Original Research Article

Heuristic Bus Route Optimization Incorporating Taxi-Derived Route Preferences under NYC Congestion Pricing

SeungYun Lee

CheongShim International Academy, 40 Dongsan-ri, Gapyeong-eup, Gapyeong-gun, Gyeonggi-do, 32424, Republic of Korea

ABSTRACT

This study proposes a data-driven optimization framework for designing efficient bus routes in Manhattan under the new congestion toll policy. Using origin-destination data from the New York City Yellow Taxi Trip Records, urban mobility is modeled as a directed graph, and a HybridScore function is defined to integrate congestion surcharges, Central Business District (CBD) fees, and trip frequency. Three metaheuristic algorithms, Genetic Algorithm (GA), Simulated Annealing (SA), and Ant Colony Optimization (ACO), are applied to maximize the weighted sum over a single K-cycle, representing a feasible bus loop. Comparative experiments at K = 10, 25, and 50 show that ACO consistently achieves the best performance, balancing solution quality, convergence speed, and stability, while SA rapidly produces feasible initial solutions. Beyond algorithmic performance, the results demonstrate the framework's potential to inform real-world transit design by identifying corridors where expanded bus services could reduce congestion within Manhattan's toll zone. This approach links computational optimization to sustainable policy implementation, offering a scalable method for building adaptive public transportation networks in dense urban environments.

Keywords: Congestion Pricing; Transit Network Design Problem (TNDP); Graph-Based Route Optimization; Yellow Taxi Trip Records; Heuristic Algorithms; Ant Colony Optimization (ACO); Simulated Annealing (SA); Genetic Algorithm (GA)

INTRODUCTION

Urban traffic congestion remains a pervasive challenge for cities across the globe. According to traffic analytics firms such as INRIX, drivers in major metropolitan areas lose dozens to hundreds of hours annually in stalled traffic, with significant social costs: wasted fuel, lost productivity, environmental degradation, and increased accident risk (1). Conventional infrastructure expansion often fails to keep pace with rising vehicle demand (2). As a result, policy makers have increasingly turned to economic instruments to manage demand on heavily trafficked roads.

A congestion toll (or congestion pricing) is a demand-management policy that requires drivers to pay an additional fee to enter or operate within areas of heavy traffic. Unlike traditional road tolls that finance highway construction or maintenance, congestion tolls are primarily designed to influence driver behavior. The toll amounts, designated areas, and operating hours are determined by governments and managed by transportation authorities, with revenues often reinvested

Corresponding author: SeungYun Lee, E-mail: serry0827@gmail.com.

Accepted February 5, 2026
https://doi.org/10.70251/HYJR2348.41623634

into public transit infrastructure.

The most recent large-scale implementation in the United States is New York City's congestion toll. Beginning January 5th, 2025, drivers entering Lower Manhattan's Central Business District (CBD), defined as south of and including 60th Street, but excluding highways such as the FDR Drive, West Side Highway (Route 9A), and the Hugh L. Carey Tunnel connections, are required to pay a congestion fee (3). The main objectives of congestion tolls are traffic reduction by encouraging commuters to use mass transit, environmental benefits that follow, and financial funding, as the tolls generate revenue that can be reinvested in public transportation systems, creating a virtuous cycle (4).

Meanwhile, another policy implemented called Congestion Surcharge, this time in January 1st, 2019, applies to for-hire vehicle (FHVs) trips and taxis for trips that start, end, or traverse the "Congestion Zone" (generally Manhattan south of 96th Street) (5). The standard surcharge amounts are, for example: \$2.50 per trip for non-shared medallion taxis, \$2.75 per trip for non-shared non-medallion for-hire vehicles, and \$0.75 per trip for shared / pooled rides (i.e. in any eligible vehicle). Congestion Surcharge was intended to influence travel in the for-hire and taxi sector, which has been known to contribute disproportionately to congestion in dense Manhattan streets (through pickups, drop-offs, deadheading, cruising) (6).

Observably, both the congestion toll and the congestion surcharge share a common underlying vision: reducing vehicular dependence and transitioning toward a less congested, sustainable urban mobility framework. However, for such pricing mechanisms to yield long-term benefits, a robust and efficient public transportation network must serve as a viable alternative to private vehicle use (7). Merely discouraging car trips through fees is insufficient unless commuters are provided with accessible, reliable, and well-planned transit options (8).

Yet, devising new public transit routes presents a persistent challenge. Traditional route planning methods often rely on static demand data or historical ridership patterns, which fail to capture the dynamic flow of human movement shaped by new policies such as congestion tolls and surcharges. To maximize the effectiveness of public transit expansion, bus routes must be established along corridors that reflect actual commuter behaviors and evolving mobility flows.

This study proposes a data-driven approach to designing efficient bus routes across Manhattan, integrating human mobility and policy implementation.

METHODS AND MATERIALS

Collecting Data

The Yellow Taxi Trip Records dataset is a publicly available dataset provided by the New York City Taxi and Limousine Commission (TLC) that contains detailed trip-level data for all yellow taxi rides recorded in New York City (9). It is one of the most comprehensive open urban mobility datasets in the world and is frequently used for research in transportation planning, machine learning, and urban systems analysis. The dataset used in this research covers a six-month period from January to June 2025, during which New York City's congestion toll had already been fully implemented, ensuring that all observations reflect post-policy travel behavior.

Defining the Optimization Problem

First, we define the optimization problem: Every OD (origin-destination) pair starts at i and ends at j.

Through this we define the set of trips S_{ij} to be:

$$S_{ij} = \{\, t \in T \mid \mathrm{PU}(t) = i, \mathrm{DO}(t) = j \,\} \qquad \text{Eq. (1)}$$

T: set of taxi trips
t: index over individual taxi trips
PU(t): pickup location of trip t
OD(t): drop-off location of trip t

We then define the costs of a certain route, factoring in both congestion fee/congestion surcharge and route usage, as such:

$$\mathrm{HybridScore}_{ij} = \left(\textstyle\sum_{t\in S_{ij}} (\mathrm{congestion\_surcharge}_t + \mathrm{cbd\_congestion\ \_fee}_t)\right) \cdot \left|S_{ij}\right|^{\lambda-1} \qquad \text{Eq. (2)}$$

λ: weighting parameter controlling the relative importance of congestion pricing effects versus observed trip frequency in the arc reward. For our study, the value 0.3 was selected experimentally as an adequate number to avoid overemphasizing either trip frequency or congestion pricing effects.

Then we use a directed graph of stops and observed OD arcs G, defined by:

$$G = (V, E) \qquad \text{Eq. (3)}$$

V: the set of candidate bus stops, indexed by i, j
E: the set of directed edges representing the OD taxi flows

Objective Function

The objective function is designed to maximize the sum of the costs assigned to all edges that make up the route, so that overall energy efficiency is improved if introduced to eco-friendly public transportation.

$$\max \sum_{(i,j)\in E} r_{ij}x_{ij} \qquad \text{Eq. (4)}$$

$r_{ij} \geq 0$: reward (or weight) associated with arc

$x_{ij} \in 0,1$: 1 if arc is included in the route

$y_i \in 0,1$: 1 if stop is visited

$u_i \in [0,K]$: Miller-Tucker-Zemlin (MTZ) ordering variable (subtour elimination)

$z_i \in 0,1$: symmetry-breaking anchor selector

Here, the term "cost" does not represent an expense incurred by the proposed bus route itself. Instead, it represents the congestion-related penalties currently borne by private vehicles and taxis along each origin–destination corridor. Edges with high congestion surcharges or CBD fees correspond to routes where vehicular demand is high and where policymakers explicitly seek to discourage private vehicle usage.

Maximizing the HybridScore therefore does not imply promoting high-cost travel, but rather identifying corridors where a large amount of policy-induced and environmental penalty is currently accumulated. Introducing bus services along these corridors enables the displacement of high-emission, fee-paying vehicle trips with more energy-efficient public transportation. In this sense, the objective function maximizes the potential for avoided congestion costs and emissions by prioritizing routes where modal substitution yields the greatest systemic benefit.

Constraints:

1) Maximum Bus Stops

$$\sum_{i\in V} y_i \leq K\,d \qquad \text{Eq. (5)}$$

$K \in \mathbb{Z}_+$: upper bound on the number of visited stops

2) Each stop used must have exactly one edge coming in and one edge going out.

$$\sum_{j:(i,j)\in E} x_{ij} = y_i \quad \forall i \in V \qquad \text{Eq. (6)}$$

$$\sum_{j:(j,i)\in E} x_{ji} = y_i \quad \forall i \in V \qquad \text{Eq. (7)}$$

3) The route must be closed and consistent (number of active edges equals the number of used stops.

$$\sum_{(i,j)\in E} x_{ij} = \sum_{i\in V} y_j \qquad \text{Eq. (8)}$$

4) The sequence variable enforces sequential order through:

$$1 \leq u_i \leq K y_i \quad \forall i \in V \qquad \text{Eq. (9)}$$

$$u_j \geq u_i + 1 - M\left(1 - x_{ij}\right) \quad \forall (i,j) \in E\,, i \neq j \qquad \text{Eq. (10)}$$

In this mathematical formulation, M denotes a sufficiently large constant used to express the subtour elimination constraint in linear form. In practice, M is not instantiated numerically, as the optimization is solved using metaheuristic algorithms rather than a mixed-integer linear programming solver. Its inclusion serves only to formally define the feasibility conditions of a simple cycle; subtour prevention is enforced algorithmically during solution construction and evaluation.

5) There is exactly one starting location in each group of stops selected for a route:

$$\sum_{i\in V} z_i = 1 \qquad \text{Eq. (11)}$$

$$z_i \leq y_i \quad \forall i \in V \qquad \text{Eq. (12)}$$

$$u_i \geq z_i \quad \forall i \in V \qquad \text{Eq. (13)}$$

6) Each variable exists within the following domain:

$$x_{ij} \in \{0,1\},\quad y_i \in \{0,1\},\quad z_i \in \{0,1\},\quad 0 \leq u_i \leq K \qquad \text{Eq. (14)}$$
$$a = b$$

The optimization problem seeks a simple directed cycle (K-cycle), meaning that each node may be visited at most once and the route forms a closed loop without repeated stops. Feasibility is enforced through degree constraints and MTZ subtour elimination constraints, ensuring that the resulting route constitutes a single simple cycle of length at most K.

Data Preprocessing and Graph Construction

The raw Yellow Taxi Trip Records were originally provided in Parquet format and were first converted to CSV files for preprocessing. From each record, only the variables PU(t) which was named PULocationID, DO(t) which was named DOLocationID, congestion_ surcharge, and cbd_congestion_fee were retained. Trips were filtered to include only those with RatecodeID = 1, excluding special cases such as airport trips or group fares, in order to focus on standard passenger travel behavior.

The pickup and dropoff location identifiers were cast to int16, while congestion surcharge and CBD congestion fee values were cast to float32 to ensure numerical stability. Any missing, NaN, or negative surcharge values (treated as invalid) were replaced with zero. Trips for which both congestion_surcharge and cbd_congestion_fee were zero were excluded from further analysis.

The remaining trips were aggregated by identical (PULocationID, DOLocationID) pairs to construct OD flows. This aggregation produced 18,005 unique OD pairs (E) and 262 nodes (V), with the total number of aggregated trips equal to 32,958,808.

The optimization graph is constructed as a zone-based directed graph, where each node corresponds to a New York City taxi zone identified by its official Location ID. No spatial remapping to bus stops, intersections, grid cells, or other geometric representations was performed. Instead, nodes and edges were defined directly based on the zone identifiers provided in the original dataset. While this study operates at the taxi-zone level, the proposed optimization framework is not inherently restricted to this resolution. If future datasets provide publicly accessible identifiers for finer-grained spatial units such as individual buildings, parks, or transit stops, the same methodology can be applied without modification.

Algorithmic Execution

We then use our objective function as basis to run 3 different types of algorithms: Genetic Algorithm, Simulated Annealing, and Ant Colony Optimization, as well as a baseline code for comparison. Given the nonlinear, combinatorial, and constraint-intensive nature of the K-cycle routing problem, heuristic and meta-heuristic optimization approaches are employed to obtain high-quality solutions within practical computational budgets (10).

To ensure reproducibility, randomness was strictly controlled across all experiments. Before each run, both Python's built-in random number generator and NumPy's random number generator were initialized with the same seed value (42) to ensure identical stochastic behavior at the start of each experiment. For performance evaluation, each algorithm and each value of $K \in \{10,25,50\}$ were then evaluated over five independent runs using a unified seed set {100, 101, 102, 103, 104}, while keeping the input data and hyperparameter settings fixed. This procedure ensures that reported performance differences arise from algorithmic behavior rather than uncontrolled randomness.

Baseline Code

The baseline method constructs a feasible K-cycle by greedily selecting edges based on descending HybridScore, subject to cycle feasibility constraints including non-repetition, directionality, and closure. At each step, candidate edges that violate feasibility are discarded, and selection continues until a valid cycle of length K is formed or no further feasible extensions are available. This approach provides a deterministic reference solution against which metaheuristic methods are evaluated. In this baseline implementation, W is set equal to K, ensuring that the branching factor scales consistently with the target cycle length and preventing excessive candidate expansion for larger values of K. We condensed the baseline description into a single paragraph, formalized it using algorithmic notation, and added a comparative table (Table 8) summarizing complexity, advantages, and limitations. A schematic illustration of the baseline model has also been added as Figure 3.

In Table 1, *Status* indicates whether a feasible K-cycle was found within the time limit. *Objective value* denotes the total arc reward of the resulting cycle. *Gap* represents the relative difference between the obtained solution and the best-known solution among all methods. Entries marked as N/A when K=25 and K=50 indicate that the algorithm failed to converge to a feasible cycle or a meaningful local minimum within the allotted runtime, making quantitative evaluation impossible. The baseline greedy algorithm terminated normally after exhausting all restart attempts without identifying a feasible closed cycle, and no crashes or memory errors occurred during these runs. Because this greedy algorithm reflects the objective function and constrains only one step ahead, it can quickly find plausible solutions within the time limit, but for large K or sparse graphs, it easily fails to find a feasible answer or gets stuck in local optima.

Therefore, we infer that metaheuristics such as Ant Colony Optimization (ACO), Simulated Annealing (SA), and Genetic Algorithm (GA) may help overcome this shortcoming with identical input and time budget, which we will demonstrate through comparative experiments.

Table 1. *Performance of the baseline greedy K-cycle construction method for K = 10, 25, and 50. Status indicates whether a feasible simple directed K-cycle was successfully identified within the fixed runtime. Objective Value denotes the total HybridScore of the resulting route, while Time (s) reports wall-clock runtime. |V| and |E| represent the number of vertices and edges included in the final solution. The baseline successfully constructs a feasible cycle for K = 10 but fails for K = 25 and K = 50 (N/A entries), demonstrating that greedy edge selection without global exploration becomes insufficient as cycle length and feasibility constraints increase.*

K	Status	Objective value	Gap	Time (s)	|v|	|E|
10	Feasible	863.475	0.158	160.11	10	10
25	N/A	N/A	N/A	160.10	0	0
50	N/A	N/A	N/A	160.12	0	0

Simulated Annealing

Simulated Annealing is another popular meta-heuristic approach to optimization problems. Based on the annealing process of metallurgy, SA works by first setting an initial solution (11, 12). Randomly modifying it, it assigns a probability of acceptance.

In SA, the probability of acceptance is a rule that decides whether the algorithm will move from its current solution to a new, modified one. If the new solution is better than the current one, it is always accepted. If the new solution is worse, it is not immediately rejected; instead, the algorithm flips a kind of weighted coin to decide whether to accept it anyway. This "coin" is biased by two factors: how much worse the new solution is and what the current temperature is.

At the start of the algorithm, the temperature is set high. A high temperature means that even significantly worse solutions have a fair chance of being accepted. For example, in the early stages the algorithm might accept a new solution that is much worse than the current one with, say, a 40–50% chance. This keeps the search moving widely across the landscape instead of getting stuck early. As the algorithm proceeds, the temperature is gradually lowered according to a cooling schedule. When the temperature is lower, the chance of accepting a worse solution drops sharply. At a moderate temperature, a slightly worse solution might still have a small chance of being accepted (perhaps 5–10%), while a much worse one would be nearly impossible to accept. By the time the temperature is very low, even small deteriorations are almost always rejected. This mechanism is what allows SA to "jump out" of local minima early in the search, while still focusing in on the best regions as the run progresses. The probability of acceptance starts high and gradually shrinks to near zero, mirroring the physical idea of atoms in a hot metal being free to move at first, but eventually settling into a stable crystal structure as they cool.

For our runs, the algorithm was configured using the following parameters: *Initial Temperature* (T_0), which denotes the starting temperature controlling early exploration; *Cooling Rate* which specifies the decrement applied to temperature at each step; *Cooling Multiplier* (α), which defines the geometric decay factor for temperature reduction; and *Final Temp Factor*, which determines the termination threshold as a fraction of T_0. The fixed parameter values are summarized in Table 2.

Table 2. *Fixed parameter values used for Simulated Annealing (SA) across all experiments. Initial Temperature (T_0) controls early-stage exploration, Cooling Rate specifies the decrement applied to temperature per iteration, Cooling Multiplier (α) defines the geometric decay factor, and Final Temperature Factor determines the termination threshold as a fraction of T_0. These parameters were held constant across all K to ensure that observed performance differences arise from problem size rather than parameter tuning.*

Attribute	Value
Initial Temperature (T0)	200
Cooling Rate	0.0005
Cooling Multiplier (Alpha)	0.9995
Final Temp Factor	1e-3

Table 3 shows run-specific settings for each value of $K \in \{10,25,50\}$: the maximum number of iterations allowed per run (MAX_ITERS) and the maximum number of consecutive iterations without objective improvement before early termination (STALL_ITERS). The computational budget was scaled with *K*: for *K = 10*, *max_iters = 20,000* and *stall_iters = 1,500* were used; for *K = 25*, *max_iters = 40,000* and *stall_iters = 2,000*; and for *K = 50*, *max_iters = 80,000* and *stall_iters = 3,000*. The common parameters were *T0 = 200*, *ALPHA = 1 − 0.0005*, and *T_FINAL_FACTOR = 1e−3*.

Simulated Annealing terminates when any of the following conditions is met: the maximum number of iterations is reached, the temperature T drops below T_0 × T_FINAL_FACTOR, or no improvement in the best objective value is observed for *stall_iters* consecutive iterations (stall-based early stopping).

Table 3. *Run-specific computational budgets for Simulated Annealing as a function of K. MAX_ITERS denotes the maximum number of iterations allowed per run, and STALL_ ITERS specifies the maximum number of consecutive iterations without improvement before early termination. Budgets scale with K to account for the increased complexity of maintaining feasibility in longer cycles, enabling fair comparison across problem sizes.*

K	MAX_ITERS	STALL_ITERS
10	20,000	1,500
25	40,000	2,000
50	80,000	3,000

Genetic Algorithm

The Genetic Algorithm is an optimization strategy based on the functions of "survival of the fittest" in natural selection of Darwinian evolution. It operates by repeating a simulation set on a set of decided conditions and rules coordinated to direct towards a certain desired result. The algorithm, first creating a random population, follows the given guidelines to score and scale the population, such that when selecting a percentage of individuals from the population to be a parent generation, they choose parents closest to the intended outcome (11, 13); this process is called selection. Our algorithm uses tournament selection, in which the 10 individuals (Tournament Size) are randomly selected, compared in fitness and 2 top-performing individuals are chosen to be the parent generation (Elitism). Then, using the parents' characteristics to create a second generation, the algorithm uses two main rules of crossover and mutation. In crossover, it can use two parents to combine their characteristic to form a child. In mutation, random changes are applied to the child. Crossover Rate and Mutation Rate refers to the probability of the two processes being carried out. All parameter values used can be found in Table 4.

The Genetic Algorithm is executed for a fixed number of generations and does not employ early stopping in the current implementation. For $K = 10$, a population size of $pop = 100$ and $gen = 500$ generations were used; for $K = 25$, $pop = 150$ and $gen = 1{,}000$; and for $K = 50$, $pop = 200$ and $gen = 1{,}500$.

Table 5 shows run-specific settings for each value of $K \in \{10,25,50\}$: Population Size refers to the number of candidate solutions per generation, while Generations specifies the fixed number of evolutionary iterations executed per run.

Table 4. *Fixed parameter settings for the Genetic Algorithm (GA). Mutation Rate and Crossover Rate control genetic variation, Tournament Size determines selection pressure, and Elitism specifies the number of top-performing individuals preserved across generations. These values were kept constant across all experiments to isolate the effect of population size and number of generations on performance.*

Attribute	Value
Mutation Rate	0.10
Crossover Rate	0.90
Tournament Size	10
Elitism (kept elites)	2

Table 5. *Genetic Algorithm run settings for K = 10, 25, and 50. Population Size indicates the number of candidate solutions per generation, and Generations denotes the total number of evolutionary iterations executed per run. Larger values of K are assigned larger populations and longer runs to compensate for the expanding solution space, which contributes to GA's strong performance for small K but rapidly increasing runtime for larger K.*

K	Population Size	Generations
10	100	500
25	150	1,000
50	200	1,500

Ant Colony Optimization

Ant Colony Optimization is a metaheuristic inspired by the foraging behavior of real ants (14). In nature, ants find paths between their nest and food sources by laying down chemical substances called pheromones. As ants travel, they deposit pheromone on the paths they take. Other ants sense this pheromone and are more likely to follow paths with stronger pheromone levels. Over time, shorter and more efficient paths receive more traffic and more pheromone reinforcement, while longer or less effective paths lose pheromone due to natural evaporation. This decentralized process, based on simple individual rules, results in a colony collectively finding optimal or near-optimal routes.

Table 6 contains the parameters for our ACO model. ACO involves a colony of artificial ants repeatedly constructing solutions to a given problem. Each ant starts from some initial position and builds a solution incrementally, making decisions at each step about which component (or path) to add next. Unlike deterministic algorithms, the decision is made probabilistically, influenced by pheromone level (Alpha), which represent the algorithm's shared memory. At the start of the algorithm, all edges are initialized with a small constant Initial_pheromones (τ₀) to promote early-stage exploration before sufficient solution feedback is available. Components (such as edges in a graph, tasks in a schedule, or variable assignments) that have appeared in good solutions receive higher pheromone levels. This makes ants more likely to select them in the future. Heuristic information is problem-specific knowledge that estimates how good a particular move is. The variable Beta controls the weight of heuristic information derived from the arc reward. After all ants construct their solutions, the quality of each solution is evaluated. Paths or components used in better solutions receive pheromone reinforcement, which increases their probability of being chosen in subsequent iterations. Meanwhile, pheromone evaporation occurs on all paths, reducing their strength gradually according to the Pheromone_persistence parameter, which determines the fraction of pheromone retained between successive generations. This dual mechanism ensures a balance between exploitation (reinforcing good solutions) and exploration (preventing the colony from converging too quickly on a suboptimal path).

Like GA, ACO also runs for a fixed number of generations under the same *K*-dependent schedule, with a fixed number of ants per generation; each ant's route construction is bounded by a maximum of 200 trials to prevent infinite loops. Table 7 shows run-specific settings for each value of $K \in \{10,25,50\}$: Number of Ants denotes the size of the ant colony per generation, and Generations specifies the total number of iterations executed for each experiment. For $K = 10$, $ants = 20$ and $gen = 500$ were used; for $K = 25$, $ants = 40$ and $gen = 1{,}000$; and for $K = 50$, $ants = 60$ and $gen = 1{,}500$.

Table 6. *Parameter configuration for the Ant Colony Optimization (ACO) algorithm. Alpha controls the influence of pheromone trails, Beta weights heuristic information derived from HybridScore, Pheromone Persistence determines evaporation strength, and Initial Pheromone (τ₀) sets the starting pheromone level for all edges. These parameters govern the balance between exploration and exploitation in path construction.*

Attribute	Value
Alpha	1.0
Beta	3.0
Pheromone_persistence	0.9
Initial_pheromones (τ₀)	1e-3

Table 7. *ACO run settings by K. Number of Ants denotes the colony size per generation, and Generations indicates the total number of iterations. Computational budgets increase with K to maintain solution quality as cycle length and combinatorial complexity grow, contributing to ACO's stable performance and relatively low variance for medium and large K.*

K	Number of Ants	Generations
10	20	500
25	40	1000
50	60	1500

Table 8. *Comparative overview of the baseline greedy method, SA, GA, and ACO. The table summarizes search structure, dominant operations per iteration, stopping criteria, and scalability characteristics. The comparison highlights that SA offers minimal runtime overhead but limited global exploration, GA provides strong performance for small K at the cost of rapidly increasing computation, and ACO balances population-level exploration with edge-based memory, enabling stable and scalable performance for larger problem instances.*

Method	Search structure	Per-iteration / per-run work	Time complexity (typical)	Stopping rule
Baseline	Exact MIP/CP search	Branch-and-bound / propagation over binary variables	Worst-case exponential in (NP-hard)	Time limit T_{max} TIMAL if proven; else best FEASIBLE)
SA	Single-trajectory	Evaluate neighbor + accept/reject	$O\,(I \cdot c_{nbr})$	I iterations or $T \downarrow$ to T_{min}
GA	Population-based	Selection + crossover + mutation + repair + fitness for P	$O\,(G \cdot P \cdot c_{eval})$	G generations (or no improvement if enabled)
ACO	Colony-based	Construct a solution + pheromone update	$O\,(G \cdot A \cdot K \cdot c_{step})$	G generations (or convergence threshold)

RESULTS AND DISCUSSION

Convergence Behavior Across Algorithms

The processing time (Time-average computation time per fun and mean scores (Mean Score-average object value over five independent runs) of the three methods are shown in Table 9, with the best objective value stated in parentheses.

For the Genetic Algorithm, a clear convergence pattern is observed in which the objective value increases sharply during a very short initial phase and then rapidly plateaus for *K = 10* and *K = 25*. That is, high-quality solutions are secured quickly in early generations, while the average convergence curve shows relatively limited improvement in later generations. In contrast, for *K = 50*, GA exhibits a longer period of gradual improvement following the initial rapid increase, and in some seeds, additional late-stage jumps (stepwise increases) are observed. This suggests that as the solution space expands for larger *K*, crossover, mutation, and repair operations may occasionally generate better combinations even at later stages of the search. At the same time, as *K* increases, differences in the final convergence levels across seeds remain non-negligible, indicating that outcome variability depending on the search trajectory (i.e., initial population) persists.

Compared with GA and ACO, Simulated Annealing exhibits a convergence pattern in which improvements accumulate incrementally and require a relatively long (log-scale) iteration horizon to reach convergence. For *K = 10*, the score increases gradually from the early stage, but the timing and magnitude of improvements vary across seeds, resulting in a relatively wide shaded region during the initial phase. For *K = 25*, some seeds remain at low objective values for an extended period before eventually catching up, indicating that in a single-trajectory search, an unfavorable initial path can delay improvement. For *K = 50*, a sharp initial increase is followed by a prolonged stagnation phase. This behavior can be interpreted as a consequence of the increasing difficulty of achieving structural improvements through local neighborhood moves while maintaining cycle feasibility constraints (closure, non-repetition, and directionality) as *K* becomes large.

Taken together, the convergence curves indicate that ACO and GA achieve faster early-stage performance gains than SA. In particular, ACO tends to enter promising regions of the search space rapidly during early generations and then converge smoothly toward a stable plateau. GA demonstrates strong early convergence for small *K*, while for larger *K*, additional late-stage improvements remain possible, albeit with persistent seed-dependent variability. Although SA may have shorter wall-clock runtime, its convergence curves reveal increasingly long stagnation phases as *K* grows. Consequently, for the problem structure considered here, ACO's combination of rapid initial convergence and stable plateau behavior becomes an increasingly advantageous property as *K* increases.

From a structural perspective, this behavior reflects the alignment between each algorithm's search mechanism and the constraints of the problem. In particular, the K-cycle constraint induces a search space in which high-scoring edges tend to cluster, but must be connected into a single feasible loop without repetition. As K increases, maintaining global connectivity while incorporating multiple high-weight edges becomes increasingly difficult. ACO benefits from edge-level memory through pheromone reinforcement, allowing promising partial structures to be reused and refined across generations, which supports stable convergence even for large K. In contrast, GA relies on population-level recombination that may still generate improvements at later stages but retains seed-dependent variability, while SA's single-trajectory, local-move–based search is more prone to stagnation once early high-score regions are reached.

Table 9. *Mean HybridScore and mean runtime across five independent runs for SA, GA, and ACO at K = 10, 25, and 50. Values in parentheses indicate the best score observed among all runs for each configuration. At K = 10, GA achieves the highest mean score, while SA exhibits the shortest runtime. As K increases, ACO maintains competitive or superior mean scores while remaining substantially faster than GA, indicating a more favorable balance between solution quality, computational cost, and reproducibility for medium to large K.*

K	GA Mean Score (Best)	GA Time (s)	ACO Mean Score (Best)	ACO Time (s)	SA Mean Score (Best)	SA Time (s)
10	916.516 (924.094)	~6.57	890.812 (906.540)	~3.16	883.216 (906.846)	~0.65
25	1,998.440 (2,026.618)	~44.33	1,995.725 (2,019.553)	~25.07	1,899.956 (1,932.094)	~1.56
50	3,129.209 (3,145.527)	~155.44	3,147.505 (3,156.598)	~98.14	2,840.943 (2,870.708)	~2.47

Runtime and Variability Analysis

Figure 1 compares the mean runtime and standard deviation of SA, GA, and ACO for *K* = *10, 25,* and *50.* SA is consistently the fastest method across all values of *K*, with mean runtimes of approximately 0.45 s for *K* = *10*, 1.11 s for *K* = *25*, and 1.72 s for *K* = *50.* The corresponding standard deviations are also small, indicating very low temporal variability. This behavior is attributable to SA's single-trajectory search, which evaluates only one neighboring solution at each step and avoids population-based or generation-level operations, resulting in low per-iteration computational overhead.

In contrast, GA exhibits the largest increase in runtime as *K* grows. Mean runtimes rise sharply from approximately 4.90 s at *K* = *10* to 36.01 s at *K* = *25* and 119.09 s at *K* = *50.* In particular, the standard deviation at *K* = *25* is relatively large (approximately 4.76 s), indicating seed-dependent variability in execution time. This trend can be attributed to the cumulative cost of selection, crossover, mutation, and repeated repair (projection) into feasible cycles as the number of generations increases.

ACO is consistently faster than GA across all problem sizes and demonstrates particularly strong time efficiency for large *K*. Mean runtimes are approximately 2.37 s for *K* = *10*, 19.38 s for *K* = *25*, and 75.16 s for *K* = *50*, corresponding to speedups of roughly 2.1×, 1.9×, and 1.6× relative to GA, respectively. Moreover, standard deviations remain small, indicating high runtime reproducibility. This efficiency can be explained by restricting candidate neighbors to high-weight edges (Top-W cut) and enforcing cycle closure constraints during solution construction, thereby reducing the computational cost associated with generating infeasible paths.

Beyond mean runtime, the observed standard deviations highlight differences in execution stability across algorithms. SA exhibits consistently low variance in runtime due to its deterministic iteration structure and single-trajectory evaluation. In contrast, GA shows increasing runtime variability for medium to large K, reflecting sensitivity to population evolution and the cumulative cost of repair operations needed to enforce feasibility. ACO maintains relatively low runtime variance despite its population-based nature, indicating that its construction-based solution process and early pruning of infeasible paths yield reproducible computational behavior. This distinction is important in practical settings where predictable execution time is as critical as average performance.

Final Performance with 95% Confidence Intervals

Figure 2 (Performance with 95% CI across five seeds) presents the final best scores for each method using mean values and 95% confidence intervals computed from five seeds. The confidence interval represents the range

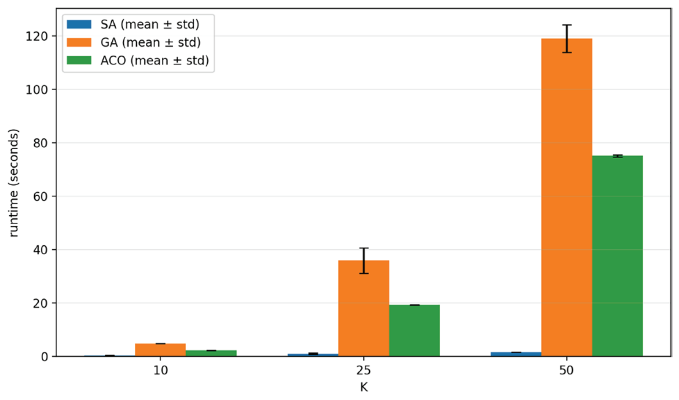

Figure 1. Mean runtime and standard deviation across five independent runs for K = 10, 25, and 50. SA consistently exhibits the lowest runtime and minimal variability due to its single-trajectory search. GA shows the steepest growth in runtime as K increases, with elevated variance at K = 25 reflecting sensitivity to stochastic initialization. ACO remains consistently faster than GA while maintaining low variance, indicating efficient scaling achieved through early pruning and restricted candidate-neighbor evaluation. Error bars represent one standard deviation.

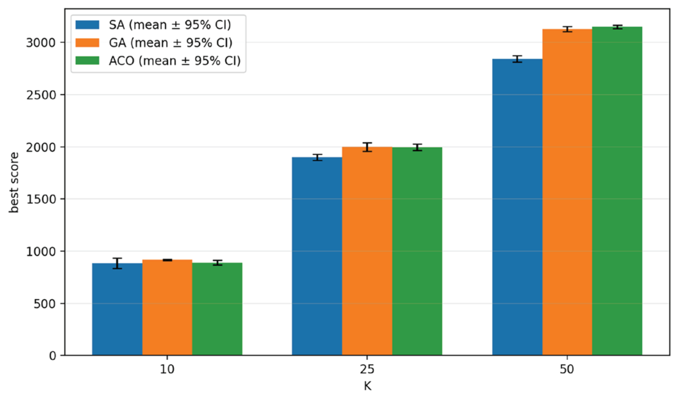

Figure 2. Final solution quality measured as mean HybridScore with 95% confidence intervals across five independent runs for K = 10, 25, and 50. Narrower confidence intervals indicate lower seed-induced variability and higher reproducibility. ACO demonstrates the most stable performance for larger K, while GA shows higher variance and SA exhibits lower mean performance as K increases.

within which the mean performance is likely to lie under repeated experiments, with narrower intervals indicating lower seed-induced variability and higher reproducibility. For *K = 10*, GA achieves the highest mean score, followed by ACO and SA (GA ≈ 916.5, ACO ≈ 890.8, SA ≈ 883.2). GA's confidence interval is relatively narrow, confirming its ability to consistently identify high-quality solutions for small *K*. In contrast, SA exhibits a wider interval, suggesting stronger dependence on the initial trajectory.

For *K = 25*, GA and ACO exhibit nearly identical mean performance, with substantially overlapping 95% confidence intervals, indicating comparable solution quality. In this regime, it is reasonable to interpret GA and ACO as performing at a similar level. SA, however,

```
**Algorithm** Baseline Model: Maximum-Weight Simple Cycle with $|V(C)| \le K$
**Require:** Directed graph $G = (V, E)$, weights $w_{uv} > 0$ for $(u, v) \in E$, stop
    limit $K$, time limit $T_{\max}$, workers $W$
**Ensure:** A feasible simple cycle $C$ (OPTIMAL if proven within $T_{\max}$; otherwise
    best FEASIBLE)
**Variables:**
    $x_{uv} \in \{0, 1\}$ for each $(u, v) \in E$                ▷ select edge
    $y_v \in \{0, 1\}$ for each $v \in V$                        ▷ visit node
    $u_v \in \{0, 1, \dots, K\}$ for each $v \in V$              ▷ MTZ order variable
Choose an anchor node $r \in V$ and enforce $y_r = 1$           ▷ anchor for MTZ
**Objective:** max $\sum_{(u,v)\in E} w_{uv}\, x_{uv}$
**Cardinality:** $\sum_{v\in V} y_v \le K$
**Degree constraints (simple cycle feasibility):**
**for all** $v \in V$ **do**
    $\sum_{(v,j)\in E} x_{vj} = y_v$                             ▷ visited ⇒ exactly one outgoing
    $\sum_{(i,v)\in E} x_{iv} = y_v$                             ▷ visited ⇒ exactly one incoming
**end for**
**Edge-node consistency:** $\sum_{(u,v)\in E} x_{uv} = \sum_{v\in V} y_v$     ▷ $|E_{\text{sel}}| = |V_{\text{sel}}|$
**Subtour elimination (anchored MTZ):**
**for all** $v \in V$ **do**
    $u_v \le K \cdot y_v$                                        ▷ if not visited then $u_v = 0$
    $u_v \ge y_v$                                                ▷ visited ⇒ $u_v \ge 1$
**end for**
$u_r = 1$
**for all** $(u, v) \in E$ with $u \ne r$ and $v \ne r$ **do**
    $u_u - u_v + K \cdot x_{uv} \le K - 1$
**end for**
Configure **Baseline Model** with time limit $T_{\max}$ and workers $W$
Solve the model and obtain status $\in$ {OPTIMAL, FEASIBLE, ...}
**if** status is OPTIMAL or FEASIBLE **then**
    $C \leftarrow \{(u, v) \in E \mid x_{uv} = 1\}$
    $z \leftarrow \sum_{(u,v)\in C} w_{uv}$                      ▷ objective value
    $b \leftarrow$ solver best objective bound
    $g \leftarrow |z - b| / \max(1, |z|)$                        ▷ report gap when not optimal
**else**
    $C \leftarrow \emptyset$, $z \leftarrow$ N/A
**end if**
**return** ($C$, $z$, status, $g$)
```

Figure 3. *Schematic illustration of the baseline greedy K-cycle construction method. Edges are selected in descending order of HybridScore subject to feasibility constraints, including directionality, non-repetition, and cycle closure. Infeasible candidate extensions are pruned at each step, and the process terminates when a closed K-cycle is formed or no further feasible extensions remain.*

shows a clearly lower mean score and a separated confidence interval, suggesting that single-trajectory search becomes less effective for identifying high-quality solutions at intermediate problem sizes. The substantial overlap of confidence intervals for GA and ACO at K = 25 suggests that their performance differences in this regime may not be statistically significant. In such cases, algorithm selection should be guided not only by mean objective value but also by computational efficiency and stability. Given ACO's lower runtime and smaller execution-time variance, it offers a more favorable performance–cost trade-off at intermediate problem sizes.

For *K = 50*, ACO achieves the highest mean score, followed by GA and then SA (ACO ≈ 3147.5, GA ≈ 3129.2, SA ≈ 2840.9). Notably, ACO's confidence interval is particularly narrow, supporting the stability and reproducibility of its performance even at large problem sizes. While GA also attains high scores, its longer runtime weakens its overall efficiency. SA exhibits a substantial drop in mean performance, consistent with the increasing difficulty of improving global structure through local neighborhood moves in large cycles, and aligns with the prolonged stagnation observed in the convergence curves. The separation of confidence intervals at K = 50 indicates a more decisive performance distinction. ACO's higher mean score combined with a notably narrow confidence interval supports the conclusion that it reliably converges to high-quality solutions for large K. SA's lower mean score and wider variability are consistent with its convergence behavior, where prolonged stagnation limits further improvement. These results suggest that as the search landscape becomes more constrained and multimodal with increasing K, algorithms capable of accumulating and reusing high-quality substructures, such as ACO, gain a clear advantage.

Taken together, the two figures indicate that while SA can produce solutions extremely quickly, its performance limitations become pronounced as *K* increases. GA demonstrates strong performance for small *K* but incurs rapidly increasing computational cost as problem size grows. In contrast, ACO maintains high performance for medium to large *K* (*K ≥ 25*) while remaining faster than GA and exhibiting smaller confidence intervals and standard deviations, indicating superior reproducibility. Therefore, when performance, runtime, and stability must be considered simultaneously, ACO represents a balanced and practically advantageous choice, while SA can be justified as an initialization strategy when extremely fast approximate solutions are required.

Overall, the results indicate that the underlying search landscape of the proposed K-cycle optimization problem is multimodal, shaped by skewed edge-weight distributions and strict cycle feasibility constraints. Under this structure, ACO's combination of population-level exploration and edge-based memory enables it to both explore multiple promising regions early and converge stably within high-quality solution clusters. GA remains effective for small K but incurs increasing computational cost and variability as repair operations dominate at larger scales. SA, while exceptionally fast, is structurally limited by local neighborhood moves and temperature decay, making it less effective for large K where global reconfiguration is required. These observations provide a mechanistic explanation for the empirical performance differences observed across algorithms.

CONCLUSION

This study evaluated metaheuristic methods to approach public transportation route configurations, implementing Yellow Taxi Trip Records data to integrate human mobility and recent policy enactments. Using Genetic Algorithms, Simulated Annealing, and Ant Colony Optimization, we ran experiments at K with the value 10, 25 and 50. ACO consistently achieved the highest overall performance, balancing solution quality, runtime, and stability. SA demonstrated remarkable speed due to its lightweight single-trajectory search, making it ideal for rapid initialization, whereas ACO effectively exploited heuristic information and collective memory to converge toward globally optimal solutions even for large problem sizes. Integrating SA as an initial solution generator followed by ACO refinement was therefore identified as the most efficient and reliable framework. From our results, considering solution quality, runtime, and reproducibility, adopting ACO as the primary algorithm for this problem is the most reasonable choice; when ultra-low latency is required, generating an initial solution with SA and then refining it using ACO is recommended.

The findings of this study demonstrate that algorithmic optimization of public transportation systems can complement economic instruments such as congestion tolls, ensuring that policy implementation is supported by data-driven infrastructure design. Nevertheless, this research is limited to static congestion and trip datasets; future work should integrate dynamic temporal data, multimodal interactions, and real-time feedback to better capture evolving urban mobility patterns.

Beyond computational performance, the results carry practical implications for real-world transit design. By mapping optimized bus loops directly onto New York City's spatial and demand data, the proposed algorithmic framework can be used by transportation authorities to identify corridors where expanded or newly introduced bus services would yield the greatest congestion relief. From an environmental standpoint, the rationale is clear. Transportation remains one of the largest contributors to greenhouse gas emissions in New York City, accounting for approximately 30% of total CO_2 output (15). Reducing the number of vehicles entering Manhattan's core not only alleviates congestion but also directly decreases fuel consumption and tailpipe emissions.

In this context, expanding and optimizing public transit networks presents a practical pathway toward achieving both congestion mitigation and emission reduction goals. The framework using ACO proposed here provides both a methodological foundation and a practical blueprint for building smarter, cleaner, and more responsive public transportation networks in congested cities worldwide. For future studies, we look forward to incorporating multimodal data integration, real-time feedback, and equity-based accessibility metrics to ensure that algorithmic efficiency translates into inclusive, sustainable, and policy-relevant transit design.

CONFLICT OF INTEREST

The author declares that there are no conflicts of interest related to this work.

REFERENCES

1. INRIX. Global Traffic Scorecard [Internet]. Kirkland (WA): INRIX; 2024. Available from: https://inrix.com/scorecard/ (accessed on 2025-9-1)
2. Downs A, Still stuck in traffic : coping with peak-hour traffic congestion. Washington (DC): Brookings Institution Press; 2004.
3. MTA. Congestion Pricing Program in New York - MTA | Tolling [Internet]. MTA. Available from: https://congestionreliefzone.mta.info/tolling (accessed on 2025-9-1)
4. City of New York. Congestion Pricing Program· NYC311 [Internet]. portal.311.nyc.gov. Available from: https://portal.311.nyc.gov/article/?kanumber=KA-03612 (Accessed on 2025-9-1)
5. Congestion surcharge [Internet]. www.tax.ny.gov. Available from: https://www.tax.ny.gov/bus/cs/csidx.htm (Accessed on 2025-9-14)
6. Taxi and For-Hire Vehicle Congestion Surcharge· NYC311 [Internet]. Nyc.gov. 2025. Available from: https://portal.311.nyc.gov/article/?kanumber=KA-03191& (Accessed on 2025-10-6)
7. Leape J. The London congestion charge. *J Econ Perspect*. 2006; 20 (4): 157-176. doi:10.1257/jep.20.4.157
8. Weber BS, Moghtaderi A, Cappellari P. Effects of congestion surcharges: From ridership to competition and safety. *Economics of Transportation*. 2025; 43: 100426. https://doi.org/10.1016/j.ecotra.2025.100426
9. TLC Trip Record Data [Internet]. Nyc.gov. Available from https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page (Accessed on 2025-10-6)
10. Christensen J, Bastien C. Chapter seven: Heuristic and meta-heuristic optimization algorithms. In: Christensen J, Bastien C, editors. Nonlinear Optimization of Vehicle Safety Structures. *Oxford: Butterworth-Heinemann*; 2016; p.277-314. doi:10.1016/B978-0-12-417297-5.00007-9.
11. Delahaye D, Chaimatanan S, Mongeau M. Simulated annealing: From basics to applications. In: Gendreau M, Potvin JY, editors. Handbook of Metaheuristics. 3rd ed. Cham (CH): Springer; 2019; p.1-35. (International Series in Operations Research & Management Science; vol. 272). doi:10.1007/978-3-319-91086-4_1.
12. Bertsimas D, Tsitsiklis J. Simulated annealing. *Stat Sci*. 1993; 8 (1): 10-15. doi:10.1214/ss/1177011077.
13. Whitley D. A Genetic Algorithm Tutorial. Fort Collins (CO): Colorado State University, Department of Computer Science; 1994.
14. Dorigo M, Stützle T. Ant Colony Optimization. Cambridge (MA): MIT Press; 2004. ISBN: 0-262-04219-3. https://doi.org/10.7551/mitpress/1290.001.0001
15. New York City Mayor's Office of Climate & Environmental Justice. New York City Greenhouse Gas Emissions Municipal Inventory [dataset]. New York (NY): City of New York; 2025. Available from: https://data.cityofnewyork.us/Environment/NYC-Greenhouse-Gas-Emissions-Inventory/wq7q-htne (Accessed on 2025-12-25)