called the **g-cost**, and `g(Successor(n))> g(n)` is a *necessary condition* for completeness as well as a *sufficient condition* for optimality.

Iterative Deepening

Strategy: Increase the artificial limit by 1 every iteration, and traverse using that limit with DFS.

Even though it has good space complexity, is optimal, and is complete, the downside is that the shallower nodes are being checked over and over again during each iterative deepening iteration.

Bidirectional Search

Search from both the start and the goal nodes.

If we use BFS from both ends, then the time complexity is $O(b^{\frac{d}{2}})$ while the time complexity fr normal BFS is $O(b^{d+1})$. One example is that if the branching factor is 10 and the depth is at 6, BFS needs over 1 million nodes and bidirectional search only needs around 2,000.

The limitation is that we need a *predecessor* function to go backwards, and we might also run into problems with a lot of goal states and having to do bidirectional searches for all of them.

Generic *Best*-First Search

Similar to BFS, but use a function `f` that determines the cost of a node based on how "promising" it is. The difference between this and a BFS/DFS is that we are using a heap. This is because we are selecting the node with the minimum "cost" (most promising), which can be implemented as a heap.

If we view DFS and BFS as a specific form of the generic best-first search, we can see that BFS has a function of `f(n)=depth(n)` since the lowest depth nodes are scanned first, then the next level, and etc. On the other hand, DFS has a function of `f(n)=1/depth(n)`, since the largest depth node is explored first.

The key idea is to notice that if we can find a function `f(n)` that can help us find the path in the shortest time, then we would have the optimal searching algorithm for that problem.

Informed Methods: Heuristic Search

If we have a function `h(n)=estimated cost of the cheapest path from n to target`, then we can define the `f` function as `f(n)=g(n)+h(n)`, where `g(n)` is the distance from the initial state to the current state. Note that `g(n)` is *known* (since we traversed that path from initial to current already), and `h(n)` is an estimate. This approach is known as **A* Search**.

If we define `f(n)=h(n)`, this is known as the **Greedy Best-First Search**.

For the 8-puzzle problem, one possible heuristic function is the sum of each Manhattan Distance, another is simply the number of incorrectly placed blocks.

Day 5: 2/3

Generic Best-First Search

Reviewed **Generic Best-First Search**: what is the cost function for each of these functions?

- **BFS**: $f(n) = depth(n)$
- **DFS**: $f(n) = \frac{1}{depth(n)}$
- **UCS**: $f(n) = g(n)$
- **Greedy**: $f(n) = h(n)$
- **A***: $f(n) = g(n) + h(n)$

Heuristics

For example, if our task is to find a path from city A to city B, one possible heuristic is using the straight-line distance between two cities. The **greedy** approach would rely solely on the heuristic, and thus pick the closest city (in terms of straight line distance) in each iteration.

Graph vs Tree Search

It is important to note that in **tree search**, we do not visit nodes that have been visited already. In **graph search**, we are allowed to do that. Graph search is more space efficient because we don't have to keep track of what nodes have been visited before.

Continuing from the example above, A-star would take into account the path from the source vertex (which is $g(n)$). In this example, it is able to return the lowest-cost path.

Admissible Heuristic

In order for A-star to always return the lowest-cost solution, the heuristic must be **admissible**. This means that $h(n) \leq h^*(n)$ for all nodes n ($h^*(n)$ is the actual cost to reach the goal state from n). If this is true, A-star will never return a suboptimal goal.

Another way of putting it is that admissibility guarantees that we never overestimate the cost of the path.

Note that the **perfect heuristic** is exactly equal to the actual cost, and the **trivial heuristic** is just $h(n) = 0$ because it is never overestimating the cost ($h^*(n) \geq 0$).

Consistency of Heuristics

Heuristic "arc" cost $\leq$ actual cost for each arc. This is to ensure that a child node does not have a lower $h(n) = f(n) + g(n)$ than its parent. Note that if you use graph search, you need **consistency** for A-star to optimal. If you use tree search, **admissibility** alone is enough to guarantee optimality.

For example, in the 8-puzzle problem, the *out-of-place* heuristic is always less than or equal to the *Manhattan* heuristic. This means that the Manhattan heuristic is better to use.

A-Star Analysis

A* is optimally efficient (expand the least amount of nodes), is complete, but has exponential complexity due to its bread-first nature.

Day 6: 2/8

Inventing Heuristics

Breaking down the 8-puzzle problem:

a. A tile can move from A to B if A is adjacent to B: Manhattan distance
b. A tile can move from A to B if B is blank
c. A tile can move from A to B regardless: number of misplaced tiles

We can take any of these three perfect heuristics, and they will be admissible for the original problem. Thus, we can then take the maximum of these three since it would dominate the others.

Optimized A* can only go up to around 10^{100} states; however, real world problems have up to $10^{30,000}$ states. To handle more states, we have to give up some information. With this in mind…

Local Search Method

In this method, we do not care about the path and only care about the goal state. Thus, we do not need to store the parent structures.

However, we generally need a complete state description. In the 8-queens problem, this means that we start with a complete board that already has the 8 queens.

Hill Climb

This method is quite naive and only moves up hill, even if there is a higher global maxima on the downhill side.

GSAT vs Davis-Putnam

GSAT always outperforms Davis-Putnam in boolean expression problems. Davis-Putnam performs a DFS search which could take a lot longer. Since we only need one solution, GSAT would be complete in that regard and be able to find the solution a lot faster.

Improvements to Basic Local Search

The issue arises when we want to escape the local minima. One possible approach is using a *fixed length queue* that prevents 2-step cycles. This means that we would not revisit the original step that we have just visited. This does not require a lot of storage space but reduces the complexity.

Day 7: 2/10

Note that from last lecture, local search starts from a complete state and makes edits to get successors.

GSAT attempts to get out of local maxima by restarting in a random different location. However, this means that it is incomplete, but is more efficient compared to DFS.

Tabu maintains a small queue to detect small cycles and prevent going back through already visited areas.

Simulated Annealing allows moving downhill, but with a lower probability as time increases.

Random Walk is very slow and not feasible for larger problems. Walk SAT performs better compared to Simulated Annealing for larger problems.

Genetic Algorithms

Maintain a population of individuals, and evaluate them using a **fitness function**. From the current generation, we select pairs of individuals (that scored well) and generate new individuals (this step is called **crossover**). To introduce some noise (and get out of local maxima), we use the idea of **mutations** to introduce random noise into the new generation.

High-level Overview: The initial population is first evaluated by the fitness function (probability determines if they would be chosen as the parent), then cross-over happens between pairs of individuals by randomly selecting a random "chopping point" and selecting the opposite portions alternatively to create two new children that are the combination of the two parents. Finally, we mutate an individual by randomly changing some of its values (if it should be mutated at all).