Local Search

Elijah Dangerfield CSCI 4350

Introduction

In artificial intelligence, formulating a problem as a search problem with a search space, states, and a goal state(s) can make a seemingly complex problem become relatively simple to solve by finding the path to the goal. However, there are many problems in which you may be more interested in the final goal state and less about the path taken to get there. For these problems we can use techniques under the toolbelt of what is known as local search. Broadly speaking, local search is the category of search in which the agent does not hold in memory its path, it only uses its current state and the possible next states to search a space (think one step at a time without looking back). On top of saving memory by only looking at the current state and the next states, local search also tends to perform well in large or infinite continuous search spaces.

Local search works particularly well in optimization problems as we generally care about obtaining the value of the maximum or minimum of the search defined by a problem formulation rather than the path to get there. In this lab we use the local search methods, Greedy Hill climbing and Simulated Annealing, in order to find the maximum in a search space defined by the Sum of Gaussians function. The Sum of Gaussians function is a tool used to give us randomized search spaces while letting us define the number of dimensions (D) and the number of hills (N) in the space. Figure 2 below exemplifies a search space generated by the Sum of Gaussians function with a D = 2 and N = 10.

Algorithms

At a high level, the hill climbing search algorithm works by evaluating the value of the current state vector and taking its partial derivative in order to get the gradient at that point. The gradient is simply a unit vector that points in the direction of steepest ascent/descent towards optimal in a search space at a given point. Given a gradient, we can multiply it by some number and add this to the current point in order to take a step in the direction of the steepest ascent/descent towards optimal. The number we multiply the gradient by is referred to as the step size and is held at a constant 0.01 for this lab. The algorithm continually calculates the steepest ascent/descent towards optimal at a point and takes a step in that direction until the increase in value from one point to the next is below a specified tolerance. For this lab the tolerance was held at a constant 1e-8.

Simulated Annealing works similarly but rather than taking a gradient to define the next step, it chooses a step in a random direction and accepts it if it is a step towards the optimal. Otherwise, if the step is not towards the optimal, the algorithm takes the step based on what is known as metropolis criterion (see figure 1.) The metropolis criterion works such that the larger the

temperature, the larger the probability of accepting a non optimal decision. The idea of simulated annealing is to start with the highest probability of making random actions even if they are not optimal in order to explore the space and help prevent getting caught in one area of the search space while lowering that probability to only make logical moves over time. I chose to start with my temperature at 10000.0 and to multiply it by 0.99 on every iteration in order to decrease it over time with the idea that the algorithm will start out with the most energy, and decrease quickly. I made this decision based off the fact that the more random moves the algorithm makes, the closer it comes to being a completely random search and the more logical moves the algorithm makes, the closer it comes to being basic hill climbing. One other choice I made in my algorithm was to prevent the algorithm from taking steps outside of the domain of the Sum of Gaussians function (0 -10). If a random step was made outside of this boundary, the step was wrapped to the other side of the search space. This effectively provides the simulated annealing algorithm with an ability to search opposite sides of the search space. This effect is shown in figure 2 where both algorithms start at the same point but simulated annealing searches the opposite side of the space, resulting in if finding a much better local max than that of greedy hill climbing.

Figure 1:

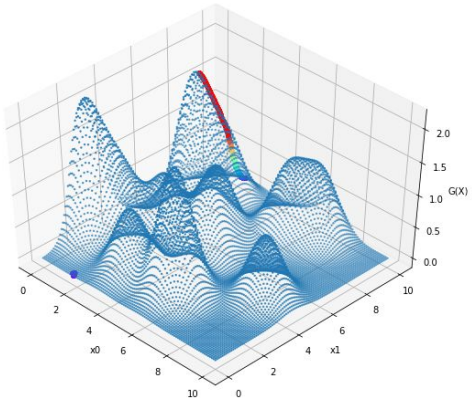
Metropolis criterion

$$\text{probability of accepting step away from ascent} = e^{(G(Y) - G(X)) / T}$$

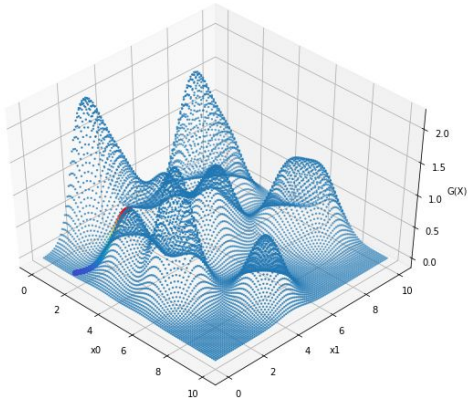
Where G(Y) is the evaluation of New point in search space, G(X) Is evaluation of current point in search space and T is temperature

Figure 2. Comparison of search by simulated annealing and greedy hill climbing

Simulated Annealing



Greedy Hill Climbing



In order to test the performance of the search algorithms, each algorithm was ran with every combination of N = {10 , 50 , 100, 1000 } and D = { 1, 2, 3, 5}, where N is the number of Gaussians (hills) and D is the dimension, for 100 runs each with a different random seed for a total of 3200 runs (1600 each). For each configuration the number of times one algorithm out performed the other was measured. For this lab outperformance is defined as one algorithm finding a better local maximum by at least 1e-8.

Results

Sa = Simulated Annealing D = Dimensions
Greedy = Greedy Hill Climbing N = Gaussians

Figure 3: Performance statistics of simulated annealing vs greedy hill climbing

	N = 10	**N = 50**	**N = 100**	**N = 1000**
D = 1	Sa - 91 Greedy - 9	Sa - 88 Greedy - 7	Sa - 80 Greedy - 11	Sa - 28 Greedy - 45
D = 2	Sa - 50 Greedy - 43	Sa - 48 Greedy - 51	Sa - 40 Greedy - 58	Sa - 10 Greedy - 90
D = 3	Sa - 31 Greedy - 57	Sa - 18 Greedy - 82	Sa - 22 Greedy - 78	Sa - 10 Greedy - 90
D = 5	Sa - 22 Greedy - 12	Sa - 47 Greedy - 35	Sa - 49 Greedy - 46	Sa - 12 Greedy - 88

Conclusion

The results show that simulated annealing generally outperformed greedy hill climbing in lower dimensions and with a lower number of gaussians. However, greedy hill climbing performed better than simulated annealing with a higher number of gaussians. This intuitively makes sense under the idea that greedy hill climbing has a higher chance of landing in a flat area to start will and terminating before finding a hill to climb with a lower number of gaussians whereas a high number of gaussians would provide a diverse search space. Additionally greedy hill climbing out performed simulated annealing with all number of gaussians in 3 dimensions. Which is very interesting. I would not go as far as to say that this would generalize to every problem. Every search space is different and there is likely a problem that gives a search space in which one of these algorithms shines over the other. The trends found in this example may be a consideration for future problems with different search spaces. However, for this specific speech space, and for my implementation, there were trends with dimensions and gaussians.