Monte Carlo Optimization and the Traveling Salesman Problem

Calen Jackman

April 26, 2019

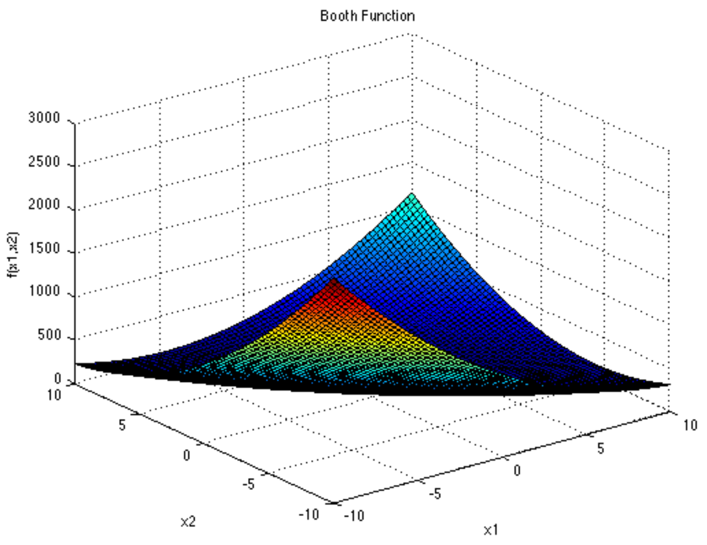
1 Introduction

Application of Monte Carlo to optimization is a widely researched subject. Optimization problems like the Traveling Salesman problem (TSP) are one of the hallmark problems used to illustrate the power of MC methods when applied to higher dimensional optimization problems. The TSP is a NP-hard problem in combinatorial optimization. The solution to it is finding the shortest path based on distance between N points/cities. In this paper we will explore a beginner optimization method known as Stochastic Gradient Descent that is used on basic, lower dimensional problems and then a method known as Simulated Annealing to solve the TSP.

2 Stochastic Gradient Descent with RNGs

Gradient descent, also known as steepest descent, is a method of finding the minimum of a function using the gradient of the corresponding function. Stochastic Gradient Descent is another method that involves estimating the gradient of the function by use a finite difference method. One can use random number generators to help determine the learning rate or step-size(α) and the difference (β). Using constants for these values is another option but is it faster than using RNGs?

In this problem I chose to find the global minimum of the Booth Function.

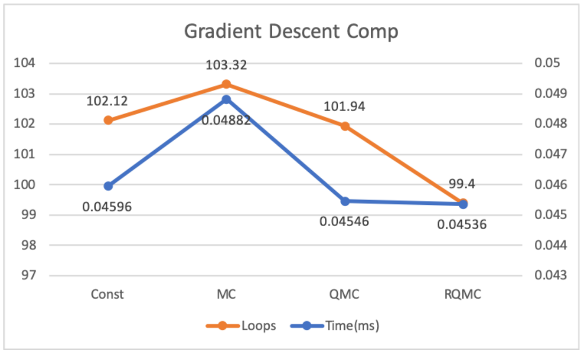


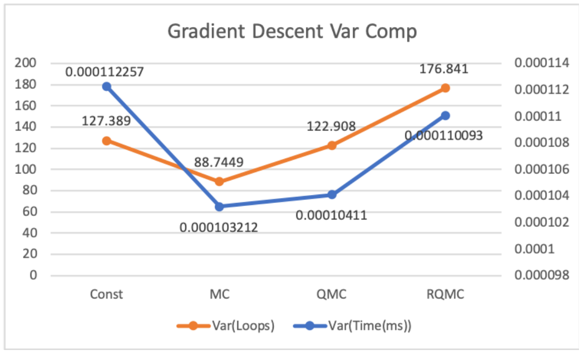
$$f(x) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2$$

It has 2 variables and is a relatively simple function to optimize. You may also apply the algorithm (although it may need some adjustments as far as the gradient calculation) to other higher dimensional optimization problems but for simplicity I stayed here.

To determine the effectiveness of using a RNG vs. constants as the learning rate and difference factor I would use 3 RNGs (Mersenne Twister, QMC (Halton), RQMC (Random Shift Halton)) to determine α and β. The range of values were 0 to 0.1 for each of the RNGs to make sure step sizes were not too large. The constants used were 0.05 for each of the values. Each of the methods were very efficient at finding the correct root so error was not a problem that I felt need to be explored in this context. Each of the four methods were sampled 50 times using the same starting points for each of the methods during each loop. The starting points were determined in all cases by Mersenne Twister from a $U(-10, 10)$. I measured the time it took and the number of loops it took to find the root. To show true depictions of the speed of the actual computing time for the method I constructed an array of values for each of the RNGs before I started the timing. This enabled the accurate timing of the methods so that it would not be skewed in favor of the constant method or the Mersenne Twister while the Halton sequence would have been seen to be slower if this process was included.

What did I find? (In the pictures below you see the Means and Variances for each of the methods in the two metrics measured)





The normal MC method (Mersenne Twister) performed the worst among the three methods while RQMC performed the best in terms of the mean. The constant method was the least consistent in terms of the time taken while the RQMC method, surprisingly, was the least consistent in terms of amount of loops. Mersenne twister, although performing worse than the other methods in time and amount of loops was the most consistent in both categories. As previously stated, error was not investigated because after repeated trials of each method there seemed to be very negligible errors in all three methods. In other words, they were all finding the global minimum regardless of starting point within the bounds.

Although each of the methods performed very well when finding roots to a simple function like the Booth function the methods quickly start to not converge to a global minimum when using a function with multiple local minimums. This is not a symptom of the method chosen such as RNG vs. constants but it is a symptom of the larger choice of optimization method. Gradient Descent is a method that does not do well with functions with almost any amount of local minimums. It will get stuck around a local minimum and does not achieve global optimization. More sophisticated methods are needed to solve more complex optimization problems.

3 Simulated Annealing and the Traveling Salesman Problem

Simulated Annealing (SA) is an optimization method that is widely applied to optimization problems of higher dimensions. It is known as a heuristic algorithm meaning that is not guaranteed to converge to a solution but can be very useful. In the TSP we are looking at a function that maps the path from city to city for N cities. As N grows larger you can see that a brute force method will become difficult. SA helps simplify this problem so that it may be solved for an arbitrary amount of cities.

SA takes a random initial guess for the path of the salesman. From this point you could take multiple paths. One method is to take a "neighboring state" of the initial guess by randomly selecting a step in the path and switching it with a nearby city in the path (usually something like the city visited before). This new candidate path will now be measured. If the path has a shorter distance (function value) the path is accepted and the process repeats. If the path has a larger distance we will compute an acceptance probability.

This acceptance probability requires a few more additional initial conditions to be set before the initial random guess. This probability is one that depends on the magnitude of the difference between the two distances (function values) and a Temperature (scaling factor). The initial temperature can be set in multiple different manners. One way is to first set the inital temp $T = k\sigma_\infty^2$ where σ_∞^2 is the variance of the distance distribution when initial temperature is set to a very large number and k is a constant value from 5 to 10.[1] After determining the initial temperature it will be scaled down by a "cooling schedule" that can take many forms such as $T_k = \frac{\alpha T_0}{\ln(1+k)}$ or geometric schedule such as $T_k = \alpha^k T_0$ where α is usually taken to be between 0 and 1 with more optimal choices being close to 1.[2] For my algorithm I chose the latter.

$$P = \exp\left(-\frac{\Delta E}{T_k}\right)$$

You also must set a stopping criteria for the algorithm. In my case I chose to use a criteria where I limited the number of consecutive acceptances of the same value (candidate not accepted for a number of consecutive loops).[3]

To explore this method thoroughly I decided to take some different variables from my algorithm and estimate the dependence of the error of the final solution on each of these variables. They include:

1) initial temperature

2) cooling constant

3) consecutive acceptances

4) number of variables (cities)

5) method of RNG (Mersenne, QMC, and RQMC)

To examine the effect of changing these variables I had a control set to use for the other variables while testing one of them:

1) initial temperature = 5

2) cooling constant = 0.95

3) consecutive acceptances = 10

4) number of variables (cities) = 16

5) method of RNG (Mersenne, QMC, and RQMC) = Mersenne

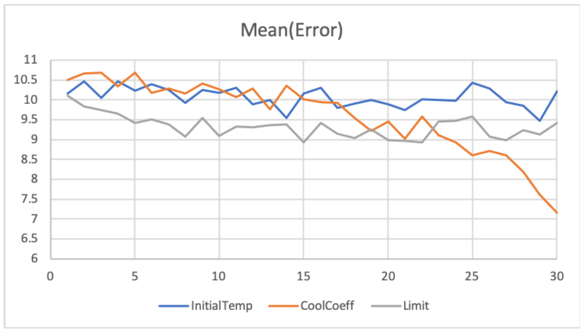
For the first 3 variables in the above lists I generated 30 different values for each of them.

1) initial temperature (1 - 10.667)

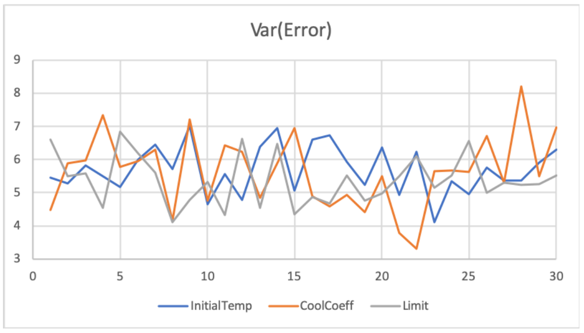
2) cooling constant (0.916 - 0.999)

3) consecutive acceptances or "limit" (10 to 68)

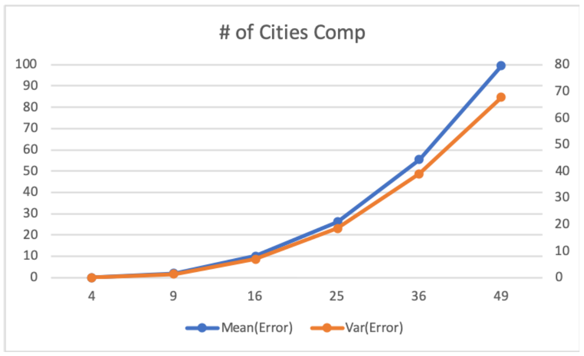
I then calculated the mean and variance of 100 final solutions for each value of the variable. For the last two, instead of 30 different values for each variable I decreased the amount of values based on the variable (# of cities = 6 and RNG = 3). Each value of the number of cities used were squares of a consecutive set of numbers. This is because when these are the values for this variable it is easy to determine the optimal distance that we should get and we can therefore determine the error easily. The graphs for this are on the next page.



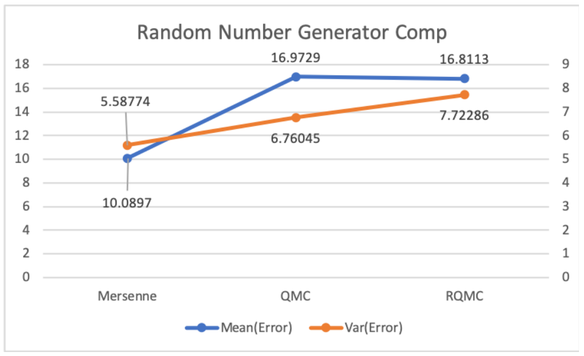
(1)



(2)



(3)



(4)

(1) From this graph we can see that the change in cooling coefficient affected the final error the most as it increased, starting out with the highest error and steadily decreasing to the lowest. The other two variables did not have as much as an effect.

(2) None of the variables helped variance much as they increased.

(3) As the number of cities increased both the mean and variance of the error increased as well. This may be an effect of the method used to switch cities which I will discuss later.

(4) The effect of the RNG seems to be pretty significant when looking at the means of the error with Mersenne being the most accurate and having the least variance.

The significant increase in error based on the amount of cities may be due to the choice to switch cities that are directly next to each other in the initial path. In my opinion the reason this method is a bad choice is because when we choose an initial path the cooling coefficient must be so close to 1 so that the random switches can explore many more possibilities. The method of switching does not allow for enough variety in candidate paths. In this way the initial path seems to have a large effect on the final solution given by the algorithm. This may not play a large roll when the amount of cities are something like 9 or 16 but when the number jumps to an arbitrarily large value we need to change the method of choosing candidate paths. To improve upon the method I believe the best course of action would be to change to some of the following:

(a) generate 2 random picks for steps in the path and switch those

(b) generate a relatively large number of random picks and then switch the steps based on each pick in sequence (if you have a pick sequence of 2, 11, 8, 15 you would switch steps 2 and 11, then 11 and 8, then 8 and 15)

(c) generate an entirely new random path (most costly method)

4 Conclusion

While Stochastic Gradient Descent seems to be a good way to solve simple optimization problems it is limited in its scope. It is a good way to explore the basics of optimization. In higher dimensions this method would seem to become very costly when calculating gradients of each variable. Simulated Annealing is known as a Heuristic algorithm and there exist many other methods like it. A few examples of these algorithms are Particle Swarm Optimization, Stochastic Tunneling, Differential Evolution, etc. Simulated Annealing and these other heuristic algorithms are more versatile optimization methods and can be applied to those same higher dimensional optimization problems where Stochastic Gradient Descent would be very costly and my fail to find global minimums.

References

[1] Walid Ben-Ameur. Computing the initial temperature of simulated annealing. *Computational Optimization and Applications*, 29(3):369–385, Dec 2004.

[2] Serdar Cellat, Yu Fan, Washington Mio, and Giray Ökten. Learning shape metrics with monte carlo optimization. *Journal of Computational and Applied Mathematics*, 348:120 – 129, 2019.

[3] Rafael E. Banchs. Simulated annealing. Research Progress Report 15, The University of Texas at Austin, 1997.