Project: Optimization Algorithms, HashCode 2017

Aiven Arones

Trinity College Dublin

CSU22012

Anthony Ventresque

01/04/2024

Optimization Algorithms, HashCode 2017

Introduction

The project is an exploration of optimization algorithms, with an emphasis on Heuristic techniques. The primary task is the design and implementation of such algorithms in the context of a Google HashCode 2017 problem. The problem requires the development of optimized solutions for a "video-serving infrastructure"[1] through strategic placement of videos/files within a series of cache servers to "minimize the average waiting time for all requests." [1]

System Requirements

Regarding the Hashcode 2017 problem [2], the overall purpose of the system is to create an optimized solution(s) that minimizes the average wait time for all requests when "given a description of cache servers, network endpoints and videos, along with predicted requests for individual videos" [2]. In this case, a solution is a specific assignment of files in cache servers. Below are system requirements that the designed system must be able to do in addition to completing the primary purpose.

1. The system must be able to read input files containing different instances of the problem
2. The system must be able to represent and score solutions through a fitness function
3. The system must use hill-climbing and genetic algorithms to complete the primary purpose
4. The system must ensure that each solution adheres to the below constraint

Constraint

Each cache has a maximum capacity, the sum of all file sizes contained within said cache must not exceed its maximum capacity, this applies to all caches.

Architecture/Design

My solution makes use of heuristic functions and techniques when generating, evaluating and mutating solutions. I've created a file, endpoint and solution matrix class containing necessary information. Each class has a constructor which takes a HashMap <String, Object> as a parameter, the system will create a data field and pass it into each class where necessary. My design implements a multi class structure to take advantage of Java's Object-Oriented Principles.

Requirement 1

The system can read input files and create a data field of type HashMap <String, Object>, this will be passed into all classes. This allows each component of the system to interact with each other using the same information. However, input files must be specifically formatted using the guidelines stipulated in the Hashcode 2017 problem statement. [2]

Requirement 2

Solutions are represented using a 2D-Array, in the form of solution [cache][file] and in the submission file format found in the Hashcode 2017 problem statement. Both representations are used for calculations, alterations and presenting the solutions in a visually appealing way. A binary representation is used in the genetic algorithm for crossover and mutation purposes. Within the genetic algorithm, 2D-Array representations are encoded as a binary string for string manipulations.

The scoring class (calculateScore) is an essential component of all subsequent Heuristic algorithms. The fitness function uses this class, invalid solutions are given a score of – 1. The pseudocode used was given by the college, a more in-depth explanation can be found within the Hashcode 2017 problem statement.

Requirement 3

Hill-climbing algorithm

To complete the primary purpose of the system, a hill-climbing algorithm is used to generate a population of solutions and define a fittest solution. This fittest solution minimizes the average wait time for all requests thus satisfying the primary purpose of the system. Hill-climbing defines an initial state, makes a small move and updates the fittest solution if there is an improvement. I have designed two variations (first choice and random restart) based on the instructions below found in the project brief.

• generate all the neighboring solutions, by making simple "moves" [1]

• keep the best (fittest) solution [1]

My first design implements first choice hill-climbing, in this case the simple move places a random file in a random cache and accepts it if the resulting solution is an improvement. To go "beyond" [1] I tried to explore different algorithms, my second design makes use of random restart hill-climbing (shotgun hill-climbing). In this design the simple move is the creation of a randomized solution by allocating a random number of files into a random cache. In both designs the initial solution is a randomly generated solution to create more diversity.

Genetic algorithm

My design uses a binary encoding when applying genetic operators to solutions. The initial population is created using Random Initialization and Random Restart hill-climbing. This makes use of

Heuristic principles and is a greedy/random approach. This design is a combination of both options given in section 5 of the project brief. This greedy/random approach creates a diverse initial population and lessens the chance of pre-mature convergence.

The design uses two genetic operators, one-point crossover and bit-flip mutation. The probability of mutation in my design is 0.01. One-point crossover randomly partitions two parent solutions, two different children are created by taking one partition from both parents. Bit-flip mutation takes one or several bits from a binary encoded solution and flips them. In my design, there is a 25% chance that more than one bit will be flipped, otherwise only a single bit will be flipped. The specific bit to be flipped is chosen at random. This approach adds further uncertainty to maintain diversity.

The selection of parents for the crossover operation is chosen using rank selection. Selection probabilities are assigned using "p(i) = (2 - s) / N + 2 (i - 1) (s - 1) / (N * (N - 1))"[7], where s is a selection parameter that allows for solutions to tend towards fitter scores N is the number of solutions. The formula can be attributed to Ali Karazmoodeh[7]. After five generations, a diverse population of fifty solutions will be produced and will provide a satisfactory solution to the problem at hand.

Design Philosophy

The use of hill-climbing and genetic algorithms allows for the creation of solutions and a fittest solution to complete the primary purpose. But my guiding philosophy was the use of uncertainty to create diverse solutions to maintain diversity and lessen the chance of getting stuck in local optima. Each algorithm can complete the primary purpose using a greedy/random approach.

Requirement 4

A safetyConstraintTest class uses a 2D-Array representation of a solution and iterates through each cache calculating the sum of file sizes. It returns a Boolean on whether the solution satisfies the constraint. This constraint test is an essential component and is found at every stage of the system.

Design Considerations

Choosing Initial Solution and Simple Moves

Greedy algorithms, especially hill-climbing ones are "sensitive to the choice of initial solution". [3] A poorly chosen initial solution may result in a poor fittest solution. This was something I observed when I used a blank solution. The resulting fittest solution often had an unsatisfactory score, I also observed that it took longer for the hill-climbing algorithm to reach a good level of local optima. I decided to use a random approach with the optional condition of creating a solution with a greater score than 0. This made reaching a good level of local optima quicker and created more diverse solutions. This initial solution uses a series of randomized file placements to create a unique solution. This method of choosing an initial solution is used in both of my hill-climbing designs.

Within the project brief [1], a simple move was to be used to generate Neighbours. I pondered on what would be considered a simple move. I concluded that since I was using a greedy approach and explored all possibilities, a random file placement was satisfactory. It also had the added benefit of leading to more diverse solutions. This line of thinking eventually led to the development of my first choice, random restart algorithm and subsequent genetic algorithm designs. The use of uncertainty was the guiding principle of my system design going forward.

Plateau and Premature convergence

A plateau in hill-climbing "makes it impossible to choose a direction as well as reach the goal state".[4] In my initial designs, prior to my decision to use a greedy/random approach as my guiding principle. I found that my solutions plateau often and too quickly. Despite using a random simple move for my first-choice algorithm the problem remained at times. I decided to redefine the simple move as a collection of random simple moves, inadvertently creating a random state of the board. I later realized that this design resembled Random Restart hill-climbing.[4] After this I began researching this algorithm variant and decided to incorporate it into my system as a separate algorithm in my attempt to go beyond.

A similar issue was found when designing the genetic algorithm, I wanted to avoid premature convergence. It was important to maintain the diversity of the population after every generation, to do so I decided to use bit-flip mutation. I added a further level of uncertainty by varying the number of bit flips and randomizing which bit of the solution was to be altered. If the mutation led to a valid solution, it would be added to the population to maintain diversity.

Genetic Algorithm: Representation, Operations, Selection

While it was possible to represent genotypes, apply genetic operations and select parent using a 2D-Array. It was more common to use Binary Representation for Genotype Representation. While my initial ideas involved the use of 2D-Array representation I decided to use binary encoding. This made genetic operations easy to implement as it merely involved simple string manipulation. I decided to use the 2D-Array representation only for calculating fitness and representing the final population.

I chose to use one-point crossover and bit-flip mutation as my genetic operators, these were simple to implement but added a greater level of uncertainty. There existed more complex operations such as scramble mutation and multi point crossover, but I deemed it unnecessary as the system already had a high degree of uncertainty complexity. Though my initial design wanted to implement both scramble and bit-flip mutation to experiment with the concept of multi-mutations.

The most difficult part of the genetic algorithm for me was choosing the selection method for parents. I originally tried to implement roulette wheel selection but found I had trouble in testing as it led to various errors, I later realized it was because I was adding children to the population that were not valid solutions. My current method of selection uses Rank Selection, by sorting the population by fitness score I could use its position in the list as its rank. One point of contention was finding a formula for selection probabilities. I later found an article on Indeed with a satisfactory formula. [7]

Challenges

How often to crossover and mutate?

The rates at which we apply these genetic operations varied my solutions greatly, I observed that higher probability of mutation led to a more diverse population. But in the project brief [1] it was recommended to keep the probability of mutation low, so I experimented with different probabilities and settled on a probability of 1%. Since the probability of mutation was 1%, the probability of a crossover operation was 99%.

Parent Selection and creating children

While my genetic algorithm design was easy to implement, parent selection was the hardest part as there were many options to choose from tournament selection to roulette wheel selection. I first used roulette wheel selection, but it was poorly implemented and led to unfit children. I implemented rank selection, sorting the solutions was particularly difficult as the list only stored the encoding and not the corresponding score. I thought it was possible to sort the population by score if I created a HashMap with the key being the solution and value being its fitness. I found a way to sort Hash Maps by key from geek for geeks.[6] I also had trouble with creating children as I sometimes got null children and infinite loops, I was able to resolve it by adding a constraint condition within the genetic algorithm.

Going Beyond

While I aimed to go beyond to get more marks, I also wanted to expand my horizons farther than the project brief's scope. To this end I tried to learn new things, implement new strategies and create my own solutions. This led me to research in-depth the theory of heuristics, develop my own designs (Random Restart) and decide to use the idea of greedy/random in my guiding philosophy. For this report I also learned about AMA formatting and used it to showcase my ideas in a professional format.

Conclusion

In conclusion, my system uses a heuristical and greedy/random approach to solve the Hashcode 2017 problem. One can clearly see how my design evolved throughout the development cycle. From first choice hill-climbing to random restart hill-climbing then to the genetic algorithm. Each component communicates with each other to solve the given problem.

References

[1]Trinity College Dublin. (2024, February 21). "Project: Optimization Algorithms" instructions.

https://tcd.blackboard.com/ultra/courses/_82334_1/cl/outline

[2]Google. (2017). "Problem statement for Online Qualification Round, Hash Code 2017".

[3]GeekForGeeks. (2023, April 20). "Introduction-hill-climbing-artificial-intelligence".

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

[4]Ashul Jain. (2020, April 22). "Hill Climbing Algorithm"

https://www.professional-ai.com/hill-climbing-algorithm.html

[5]Tutorialpoint. "Genetic Algorithms Tutorial"

https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_population.htm

[6]GeekForGeeks. (2022, April 6). "Sort HashMap by Value"

https://www.geeksforgeeks.org/sorting-a-hashmap-according-to-values/

[7] Ali Karazmoodeh. (2023, December 9). "selections-genetic-algorithms"

https://www.linkedin.com/pulse/selections-genetic-algorithms-ali-karazmoodeh-g9yyf/