Giuliano Cancilla

Intro to Artificial Intelligence

OLA 2: Greedy and Simulated Annealing Report

The Problem

There is a mathematical principle known as a Gaussian Distribution. A Gaussian Distribution is more commonly known as a bell curve. Figure 1 is an example of a normal Gaussian Distribution. Normal meaning symmetrical and no further transformations have been made (See figure 1). We can see that the curve resembles that of a hill. The maximum coordinate of the curve is a coordinate (x,y) such that y is the largest value possible for the given distribution. In Figure 1, the maximum Y value is 1.0 and the x is 5, giving us the maximum as (5,1). Now If I where to pick a random starting point on this graph and I told myself that I wanted to climb the hill I would have to make some decisions. If I start to the right of the curve, I need to backtrack to get to the top. If I start to the left of the curve, I need to move forward to reach the peak.

With the Sum of Gaussians, we can introduce more peaks or more valleys and we can make a program that finds the peak closest to it. By stringing together more Gaussian Distributions we can make a new graph (see figure 2). We can randomly choose where to start and find the closest peak near us. Sum of Gaussians (SoG) can also be multidimensional, meaning that you can make it 1D (like figure 1 and 2), 2D, 3D or more dimensions. Even the multidimensional gaussians can be solved to find a peak. (See Figure 3).

The problem we are asked to solve in this experiment is "How do we find a peak in a Sum of Gaussian Distribution?". This process of finding the peak is known as hill climbing. In this experiment we will choose a random place to start on the distribution and climb to a peak. We will do this in two different ways, Simulated Annealing and Greedy Hill Climbing Search. We will then compare which hill climbing search algorithm will be better for the given problem. It is important to note that these two methods of search are not just confined to the Sum of Gaussians problem but can be changed to incorporate a wide range of other problems as well.

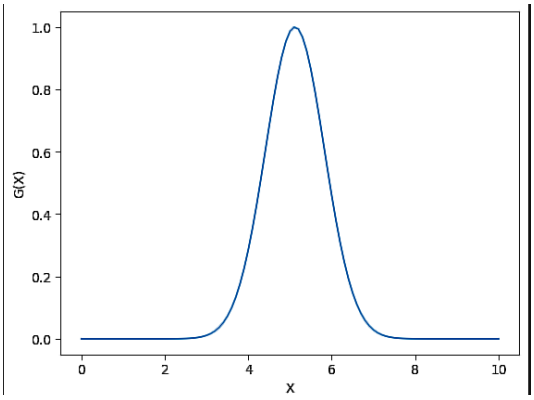

Figure 1: an example of a Normal Gaussian Distribution

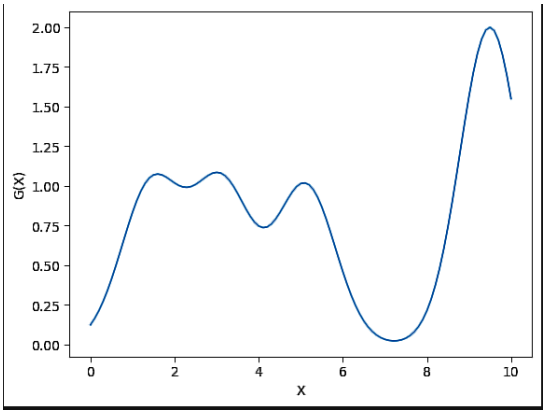

Figure 2: The result of performing the Sum of Gaussian Distribution on 5 Gaussian Distributions.

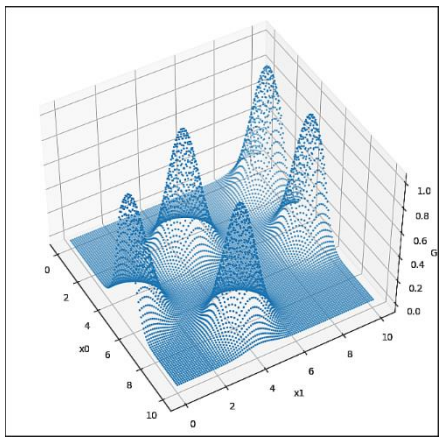

Figure 3: An example of 2D SoG

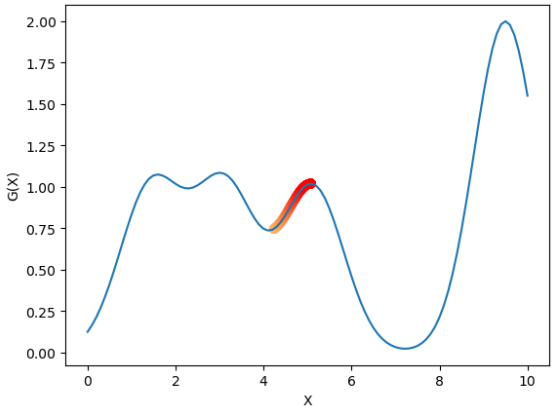

Figure 4: Greedy Search Algorithm on a 1D, 5 Sum of Gaussians

Search Methods Explained

Greedy Hill Climbing Search (See greedy.py for code)

So now that we know what a Gaussian Distribution is we can now proceed to search through it. The first method we use is known as Greedy Hill Climbing Search. In Greedy Search, we first find a random point on the graph and choose to start there. We calculate the x and the y for the point that we start at. Now we need to make a step, a step is consistent of a pre-defined step size (ours is 0.01) * the gradient, where the gradient is the slope at that point. We update our current point to incorporate the step that we just took. The idea is to try and maximize the value of G(x), which is the Y value after plugging in an X to our Sum of Gaussians function. When we compare the values, we will update the currentX to the highest value. Since we are favoring a higher G(x) value we will update the currentX to take that step. When the steps we are taking are not making significant changes to Y, we can terminate early, or we terminate once we have reached 100,000 iterations. We need to test on each iteration if the change in Y from the currentX and the next X is less than 1e^-8. If we can say that difference is insignificant, then we can say that we have reached a local max, because the Y isn't going any farther up. A great way to visualize this is by looking at a graph of how greedy search is performed (see Figure 4).

This method is good at finding a local max, but it may not find the most optimal maximum. If we look at Figure 4, we can see that the greedy algorithm finds a max somewhere around 1.00 for G(x). This looks like it is the smallest maximum G(x) value in terms of maxima on the graph. This is because greedy stops when it finds the first maximum and in our case, terminates once the difference in G(x) values is less than 1e^-8. However, it doesn't fail to find a maximum. It is reliable, but may not be optimal for certain cases.

Simulated Annealing Search (Check figure 5 for example on how SA searches)

Simulated annealing starts a random position. In simulated annealing we are going to randomly change the start position with a vector between -0.05 and 0.05. We use a vector because SoG functions can be multidimensional, and vectors can also be multidimensional. We make a step using that random vector much like taking a step using greedy and we find the G(x) of the new point. If the G(x) of the new point based off the step taken is larger than that of the current point, then we update the current point to the new point. However, if it is not larger then we set up a probability and make a roll to see if we take that move. We are introducing randomness into our solution by doing this. The probability is based off this variable called temperature. In simulated annealing, over time, we want to lower the temperature (a.k.a lower the probability that we are going to make a bad move). The reason why we don't set temperature immediately to 0 is because we may have to make some bad moves to find a more optimal solution.

Temperature can be played around with to find a more optimal solution. In my program I used a linear regression for temperature. After every iteration I multiplied the current temperature by .9995 to slowly decrease it over time. So, over time we are less likely to make a bad move. At the end of the main loop for my simulated annealing program I added a conditional check. If the temperature gets below 0.001, then we can terminate. This was to speed up the time that it took to run. This makes sense because since the temperature is approaching 0, we aren't making many moves. If the temperature is approaching 0.001 the probability of us making a move is low.

The cooling of the temperature over time based off the annealing schedule allows Simulated Annealing to explore more of the function than greedy search does. It allows for a wider range of values to be explored. This may cause Simulated Annealing to find a higher maximum. If the amount of iterations for simulated annealing was infinite it would always find the maximum of the SoG. However, the time complexity to perform that, especially on higher dimensional objects goes up drastically. So, an optimal solution can be found, but not necessarily in a timely manner. Figure 5 (see below) gives a good representation of how simulated annealing searches through the search space and settles on the local maxima.

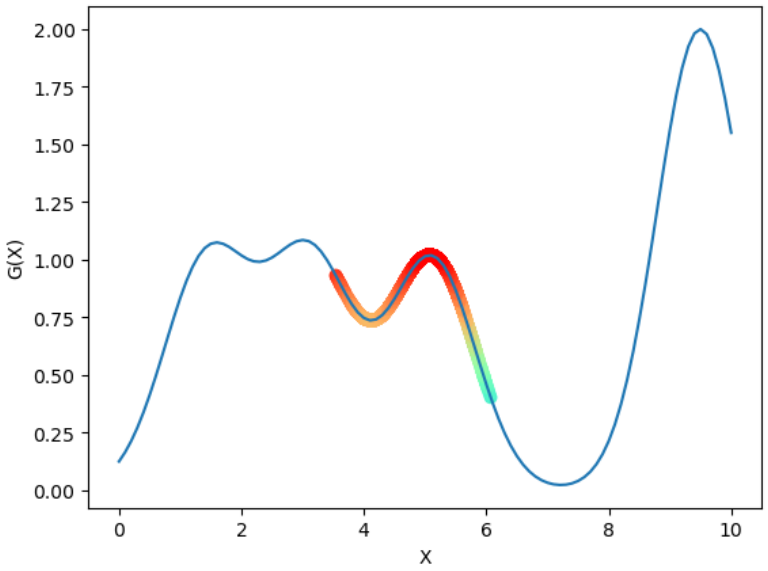

Figure 5: The explorative nature of Simulated Annealing

Results

Dimensions	Number of Gaussians	Greedy Higher G(x) than SA (win)	SA Higher G(x) than Greedy (win)	Overall Winner	Ties	Max Difference
1D	5	80	20	Greedy	0	1.988 (SA)
	10	77	23	Greedy	0	1.979 (SA)
	50	93	7	Greedy	0	4.388 (SA)
	100	89	11	Greedy	0	13.521 (SA)
2D	5	77	12	Greedy	11	2.0548 (SA)
	10	83	17	Greedy	0	1.283 (SA)
	50	68	32	Greedy	0	2.653 (SA)
	100	79	21	Greedy	0	3.624 (G)
3D	5	57	22	Greedy	21	.999 (G)
	10	71	18	Greedy	11	.999 (G)
	50	68	32	Greedy	0	1.167 (SA)
	100	62	38	Greedy	0	2.404 (SA)
4D	5	31	18	Greedy	51	.999 (G)
	10	47	21	Greedy	32	1.000 (G)
	50	78	21	Greedy	1	1.000 (G)
	100	72	28	Greedy	0	1.055 (SA)
5D	5	19	6	Greedy	75	.999 (G)
	10	27	18	Greedy	55	.999 (SA)
	50	64	21	Greedy	15	1.000(G)
	100	75	22	Greedy	3	1.000(G)

Wins and losses are determined by the maximum height of the Y value that each algorithm produced. 100 runs were performed for each case using both algorithms. The Y values are then compared between the outputs of the Greedy (G) and Simulated Annealing Algorithms (SA). The Max Difference is the difference between the local maxima found by each algorithm in all 100 run comparisons. (i.e. if Greedy produced point (1,1) for dimension: 1 Gaussians: 50 and SA produced (2,6) dimension:1 Gaussians: 50, then the max difference is 5 and SA found the higher point for that particular seed).

In the table I have shown we can see that Greedy finds a higher local max more often than Simulated Annealing. However, for the 40 sets of 100 runs when Simulated Annealing won, it ends up finding a much higher maxima than greedy does. If we look at a one dimensional and 100 sum of gaussians we see that Simulated Annealing found a local maximum that was 13.521

units above that of Greedy search. Since Simulated Annealing is an explorative algorithm (*see Figure 5*), it makes sense that it would find a more optimal solution than greedy would. Greedy just takes the first local max it can find, but simulated annealing will search across the function until it finally settles into a local maximum, allowing it to consider more options which may lead to better optimality.

In Conclusion, when Simulated Annealing does find a higher value for G(x), then it performs more optimally by a large margin than that of greedy. However, in the case of this problem and with the annealing schedule I had defined for Simulated Annealing, greedy would be a better approach if trying to just find a maximum. If the goal is to find the absolute max, simulated annealing may be a better approach, but if the goal is to find a single maximum, then greedy is the better approach.

Work Cited

https://www.baeldung.com/cs/simulated-annealing

Overview on simulated Annealing.

https://www.mathworks.com/help/gads/understanding-simulated-annealing-terminology.html

This page talks about temperature and terminology used in SA.

https://towardsdatascience.com/optimization-techniques-simulated-annealing-d6a4785a1de7

Article on optimization techniques.

https://www.khanacademy.org/math/statistics-probability/modeling-distributions-of-data/more-on-normal-distributions/v/introduction-to-the-normal-distribution

Gaussian Distribution video from khan academy

https://www.geeksforgeeks.org/introduction-hill-climbing-artificial-intelligence/

Introduction to hill climbing through greedy

https://courses.cs.washington.edu/courses/cse573/12sp/lectures/04-lsearch.pdf

Really good powerpoint on SA and Greedy.

https://chat.openai.com/share/0f2d9149-add6-496b-b831-ae611989ab35

ChatGPT to help with figuring out how to format the data we made through our 100 iterations for each case.