MBTA Optimal Schedule Finder

Ashwin Gurumurthy
gurumurthy.as@northeastern.edu

Sujan Dhandapani Murali Balakrishnan
dhandapanimuraliba.s@northeastern.edu

Chittela Ajayanath
ajayanath.c@northeastern.edu

***Abstract*—Scheduling trains effectively in busy city transits like Boston's MBTA is a complex task, since there are a lot of factors to be balanced like the rider demand, service quality and other operational limits. In order to improve how train frequencies are assigned across various hours, day types and directions, this project utilizes actual passenger flow data to create an optimization framework that uses 3 algorithms : Hill Climbing, Simulated Annealing and Genetic Algorithm. The primary goals are to reduce passenger wait time, prevent both congestion and underuse, and maintaining service frequency under limits. A custom cost function was designed to penalize inefficiencies including excessive wait times, overcapacity, underuse and frequency violations. All the three algorithms were able to generate practical schedules, with Simulated Annealing consistently obtaining lower costs. The visualizations and performance analysis further show the effectiveness of our project, which offers useful information for transit planning. The results show us how AI-based optimization offer drastic improvements to public transit operations.**

Introduction

Public transport is important to city life, moving millions of people daily. Boston's MBTA system highlights the challenge of balancing changing passenger demand with reliable service. Traditional methods like fixed schedules or manual tweaks often fall short, struggling to adapt to real-time shifts like commute changes, events, or delays.

With detailed passenger data and better optimization techniques, we can better predict demand and adjust train schedules accordingly. However, the problem is complex, involving constraints like train capacity, frequency limits, resource availability, and multiple trade-offs that must be carefully balanced.

This project addresses the challenge using real MBTA data and optimization algorithms like Simulated Annealing, Hill Climbing, and Genetic Algorithms. A custom cost function was designed to penalize inefficiencies such as long wait times, overcrowding, underutilization, and frequency violations.

This project also involved building a Streamlit-based web app to enhance user experience. The tool allows users to visualize passenger demand and experiment with different scheduling outputs. Overall, the project shows how AI-based methods can improve public transit planning and offer a practical alternative to traditional scheduling techniques.

Related Work

Zhang et al. aimed to optimize irregular train schedules in urban rail systems by minimizing the maximum train loading rate and the average deviation of departure intervals under time-varying demand, subject to operational constraints like headway, running time, dwell time, and train capacity[4]. They used simulated annealing combined with a large neighborhood search (LNS) metaheuristic to effectively explore the solution space and escape local minima.

Oliveira and Smith proposed a hybrid scheduling method using hill climbing, modeling the problem as a disjunctive graph to detect track conflicts and job sequences[5]. Constraint propagation ruled out infeasible train orderings, while hill climbing was used to iteratively adjust the schedule along the "critical path" the route that determines total schedule length thereby reducing delays and improving track utilization.

Barman et al. introduced a hybrid approach combining a Fixed Path model with a Genetic Algorithm (GA) to minimize total train delays [6]. The GA explored route possibilities, and the Fixed Path model evaluated each route's total delay as a fitness score. They used roulette wheel selection, single-point crossover, and limited mutation to evolve better solutions, effectively separating routing from scheduling for efficient optimization.

Problem Statement

The MBTA rapid transit system needs a smarter scheduling approach to allocate trains based on real passenger demand. Traditional fixed schedules fail to adapt to dynamic ridership patterns, resulting in overcrowding during peak hours and underutilization during off-peak periods.This project addresses the issue by designing and implementing an AI-based optimization system that determines optimal train frequencies across time slots, days, and directions—minimizing costs while meeting demand and operational constraints..

This project aims to minimize passenger wait times through optimal frequency allocation and prevent overcrowding by keeping train capacity under 1000 passengers. It also focused on optimizing resource use by reducing underutilization during off-peak hours, maintaining service quality with 6–30 minute frequency limits, and balancing efficiency by aligning service with demand.

Methods

A. Dataset

For this project, we used publicly available datasets from the Massachusetts Bay Transportation Authority (MBTA) through the MassDOT Open Data Portal[2]. The primary dataset was the MBTA Rail Ridership by Time Period, Season, Route, Line and Stop dataset which provides passenger counts broken by the time of the day(time period), stop, direction and day type (weekday, Saturday, Sunday). This data helped us understand the actual demand patterns and informed us about the MBTA's

schedule of the trains throughout the day. We also used the GTFS (General Transit Feed Specification)[3] data to extract data to find transit information like route sequences and train timings, which was basically essential for modelling the trip planning and building the web app.

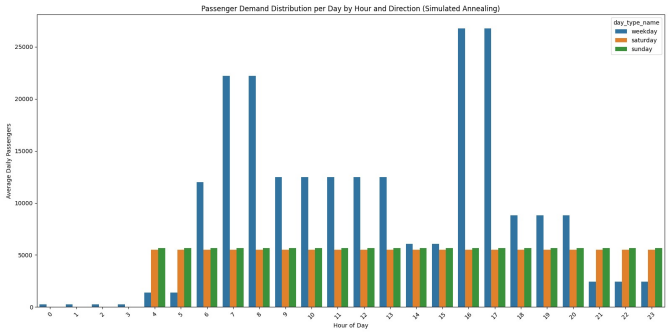


Fig. 1: Passenger Demand Distribution

B. Data Preprocessing

The raw MBTA passenger flow data was cleaned and standardized to remove missing and inconsistent entries. Passenger counts were recorded for different time periods of the day which have to be converted into hourly bins for each day type and direction. Demand was normalized by number of service days and cumulative loads were calculated to estimate train occupancy. Additionally, the geometric mean was applied to the demand values for each time slot and direction to provide a more stable and representative demand profile. The final preprocessed dataset acts as the foundation for the optimization algorithms, making sure that scheduling decisions are based on a more representative data. Figure 1 shows the variation in passenger demand for each hour of the day, separately plotted for weekdays, Saturdays, and Sundays.

Each scheduling slot is uniquely defined by a combination of day type (such as weekday or weekend), hour of the day, and travel direction. These combinations are used as keys in a dictionary to efficiently store and retrieve information like train allocations and passenger demand. This structure allows the optimization algorithms to manage and update train schedules across all relevant time periods and travel directions throughout the week.

C. State Representation in the Search Space

In this project, each state in the search space represents a complete weekly train schedule. A state is basically a dictionary, where each key is a unique combination of day type, hour, and travel direction such as ('weekday', 8, 0) and the corresponding value indicates how many trains are scheduled for that slot. For example, an entry like ('weekday', 8, 0): 10 means 10 inbound trains are scheduled at 8 AM on weekdays. To explore better schedules, the algorithm performs small changes or moves by randomly selecting a time slot and adjusting the number of trains assigned to it, either increasing or decreasing one or two trains at random. This flexible structure enables the optimization algorithms to evaluate a wide variety of scheduling configurations in search of more efficient, lower-cost solutions.

D. Hill Climbing

In this project, Hill Climbing was one of the algorithms used to optimize train allocations in the MBTA scheduling problem. The algorithm starts with an initial solution, that is a randomly generated schedule that assigns a reasonable number of trains to each time slot. Here, feasibility refers to a train frequency per hour that is within acceptable operational bounds and meets all the other requirements. At each iteration, the algorithm chooses a random time slot and proposes a small change to the number of trains in that slot by increasing or decreasing by 1 or 2. This process creates a new neighboring solution. Then the cost of the modified schedule is evaluated using a penalty based cost function explained in Equation 1.

If the new proposed schedule results in a lower total cost, the change is accepted and becomes the new current state. If the cost increases or stays the same, the algorithm returns to the previous solution. This greedy approach allows the algorithm to iteratively find a locally optimal solution. Also, to improve the robustness and to avoid getting stuck at local minima, the algorithm uses two main mechanisms.

1) Random Restarts : After each full run, the algorithm restarts with a new random schedule , we allow up to 3 random restarts , helping the search explore different regions of the solutions.
2) Early Stopping : If no improvement was observed in the cost for 200 consecutive steps, the algorithm stops and terminates the current run early.

Although Hill Climbing is computationally efficient and easy to implement, it is very prone to local optima because of its greedy nature. With well tuned parameters, Hill Climbing consistently produced efficient schedules that reduced costs, but still it did not outperform Simulated Annealing and Genetic Algorithm.

E. Simulated Annealing

Simulated Annealing is a probabilistic algorithm inspired by the annealing process in metallurgy, where materials are heated and gradually cooled to achieve a low-energy crystalline state. In optimization, simulated annealing begins with an initial solution and explores the solution space by accepting both improving and, with some probability, worsening moves. This allows it to escape local minima. Over time, the probability of accepting worse solutions decreases based on a cooling rate(0.995), which mimics the decline in temperature(initially 1000) during physical annealing[1].

Train dispatch frequencies on the MBTA Orange Line are optimized through the use of Simulated Annealing, which minimizes a cost function (Equation 1) that takes into account underutilization, passenger wait times, train overload, and frequency violations. The technique iteratively investigates nearby configurations after beginning with a random solution.

To prevent local minima, worse solutions are accepted probabilistically, with the likelihood of acceptance declining

Algorithm 1 Hill Climbing

Initialize current solution x with random train allocations
Set best solution $x_{\text{best}} \leftarrow x$
Compute current cost $C \leftarrow f(x)$
for each restart $r = 1$ to r_{max} **do**
    Set `no_improve_rounds` $\leftarrow 0$
    for iteration $i = 1$ to N **do**
        Randomly select one slot to modify
        Propose change $\delta \in \{-2, -1, 1, 2\}$
        Modify train allocation $\rightarrow x'$
        Compute new cost $C' \leftarrow f(x')$
        if $C' < C$ **then**
            Accept $x \leftarrow x'$
            $C \leftarrow C'$, reset `no_improve_rounds`
            if $C' < f(x_{\text{best}})$ **then**
                $x_{\text{best}} \leftarrow x'$
            end if
        else
            Reject and revert to previous x
            Increment `no_improve_rounds`
        end if
        if `no_improve_rounds` $>$ threshold **then**
            break
        end if
    end for
    Reinitialize x randomly
end for
return best solution x_{best}

Algorithm 2 Simulated Annealing

Initialize temperature $T \leftarrow T_0$
Initialize current solution x with random train allocations
Set best solution $x_{\text{best}} \leftarrow x$
Compute current cost $C \leftarrow f(x)$
while $T > T_{\text{min}}$ **do**
    Randomly select k time slots to modify
    for each selected slot **do**
        Modify train allocation to form neighbor x'
    end for
    Compute new cost $C' \leftarrow f(x')$
    $\Delta \leftarrow C' - C$
    if $\Delta < 0$ **then**
        Accept $x' \leftarrow x$
    else if random() $< \exp(-\Delta/T)$ **then**
        Accept $x' \leftarrow x$ (probabilistically)
    else
        Reject and revert to previous x
    end if
    if $f(x) < f(x_{\text{best}})$ **then**
        $x_{\text{best}} \leftarrow x$
    end if
    Update temperature: $T \leftarrow \alpha \cdot T$
end while
return best solution x_{best}

from higher cooling temperature. We have considered a higher cooling rate to get more exploration and a large search space which help to get minimum convergence. The optimal schedule is determined by the procedure and is the best configuration discovered.

F. Genetic Algorithm

The genetic algorithm is inspired by Darwinian evolution and the concept of "survival of the fittest." For consistency, we used the same model parameters as in hill climbing and simulated annealing. Each individual represents a state, mapping slot times to train counts, with the train counts acting as chromosomes that impact the cost function.

We begin with an initial population of 50 slots, each assigned a random chromosome representing train counts, selected within a defined range. The fitness of each chromosome is calculated using the $computefitness$ function, which utilizes the cost function. Since the genetic algorithm is a maximization algorithm, we compute the inverse of our cost function as 1/(1+cost). This ensures the cost remains positive, and a lower cost results in a higher fitness.

In the genetic algorithm, parents are selected using roulette wheel selection: a random number is drawn between 0 and the total population fitness, and the first individual exceeding it is chosen. This method encourages broader exploration by giving all individuals a chance. Before selection, we retain the top 5 fittest individuals an approach known as elitism,which helps preserve strong solutions while still allowing exploration and aiding convergence.

With a crossover rate of 0.8, selected parents undergo uniform crossover, where each gene is picked from either parent with a chance of 50 After generating a child schedule, each gene has a 0.2 probability of mutating to a value between the minimum (3) and maximum (10) allowed trains.This process of parent selection and child generation continues for 1000 generations. The number 1000 was chosen to maintain consistency with the other optimization methods, enabling a fair comparison.

Genetic algorithms are powerful for local search in large spaces, but in smaller ones like ours, they risk premature convergence. Still, with well-tuned parameters, our implementation was able to effectively optimize the cost function a result discussed further in a later section.

This approach may be summarized in Algorithm 3:

EXPERIMENTS AND RESULTS

We designed a custom cost function that accounts for significant operational inefficiencies in order to evaluate how well various local search algorithms optimize train schedules. Passenger overload, under use of train capacity, long wait times, frequency infractions, and outages during peak hours are all taken into consideration by the objective function.

For each time slot defined by the combination of *(day_type, hour, direction)*, the total cost is computed as:

Algorithm 3 Genetic Algorithm

Input: slots, geometric_mean, penalties, train_capacity, min_trains, population_size, generations, mutation_rate, number_of_elites, crossover_rate
Output: optimized_slots, best_cost, cost_progress
Initialize first generation with random chromosomes
Initialize *best_solution*, *best_fitness* $\leftarrow -\infty$, *best_cost*
for each individual in population **do**
    Calculate fitness using *ComputeFitness(individual)*
    if fitness $>$ best_fitness **then**
        Update *best_fitness*, *best_solution*, and *best_cost*
    end if
end for
for generation = 1 to generations **do**
    Evaluate fitness of all individuals using *ComputeFitness(individual)*, store as *fitness_scores*
    Identify individual with maximum fitness as *current_generation_max*
    if *current_generation_max* $>$ *best_fitness* **then**
        Update *best_solution*, *best_fitness*, and *best_cost*
    end if
    Preserve *number_of_elites* fittest individuals
    Select parent1, parent2 = *roulette wheel selection(population, fitness_scores)*
    child1, child2 using *uniform_crossover(parent1, parent2)*
    Apply random mutations to offspring with probability *mutation_rate*
    Form new generation by combining elites and mutated offspring
end for
return *best_solution* as *optimized_slots*, *best_cost*, *cost_progress*

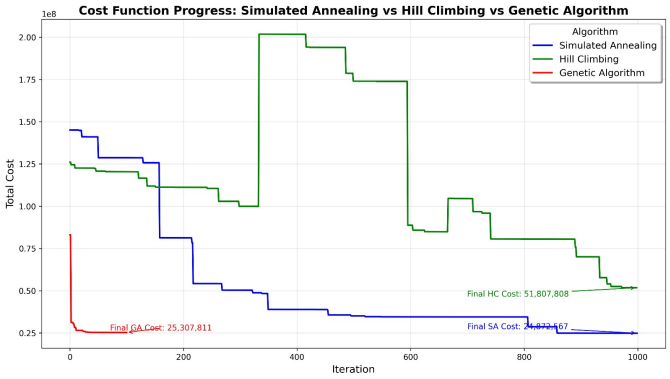
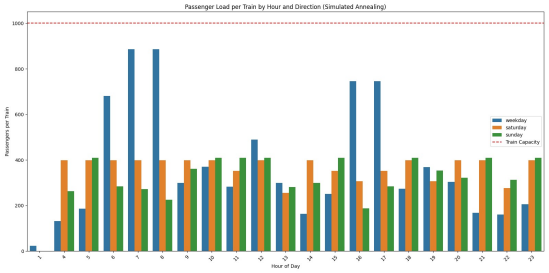


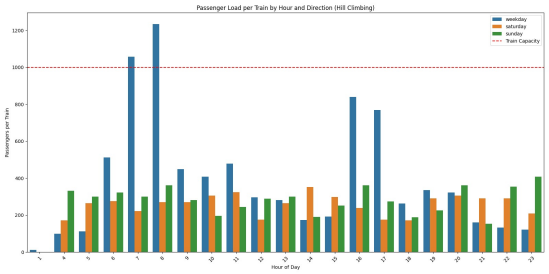
Fig. 2: Cost Comparison

$$C_{\text{total}} = \sum \left(C_{\text{overload}} + C_{\text{underutilization}} + C_{\text{wait\_time}} + C_{\text{frequency\_violation}} + C_{\text{no\_trains}} \right) \quad (1)$$

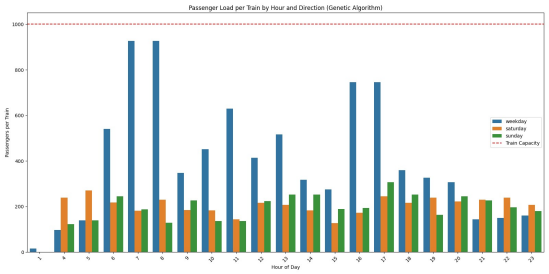
Where:



(a) Simulated Annealing



(b) Hill Climbing



(c) Genetic Algorithm

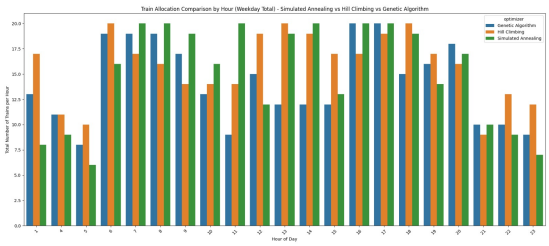
Fig. 3: Passenger Load per Train by Hour for Three Optimization Algorithms

- C_{overload}: Penalty added when passenger load exceeds the maximum train capacity (1000 passengers).
- $C_{\text{underutilization}}$: Penalty applied when train load is below 30% capacity (under 300 passengers), to promote efficient operations.
- $C_{\text{wait\_time}}$: A linear cost based on average wait time, encouraging shorter and more predictable intervals.
- $C_{\text{frequency\_violation}}$: Applied when train dispatch frequency falls outside the allowed 6–30 minute range.
- $C_{\text{no\_trains}}$: A large penalty (100,000,000) imposed when no train is scheduled during a time slot with demand, ensuring coverage.

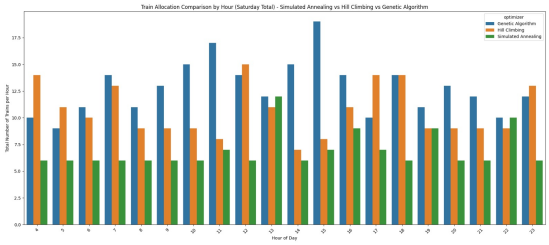
The daily passenger load per train for each of the three optimization algorithms are shown in Figure 3, broken down by weekday, Saturday, and Sunday. The maximum train capacity of 1000 passengers is shown by a red dashed line. In order to achieve a balanced and effective distribution, Simulated Annealing and Genetic Algorithm (Figure 3a) continuously keep the passenger load below the maximum threshold throughout the day. Simulated Annealing shows resilience in capacity planning by preventing crowding during peak hours and main-

taining utilization over the underuse penalty threshold.

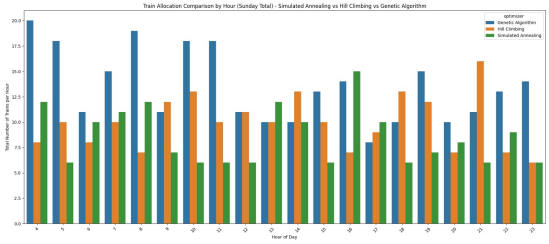
In contrast, Hill Climbing (Figure 3b) causes overcrowding during a number of time slots, particularly around morning and evening peaks. The algorithm tends to get trapped in local minima, often failing to explore alternative configurations that could improve load balancing. In comparison to Simulated Annealing, the Genetic Algorithm (Figure 3c) shows a little more fluctuation but still maintains respectable passenger loads. Some time slots exhibit under-use, despite avoiding over crowding. Overall, Simulated Annealing performs better than the others by generating the most reliable and practical load distribution that satisfies passenger demand and operational limitations, demonstrating its usefulness in optimising train scheduling.



(a) Weekday



(b) Saturday



(c) Sunday

Fig. 4: Train per Hour comparison for all Three Optimization Algorithms

The average passenger load per train for three optimization algorithm Simulated Annealing, Hill Climbing, and Genetic Algorithm is compared for various hours and day types (weekday, Saturday, and Sunday) in Figure 4. Simulated Annealing, out of the three, constantly distributes the passenger according to the actual demand throughout the day, avoiding excessive peaks and adhering to the 1000 passenger train capacity restriction. In contrast, Hill Climbing reacts more forcefully to changes in demand, but it frequently leads to overcrowded trains during rush hour, particularly on weekdays, which results in cost function penalties. In order to avoid both underutilization and overload.

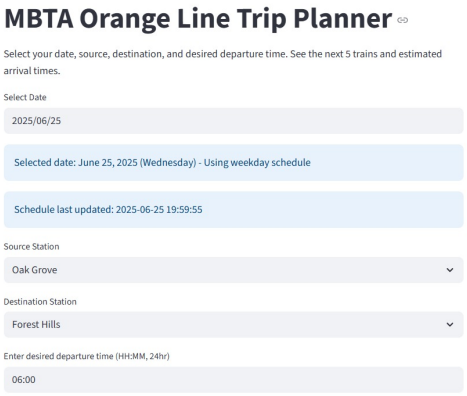


Fig. 5: Streamlit Web Application Option Page

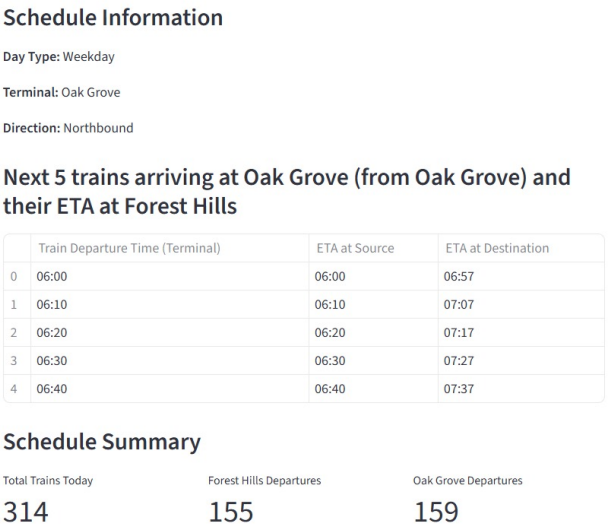


Fig. 6: Streamlit Web Application Schedule Information

For the MBTA Orange Line, we created a responsive Streamlit-based web application to improve user interaction and offer real-time train schedule details. The application enables users to conveniently plan their trip by choosing a certain day, source station, destination station, and desired departure time, as seen in Figure 5.

The application dynamically shows the next five trains leaving from the chosen terminal after the user enters their preferences. The user chooses a source and destination and the departure time in the example shown in Figure 6. The next five scheduled train departures, beginning at the chosen time, are then retrieved and shown by the system, together with an estimate of when they will arrive at the source and destination stations. Furthermore, an overview of the total number of trains

scheduled for the chosen day, broken down by direction, is given in the timetable summary section at the bottom.

The real-time schedule visibility provided by this tool not only gives users the confidence to plan trips, but it also lays the groundwork for future improvements like congestion prediction based on optimized dispatch outputs and delay-aware trip planning.

Conclusion and Future Work

This project shows how local search can address a common commuter issue. By combining MBTA GTFS schedule data with MassDOT's ridership data, it builds a strong foundation for analyzing passenger flow and generating optimal schedules. A key challenge for any search algorithm is balancing exploitation and exploration, especially within a tightly constrained search space. The publicly available data offers only seasonal averages, lacking detailed insight into actual ridership patterns.

A purely greedy approach didn't work well our early tests quickly converged to nearly uniform train distributions and got stuck in local optima. To encourage better exploration, we adjusted each algorithm accordingly: random restarts in hill climbing, a higher cooling rate in simulated annealing for more randomness, and in genetic algorithm, we used uniform crossover[7], roulette wheel selection to allow less fit individuals a chance, and a moderately high mutation rate to promote diversity despite GA's inherently greedy nature.

Another important point is that we didn't use any heuristics or fixed starting state, all algorithms begin from random feasible schedules. As a result, outputs vary with each run, with Hill Climbing showing the most fluctuation, followed by Simulated Annealing, while Genetic Algorithm remained relatively stable Simulated annealing and genetic algorithm generally kept passenger loads under the 1000 limit, though occasionally it failed to do so. Hill Climbing, however, often failed to meet this requirement. Our loss function plot (Figure 7) shows simulated annealing as the top performer in minimizing cost across most runs, with genetic algorithm close behind and hill climbing trailing. In some cases, the genetic algorithm outperformed simulated annealing. To illustrate this, inspired by [8], we ran all three algorithms 30 times and tracked which gave the best result per run. The graph below summarizes these outcomes.

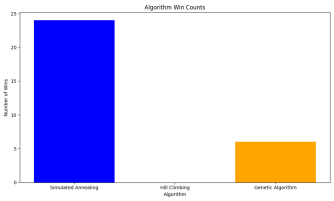


Fig. 7: Comparison of cost function for 30 runs

Overall, it is clear simulated annealing produced the most optimal solution most number of times, and its the best approach, all things considered.

Our work is not exhaustive, there is a lot more that may be done to take this to the next level.While the project showed clear improvements across the three optimization algorithms, it didn't include a direct comparison with the actual MBTA schedule or a simple uniform baseline. As a next step, comparing the optimized results with a real-world reference schedule would help better understand the practical benefits and real-life impact of the proposed solutions. Currently, only the MBTA orange line is taken into consideration for schedule optimization. We may extend this project to other lines including other modes of transport such as the commuter rail, the silver line and the bus system and even inter- commutability within lines and modes of transport. Further, we may integrate this with more real time data by making use of MBTA's API and google's API for crowd data at various times of the day. More anomalies and external factors can be taken into account such as public holidays, special event, track and train servicing down times etc. Infact, an even more comprehensive cost function may be designed for this purpose, improving existing penalties such as penalizing every additional passenger over the limit, rather than set a hard limit. In the sense, 1 or 100 more than the limit is currently penalized the same way rather a more dynamic penalization system would make more sense. To design a better cost function, a more detailed data set may be needed and this may be possible with better collaboration with the MBTA.

Github Repository

For more details and source code, visit our GitHub repository: https://github.com/SujanDM04/MBTA-LATEST

References

[1] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[2] MassDOT MBTA Passenger Flow Dataset. [Online]. Available: http://mbta-massdot.opendata.arcgis.com/datasets/MassDOT::mbta-rail-ridership-by-time-period-season-route-line-and-stop/about

[3] MBTA GTFS Data. [Online]. Available: https://www.mbta.com/developers/gtfs

[4] Zhang H, Ni S, Gao Y. Train Scheduling Optimization for an Urban Rail Transit Line: A Simulated-Annealing Algorithm Using a Large Neighborhood Search Metaheuristic. *Journal of advanced transportation*. 2022;2022(1):1-17. doi:10.1155/2022/9604362

[5] Oliveira, E., Smith, B.M. (2001). A Combined Constraint-Based Search Method for Single-Track Railway Scheduling Problem. In: Brazdil, P., Jorge, A. (eds) Progress in Artificial Intelligence. EPIA 2001. Lecture Notes in Computer Science(), vol 2258. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-45329-6_36

[6] R. Barman, C. J. Baishya, B. Kharmalki, A. Syiemlieh, K. B. Pegu, T. Das, and G. Saha, "Automated Train Scheduling System Using Genetic Algorithm," in *2015 International Symposium on Advanced Computing and Communication (ISACC)*, Silchar, India, Dec. 2015, pp. 50–55. doi: 10.1109/ISACC.2015.7377333

[7] https://algorithmafternoon.com/books/genetic_algorithm/chapter05

[8] https://rajagopalvenkat.com/teaching/resources/CityLayout.pdf