Use this document to reflect on your understanding of the week's material.
After entering your responses, convert into pdf and upload in Cougar Courses.
Deadline to submit is Saturday at 9 PM.

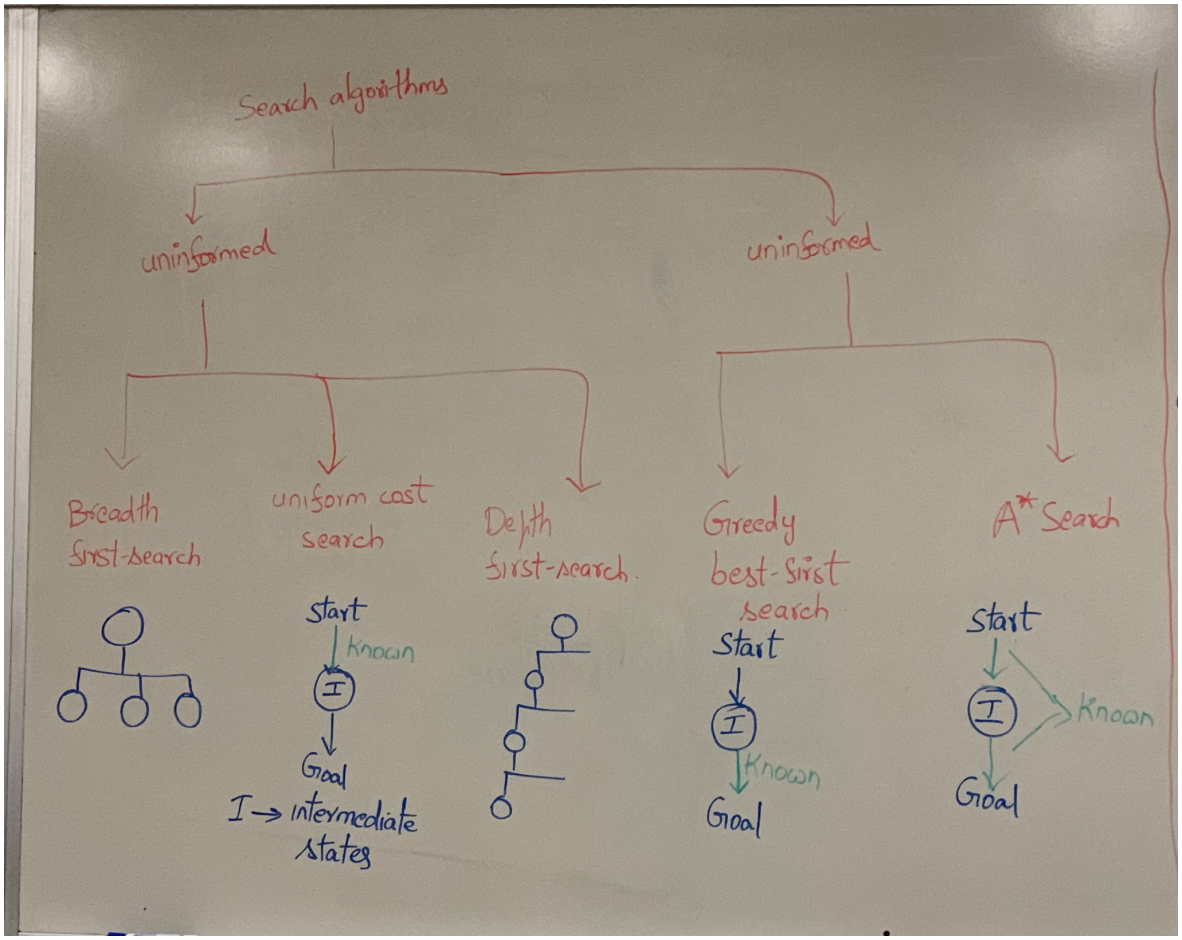
Review the class material and provide a detailed description of the key topics you have developed a good understanding.

An algorithm is a sequence of steps to find a solution for a problem. Search algorithms can be applied to a wide range of problems, but to decide which algorithm to use, we first need to have a well-defined problem. A problem should have an initial state, a set of actions, a transition model with the description of the results of the taken actions, a set of goal states, and an action cost function.

This past 2 weeks we learned about Search Algorithms and Local Search Algorithms. Both categories of search are for optimization problems. The main difference between them is that the Search Algorithms care about the path and keep track of it, while the Local Search Algorithms focus on finding a solution.

Search Algorithms

Search Algorithms have an initial state, they keep track of all intermediate states, and they find a solution at the goal state. There are 2 types of Search Algorithms, Uninformed and Informed. The Uninformed Search Algorithms does not know how far it is from the goal, while that on the Informed Search Algorithms, the cost/distance to reach the goal is known.



The Uninformed Search Algorithms:

1. Breadth-First Search
   . Complete.
   . Optimal for unit action costs
   . Search from the root expanding layer by layer
   . The search stops at the shallowest goal found.
   . Good strategy to be applied when all actions have the same cost.
   . It always find an optimal solution (post optimal)
   . At the end of the expansions, it compares the results of all options and return the optimal solution
   . Exponential space/time complexity
   . Computational expensive: time and space complexity are $O(b^d)$, where b= branching factor, d = depth
   . Good for small instances
   . Not feasible for very large datasets

2. Uniform-Cost Search
   . Complete.
   . Optimal for general action costs.
   . Spreads out in waves of uniform path-cost. Starts form an initial state and compares both path options. It takes the least cost direction, but keeps the bigger on hold. On each step, it expands and compares again.
   . Cost-optimal. It considers all paths systematically in order of increasing cost.
   . It always finds an optimal solution when the cost of each step is a positive number.
   . Time and space complexity can be greater than $O(b^d)$ in the worst-case scenario.
   . Should be applied only if the costs are different, for the same cost BFS is better.

3. Depth-First Search
   . Complete only in finite state space.
   . Neither complete nor optimal.
   . Search from the root, expanding the deepest branch of the tree until it finds the solution. If it reaches a leaf that is not the goal, then it backs up and follows the ancestor's other branch up to the deepest node, and so on.
   . Linear space complexity.
   . Space complexity is O(bm) and time complexity is O().
   . Requires less memory than BFS
   . Can be implemented with recursion
   . Not cost-optimal, it may find a non-optimal solution because it returns the first solution it finds, even if it's not the cheapest one.

<u>The Informed Search Algorithms:</u>

Can find solutions more efficiently than an uninformed strategy. Uses a heuristic function h(n) to estimate the cheapest cost to reach the goal from n.

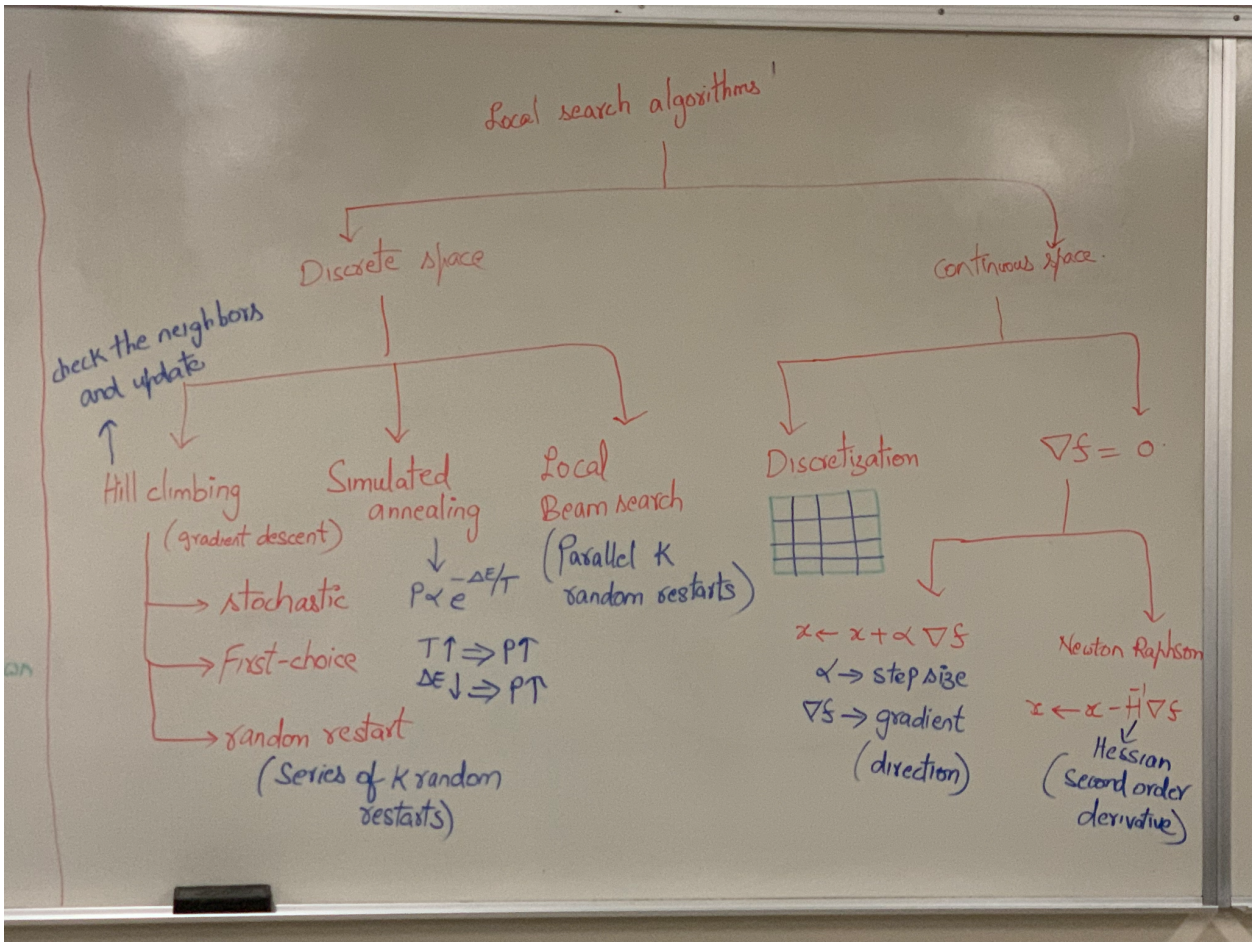
1. Greedy Best-First Search
   . Complete in finite state spaces.
   . Is a form of best-first search. It chooses to expand to the next node with the lowest h(n) value, which seems to be the closest point to reach the goal.
   . Does not always find the optimal solution (with the optimal cost).
   . On each iteration it tries to get closer to the goal. It does not backtrack.

2. A* Search
   . Complete and optimal.
   . It may be cost-optimal depending on the heuristic function.
   . The most common informed search algorithm.

. It's a form of best-first search with a combination of uniform search and greedy algorithm, where the cost from the current state to the origin g(n) and to the goal h(n) are known.
. From the initial state it goes on through the lowest cost path, but it keeps track of the last iteration costs. It expands from the lowest cost, and then it compares the costs again, always choosing to expand from the lowest cost node.

Local Search Algorithms

Local Search Algorithms start searching from an initial state to its neighbors, but it does not record the path or the set of states it visited. These algorithms have the advantage of using little memory (not storing the path), and are able to find good solutions even in large or infinite state spaces.



Search in Discrete Spaces:

1. Hill-climbing Search

. Goal is to find the highest peak, the global maximum

. Keeps track on the current state and moves to the neighbor with the highest value.

. It may get stuck in a local maxima, when it reaches a ridge, or a plateau

. Reasonable output, able to find a good local maxima after a small number of restarts.

1.1. Stochastic Hill Climbing

. Choose a random state among the uphill, the probability of selection depends on the slope of the hill.

. Usually finds a good solution

1.2. First-choice Hill Climbing

. Chooses a random state as successors, updates the current state when a better one is found.

. Good strategy when a state has many successors.

1.3. Random-restart Hill Climbing

. Tries random successors until it finds a better one.

2. Simulated Annealing

. Complete and efficient.

. Similar to Stochastic Hill-Climbing, but it accepts some downhill moves.

. Combines hill-climbing with random walk.

. Starts at a random state.

. Schedule a temperature T

. Compute ΔE = value (current) - value (next)

. If the next random choice has a better value, it always updates the current state, or it uses probability to update the state.

. $P = e^{-\Delta E / T}$

. If probability is high $P \uparrow$, it moves around more (more randomness of current updates)

. $T \uparrow = P \uparrow$

. $\Delta E \downarrow = P \uparrow$

. Make some “shakes” (using T) to allow the state to change if it gets stuck. At the beginning, the shakes are more intense, allowing for bigger moves, and gradually reduces the intensity of the shakes.

. Gets better results because it keeps adjusting T

. Goal is to find highest or lowest point

3. Local Beam Search
   . Initiates k random initial states and, at each step, all k successors are generated.
   . Compare current values on each parallel search.
   . All k states run in parallel, and if one state is too far from the goal, it is dropped and a new random state restarts closer to the location of the best state.

Search in Continuous Spaces:

Continuous spaces have infinite branching factor, so they cannot be solved by the previous algorithms.

1. Discretization
   . Limit the space into a grid with spacing of size δ (lowercase delta).
   . Once it is discretized, apply the discrete algorithms.
2. $\nabla f = 0$ (Gradient)
   . Use calculus to solve a problem analytically
   . Find a x where $\nabla f(x) = 0$ to find a maximum or minimum
   2.1. Compute the gradient locally
   . $x \leftarrow x + \alpha \nabla f(x)$
   . α is the step size. If it's too big, it may miss the global maxima; if it's too small, it may need too many steps.
   . $\nabla f(x)$ tells if the slope is positive or negative
   2.2 Newton-Raphson method
   . It's a general technique for finding roots of functions
   . $x \leftarrow x + H_f^{-1} \nabla f(x)$
   . $H_f(x)$ is the Hessian matrix of second derivatives
   . Challenge: computational expensive
   . Advantage: No hyper parameter

Constraint Satisfaction Problems

An optimization problem is constrained if there are constraints for the variables' values. A Constraint Satisfaction Problem - CSP - is a problem whose solution satisfies all the constraints. It helps reduce the scope of the problem by eliminating the search space. Many problems can be reduced to a CSP. Discard any solution that violates the constraints and solve it faster as CSP.

What concepts do you need more support?

Every algorithm that has math functions in it gets overwhelming to me. It’s been many years since I’ve studied math, and I’ve learned it in Portuguese. It takes me time to remember, and to connect English names to the concepts. Many times I need to revisit the topic to refresh it.