A Review and Trade Study of Simulated Annealing

Nicholas Andrews

***Abstract*—Simulated annealing is a stochastic global optimization algorithm that has demonstrated efficient and consistent performance for approximating global solutions to large scale optimization problems. Applying simulated annealing to a particular problem requires the user to provide 2 hyperparameters: a perturbation function and annealing schedule. This report develops a methodology for selecting these parameters through numerical simulations of the travelling salesman problem. It was found that a reannealing schedule paired with a perturbation scheme that swapped 2 randomly selected cities in the route provided the best performance on average. The results found from these experiments can easily be extrapolated to other combinatorial problems, and on an even broader scale, to other classes of optimization problems.**

I. Introduction

In some instances an optimization problem cannot be formulated as convex and robust algorithms are required to approximate a solution. Simulated annealing is one such algorithm and can be applied to smooth, discontinuous, or combinatorial objective functions. The objective of this project is to develop a deeper understanding of the simulated annealing algorithm so that it can be effectively implemented in real world application. In particular, experiments with different hyperparameters were conducted to understand the performance sensitivity of each and how to design them according the application. Lastly, directions for future work with applications to reinforcement learning and global optimization are proposed.

II. Background

Simulated annealing is a stochastic global optimization algorithm that was popularized in 1983 by S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi [1]. The algorithm takes inspiration from the field of statistical mechanics and the study of thermodynamic energy states during the annealing process in metallurgy. The objective during the annealing process is to slowly heat and cool metals in to lower energy crystalline states so that they have more advantageous material properties for manipulation and crafting. The Metropolis-Hastings algorithm proposed in 1953 was designed to model and predict the most probable minimum energy structure during the annealing process [2]. The fundamental idea behind the simulated annealing algorithm comes from connecting the parallel between minimum energy states in statistical mechanics and minimum costs/objectives in optimization.

Global optimization is the broadest class of optimization problems and seeks to approximate a global minimum in a cost landscape with many local minima. While gradient methods are known for their precision, they are not well suited for this problem because they are sensitive to the initial condition, fail to explore the cost landscape, and thus converge to the nearest local minimum. As a whole it is difficult to provide guarantees on convergence to a global minimum because of the vast variety of functions covered in this problem set. In some specific applications the cost landscape may be low dimension and well known, in which case a global optimization algorithm may be tailored specifically for this use case. However, an ideal global optimization algorithm should be robust and perform well across all cost landscapes. Simulated annealing is one of the global optimization algorithms that has demonstrated consistent performance at approximating global minima. Some other common global optimization algorithms are: genetic algorithms and particle swarm optimization. The common theme across all global optimization algorithms is balancing exploration vs exploitation; exploring the cost landscape vs exploiting low cost basins.

N. Andrews is a PhD student with the Department of Aeronautics and Astronautics, University of Washington, Seattle, WA, 98195 USA e-mail: nian6018@uw.edu

III. Algorithm

Pseudocode for simulated annealing is shown in algorithm block 1. The algorithm requires the user to provide an initial guess x_0, a function for randomly perturbing the state (**perturb**), and an annealing schedule (**schedule**). The annealing schedule sets the "temperature" of the algorithm which subsequently controls how many iterations the algorithm runs for as well as how much exploration the algorithm will do. The while loop beginning on line 2 progresses through the annealing schedule until the final temperature is reached.

At each iteration inside the loop the current point x^* is randomly perturbed to generate a candidate point $x_\triangle$. If the cost of the candidate point is less than the cost of the current point, then the current point is updated to the candidate point. If the cost of the candidate point is greater than the cost of the current point, then the candidate point is randomly accepted or declined based on a transition probability that is a function of the current annealing temperature. At higher temperatures the algorithm is more likely to transition to a higher cost point; this is equivalent to exploring the cost landscape. As the temperature decreases so does the probability of accepting a higher cost point, and the algorithm transitions from exploration to exploitation.

It is important to note that the algorithm presented here is the original form of the simulated annealing algorithm proposed in [1]. There have since been several extensions which will be discussed later. The presented algorithm works for discrete and continuous cost functions, and with some slight modifications, can handle constraints as well. The perturbation function and annealing schedule are the 2 primary tuning parameters and require the operator to apply domain

Algorithm 1 Simulated Annealing

$x^* = x_0$
while $T \geq T_f$ **do**
  $x_\Delta = \textbf{perturb}(x^*)$ ▷ Perturb current point
  $P = \exp(-\frac{\Delta}{kT})$ ▷ Uphill transition probability
  if $f(x_\Delta) < f(x^*)$ **then** ▷ Downhill update
    $x^* = x_\Delta$
  else if $\mathcal{U}(0,1) \leq P$ **then** ▷ Uphill update
    $x^* = x_\Delta$
  end if
  $T = \textbf{schedule}(T)$ ▷ Update temperature
end while

expertise when choosing them; these are discussed further in the following subsections. The transition probability function is also a tuning parameter, but to a lesser extent. Tuning the annealing schedule achieves a similar effect to manipulating the transition probability function.

A. Perturbation Function

The purpose of the perturbation function is to introduce a degree of randomness in to the system and produce exploration. The degree of a perturbation is proportional to the amount it can change the cost in a single step. Keeping this in mind, we want a perturbation function that induces enough of a perturbation to escape local minima and explore the global cost landscape, but not too much that it never exploits. It is difficult to quantify too much or too little randomness a priori, which is why the perturbation function is considered a tuning parameter. The following subsections will introduce specific use cases of this trade off in the continuous and discrete action spaces and how the user could tune appropriately.

1) Continuous: In the continuous action space a commonly used perturbation function is to draw the candidate point from a Gaussian distribution with mean centered at the current point. In this case the covariance of the distribution determines the size of the perturbation. Another valid perturbation function would be a uniform distribution centered about the current point. Similarly, the bounds on the uniform distribution determine the size of the perturbation.

In both instances it can be difficult to determine what appropriate values are to assign to the perturbation bounds. If the problem is bounded, then setting the bounds of the uniform distribution to the bounds of the problem reduces simulated annealing to a simple random search. In this instance the bounds are clearly too large because a majority of candidate points will be rejected as the temperature decreases and the exploitation benefit is lost. The covariance or distribution bounds are then hyperparameters for which multiple values should be tested when optimizing.

2) Discrete: Perturbation functions for discrete actions spaces are typically problem specific and require some domain knowledge when crafting them. Consider the travelling salesman problem where the decision variable is a list which defines a route through the cities. 3 possible perturbation functions for the route are: flip the order of any 2 adjacent cities, swap positions of any 2 cities, and swap positions of n cities. Of the 3 methods it is clear that the first method induces the smallest perturbation and the last method induces the most. Flipping the order of any 2 adjacent cities will take more iterations to explore different routes, which can be particularly problematic if the initial guess was not very good. In comparison, swapping n different cities will do a good job of exploring the landscape but will struggle to exploit as the temperature decreases. In a scenario where there are clusters of cities, swapping n cities will perform well at finding an optimal route for connecting the clusters, but struggle to optimize the routes within each clusters.

B. Annealing Schedule

The annealing schedule determines the probability of accepting a higher cost candidate point. A slow annealing schedule allows for more of the landscape to be explored, but results in a longer runtime. The most common approach to designing an annealing schedule is to choose uniformly spaced decreasing temperatures and then spend an arbitrary amount of iterations at each temperature. Another approach, called "reannealing", follows a similar approach but the temperature will occasionally increase during the global decline. The benefit of reannealing is that it allows for intermediate exploration phases as it approaches an optimal solution. Lastly, an adaptive annealing schedule is one that uses a metric to monitor the progression of the algorithm and to slow or accelerate the cooling process accordingly. In [1] the derivative with respect to temperature of the average value of the objective function observed at a given temperature is proposed as an effective heuristic for adaptive cooling.

C. Extensions

1) Dual Annealing: Dual annealing is the process of running the simulated annealing algorithm and then using the solution as the initial guess for a gradient method. The idea behind this pairing is that simulated annealing provides a good approximation of the global solution and then a gradient method is used to further refine the solution.

2) Generalized Simulated Annealing: Generalized simulated annealing (GSA) uses the same overall framework as classic simulated but has better runtime and convergence properties. GSA is able to achieve this by using the sample points to estimate a distribution over the cost landscape. This distribution is then used to draw candidate points and generate the probability of accepting a point with higher cost. GSA is able to achieve better runtime and convergence results because it samples its perturbed points in a more informed way, as opposed to classic simulated annealing's purely random sampling [3].

IV. Results

The primary objective of this project was to develop code for simulated annealing, experiment with different cost functions, perturbation functions, and annealing schedules, and then compare results in hope that the lessons learned from

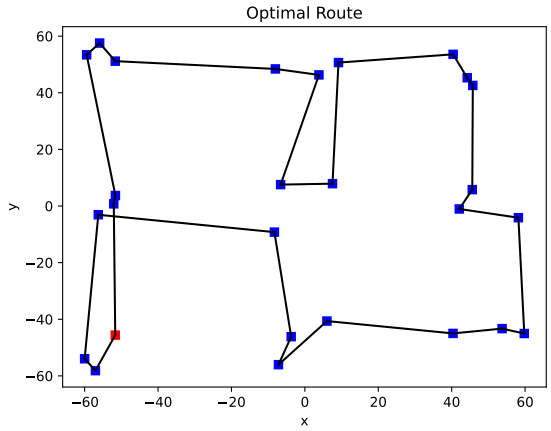

Fig. 1. Optimal route using a reannealing schedule and swapping any 2 cities.

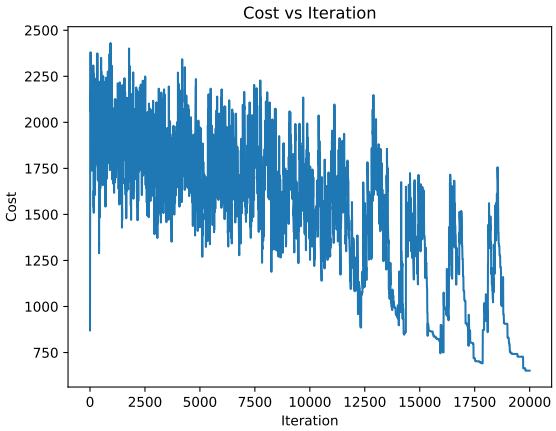

Fig. 2. Cost vs number of iterations using a reannealing schedule and swapping any 2 cities.

these experiments will make it easier to then apply simulated annealing to other research or industry applications. SciPy's optimization module provides a dual annealing function, but it is only designed for continuous action spaces and does not provide intermediate results, making convergence and tuning sensitivities difficult to quantify. In this project simulated annealing was applied to the continuous and discrete action spaces. In the continuous space several nonlinear functions, such as the Rastringin function, were used to validate performance. In the discrete case the travelling salesman problem was chosen as a baseline test problem. For the sake of brevity and consistency with how performance is compared in [1], the travelling salesman problem was chosen to be the sandbox for comparing different tuning parameters.

Replicating the baseline used in [1], the layout of the cities is such that there are 9 clusters, each cluster containing 3 randomly placed cities. The reasoning behind this design is that it should make it easier to observe simulated annealing reducing its exploration by first teasing out the route between clusters, and then optimizing the route within each cluster. The cost function we wish to minimize is the distance required to visit all points and return to the starting location.

Two different perturbation functions were experimented with when perturbing the route: flip the order of 2 adjacent cities and swap positions of any 2 cities. Three different annealing schedules were developed for decreasing the temperature: linear, logarithmic, and logarithmic with sine. The logarithmic with sine schedule is designed to replicate the reannealing behavior. For consistency across the annealing schedules each schedule is evaluated at 200 temperatures and for 100 iterations at each temperature, for a total of 20,000 iterations per run. Sample results for swapping any 2 cities with a reannealing schedule are shown in figures 1 through 4. A single run of this setup takes 13.5 seconds to run on a 2015 MacBook Air.

Compared to the other possible configurations of annealing schedules and perturbations, a reannealing schedule and swapping positions of any 2 cities gave the best consistent performance. Due to the stochastic nature of simulated annealing,

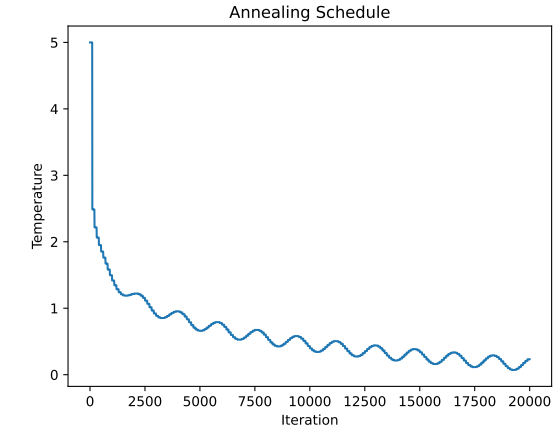

Fig. 3. Reannealing temperature schedule.

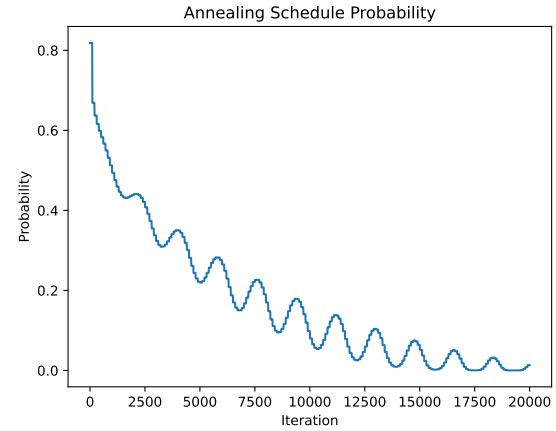

Fig. 4. Reannealing probability schedule.

each perturbation and annealing schedule configuration was run 10 times and the optimal cost was compared across all runs. The best overall configuration is then whichever had the lowest cost a majority of the time. A configuration that

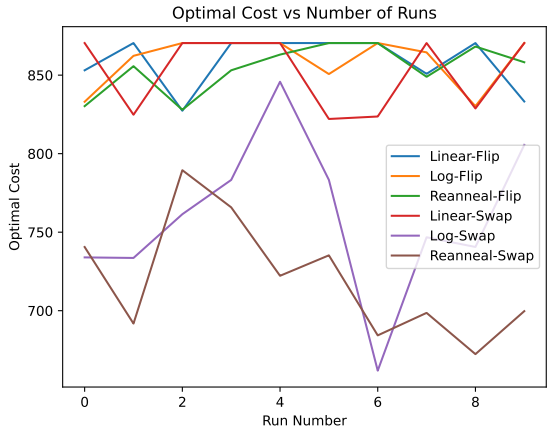

Fig. 5. Optimal cost across 10 runs for different perturbation function and annealing schedule configurations.

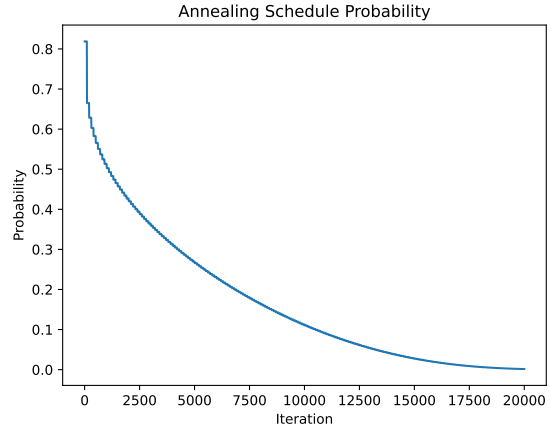

Fig. 6. Logarithmic probability schedule.

has lower optimal cost variance is indicative of consistent convergence to a minima, whereas a configuration with high variance is the result of cooling too quickly or perturbing too much. Results are shown in figure 5. The logarithmic annealing schedule had comparable performance but had much higher variance. The logarithmic annealing schedule is shown in figure 6.

V. Future Work

Now that the simulated annealing framework has been developed and explored I would like to apply what I've learned in my research and work. Some proposed future directions are discussed in the following subsections.

A. Reinforcement Learning

One of the original objectives of this project was to apply simulated annealing to the reinforcement learning framework. A particular application that first came to mind was to use simulated annealing in policy or value iteration for non-quadratic costs with large action spaces. The value iteration optimization equation is shown in equation 1.

$$(Tv)(x) = \min_{u \in U} \sum_{x^+ \in X} P(x^+|x,u)(\mathcal{L}(x,u) + \gamma v(x^+)) \quad (1)$$

This result was then going to be compared against the simple random search results from [4]. Unfortunately this objective was not achieved within the time frame of this project, but will be explored in the following months.

B. Comparison to Other Methods

Another future line of work is to compare simulated annealing to other global optimization algorithms such as: genetic algorithms, swarm algorithms, and gradient methods with random initial conditions. The objective of this work is to understand the benefits and limitations of each method so as to better choose the right algorithm for application in the real world. The "correct answer" is rarely verifiable in global optimization problems, so it is often a good idea to compare performance from a variety of approaches to achieve the best approximate solution.

VI. Conclusion

Simulated annealing is a proven and effective global optimization algorithm. It was developed for continuous action spaces but naturally extends to discrete and constrained action spaces as well. The algorithm was implemented in Python and tested against the travelling salesman problem for a variety of perturbation functions and annealing schedules. It was found that a reannealing schedule and perturbation function with larger induced randomness gave better results.

All code developed for this project is available on GitHub at: github.com/nian6018/Simulated-Annealing

References

[1] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, "Optimization by simulated annealing," en, *Science*, vol. 220, no. 4598, pp. 671–680, May 1983.

[2] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, Jun. 1953. DOI: 10.1063/1.1699114.

[3] C. Tsallis and D. A. Stariolo, "Generalized simulated annealing," *Physica A: Statistical Mechanics and its Applications*, vol. 233, no. 1-2, pp. 395–406, Nov. 1996. DOI: 10.1016/s0378-4371(96)00271-3.

[4] H. Mania, A. Guy, and B. Recht, *Simple random search provides a competitive approach to reinforcement learning*, 2018. DOI: 10.48550/ARXIV.1803.07055.