Kyle Buie (kbuie1)
Aaron Cummings (acummi28)
April 24, 2021
CS 3642 Section W01
Research Project

Machine Learning, Stochasticism, & Gradient Descent: A More Efficient Approach to Simulated Annealing Using Optimization

New Contributions

Our new algorithm applies the realm of machine learning to the existing Simulated Annealing algorithm. We demonstrate how adjusting the annealing algorithm to make more moves per temperature value can increase the probability of finding a solution. We also explore how a learning algorithm can be applied to optimize and automate selection of the important variables, such as the maximum temperature, step size, and annealing rate, for a given type of optimization problem.

Abstract

We begin this paper by providing general background information on Simulated Annealing and learning algorithms followed by our problem definition. We also include a brief section on literature reviews pertaining to these two fields of AI. Next, we propose our enhancement to the annealing algorithm and then describe our implementation and experimental results. We also discuss the advantages and drawbacks of the algorithm's design and implementation.

BACKGROUND INFORMATION

Simulated Annealing is a greedy hill climbing algorithm based on the process of annealing in metallurgy. The algorithm applies to optimization problems in order to find a solution or near-solution state. The algorithm starts at a high temperature, and gradually reduces the temperature by a given rate, until it reaches a quenching point. As the temperature is reduced, the algorithm gradually reduces the probability of accepting states of a lower energy, which means bad moves are less favorable. Near the beginning of the problem, the algorithm is more likely to accept bad moves in order to navigate around local maxima or minimums.

In a learning algorithm, the outcome of a problem adjusts the algorithm based on an expected outcome. Training data is used to adjust the solution to the problem and can then be tested with other data.

PROBLEM TO BE SOLVED

The central issue that we are addressing is improving the success rate of the annealing algorithm. When applying simulated annealing to an optimization problem with random start states, there is an ideal range of values for temperatures, the step size, and the annealing rate. In order to find these values, we underwent trial and error to find ideal conditions to solve this problem.

LITERATURE REVIEWS

Handbook of Metaheuristics: Chapter 10: The Theory and Practice of Simulated Annealing

This essay provides an overview of the Simulated Annealing algorithm. The author begins by describing the history of the algorithm and its applications to discrete and continuous optimization problems. The author also explores the convergence aspect and the performance (in terms of time) of the algorithm. Next, the author compares other existing search algorithms to Simulated Annealing. Finally, the author discusses the practical implications of simulated annealing (Henderson et al. 287-319).

Simulated Annealing: Practice versus Theory

This paper proposes the problem that in practice, Simulated Annealing is quickly out paced by other algorithms. The author proposes his application of Simulated Quenching, an approach which utilizes exponential or logarithmic functions to adjust the rate of temperature change or the temperature schedule. The author expands on several ideas of hybridizing the traditional simulated annealing algorithm with other optimization approaches, such as genetic algorithms and neural networks (Ingber 29-57).

PROPOSED ALGORITHM ENHANCEMENT AND APPLICATIONS

We propose that finding the ideal conditions for using simulated annealing to solve a problem that has randomized initial states can be optimized through a learning algorithm. This algorithm will find the best condition for one variable given static variables for the other conditions.

This enhancement of the algorithm will cut down on time spent optimising new Simulated Annealing applications. In problems with many to infinite starting states and solutions, allowing the algorithm to train itself on test data will improve the rate at which it can produce meaningful results given a new problem. For example, in creating an application to optimize production line layout, there are many combinations and layouts of machinery given a space to organize them. With this enhancement, giving the algorithm a new layout to organize, knowing the expected results as testing data, it can allow itself to optimize a particular variable. Then the optimized application can be applied to new problems. Given this possibility, it should also be feasible to have a simulated annealing application that optimizes all variables given a random starting condition for each.

IMPLEMENTATION AND EXPERIMENTAL RESULTS

Here is the pseudocode demonstrating the implementation of the learning algorithm:

Annealing Algorithm Class

```
class algorithm{
    double temp;
    double tempMax = value(passed);
    double tempMin = value(.01);
    double delta;
    double eOld;
    double eNew;
    int iMax = value(10);
    Random rand = new Random();
```

```
        NodeObject<> tempMove = new NodeObject();
        Double annealingRate;

        constructor (double tempMax){}

        public double annealingAlg(NodeObject Y) {
                for (temp = tempMax; temp >= tempMin; temp = nextTemp(temp)){
                        for (int i = 0 ; i <= iMax; i++){
                                tempMove = Y.successorFunction();
                                eOld = Y.energyFunction(Y.state);
                                eNew = Y.energyFunction(tempMove);
                                delta = eNew - eOld;
                                if (delta<0){
                                        double random = rand.nextDouble();
                                        //System.out.println(random);
                                        if (random > Math.exp(delta/temp)){
                                                //reject bad move
                                        }
                                        else{
                                                //accept bad move
                                                Y.acceptSuccessor(tempMove);
                                        }
                                }
                                else{
                                        //always accept good moves
                                        Y.acceptSuccessor(tempMove);
                                }
                        }
                }
                return goalStateEnergy – currentStateEnergy;
        }

        //reduces temp
        private double nextTemp(double tempIn){
                return tempIn * annealingRate;
        }
}
```

Learning Algorithm

```
Double successRateGoal = 0.05
Queue<bool> lastjMaxMoves = new queue
Double currentSuccessRate = 0
Int jMax = 20
Double deltaEnergy = 0.0

While (currentSuccessRate < successRateGoal){
    For 0 to jMax {
        NodeObject<> temp = new NodeObject<>
        deltaEnergy = annealingAlg(maxTemp);
        Boolean isGoal = False
        If (deltaEnergy  = 0)
            isGoal = True
        lastjMaxMoves.push(isGoal)
        If (last_20 size > iMax)
            Pop lastiMaxMoves
    }
    Int numGoal = 0;
    For 0 to jMax{
        Boolean temp = lastjMaxMoves.peek()
        If temp == true
            numGoal++
        lastjMaxMoves.pop()
    }
    currentSuccessRate = numGoal / jMax
    If (currentSuccessRate < successRateGoal) {
        iMax = iMax + 3
        // OR (these are the learning functions for the variables adjusted in experiments)
        tempMax = tempMax * (1-(successRateGoal - currentSuccessRate))^-1
        // OR
        annealingRate = annealingRate + 0.0001
    }
}
```

Test Problems

The problems used here are toy problems: one for commonplace testing for annealing algorithms, and one of our own design. The first problem used is the 8-puzzle problem, where there is a state of 9 places with 8 pieces and one empty place. The empty place is switched with an adjacent piece each move. The second problem used has a scrambled word that is unscrambled by switching adjacent letters one at a time.

Results

We found that this algorithm converged on values that would allow the Simulated Annealing Applications implemented to produce the desired success rates. We also found that for each problem, there is an upper limit on the success rate you can expect when optimising one variable given the other conditions. If this were to be implemented on all variables simultaneously, it is likely that the algorithm would need to adjust its own success rate goal after meeting it. Increasing the desired success rate until an upper limit is found would fully optimise the algorithm for the given application without the implentor’s input.

ADVANTAGES AND DRAWBACKS

An advantage of this algorithm is that adjustion of the number of iterations per temperature value is automated in nature. In the original annealing algorithm, the user is administered a heavy dose of trial and error when determining the values that will produce the best results. In other words, the user has to manually set the annealing variables before each execution of the program. Fortunately, the learning capability of our agent makes the user's job easier, as he/she can focus on observing the automated behavior of the algorithm without having to worry about tediously finding the range of optimal values over numerous executions.

Another advantage of this algorithm is that it boasts flexibility. Learning functions for the annealing variables (i.e. maximum temperature, iterations per temperature value, and annealing rate) can easily be inserted into the program without changing the core structure of the algorithm (refer to the pseudocode for the learning portion of the algorithm). On the other hand, a corresponding limitation is that the agent is not powerful enough to adjust/learn over all the variables at once. In other words, one learning function would be active at a time while the other two functions would be commented out. This limited capability may be an overall detriment to the algorithm's success, especially in terms of testing; however, it can actually be beneficial for training purposes. Adjusting/learning over a single variable at a time actually refines focus. For instance, if the agent were to adjust both the maximum temperature and the number of iterations per temperature value, it would be more difficult for the user to identify a pattern or behavior where he/she is supposed to.

An additional drawback of this design is that training requires a fixed initial problem state to produce clear results. In the case of the eight-puzzle, the numbers and the "blank tile" were hard-coded into the program, not randomly generated. With that being said, manually setting the initial state ensures a fair series of training trials. Upon examination of the eight-puzzle, there are approximately 1.8349 * 10^21 possible states from which to choose for initialization. Given this many possible start states, one can expect our training portion of the algorithm to exhibit far greater and less desirable variance when each problem is randomly generated. In short, learning over a fixed problem may not be as powerful but it does demonstrate consistency.

CONCLUSION

We found that given a set of test states for the algorithm, we can converge on a small range of ideal values to be applied to the simulated annealing algorithm for random starting states in an optimization problem. Given this conclusion, it should be possible to find ideal values for all variables in the annealing algorithm with random initial values and a learning function to adjust them using static testing data and a reasonable desired success rate. The resulting values can then be used in the annealing algorithm application, with any randomized starting state to find a solution more efficiently.

Works Cited

Henderson, Darrall, et al. *Handbook of Metaheuristics*, Springer, Boston, MA, 2003, pp. 287-319. https://doi.org/10.1007/0-306-48056-5_10

Ingber, Lester. “Simulated Annealing: Practice versus Theory.” *Mathematical and Computer Modelling*, vol. 18, no. 11, 1993, pp. 29-57, https://doi.org/10.1016/0895-7177(93)90204-C. Accessed 22 April 2021.