University of Cape Town

STA5071Z

Optimization

Optimization of Energy Efficiency and Execution Time in Cloud Computing

Author:
Roger Bukuru

Student Number:
BKRROG001

September 30, 2024

Contents

1 Introduction

As cloud computing grows, so does energy consumption in data centers, leading to increased operational costs and CO2 emissions. In this project we explore various optimization techniques that can enhance energy efficiency and reduce execution time in cloud environments. More particularly we will be exploring the following optimization techniques:

- Multi-Integer Linear Programming
- Simulated Annealing
- Genetic Algorithms
- Multi-Objective Goal Programming

Our goal will be to implement each of these techniques, in order to allocate a fixed number of computing tasks on a set of computing resources (servers) within the cloud environment such that we minimize execution time whilst maximizing energy efficiency. Each technique will be implemented, evaluated and thereafter the results will be compared in order to report which technique best achieves our goal. The repository of this study is available at https://github.com/rogerbukuru/OR-Cloud-Resource-Allocation-Cost-Optimization.

2 Data

The data that we will be using comprises of performance metrics in a cloud computing environment. It includes the following features:

- CPU usage
- Memory usage
- Network traffic
- Power consumption
- Number of executed instructions
- Execution time
- Task type

Our final dataset after performing standard data cleaning techniques such as removal of nulls and standardization resulted in a data set of 5236 observations. Each observation represents a computing task being executed within the cloud environment and the features represent its consumption of the available computing resources. In order to make the optimization of task allocation practical for each technique we introduced a parameter called total number of servers, which are the computational resources on which tasks are being executed within the cloud environment. The computational units highlighted above are tied to these servers i.e 10% of CPU usage means that we are utilizing 10% of the available CPU on a given server. When we have more then one server we will refer to the collection as a cluster of servers. Before proceeding in table 1 we provide brief definitions of some of the technical terminology used within the project

Term	Definition
Server	A computer that consists of resources such as CPU, memory, and is intended to run tasks within a cloud environment.
Load Balance	The process of equitably spreading task allocation across servers to ensure that no single server becomes overwhelmed.

Table 1: Definitions of Technical Terminology

3 Mixed Integer Linear Programming (MILP)

We start by undertaking the minimization of execution time and maximization of energy efficiency in isolation through the use of Mixed Integer Linear Programming (MILP). Mixed Integer Linear Programming is an improved optimization technique from standard linear programming where we can include variables that are both continuous and discrete. In our setup, which we describe below, we make use of both continuous and binary (discrete) variables.

3.1 Problem Formulation

We proceed to present our problem formulation for the MILP algorithm, the formulation highlights the decision variables, parameters, objective functions and constraints necessary to model the task assignment effectively.

Decision Variables

The decision variables presented here will be consistent across all the other algorithms, we present them below and subsequently provide an interpretation of their solution representation, this too will remain consistent across algorithms.

$$x_{i,j} = \begin{cases} 1 & \text{if task } i \text{ is assigned to server } j, \\ 0 & \text{otherwise,} \end{cases} \quad \forall i = 1, \ldots, N; \quad \forall j = 1, \ldots, M$$

Solution Representation

Each candidate solution is represented as an integer vector of length equal to the number of tasks. Each element in the vector denotes the server to which a specific task is assigned. For instance, a solution vector $[3, 1, 5, \ldots]$ implies that Task 1 is assigned to Server 3, Task 2 to Server 1, Task 3 to Server 5, and so forth. This representation ensures that each task is assigned to exactly one server.

Parameters

The unique parameters of the MILP algorithm are presented in table 2 below:

Symbol	Description
N	Total number of tasks.
M	Total number of servers.
t_i	Execution time of task i.
e_i	Energy efficiency of task i.
c_i	CPU usage of task i.
m_i	Memory usage of task i.
n_i	Network traffic of task i.
p_i	Power consumption of task i.
C_j	CPU capacity of server j.
M_j	Memory capacity of server j.
N_j	Network traffic capacity of server j.
P_j	Power consumption capacity of server j.
α	Weight for execution time in the objective function.
β	Weight for energy efficiency in the objective function.

Table 2: List of Parameters

Objective Functions

Our objectives are handled independently and are represented as follow:

$$\text{Minimize} \quad \sum_{i=1}^{N}\sum_{j=1}^{M} t_i x_{i,j}$$

$$\text{Maximize} \quad \sum_{i=1}^{N}\sum_{j=1}^{M} e_i x_{i,j}$$

Here, the objectives are to independently minimize the total execution time and maximize the total energy efficiency. We present the constraints for both cases, note these constraints will remain consistent across all the other optimization techniques.

Constraints

$$\text{1. Task Assignment:} \quad \sum_{j=1}^{M} x_{i,j} = 1, \quad \forall i = 1, \ldots, N$$

$$\text{2. CPU Capacity:} \quad \sum_{i=1}^{N} c_i x_{i,j} \leq C_j, \quad \forall j = 1, \ldots, M$$

$$\text{3. Memory Capacity:} \quad \sum_{i=1}^{N} m_i x_{i,j} \leq M_j, \quad \forall j = 1, \ldots, M$$

$$\text{4. Network Traffic Capacity:} \quad \sum_{i=1}^{N} n_i x_{i,j} \leq N_j, \quad \forall j = 1, \ldots, M$$

$$\text{5. Power Consumption Capacity:} \quad \sum_{i=1}^{N} p_i x_{i,j} \leq P_j, \quad \forall j = 1, \ldots, M$$

$$\text{6. Binary Variables:} \quad x_{i,j} \in \{0, 1\}, \quad \forall i = 1, \ldots, N; \quad \forall j = 1, \ldots, M$$

Explanation of Constraints

- **Task Assignment Constraint**: Ensures that each task is assigned to exactly one server.
- **Resource Capacity Constraints**: Ensure that the total resource usage on each server does not exceed its capacity for CPU, Memory, Network Traffic, and Power Consumption.
- **Binary Variable Constraint**: Defines the decision variables as binary, indicating whether a task is assigned to a server or not.

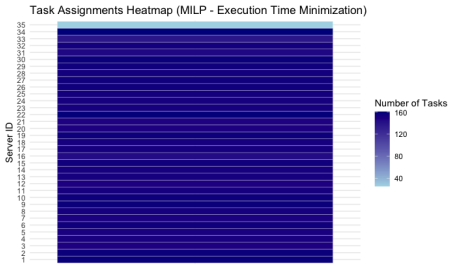
3.2 Results and Analysis

In Table 3, we show the parameters and constraints that we used to execute the MILP algorithm. The server capacities represent a bundle of units out of a total of 100. We limit it to 80 because, in computing resource allocation, we don't aim to allocate the maximum available resources as at that point it could cause a system to collapse, resulting in catastrophic failure. This means that if the allocated 80 units are exceeded, a new server would need to be introduced or a task has be assigned onto a server with sufficient resources.

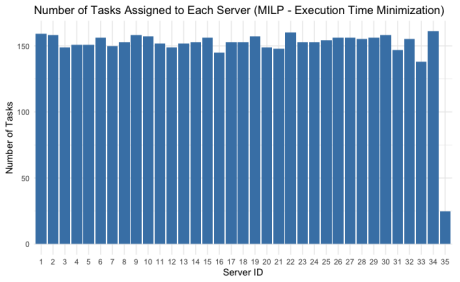
Parameter	Value
Number of Tasks	5236
Number of Servers	35
Server Capacities	
CPU Capacity per Server	80 units
Memory Capacity per Server	80 units
Network Traffic Capacity per Server	80 units
Power Consumption Capacity per Server	80 units
Total Resource Capacities	
Total CPU Capacity	$80 \times 35 = 2800$
Total Memory Capacity	$80 \times 35 = 2800$
Total Network Traffic Capacity	$80 \times 35 = 2800$
Total Power Consumption Capacity	$80 \times 35 = 2800$

Table 3: Summary of MILP Parameters and Constraints

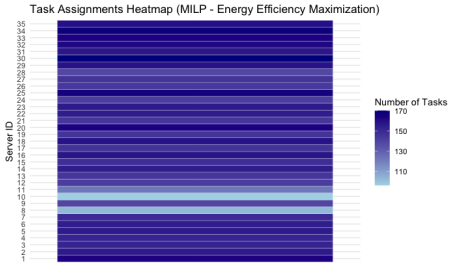
The algorithm achieved an objective value of 2597.472 when minimizing execution time and a value of 2613.82 when maximizing energy efficiency. We found that given the server constraints, in order to successfully execute all the tasks and independently optimize minimizing the execution time and maximizing the energy efficiency, a total of 35 servers would be required, we will maintain this total as our baseline for all the other techniques. Furthermore, we examined the effectiveness of the MILP algorithm to efficiently load balance task allocation across the cluster of servers to avoid situations where a server is over-utilized(overwhelmed) or under-utilized. Figure 1 below shows the result of task assignment across the servers for each of the individual objectives.



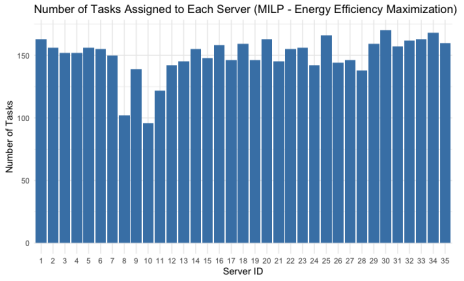
(a) Task Assignment Heatmap (Execution Time Minimization)



(b) Task Assignment Bar Plot (Execution Time Minimization)



(c) Task Assignment Heatmap (Energy Efficiency Maximization)



(d) Task Assignment Bar Plot (Energy Efficiency Maximization)

Figure 1: MILP Task Assignment

We note that in both objectives, the solver managed to effectively allocate the tasks across the servers. However we observe that the task load-balancing was not optimally managed as we observe server overloading especially when solely minimizing for execution time as shown on the heatmap of figure 1a. When maximizing for energy efficiency we observe from the heatmap shown in figure 1c a couple of lighter regions indicating better task load balancing, as we note that the solver attempted to ensure that each server is proportionally utilized with no server having disproportionately lower amount of tasks as seen when solely minimizing for execution time. This is important as it ensures that energy efficiency is optimized as it means that there would be no idling server and with fewer servers overwhelmed it would help minimize execution time.

We now proceed to more advance methods that will present us with the opportunity to simultaneously optimize both objectives, start with the Simulated Annealing (SA) method followed by Genetic Algorithms and finally wrapping it up with a Multi-Objective Goal Programming approach.

4 Simulated Annealing (SA)

Simulated Annealing is a probabilistic optimization technique inspired by the annealing process in metallurgy. It explores the solution space by allowing occasional acceptance of less optimal solutions to escape local minima, thereby simultaneously optimizing multiple objectives. In our study, we applied Simulated Annealing to efficiently assign tasks to servers with the goals of minimizing execution time and maximizing energy efficiency while adhering to resource constraints. In applying the algorithm we explored the solution space using both the geometric and logarithmic

cooling schedules.

4.1 Problem Formulation

We maintain the same structure as the MILP algorithm and present our Simulated Annealing problem formulation below. Take note that the constraints do not change and hence remain consistent, for brevity we do not re-present them here as well as in subsequent algorithms, for reference they can be found here.

Decision Variables

$$x_{i,j} = \begin{cases} 1 & \text{if task } i \text{ is assigned to server } j, \\ 0 & \text{otherwise,} \end{cases} \quad \forall i = 1, \ldots, N; \quad \forall j = 1, \ldots, M$$

Parameters

Parameter	Description
T_0	Initial temperature for SA.
α_{cool}	Cooling rate for temperature reduction.
W	Penalty weight for resource overuse.

Table 4: Parameters Used in Simulated Annealing

Objective Function

Unlike the MILP objective function we will simultaneously be optimizing for both execution time and energy efficiency, we present our formulation below, with an added penalty term.

$$Z = \alpha \times \text{Total Execution Time} - \beta \times \text{Total Energy Efficiency} + W \times \text{Penalty}$$

where:

- **Total Execution Time**: Sum of execution times for all assigned tasks.
- **Total Energy Efficiency**: Sum of energy efficiencies for all assigned tasks.
- **Penalty**: Penalty incurred if any server exceeds its resource capacities. This helps in guiding the search towards feasible solutions.

For the weight vectors α and β we utilized an equal weighting of 0.5 between the two objectives.

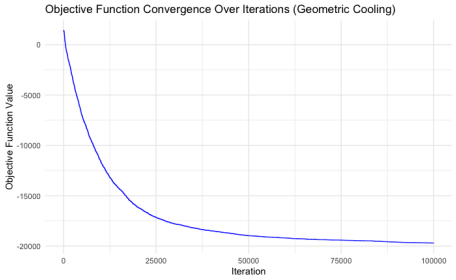
4.2 Results and Analysis

We used the exact same computing unit upper bounds to execute the Simulated Annealing algorithm. The main differences included the following:

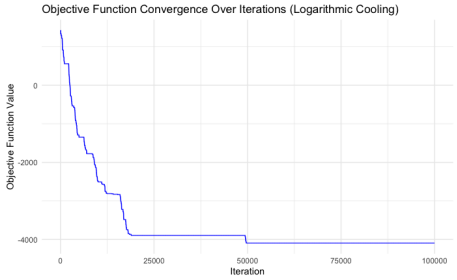
- We jointly aimed to minimize execution time and maximize energy efficiency within the solution space.
- We ran the algorithm for 100,000 iterations.
- The initial temperature was set to 50.

- Our geometric cooling rate was set to 0.995.
- Our penalty weight W was set to 10.
- Our logarithmic cooling rate of was set to 0.1.

In Figure 2, we show the results of the objective function convergence under both cooling schedules. We observe that under the geometric cooling schedule, figure 2a, the algorithm struggled to reach convergence as the loss continues to decrease, however it's lower objective value indicates a potentially better solution than that provided by the MILP algorithm. On the other hand we note from figure 2b when utilizing a logarithmic cooling schedule that the algorithm successfully reached convergence as we observe a steady state in the objective loss, this is further evident in the task distribution results which we explore next.



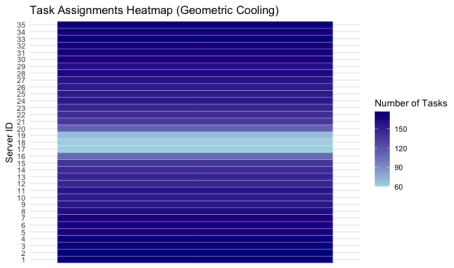
(a) Geometric Cooling Objective Function Convergence



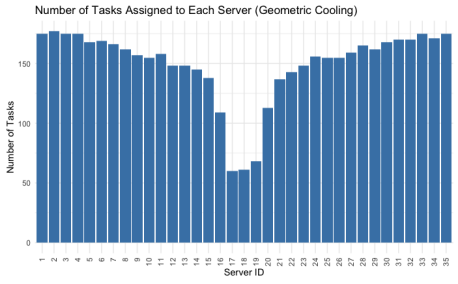
(b) Logarithmic Cooling Objective Function Convergence

Figure 2: Objective Function Convergence

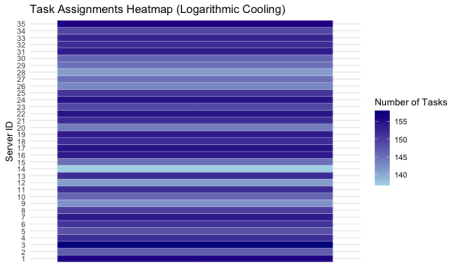
Evaluating the geometric cooling schedule task assignment results shown in figures 3a and 3b below, we note that in contrast to the MILP algorithm there was a slight improvement in task load balancing across the servers, as we observe fewer servers being overloaded or overwhelmed. However it's not completely optimal as there appears to be a specific concentration on a couple of servers having fewer tasks, and therefore demonstrating an unequal distribution of tasks. Conversely the logarithmic cooling schedule has managed to outperform both the MILP algorithm and its geometric counterpart, as we observe optimal task load balancing across the servers as shown in the figures 3a and 3d. In its heatmap we see a better balance of execution time and energy efficiency as highlighted by the lighter regions across the heatmap, indicating that less servers are overwhelmed, which would result in both optimizing for execution time and energy efficiency.



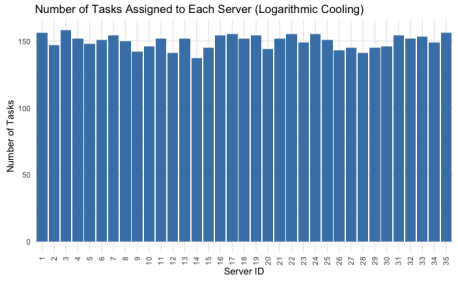
(a) Geometric Cooling Task Assignment Heatmap



(b) Geometric Cooling Task Assignment Bar Plot



(c) Task Assignment Heatmap (Energy Efficiency Maximization)



(d) Task Assignment Bar Plot (Energy Efficiency Maximization)

Figure 3: SA Task Assignment

Finally we remark that we are able to see these different insights as the SA algorithm is more flexible given it explores a wider variety of solutions (neighbors) compared to the MILP technique which provides an exact solution. We now proceed to experimenting with the Genetic Algorithm (GA).

5 Genetic Algorithms (GA)

Genetic Algorithms are population-based optimization methods inspired by the principles of natural selection and genetics. They evolve a population of candidate solutions over successive generations through selection, crossover, and mutation to find optimal or near-optimal solutions. In our study, the Genetic Algorithm is utilized to explore various task assignments to servers, aiming to minimize execution time and maximize energy efficiency while ensuring that resource constraints are met.

5.1 Problem Formulation

The problem formulation invovles the same decision variables, the major differences here are the hyperparameters the algorithm introduces, these parameters are presented in table 5 as well as the values that we used to execute the algorithm. As previously eluded the constraints remain consistent.

Decision Variables

$$x_{i,j} = \begin{cases} 1 & \text{if task } i \text{ is assigned to server } j, \\ 0 & \text{otherwise,} \end{cases} \quad \forall i = 1, \ldots, N; \quad \forall j = 1, \ldots, M$$

Parameters

Parameter	Description	Value
Population Size	The number of candidate solutions (individuals) in each generation.	100
Number of Generations	The total number of iterations the algorithm will execute.	100
Crossover Rate	The probability of performing crossover between pairs of selected parents.	0.8
Mutation Rate	The probability of mutating each gene in an offspring.	0.01
Tournament Size	The number of individuals competing in each tournament selection round.	3
Crossover Type	The method used to combine genetic information from parent solutions to produce offspring.	Single-Point, Uniform
Mutation Type	The method used to introduce variations into offspring solutions.	Random Reassignment, Swap
Generation Model	The strategy used for creating the next generation of the population.	Generational, Steady-State

Table 5: Core Parameters of the Genetic Algorithm

Objective Function

$$\text{Objective} = \alpha \times \text{Total Execution Time} - \beta \times \text{Total Energy Efficiency} + \text{Penalty Terms}$$

where:

- **Total Execution Time**: Sum of execution times for all tasks based on their server assignments.
- **Total Energy Efficiency**: Sum of energy efficiencies for all tasks based on their server assignments.
- **Penalty Terms**: Impose a cost for any violations of resource constraints (e.g., CPU, memory) to ensure feasibility.

5.1.1 Genetic Operators

To effectively explore the solution space and evolve optimal task assignments, we employed the following genetic operators:

Crossover Operators

- **N-Point Crossover**: Two crossover points are randomly selected. Offspring inherit segments from both parents between these points, promoting diverse task distributions.
- **Uniform Crossover**: Each gene is independently chosen from either parent with a 50% probability, enhancing genetic diversity.

Mutation Operators

- **Random Reassignment Mutation**: A randomly selected task is reassigned to a different server, introducing variability.

- **Swap Mutation**: The server assignments of two randomly chosen tasks are swapped, maintaining feasibility while altering distributions.

Generational Models

- **Generational Model**: The entire population is replaced by offspring each generation, ensuring broad exploration of task assignments.
- **Steady-State Model**: Only a few individuals are replaced each generation, preserving high-quality solutions and accelerating convergence.

5.2 Results and Analysis

We performed our analysis on the following combinations of parameters shown in table 6, we start by exploring the results of the generational model and then proceed to the steady state model results.

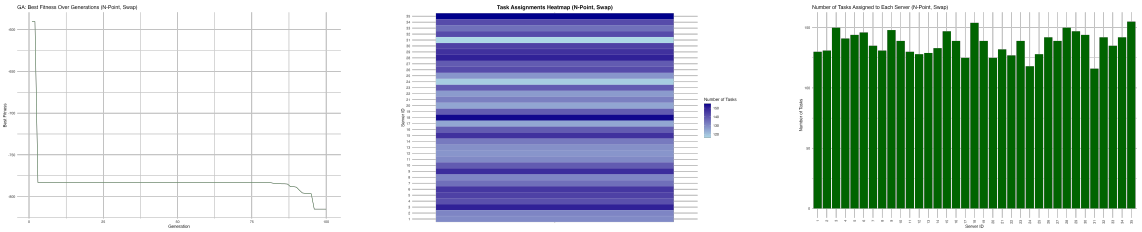
Model	Crossover	Mutation
Generational	N-Point	Swap
	N-Point	Random Assignment
	Uniform	Swap
	Uniform	Random Assignment
Steady State	N-Point	Swap
	N-Point	Random Assignment
	Uniform	Swap
	Uniform	Random Assignment

Table 6: Combinations of Genetic Algorithm Parameters for Generational and Steady-State Models

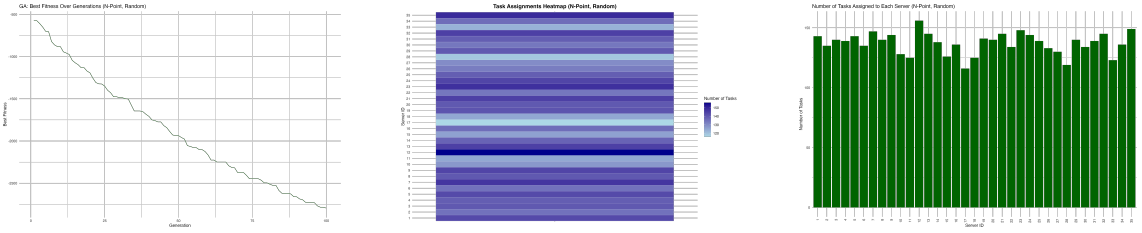
In all instances we kept all the other hyperparameters consistent such as the population size, number of generations, mutation rate and the crossover rate.

5.2.1 Generational Model Results

N-Point Crossover with Swap and Random Reassignment Mutation



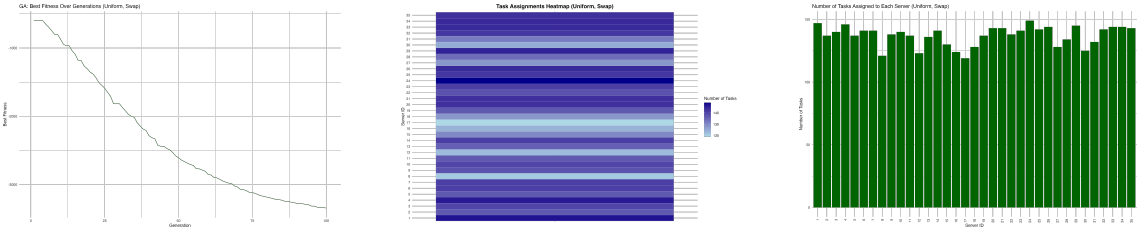
(a) N-Point Crossover with Swap Mutation



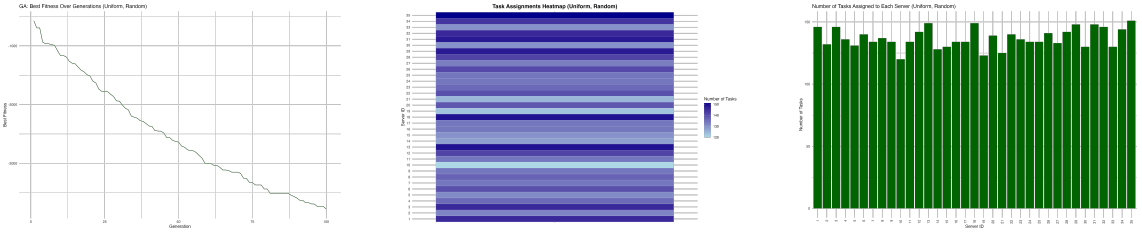
(b) N-Point Crossover with Random Reassingment Mutation

Figure 4: Generational N-Point Model

Uniform Crossover with Swap and Random Reassignment Mutation



(a) Uniform Crossover with Swap Mutation



(b) Uniform Crossover with Random Mutation

Figure 5: Generational Uniform Model

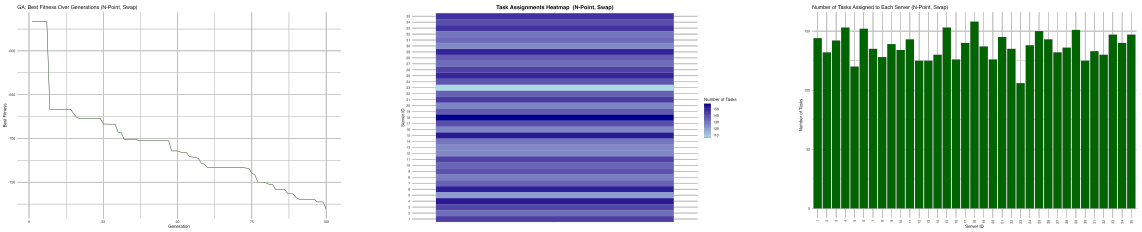
In figures 4 and 5 we show the results of executing the generational model against the given

combinations. What we observe is that the results are fairly similar with very little differences across their treatment of task allocation across the servers, they have all managed to optimally load balance the tasks across the server similar to the SA algorithm.

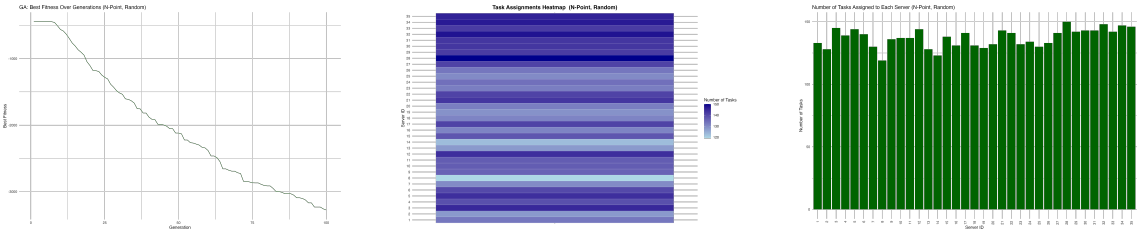
However the n-point crossover with swap mutation combination stands out as we observe that its fitness across the generations achieves convergence whereby with the other combinations we have not fully reached convergence at the given generation size. In contrast to the MILP and SA models, we observe superior optimization of the objectives compared to the MILP algorithm, however we observe comparable performance with the SA algorithm as both algorithms aimed to equitably allocate tasks across all the servers with no idling servers as we observered lighter regions within their heatmaps. This is positive as it means that we maximize efficiency given that no server is idle which subsequently means that we can minimize execution time.

5.2.2 Steady State Model Results

N-Point Crossover with Swap and Random Mutation



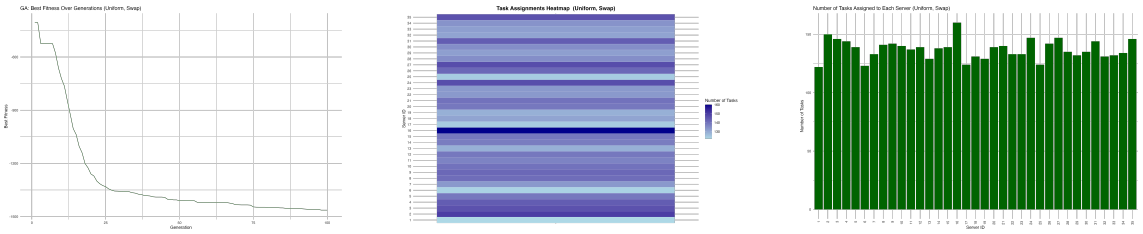
(a) N-Point Crossover with Swap Mutation



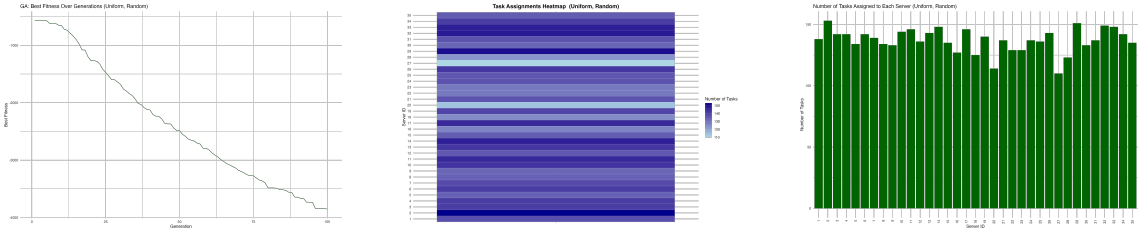
(b) N-Point Crossover with Random Mutation

Figure 6: Generational N-Point Model

Uniform Crossover with Swap and Random Mutation



(a) Uniform Crossover with Swap Mutation



(b) Uniform Crossover with Random Mutation

Figure 7: Generational Uniform Model

Although we observe similar results to the generational model, for the steady-state model we note that the uniform crossover with swap mutation combination provided the optimal convergence in fitness as shown in figure 7a, task allocation remained fairly consistent however for the optimally convergence combination we observe from its heatmap(figure 7a) that servers were were not completely overloaded as we observe more lighter regions, which is a potential indication of a superior solution given this combination would ensure that servers are able to do more which would lower execution time whilst at the same time maximizing energy efficiency.

Overall across both generational models we observe that the algorithm has aimed to ensure that it completely utilizes all the servers for task allocation and has therefore efficiently allocated the tasks without overloading a given server with tasks, their performance is on par with the SA algorithm and in one instance superior. In relation to the MILP algorithm the GA algorithm has outperfomed it with regards to our business objectives.

6 Multi-Objective Goal Programming (MOP)

Multi-Objective Programming (MOP) is a further extension of linear programming that allows for the simultaneous optimization of multiple, often conflicting, objectives. In the context of our cloud computing task assignment, MOP enables us to balance competing goals such as minimizing execution time and energy consumption while ensuring efficient utilization of server resources.

6.1 Problem Formulation

With our last method, the formulation remains consistent with the exception that we have to simultaneously optimize two conflicting goals. The constraints remain consistent.

Decision Variables

$$x_{i,j} = \begin{cases} 1 & \text{if task } i \text{ is assigned to server } j, \\ 0 & \text{otherwise,} \end{cases} \quad \forall i = 1, \ldots, N; \quad \forall j = 1, \ldots, M$$

Parameters

We present the algorithm upper bounds values as well as the weight of each goal in table 7, we have opted to provide equal importance to each objective.

Parameter	Description	Value
Weights	Relative importance of each objective in the optimization process.	Energy: 0.5, Time: 0.5
Maximum CPU Usage	Maximum CPU capacity for each server. Ensures that server resources are not overutilized.	80 units per server
Maximum Memory Usage	Maximum Memory capacity for each server. Prevents memory overcommitment.	80 units per server
Maximum Network Traffic	Maximum Network Traffic capacity for each server. Maintains network performance.	80 units per server
Maximum Power Consumption	Maximum Power Consumption capacity for each server. Controls energy usage and thermal output.	80 units per server

Table 7: Core Parameters of Multi-Objective Goal Programming

Objective Functions

We aim to achieve the following objectives simultaneously:

1. **Minimize Total Execution Time**:

$$\text{Minimize} \quad \sum_{j=1}^{J} \sum_{i=1}^{I} t_i \times x_{i,j}$$

2. **Maximize Total Energy Efficiency**:

$$\text{Maximize} \quad \sum_{j=1}^{J} \sum_{i=1}^{I} e_i \times x_{i,j}$$

Where:

- t_i and e_i respectively represents the execution time of task i and the energy efficiency of task i similar to our MILP algorithm formulation.

6.2 Implemented Methods

We have implemented two distinct methods within the MOP framework to address the task assignment problem: the Archemedian Method and the Tchebycheff Method. Both methods adhere to the defined constraints and leverage the core parameters to optimize the objectives effectively.

6.2.1 Archemedian Method

The Archemedian Method integrates the multiple objectives into a single scalar objective by applying weighted coefficients. This approach simplifies the optimization problem, allowing for straightforward minimization using linear programming techniques. The weighted objective function is defined as:

$$\text{Objective} = \alpha \times \sum_{j=1}^{J} \sum_{i=1}^{I} t_i \times x_{i,j} - \beta \times \sum_{j=1}^{J} \sum_{i=1}^{I} e_i \times x_{i,j}$$

Where:

- α and β are coefficients representing the relative importance of each objective and are weighted equally in our setup.

6.2.2 Tchebycheff Method

The Tchebycheff Method introduces an auxiliary variable Z to capture the maximum weighted deviation from the ideal objective values. By minimizing Z, this method ensures a balanced optimization that prevents the dominance of any single objective, promoting a more equitable trade-off between execution time and energy efficiency.

The optimization problem is formulated as:

$$\text{Minimize} \quad Z$$

Subject to:

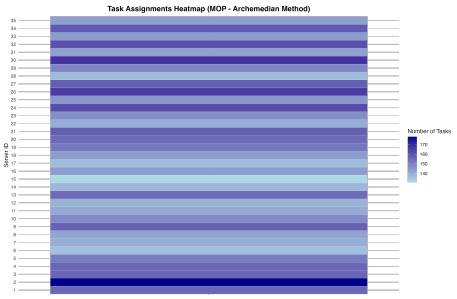
$$Z \geq \alpha \times (execution\_time^{\text{ideal}} - \sum_{j=1}^{J} \sum_{i=1}^{I} t_i \times x_{i,j})$$

$$Z \geq \beta \times (energy\_efficiency^{\text{ideal}} - \sum_{j=1}^{J} \sum_{i=1}^{I} e_i \times x_{i,j})$$

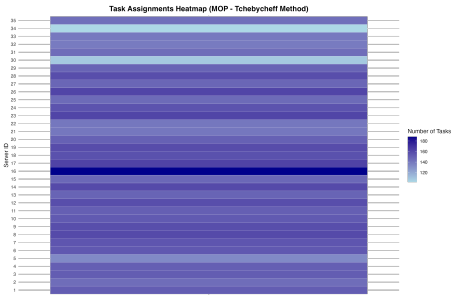
Along with the previously defined constraints

6.3 Results and Analysis

In figure 8 we present the results of the task distribution heatmaps for both the Archimedean and Tchebycheff methods and in figure 9 we present a combined barplot of their task distribution.



(a) Task Assignment Heatmap (Archemedian Method)



(b) Task Assignment Heatmap (Tchebycheff Method)

Figure 8: Task Assignment Heatmaps for MOP Methods

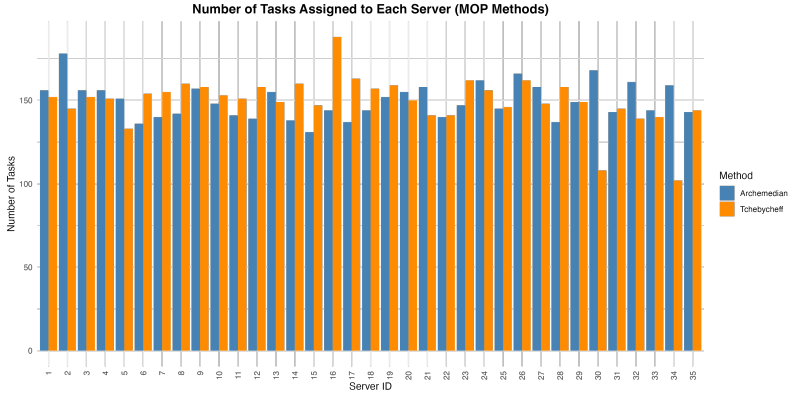


Figure 9: Number of Tasks Assigned to Each Server (MOP Methods)

We observe in figures 8a and 8b that both the Archemedian and Tchebycheff techniques optimally managed to prioritize balancing both objectives as we note a fair task allocation across the servers, with the Archemedian method demonstrating a potentially more optimal solution given it has more lighter regions within its heatmap. Their performance is therefore on par with both the SA and GA algorithms whilst it outperforms its MILP counterpart.

7 Conclusion

Across all the optimization algorithms we observed that each algorithm managed to find optimal solutions and fulffil the task allocation objectives under the given constraints. However although comparable certain algorithms outperformed others with regards to prioritizing both objectives. The MILP algorithm fail short as its optimal solutions overloaded the servers under both the execution time and energy efficiency objectives. It faired better on the energy objective however it's hard to make a conclusive decision on it's performance as the business objective is to fundamentally optimize for both objectives.

The remaining objectives aimed to balance both objectives and their optimal solutions were as follow:

- Simulated Annealing: Optimization with logarithmic cooling over 100 000 iterations
- Genetic Algorithm: Optimization with a steady state generational model coupled with a uniform crossover and a swap mutation.
- Multi-Objective Goal Programming: Archemedian Goal Programming

In all of these optimal solutions each algorithm performance was comparable as they were able to find a task distribution strategy which achieved a fair and not overwhelming allocation of tasks onto servers, therefore balancing both objectives. However the optimal GA algorithm and MOP solutions marginally outperformed the SA algorithm as we observed more lighter regions across their heatmaps indicating better load-balancing across the servers and as a result would reduce server bottlenecks therefore optimizing execution time whilst balancing energy efficiency.

As a final remark given the comparable results another factor that we should consider is when we attempt to optimize the same objectives at a larger scale i.e increasing the number of tasks, servers etc. In this instance the SA algorithm indicated the best overall speed/convergence performance, followed by the GA algorithm and finally the MOP algorithm.