Optimisation Algorithms Project

Table of Contents

Introduction

The aim of this project is to explore the theory, efficiency, drawbacks and uses of the optimisation algorithms. After reading the Hash Code brief, I wondered "What can these algorithms be used for?" - as it turns out, many sectors like Financial Markets, Healthcare, Energy, Manufacturing use these algorithms. It is a way to efficiently solve problems where resources are limited. In this report, I implement and analyse the Hill-Climbing and Genetic algorithms.

What do Optimisation Algorithms do?

The aim of the optimisation algorithms, in the context of the Hash Code problem, is to find the most efficient distribution of online videos given a set of caches, endpoints, costs, etc. On a big streaming platform like YouTube, optimising video sharing will enhance the user experience, reduce costs and strengthen scalability. In the Financial sector, optimisation algorithms are used to minimise risk in the stock market and in Health Care, they are used to manage staff, resources and optimise patient schedules.

Coming back to the Hash Code problem, there are certain constraints when it comes to optimising the content delivery such as cache sizes and endpoint latencies. The aim of the Hill-Climbing and Genetic algorithms is to deliver content efficiently to endpoints across the globe while taking the constraints into account.

Optimisation Algorithms - Design & Requirements

The Problem Requirements

THE GOAL

Objective: The main objective of the algorithm is to store videos into caches in a way that maximises the efficiency of video access.

THE CATCH

Storage Constraints: Each cache has a limited amount of storage which cannot be exceeded by the sum of all videos stored within it.

THE PLAN

Latency Optimisation: The algorithm will aim to minimise the latency from the data centre to the endpoints. This entails optimising the videos and caches while taking their varying latencies into account.

The Design

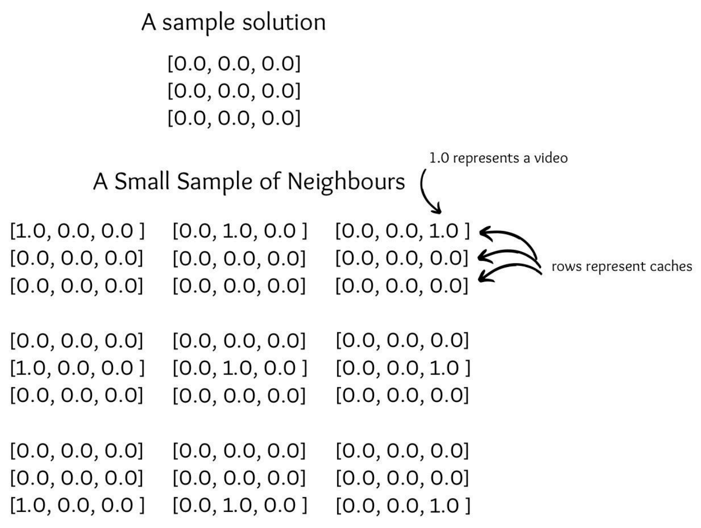
Following the problem requirements, I incorporated two popular optimisation algorithms: Hill-Climbing and Genetic Algorithms. Both algorithms are designed to provide optimal solutions toward our objective while navigating the constraints. To build these algorithms, I made use of many data structures such as 2D and 3D arrays, Array Lists and Hash Maps.

The Fitness Function

The fitness function is not the focus of this report however, it plays an important role in generating valid solutions for the optimisation algorithms. Essentially, the fitness function will take in a solution as a parameter and either return the fitness of a solution or if the solution is invalid. If the solution is invalid, this means that the sum of the video file sizes overflows the cache server.

Hill-Climbing Algorithm

The Hill-Climbing Algorithm works by generating “neighbours” of a sample solution. A neighbour is a potential solution that has a slight modification of the sample solution that was passed in. In the context of the problem, a neighbour is generated by placing or removing a video in a cache server.



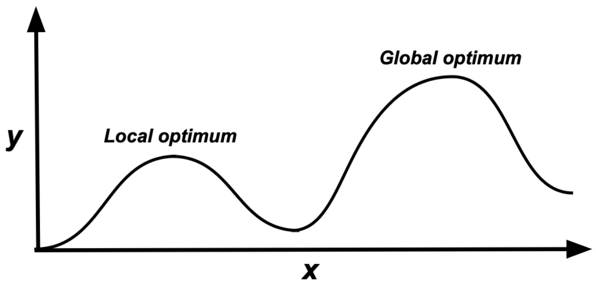
```
[0.0, 0.0, 0.0]
[0.0, 0.0, 0.0]
[0.0, 0.0, 0.0]

[1.0, 0.0, 0.0 ]   [0.0, 1.0, 0.0 ]   [0.0, 0.0, 1.0 ]
[0.0, 0.0, 0.0]    [0.0, 0.0, 0.0]    [0.0, 0.0, 0.0]
[0.0, 0.0, 0.0]    [0.0, 0.0, 0.0]    [0.0, 0.0, 0.0]

[0.0, 0.0, 0.0]    [0.0, 0.0, 0.0]    [0.0, 0.0, 0.0]
[1.0, 0.0, 0.0 ]   [0.0, 1.0, 0.0 ]   [0.0, 0.0, 1.0 ]
[0.0, 0.0, 0.0]    [0.0, 0.0, 0.0]    [0.0, 0.0, 0.0]

[0.0, 0.0, 0.0]    [0.0, 0.0, 0.0]    [0.0, 0.0, 0.0]
[0.0, 0.0, 0.0]    [0.0, 0.0, 0.0]    [0.0, 0.0, 0.0]
[1.0, 0.0, 0.0 ]   [0.0, 1.0, 0.0 ]   [0.0, 0.0, 1.0 ]
```

Once all the neighbours are generated, the algorithm will select the neighbour with the highest fitness and then generate neighbours of that solution. This will continue until there is no more improvement in the fitness of the solutions produced. The algorithm will return the solution with the highest fitness. This solution should be the most efficient set of caches which would reduce latency - optimising video sharing.

THE FACTORS & OBSERVATIONS

GREEDY ALGORITHM

The Hill-Climbing Algorithm has a tendency to only return the “local maximum”. This is because of the algorithm’s nature of always taking in the highest fitness - it doesn’t generate possible solutions outside of the neighbours already produced. This is a property of “Greedy Algorithms” and as a result, the solution that the algorithm returns might not always be the best possible solution which is a huge drawback.



INITIAL SOLUTION

The Hill-Climbing Algorithm is also heavily dependent on the initial solution that we pass in. Different starting points can result in different local optima. This means it is crucial to have a relatively fit initial solution so that the algorithm can generate possibly fitter neighbours.

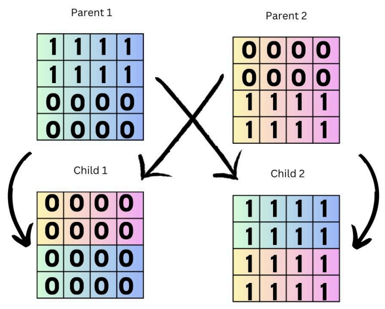
SIMULATED ANNEALING ALGORITHM

Similar to Hill-Climbing, the Simulated Annealing algorithm generates neighbours in the same way. However, to combat Hill-Climbing's greedy nature, the Simulated Annealing algorithm can backtrack and explore previous solutions. This means that there is a possibility that we can escape the local maximum and achieve the global maximum.

Genetic Algorithm

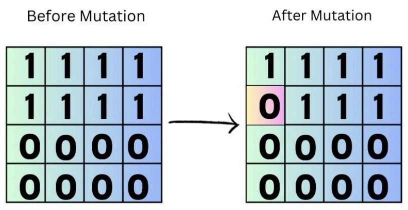
The genetic algorithm mimics the process of natural selection in a given population. This algorithm works by taking in a sample population. This sample population is generated by the Hill-Climbing function. The population will then go through the steps of crossover, mutation and selection. Iterations of this process are called "generations". By the end of this algorithm, the individual with the highest fitness is returned.

HOW IT WORKS



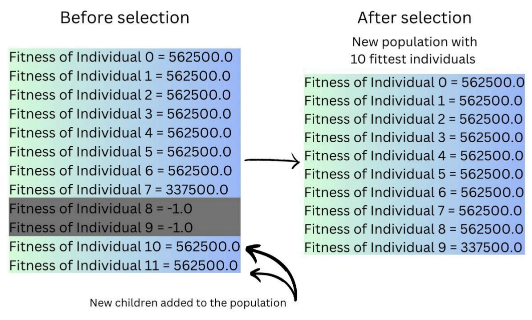
Crossover

For every pair of individuals in the population, there was a possibility that they would crossover. Crossover creates 2 children - each with one half of each parent. If these children are feasible, they will be added to the population.



Mutation

For every child produced, there is a chance that the child is mutated. This process involves inverting one or more elements in the child. This child is still added to the population regardless of its feasibility.


Before selection
After selection
New population with 10 fittest individuals
New children added to the population

```
Fitness of Individual 0 = 562500.0
Fitness of Individual 1 = 562500.0
Fitness of Individual 2 = 562500.0
Fitness of Individual 3 = 562500.0
Fitness of Individual 4 = 562500.0
Fitness of Individual 5 = 562500.0
Fitness of Individual 6 = 562500.0
Fitness of Individual 7 = 337500.0
Fitness of Individual 8 = -1.0
Fitness of Individual 9 = -1.0
Fitness of Individual 10 = 562500.0
Fitness of Individual 11 = 562500.0
```

```
Fitness of Individual 0 = 562500.0
Fitness of Individual 1 = 562500.0
Fitness of Individual 2 = 562500.0
Fitness of Individual 3 = 562500.0
Fitness of Individual 4 = 562500.0
Fitness of Individual 5 = 562500.0
Fitness of Individual 6 = 562500.0
Fitness of Individual 7 = 562500.0
Fitness of Individual 8 = 562500.0
Fitness of Individual 9 = 337500.0
```

Selection

The new population including children is now selected based on their fitness. Only the fittest individuals are passed through to the next generation where they are subjected to crossover, mutation and selection again.

THE FACTORS & OBSERVATIONS

ORIGINAL POPULATION

The original population was created by passing a sample solution into the hill-climbing algorithm. The result was then mutated some number of times to create a set of sample solutions based on the original solution. This method was used to generate a population of feasible and infeasible solutions with potential to crossover and generate fitter children.

CROSSOVER PROBABILITY

While experimenting with the crossover probabilities, I found that the higher the crossover value - the quicker the population would fill up with feasible, high fitness children. This makes sense intuitively, as more crossover occurs, the more children will be produced regardless of their feasibility.

MUTATION PROBABILITY

The opposite of crossover is said for mutation. I found that the higher the mutation value, the population would fill up with infeasible children. This also makes sense as if new feasible children are added to the population and then they are modified (mutated) - this modification will make them infeasible.

SELECTION PROCESS

During my research of the genetic algorithm, I had found many variations of the selection process. I chose to select the fittest individuals as it made the most sense to me. I figured there would be no point in carrying less fit individuals into the next generation.

I also selected the same number of individuals each time - I read that some generations increase as more iterations occur but this was not the case for me. Looking back, I think increasing the size of the population may be beneficial to allow for some more diversity.

To select the most fit individuals, I thought that using Selection Sort to pick the fittest individuals would be the most efficient. I figured that if I use Selection Sort with iterations equal to my next population size then it would be more efficient than sorting all individuals.

NUMBER OF GENERATIONS

Regarding the number of generations, I found that if the generation value was high enough relative to the population size, the population would begin to fill up with fitness values equal to each other. I believe that eventually, the crossovers begin to overlap - the parents are identical to each other which creates children identical to the parents. These children are added to the population and passed onto the next generation. Initially, I thought that my code had a bug in it however, it was the individuals inbreeding.

MY VARIABLES

There was a lot of tweaking on my part when it came to deciding on the values for the rate of crossover and mutation, population size and number of generations. In the end, I settled for the following:

Population Size = 100
Crossover Rate = 0.5
Mutation rate = 0.001
No. of Generations = 50

I found these values to be the right balance when it came to seeing the algorithm through. As reasons previously mentioned, I wanted a relatively likely crossover rate but an unlikely mutation rate. I also wanted the number of generations to be relative to the population size as if too many generations occur, the figures begin repeating.

Ant Colony Optimisation

This algorithm is based on the way that real-world ants seem to always know the shortest path to food. When ants begin to look for food, they are random and disorganised; however, over time the ants start to move in the direction where other ants have already found food because of the pheromones released. This section is entirely theory-based however it will cover the Ant Colony optimisation in terms of the Google Hash Code problem. In the context of the problem, the data centres, cache servers and endpoints will be represented on a graph.

HOW IT WORKS

Initialisation

First, the pheromone levels of the ants can be distributed evenly or it can be based on prior knowledge (most requested videos, caches closest to the endpoints, etc.). Each ant will represent a possible solution - in this case, it will return a set of videos and caches.

Solution Construction

Secondly, as the ants decide where to place each video (based on fitness), the decision can be made according to either pheromones or knowledge of least latency or video requests.

Pheromone Updates

Once all ants have created a solution (configuration of videos and caches to achieve highest fitness), the pheromone levels are updated based on the fitness of their solution. Solutions with higher fitness produce more pheromones - this makes the ants attracted to solutions similar to this.

Iteration & Termination

As the process iterates, the process repeats as long as there is always an improvement. If the program begins to return solutions of the same fitness, the program will stop. The Ant Colony Optimisation then returns the solution with the highest fitness.

THE FACTORS & OBSERVATIONS

PHEROMONE EVAPORATION

The rate of pheromone evaporation is crucial to ensure that the ants avoid converging to the local maximum. It is important to ensure a balanced pheromone evaporation rate to allow the ants to explore other potential solutions and have the opportunity to reach a global maximum.

In short, if the rate of evaporation is too low, this can lead to the outcome not being the best possible solution. This also means that a good value for this parameter should be relative to the size of the problem.

The Challenges

FITNESS FUNCTION

In the process of creating the fitness function, I had a lot of trouble wrapping my head around the idea of fitness. After discussing the issue with my friends, they explained the process step-by-step using the diagram provided in the Google Hash Code document. I found it significantly more digestible to follow each step with the diagram.

HILL CLIMBING ALGORITHM

During the creation of the Hill-Climbing Algorithm, I made the mistake of jumping straight to the code. This was a big mistake as I didn't fully understand what the algorithm was supposed to do. As a result, I had spent a lot of time debugging the simple algorithm. From this point on, I did research when it came to writing up the Genetic Algorithm and understood the ins and outs before diving head first into the code.

GENETIC ALGORITHM

Although I had done my research, the genetic algorithm was the most difficult to implement. I had spent a lot of time doing general debugging however my logic and understanding of the algorithm was correct.

I also had some trouble with the parameters of the genetic algorithm however, after discussing the topic with my coursemates, I had figured out what data structures to implement.

DEBUGGING

A lot of the time implementing the code was spent debugging. I had heard of the rubber duck method where you try to explain everything that is happening in the code however this method did not work for me at all. Instead, I ran through the code line by line comparing what was happening with what should be happening. Though repetitive, I learned that this is the best way that I can see what is happening. Through doing this method, I also got to understand the algorithm inside and out. By the end of the coding process, I had learned what worked best for me when it came to debugging.

GENERAL

Apart from the algorithm specific issues, I also had some smaller issues when it came to coding in Java. There were many times where I would go into a rabbit hole of using Pairs, HashMaps, etc. when it was not necessary. To combat this, I had to write out pseudocode before I implemented any complex data structures.

Even after I had finished the code, I began to wonder if the results I was getting were correct. To ensure that my code worked as intended, I compared the fitness values I received for the files “example.in” and “me_at_the_zoo.in” with my coursemates to ensure I was on the right track.

Performance

HILL-CLIMBING ALGORITHM

This algorithm has a straightforward and simple implementation of achieving a solution. While it may be efficient for smaller datasets, the efficiency may be halted by its greedy nature in larger datasets. With bigger sets of data, the algorithm can get stuck at the local maximum without generating neighbours far from the initial solution. This can heavily impact the efficiency of the hill-climbing algorithm.

SIMULATED ANNEALING

Like the Hill-Climbing Algorithm, The Simulated Annealing algorithm is a neighbouring algorithm. The biggest difference is the simulated annealing algorithm’s ability to backtrack and explore solutions outside of the current highest fitness in search of a more optimal solution. This allows for the performance of the algorithm to have the potential to reach the global maximum without being as greedy as Hill-Climbing.

GENETIC ALGORITHM

Though this algorithm is more complex with operations like crossover, mutation, and selection, it can explore possible solutions outside of the scope of the initial solution; however with the extra operations, this makes the algorithm less efficient. This also means it can find the global maximum rather than the local maximum - unlike the Hill-Climbing Algorithm. Genetic Algorithms can also be tweaked to have higher probabilities of crossover and mutation which can improve the efficiency depending on the context.

ANT COLONY OPTIMISATION ALGORITHM

The Ant Colony Optimisation Algorithm (ACO) shares the same issues with Genetic Algorithms such as parameter tuning and computational complexity. Similar to Genetic

Algorithms, its performance also heavily relies on the parameters. The performance of the ACO algorithm has a delicate balance between exploring (not converging to the local maximum) and exploiting (following promising paths). This means that it is able to reach a global maximum unlike the Hill-Climbing algorithm.

HILL-CLIMBING VS GENETIC

While Hill-Climbing may be quicker in finding the local maximum, there is no guarantee that the solution returned is the most efficient i.e. the global maximum. The Genetic Algorithm is more likely to find the global maximum as it explores a broader set of potential solutions. Both algorithms have their advantages and disadvantages, I believe Hill-Climbing is most useful if one is looking for a simple and sufficient solution while Genetic is best suited for bigger and more complex results.

Conclusion

When I had started this project, I thought that optimisation algorithms were niche and insignificant. However, as I progressed through the implementation, I realised that algorithms like Hill-Climbing, Genetic Algorithms, etc. are crucial to some everyday operations such as watching a YouTube video or minimising risk in the stock market. By the end of it all, I gained some valuable insight on the problem-solving technology that optimisation algorithms have to offer in the world of computer science.

References

Hill-Climbing Algorithm:

- *Introduction to Hill Climbing | Artificial Intelligence - GeeksforGeeks*
- *Hill climbing - Wikipedia*
- *Hill Climbing Search*
- *Simulated Annealing - Georgia Tech - Machine Learning*

Genetic Algorithm:

- *Genetic algorithm - Wikipedia*
- *Genetic Algorithms - GeeksforGeeks*
- *Genetic Algorithms Explained By Example*

Ant Colony Optimisation Algorithm:

- *Ant colony optimization algorithms - Wikipedia*
- *This Is How Ants Find The Shortest Way To Food (Ant Colony Optimization)*