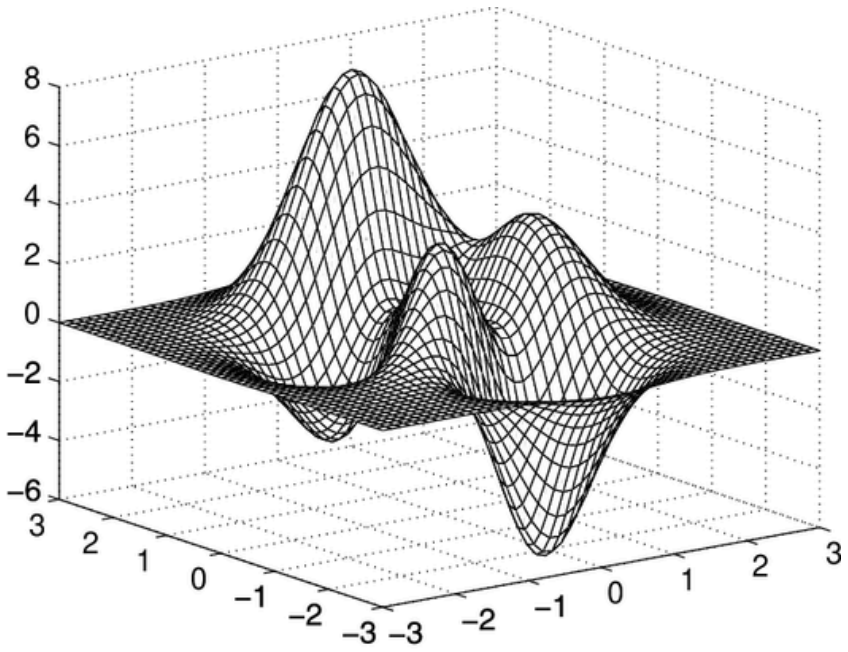
Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

SCHOOL OF COMPUTER SCIENCE AND STATISTICS



ALGORITHMS AND DATA STRUCTURES II
OPTIMISATION ALGORITHMS

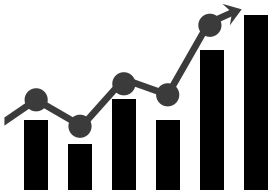
HILARY TERM 2024

SEBASTIAN BURY

22335740

CONTENTS

INTRODUCTION



In the realm of computational science and engineering, optimisation algorithms play a pivotal role in solving problems that necessitate finding the optimal solution by exploring a set or space of feasible solutions.

The essence of optimisation lies in making the most effective use of available resources while adhering to specific constraints within a solution space. This can involve minimising or maximising the objective function or a fitness function in our case, which quantifies the quality of a solution.

Sophisticated methods are often required to navigate expansive solution spaces, address non-linearities and distinguish the global optimum from local optima. Optimisation algorithms are applied across a broad spectrum of domains, including but not limited to logistics, scheduling, network design, artificial intelligence and operations research.

This project aims to investigate how various optimisation algorithms tackle a defined problem, through the implementation of hill climbing, genetic and particle swarm optimisation algorithms.

SYSTEM REQUIREMENTS

The system is designed to strategically optimise the allocation of video content across a network of cache servers to minimise the average waiting time for video requests. This is achieved by analysing network topology, video sizes, endpoint demands, and cache server capacities to make informed decisions on video placement.

The primary requirement of the system is to decrease overall latency experienced by users when accessing videos. The system achieves this by placing videos in cache servers in a manner that reduces time taken to serve a video request, thereby enhancing user experience and network efficiency.

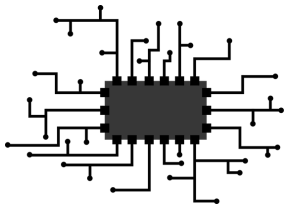
However, the system must also ensure that all outputs are feasible within the given network constraints, including cache server capacities and video sizes, more precisely the following formula must be satisfied:

$$\forall\ cache\ c_i, \forall\ file\ f_j f_j \in c_i, \sum f_j < capacity(c_i)$$

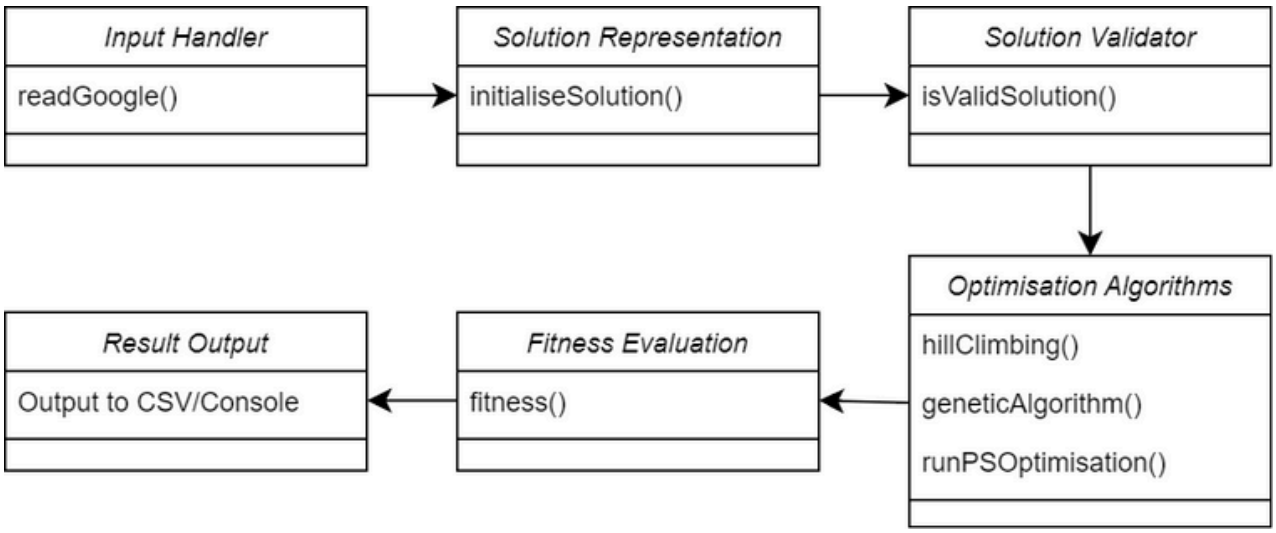
In order to meet our primary objective of decreasing overall latency, our system must satisfy the following requirements:

- **Data Interpretation:** Reads input files and processes the data that describes each network's configuration.
- **Solution Representation:** Initialises solutions, representing potential distributions of video files across cache servers.
- **Validation:** Verifies that each solution is valid by ensuring the above formula is satisfied.
- **Fitness Evaluation:** Evaluates the fitness or effectiveness of a solution based on its potential to reduce overall network latency.
- **Optimisation Algorithms:** Implements various algorithms which explore a solution space in search for optimal fitness scores.

SYSTEM ARCHITECTURE



An Overview



- **Input Handler:** The readGoogle() function employs buffered reading for efficiency, parsing natural numbers representing videos, endpoints, cache servers and requests into structured Java primitives for subsequent processing.
- **Solution Representation:** The initialiseSolution() function creates a 2D array mirroring cache servers (rows) and videos (columns), allowing us to represent the solution.
- **Solution Validator:** To align with capacity constraints, the isValidSolution() function takes in a solution and then checks that the solution respects the size limitations of cache servers.
- **Optimisation Algorithms:**
  - **Hill Climbing:** Performs an iterative hill climbing algorithm that alters the current solution incrementally by exploring neighbours and selecting the best ones in order to search for a global maximum in the landscape.
  - **Genetic:** Iteratively applies crossover, mutation and culling to explore the search space and converge towards optimal solutions.
  - **Particle Swarm Optimiser:** Inspired by the social behaviour of birds, PSO navigates the solution space by moving particles (solutions) towards the best personal and global positions, influenced by velocity vectors.
- **Fitness Evaluation:** The fitness() function calculates the highest fitness score based on the potential time saved in serving video requests from cache servers versus the data center.
- **Result Output:** Most of the aforementioned functions output the fitness score and display the solutions across many iterations, either to the console or to a CSV file.

FITNESS FUNCTION

Core Principle

The fitness function evaluates a solution based on two key metrics: latency reduction and demand fulfillment. It begins by assessing whether the distribution of videos in a proposed solution respects the capacity constraints of each cache server.
Next, it computes the potential time saved for each video request by comparing the latency from the data center to the latency from the cache server. The total gain represents the sum of all time saved across all requests, a figure that is then normalised by the number of requests to yield an average gain.

$$FitnessScore = (TotalGain \div TotalRequests) \times 1000$$

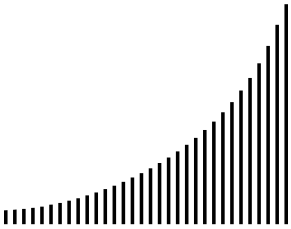
The function operates on a simple premise: videos placed closer to the user (in a cache server) should ideally reduce the time it takes to fulfill a request compared to streaming from a distant data center. Therefore, the lower the latency and the higher the number of requests served from cache servers, the greater the fitness score, indicating a more desirable solution.

Challenges Faced

- **Capacity Validation:** To prevent cache overflow, which could lead to an infeasible solution, an early exit condition was instituted. If the storage used by videos in any cache exceeded its capacity, the solution was deemed invalid and assigned a prohibitive fitness score (-1), effectively removing it from consideration.

- **Diverse Latencies:** The differences in latency between the data center and multiple cache servers required a comprehensive approach. This issue was addressed by calculating the minimum latency for each request, ensuring that each video's contribution to fitness was based on the most optimal streaming scenario.

HILL CLIMBING ALGORITHM



Description

The hill climbing algorithm applied within our system is a heuristic method used to search for an optimised solution for video placement across cache servers. Conceptually, it is similar to a climber ascending a hill, where the peak represents the optimal solution.

Mechanism of Hill Climbing

The algorithm begins with an initial solution, think of this as the climber's starting point at the base of the hill. It then examines neighbouring solutions, which are slight variations of the current solution. In short neighbours are states that can be reached with the smallest change possible. If we have a grid full of 0's, changing any spot to 1 makes it a neighbour. In our context, a "*neighbouring solution*" would be moving a single video to a different cache server, or adding a video to an initially empty cache server.

The fitness function is then used to evaluate each neighbouring solution. If a neighbour exhibits a higher score (indicating a reduction in average latency compared to current solution), then the neighbour becomes the new current solution. This step is analogous to the climber moving uphill towards a greater altitude.

This iterative process continues until no neighbouring solution can be found that improves the fitness score, in other words, the climber cannot ascend further and has reached a peak or local maximum.

Final Results

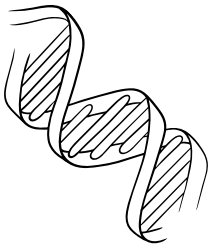
Input File	Fitness Score
example.in	562,500
me_at_the_zoo.in	516,740
trending_today.in	454,000

Challenges Faced

- **Plateaus:** I encountered a plateau, where a series of moves neither improves nor worsens the fitness score. My solution involved implementing a condition where, if no improvement was detected with a certain number of iterations, the algorithm will make a larger jump to a different part of the solution space to escape the plateau.

- **Efficiency:** Hill climbing can be computationally expensive, especially with many neighbours present. Although there were no issues with running the smaller input files, while running '*trending_today.in*', efficiency became a problem. Hence, the final fitness score for it was lower compared to the rest.

GENETIC ALGORITHM

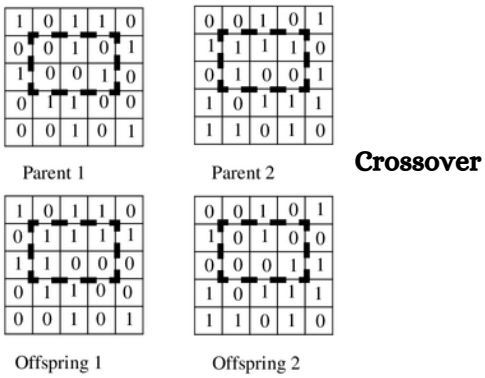


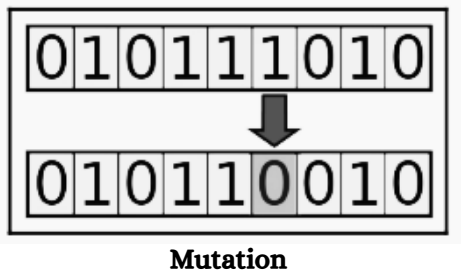
Description

A genetic algorithm is a metaheuristic inspired by the process of natural selection. They are commonly used to generate high-quality solutions to optimisation problems by relying on biologically inspired operators such as mutation, crossover and culling.

Mechanism & Evolution Strategy

- **Initial Population Generation:** A stochastic approach was implemented to combat the potential for homogeneity. Random values of 0 or 1 were assigned to the elements of the solution array, generating a variety of possible solutions. A balance was struck by introducing heuristics that limited randomness like allowing the addition of a 1, only when it didn't render the solution infeasible.
- **Crossover:** For each member in the population, there was a probability that crossover would occur. In this case, another member was selected and two children, or offspring were created. This operation swapped the latter halves of two solutions, resulting in a significant shake-up of existing genetic material. Although this led to many infeasible solutions, it was invaluable for escaping local optima.
- **Mutation:** Each individual in the population carried a chance that they might undergo mutation, in which case, the element would become inverted, what was once '0' could become a '1', and vice versa. While altering the element, this mutation did not change the individual's identity within the population.
- **Culling:** For each individual and each child within the population, we had to discard the less fit solutions, and keep the 50 fittest individuals. This selective removal was integral to forge an optimal population for the next generation of the algorithm.



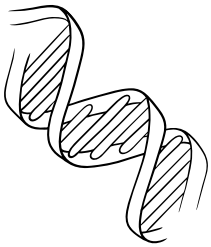


Challenges Faced

Feasibility of Offspring: The crossover operation often generated infeasible solutions. To address this, I incorporated a validation check post-crossover, discarding any solution that exceeded cache capacities.

Parameter Adjustment: Determining ideal crossover and mutation probabilities in order to yield good fitness scores required extensive experimentation. I adjusted the parameters both statically and dynamically in search for optimal performance. The efficacy of these adjustments will be assessed in the following section.

EVALUATION OF GA PARAMETERS



Crossover Probability

- The graph illustrates a clear trend where fitness scores generally increase with the crossover probability, indicating that a higher likelihood of crossover tends to produce more fit individuals. However, the rate of increase varies across different mutation rates.
- For lower mutation rates (0.001 - 0.05), we see a steep increase in fitness scores as the crossover grows from 0.1 to 0.3, suggesting that introducing more diversity contributes significantly to exploring the solution space effectively. Beyond 0.3, the increase becomes less steep but continues until 0.7, when it reaches a plateau. This indicates that excessive crossover is disruptive, potentially by creating too many homogenous offspring.
- As for higher mutation rates (0.05+), while the initial increase in fitness is steep, it seems to plateau between crossover probabilities of 0.5 to 0.7.

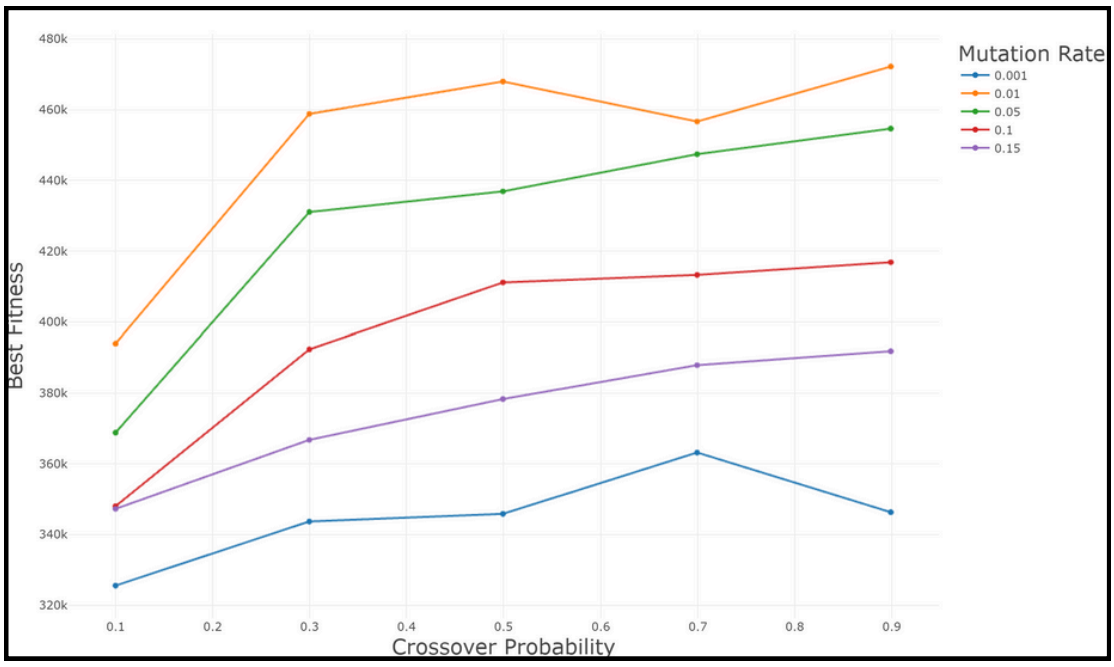
Mutation Rate

- The mutation rate has a complex relationship with the fitness score. Low mutation rates paired with moderate crossover probabilities seem to yield the highest fitness scores, while too high a mutation rate (0.15) appears to reduce the fitness regardless of crossover.
- This suggests that while mutation is necessary to prevent genetic uniformity and premature convergence, 'over-mutating' introduces too much randomness, moving away from optimal solutions.

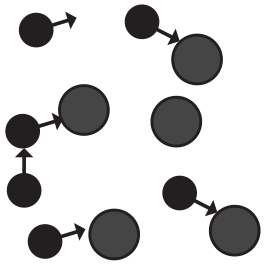
Optimal Parameters

- Based on my experimentation, the optimal combination lies around a 0.5-0.6 crossover probability with a 0.01 mutation rate.
- With the above parameters, using a population size of 50, and allowing the genetic algorithm to evolve over 500 generations with a dynamic crossover point guided by a Gaussian distribution, it reached the following scores: **562,500** for '*example.in*', **478,276** for '*me_at_the_zoo.in*' and **467,432** for '*trending_today.in*'.

Example of genetic algorithm running on file "me_at_the_zoo.in"



PARTICLE SWARM OPTIMISATION



Description

Particle swarm optimisation (PSO) is a powerful metaheuristic optimisation algorithm inspired by swarm behaviour observed in bird schooling. It iteratively tries to improve a solution with regard to a given measure of quality. It solves the problem by having an initial population of solutions, named particles, and moving these particles around in the search space according to simple mathematical formula over the particle's position and velocity.

Framework

- **Swarm Initialisation:** Each particle represents a potential solution and is characterised by its position (solution configuration) and velocity (change in position). A random, valid configuration of video allocation across cache servers is generated for each particle, serving as the starting point for the optimisation journey.
- **Particle Update:** Each particle's velocity and position are updated based on its own experience and the swarm's collective experience. The velocity adjustment is influenced by the particle's best-known position and the global best-known position, striking a balance between individual exploration and group convergence. The adjustments made in this function, must satisfy the following formula:

$$P_i^{t+1} = P_i^t + V_i^{t+1}$$

$$V_i^{t+1} = \underbrace{wV_i^t}_{\text{Inertia}} + \underbrace{c_1 r_1 (P_{best(i)}^t - P_i^t)}_{\text{Cognitive (Personal)}} + \underbrace{c_2 r_2 (P_{bestglobal}^t - P_i^t)}_{\text{Social (Global)}}$$

- **PSO Engine:** During the execution of PSO, the swarm collectively explores the solution space, with each particle searching for optimal video placements. The update rules guide the particles towards the fittest solution found, with the intent of discovering even more optimal solutions as the process iterates.

Final Results

Input File	Fitness Score
example.in	562,500
me_at_the_zoo.in	456,215

Challenges Faced

- **Diversity Preservation:** Ensuring that the swarm did not prematurely converge on suboptimal solutions was tough. However, by tuning the inertia weight (w) and the cognitive and social coefficients (c1, c2), I maintained an appropriate exploration to exploitation trade-off that preserved diversity within the swarm.
- **Efficiency:** The iterative nature of PSO and the maintenance of multiple best positions (personal and global) posed a computational challenge. As a result, I was only able to produce fitness scores for the two input files above, within a reasonable time frame.

CONCLUSION

The application of optimisation algorithms within computational science has proven to be invaluable in the quest for optimal solutions across vast and complex spaces. This project has demonstrated the efficacy of hill climbing, genetic algorithms and particle swarm optimisation in maximising resource efficiency while navigating the constraints and intricacies of solution spaces.

With the use of the aforementioned algorithms, the strategic optimisation of video content allocation across a network of cache servers has been achieved, significantly reducing average latency times for video requests. Moreover, the system ensured that all solutions adhered to feasibility within the constraints of the network capacities and video sizes.

Furthermore, through processes such as data interpretation, solution representation, validation, fitness evaluation and implementation of optimisation algorithms, the system has successfully demonstrated its capability in finding optimal solutions within the given solution space, as envisioned within the requirements.

As we bring this project to a close, we stand at the juncture where computational theory and practical application converge, illustrating the profound impact that well-executed optimisation algorithms can have on the functionality of complex networks.

REFERENCES

- Hassanat, A. et al, “Choosing Mutation and Crossover Ratios for Genetic Algorithms - A Review with a New Dynamic Approach”, 2019.

- Vrajitoru, D., “Large Population or Many Generations for Genetic Algorithms? - Implications in Information Retrieval”, 2000.

- L. Hernando, A. Mendiburu and J. A. Lozano, "Hill-Climbing Algorithm: Let's Go for a Walk Before Finding the Optimum," 2018 IEEE Congress on Evolutionary Computation (CEC), Rio de Janeiro, Brazil, 2018, pp. 1-7.

- Kour, Haneet Sharma, Parul Abrol, Pawanesh, “Analysis of Fitness Function in Genetic Algorithms”, 2015, 87-90.