T.R.

GEBZE TECHNICAL UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

ANALYSIS AND IMPLEMENTATION OF A BEAUTIFUL FLEET: OPTIMAL REPOSITIONING IN E-SCOOTER SHARING SYSTEMS FOR URBAN DECORUM

MEHDI KURTCEBE - UFUKCAN ERDEM

SUPERVISOR
PROF. DIDEM GÖZÜPEK

GEBZE
2025

T.R.
GEBZE TECHNICAL UNIVERSITY
FACULTY OF ENGINEERING
COMPUTER ENGINEERING DEPARTMENT

ANALYSIS AND IMPLEMENTATION OF A BEAUTIFUL FLEET: OPTIMAL REPOSITIONING IN E-SCOOTER SHARING SYSTEMS FOR URBAN DECORUM

MEHDI KURTCEBE - UFUKCAN ERDEM

SUPERVISOR
PROF. DIDEM GÖZÜPEK

2025
GEBZE

GRADUATION PROJECT
JURY APPROVAL FORM

This study has been accepted as an Undergraduate Graduation Project in the Department of Computer Engineering on 01/2025 by the following jury.

JURY

Member
(Supervisor) : Prof. Didem GÖZÜPEK

Member : Asst. Prof. Tülay AYYILDIZ

ABSTRACT

This project focuses on addressing the problem of improper electric scooter (e-scooter) placement in urban areas, which disrupts urban decorum. To solve this, we analyze and model the optimization problem presented by Carrese et al.[1]. The solution involves optimizing the routes and actions of ”beautificators” — individuals tasked with repositioning e-scooters within city zones, ensuring compliance with urban decorum while maximizing company profits.

We employ two approaches to solve the optimization problem: CPLEX[2], A mathematical programming tool to achieve optimal solutions, and Ant Colony Optimization (ACO)[3], A nature-inspired metaheuristic designed to solve complex optimization problems. The source code of the implementation is publicly available online[4].

The implementation of both solution algorithms is examined and complexity analyzes are performed. The running times and profits of the two algorithms are compared. Finally, the developed User Interface is introduced.

Keywords: E-scooters, Urban Decorum, Optimization, Metaheuristics, CPLEX, Ant Colony Optimization.

ÖZET

Bu proje, kentsel alanlarda kentsel düzeni bozan uygunsuz elektrikli skuter yerleştirme sorununu ele almaya odaklanmaktadır. Bunu çözmek için Carrese ve arkadaşları[1] tarafından önerilen optimizasyon problemi analiz edilmiş ve gerçeklendirilmiştir. Çözüm, şehir bölgelerinde skuterleri yeniden konumlandırmakla görevli kişiler olan ”güzelleştiricilerin” rotalarını ve eylemlerini optimize etmeyi, kentsel düzene uyumu sağlarken şirket kârlarını maksimize etmeyi içermektedir.

Optimizasyon problemini çözmek için iki yaklaşım kullanılmıştır: CPLEX[2], optimum çözümlere ulaşmak için kullanılan bir matematiksel programlama aracı ve Ant Colony Optimization (ACO)[3], karmaşık optimizasyon sorunlarını çözmek için tasarlanmış doğadan ilham alan bir metaheuristic. Gerçeklemelerin kaynak kodu İnternette halka açık olarak yanınlanmıştır[4].

Her iki çözüm algoritmasının gerçeklemesi incelenmiş ve karmaşıklık analizleri gerçekleştirilmiştir. İki algoritmanın çalışma süreleri ve kârları karşılaştırılmıştır. Son olarak, geliştirilen Kullanıcı Arayüzü tanıtılmıştır.

Anahtar Kelimeler: Elektrikli Skuter, Kentsel Düzen, Optimizasyon, Metaheuristic, CPLEX, Ant Colony Optimization.

ACKNOWLEDGEMENT

First of all, I would like to thank my supervisor Prof. Didem GÖZÜPEK. Then, I would like to thank everyone who has had any influence on me, especially my family.

Ufukcan ERDEM

I would like to thank my supervisor Prof. Didem GÖZÜPEK.
To everybody who love me.

Mehdi KURTCEBE

LIST OF SYMBOLS AND ABBREVIATIONS

Symbol or Abbreviation	:	**Explanation**
ACO	:	Ant Colony Optimization
BF	:	A Beautiful Fleet Article

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Electric scooters (e-scooters) have recently gained significant popularity, particularly in urban areas where travel distances are relatively short. For users, e-scooters offer a convenient and environmentally friendly alternative to cars in highly congested cities. In addition to being affordable, their widespread accessibility has contributed to their increasing prevalence as one of the most commonly used modes of transport for short-distance trips. However, the rapid growth in e-scooter usage has also introduced notable challenges in urban areas, especially in historic European cities and other locations that require careful preservation.

One of the primary issues associated with e-scooter usage is the frequent practice of users leaving e-scooters in arbitrary locations without regard for parking regulations or urban aesthetics. This unregulated parking behavior creates significant inconvenience for pedestrians and other city residents and disrupts the overall aesthetic appeal of urban environments. Over time, this practice has contributed to a decline in urban decorum, complicating efforts by local authorities to preserve the aesthetic and functional integrity of public spaces.

(a) Chaotically parked e-scooters

(b) E-scooters repositioned by a beautificator

Figure 1.1: Example beautifiying action (from BF, 2020) [1]

In response to this issue, local governments have begun implementing measures to address the problem. One such measure is the imposition of fines on e-scooter-sharing companies, holding them accountable for the negative impacts caused by improperly parked e-scooters. This approach aims to incentivize companies to develop improved

parking solutions and better fleet management practices. However, as fines increase, these companies face growing challenges in maintaining profitability and sustaining their operations. The indiscriminate parking of e-scooters has become a significant concern for these companies, as it directly impacts both their reputation and financial performance.

In this project, we are going to present and implement the optimization problem presented by Carrese et al.[1]. The paper gives a very innovative approach to the problem by repositioning the e-scooters in such a way as to meet not only the urban decorum concerns but also to improve operational efficiency in e-scooter-sharing systems. This problem will help companies reduce the disruption caused by improper parking of e-scooters while maximizing their profit.

Our project solves the optimization problem mentioned in the paper using IBM's CPLEX[2] and also provides our own solution algorithm based on the "Ant Colony Optimization" metaheuristic[3], a nature-inspired method that is very effective in solving complex optimization problems. By comparing the outputs of these two approaches, we would like to find out which of the two methods best achieves these twin goals of improvement in urban decorum and helps e-scooter-sharing systems, attaining financial sustainability.

1.1. Problem Definition

The solution proposed in the paper involves employing agents by the companies operating e-scooters to handle the repositioning of e-scooters. These agents, referred to as "beautificators" in the paper, are tasked with ensuring organized placement and adherence to urban decorum standards. Each beautificator is assigned specific responsibilities, beginning with pre-designated zones that are strategically determined during the planning phase. These zones divide the broader urban area into smaller, manageable regions where beautification and organizational activities are conducted.

Each zone has strategic points called "hotspots," representing locations where parked e-scooters are more likely to be rented. Hotspots include places like subway stations, hospitals, or other critical places that would normally attract people. These points are important in the process, as they guide where e-scooters should, to the best extent, be placed to ensure they remain accessible and functional for users while maintaining urban order.

The key objective of the problem described in the paper is to find such a route and sequence of actions for each beautificator that maximizes the total profit. Profit depends on the results of the actions taken by the beautificators, weighed against the costs and losses incurred. The challenge is to find this route within the fixed timeframe

for operations. Every beautificator operates within a timescale ranging from 0 to some maximum time t^{max}. Within this period, however, beautificators should complete the necessary work with minimum consumption of time and maximum efficiency.

The paper outlines four distinct actions that beautificators can perform:

1. **Beautification:** This involves correcting the positioning of a e-scooter, ensuring it adheres to urban decorum standards. The goal is to make the e-scooter blend harmoniously with its surroundings, addressing issues like blocking pathways or being improperly placed.

2. **Bring-to-hotspot:** This action entails relocating a e-scooter to a designated hotspot point within the beautificator's current zone. Moreover, this action also includes beautification of the e-scooter. By concentrating e-scooters in these hotspots, the system ensures they are available in areas of high demand, improving accessibility for users.

3. **Move:** This refers to the beautificator traveling from one zone to another to continue performing their tasks. Moving between zones is sometimes necessary to cover a larger area and address issues in multiple regions, but it comes with time and resource costs.

4. **Wait:** This is the idle state where a beautificator temporarily pauses actions. Waiting may occur due to time constraints, planning, or logistical reasons, but it still counts as part of the total operational time.

Each of these actions has its own time and resource requirement, which dictates the profit or loss a beautificator may realize in relation to a job. The trick is how to plan such acts strategically for efficiency and profitability without compromising the beautification objectives.

The solution, as indicated in the paper, suggests how the operational cost and aesthetic needs of the city may be struck in terms of balance. In short, by careful assignment of zones, hotspots identification, and optimization of actions by beautificators, e-scooter companies can deal with the urgent problem of disorganized e-scooter placement without sacrificing profitability and contributing to urban decorum.

1.2. Variables

Let us say the area the the e-scooter sharing system provides services is decomposed into a set of zones Z. The beautification actions can be executed by a set of beautificators B over a time horizon in which we identify a set of discretized time instants $T = \{0, 1, \ldots, t^{max}\}$. All time instants have identical duration of Δt. A number $n_z \geq 0$ of e-scooters is located in each zone $z \in Z$ at $t = 0$. E-scooters may be either located in the hotspot or out of the hotspot of the zone. We denote by n_z^{HOT} the number of e-scooters that are located in the hotspot of z at $t = 0$, and denote by n_z^{OUT} the number of e-scooters that are located outside the hotspot of z at $t = 0$. Note that it holds: $n_z = n_z^{HOT} + n_z^{OUT}$.

In order to model the actions executed by the beautificators over time, we rely on a multicommodity flow model based on a graph $G(N, A)$ made up of:

- a set of nodes $N = Z \times T$, which includes one node (z, t) for each zone-time instant couple $z \in Z, t \in T$;
- a set of arcs A that is made up four disjoint subsets, each representing one type of action that a beautificator can execute:
    1. A^{BEAU} - set of beautification arcs $a = \{(z, t), (z, t + m^{BEAU})\}$, represents beautifying action. m^{BEAU} means required time to complete beautifying action.
    2. A^{HOT} - set of hotspot arcs $a = \{(z, t), (z, t + m^{HOT})\}$, represents bring-to-hotspot action. m^{HOT} means required time to complete bring-to-hotspot action.
    3. A^{MOVE} - set of movement arcs $a = \{(z_1, t), (z_2, t + m_{z_1 z_2}^{MOVE})\}$, represents moving from zone z_1 to zone z_2. $m_{z_1 z_2}^{MOVE}$ means required time to complete move action.
    4. A^{WAIT} - set of waiting arcs $a = \{(z, t), (z, t+m^{WAIT})\}$, represents waiting action. m^{WAIT} means required time to complete wait action.

    Thus $A = A^{BEAU} \cup A^{HOT} \cup A^{MOVE} \cup A^{WAIT}$

Given a node $(z, t) \in N$, we denote by $\delta^{FW}(z, t)$ its forward star (i.e., the subset of arcs $a = [(z, t), (\bar{z}, \bar{t})] \in A$ having (z, t) as tail node) and by $\delta^{BW}(z, t)$ its backward star (i.e., the subset of arcs $a = [(\bar{z}, \bar{t}), (z, t)] \in A$) having (z, t) as head node.

In order to model the actions of the beautificators, we rely on a multicommodity flow model in which the actions executed by the beautificators over time and space are modeled as flows moving through the graph $G(N, A)$.

$$x_a^b = \begin{cases} 1 & \text{if beautificator } b \text{ executes the action associated with arc } a \\ 0 & \text{otherwise} \end{cases} \qquad \forall a \in A, b \in B \tag{1.1}$$

Formula 1.1 represents a set of binary values defined for every $a \in A, b \in B$, constituting unsplittable flow variables. It simply represents the action done by a beautificator. These variables are used in the following feasibility constraints.

1.3. Constraints

$$\sum_{a \in \delta^{FW}(z,0)} x_a^b = \begin{cases} 1 & \text{if } z = z_0(b), \\ 0 & \text{if } z \neq z_0(b) \end{cases} \qquad \forall z \in Z, b \in B \tag{1.2}$$

Formula 1.2 represents a beautificator must start his/her job from zone z_0 at $t = 0$.

$$\sum_{a \in \delta^{BW}(z,t)} x_a^b = \sum_{a \in \delta^{FW}(z,t)} x_a^b \qquad \forall z \in Z, t \in T : 0 < t < t^{max}, b \in B \tag{1.3}$$

Formula 1.3 guarantees the coherence of actions executed by the beautificators over space and time in the graph $G(N, A)$.

$$\sum_{b \in B} x_a^b \leq n_z^{OUT} - \sum_{b \in B} \sum_{a=[(\bar{z},\bar{t}),(\bar{z},\bar{t}+m^{HOT})] \in A^{HOT} : \bar{z}=z \wedge \bar{t}<t} x_a^b \qquad \forall a = [(z,t),(z,t+m^{HOT})] \in A^{HOT} \tag{1.4}$$

Formula 1.4 limits the total number of e-scooters that can be brought to hotspot in a zone can not be more than n_z^{OUT} minus the number of e-scooters that have been brought to the hotspot before the considered time instant t.

$$\sum_{b \in B} x_a^b \leq n_z - \sum_{b \in B} \sum_{a=[(\bar{z},\bar{t}),(\bar{z},\bar{t}+m^{HOT})] \in A^{HOT} : \bar{z}=z \wedge \bar{t}<t} x_a^b - \sum_{b \in B} \sum_{a=[(\bar{z},\bar{t}),(\bar{z},\bar{t}+m^{BEAU})] \in A^{BEAU} : \bar{z}=z \wedge \bar{t}<t} x_a^b \qquad \forall a = [(z,t),(z,t+m^{BEAU})] \in A^{BEAU} \tag{1.5}$$

Formula 1.5 limits the total number of e-scooters that can be beautified in a zone can not be more than n_z minus the number of e-scooters that have been beautified before

the considered time instant t and the number of e-scooters that have been brought to the hotspot before the considered time instant t.

1.4. Objective Function

$$\max \sum_{a \in A} \sum_{b \in B} \pi_a \cdot x_a^b \tag{1.6}$$

Objective function 1.6 aims to maximize total profit that obtained from every action done by every beautificator.

2. ACO AND CPLEX

ACO[3] and CPLEX[2] are two different approaches for solving optimization problems, each with their own characteristics. ACO is a metaheuristic that takes inspiration from the foraging behavior of ants; artificial ”ants” incrementally build solutions based on pheromone trails and heuristic information. It is really suited for solving complex, large-scale problems and puts a focus on adaptability and efficiency. On the other hand, CPLEX is a commercial optimization solver using mathematical programming — linear and mixed-integer programming. While CPLEX has major advantages of being extremely precise and reliable for small to medium-scale problems, it gets computationally expensive and somewhat slow when applied to larger datasets. ACO does not guarantee optimality but often reaches near-optimal solutions in less time and hence can be more practical when real-time applications or computational resources are scarce.

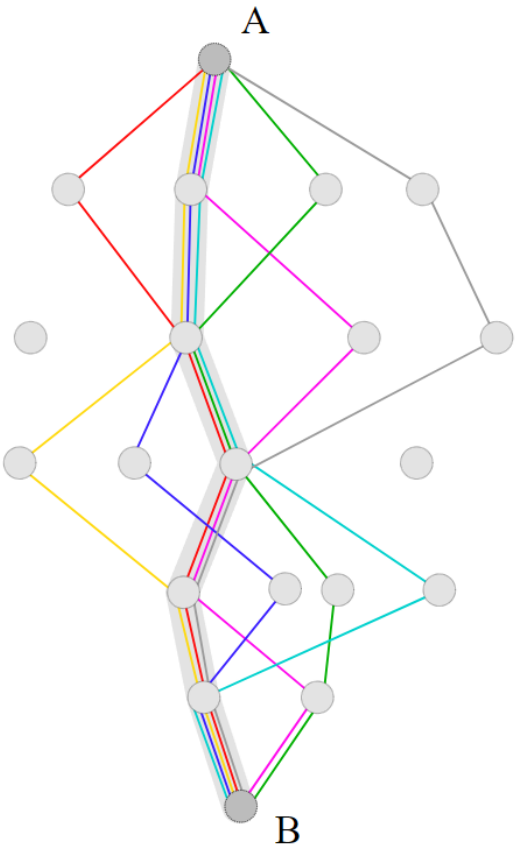

Figure 2.1: Visualization of ACO

2.1. ACO Implementation

In this chapter, we will examine implementation of ACO algorithm to solve the optimization problem. The following implementation is publicly available online [4].

2.1.1. ACO Algorithm

The Algorithm 1 shows the pseudo code of ACO algorithm.

Algorithm 1 Ant Colony Optimization for Beautification

```
1:  Initialize pheromone matrix τ
2:  Set best profits bestProfits ← −∞
3:  Set best paths bestPaths ← ∅
4:  for iteration ← 1 to num_iterations do
5:      Reset scooters (n, aco_nOut, aco_nHot)
6:      Create num_ants ants
7:      for ant ← 1 to num_ants do
8:          ant.traverse()                          ▷ Each ant constructs a solution
9:          Reset scooters (n, aco_nOut, aco_nHot)
10:     end for
11:     Evaporate pheromones: τ_{i,j} ← (1 − evaporation_rate) · τ_{i,j}
12:     Find the best ant (bestAnt) based on total profit
13:     if bestAnt.total_profit > bestProfits then
14:         bestProfits ← bestAnt.total_profit
15:         bestPaths ← bestAnt.paths
16:     end if
17:     Sort ants by total profit in descending order
18:     for each ant in sorted ants do
19:         Calculate pheromone amount based on ant's profit
20:         for each action in ant.paths do
21:             Update pheromone τ based on action type
22:         end for
23:     end for
24: end for
25: return bestPaths, bestProfits
```

The Ant Colony Optimization algorithm 1 for beautification starts with the initialization of pheromone matrix and best profits and paths to initial values. Then it performs the specified number of runs, and at each run renews the position of the scooters and creates a pack of ants. Then each ant walks through the environment in order to build a solution, and after all the ants have finished their paths, the amount of pheromone is decreased by an evaporation rate. The algorithm selects the best ant, according to the total profit achieved, and, if it is better than the current best, then the

best profits and best paths are updated. The sorting of ants in descending order of their total profits then happens, and pheromone is updated for each ant according to their actions performed and profit made. Refining solutions takes place iteration after iteration and finally returns the best paths and profits that have been discovered by the algorithm.

2.1.2. Time and Space Complexity of ACO Implementation

Time Complexity of ACO is O(k*m*n) and Space Complexity of ACO is O(n^2+m*n) where;

- k is number of iterations
- m is number of ants used in each iteration
- n is number of zones

2.2. CPLEX Implementation

In this chapter, we will examine implementation of CPLEX to solve the optimization problem. The following implementation is also available online [4].

2.2.1. CPLEX Optimization

Formulation of beautification tasks as a mathematical optimization problem in CPLEX, to solve the scooter repositioning problem, involves defining variables for scooter positions, beautificator actions, and time constraints. It uses linear programming or mixed-integer programming. The objective function maximizes total profits while considering operational costs, urban decorum requirements, and time limits. Constraints have been imposed to make it feasible: limitations of the beautificator's actions within the zones, time constraints, and repositioning of scooters. This model gives the exact solution using CPLEX. Since it is an optimal model, optimum efficiency is ensured with full consideration of urban decorum in the beautification process.

The Algorithm 2 shows the pseudo code of problem definition for CPLEX.

Algorithm 2 CPLEX Optimization for Beautification

Input: $zones$, $beautificators$, t_max, $z0$, $nOUT$, $nHOT$, $mBEAU$, $mHOT$, $mMOVE$, $mWAIT$, $\pi BEAU$, πHOT, $\pi MOVE$, $\pi WAIT$
Sets:
$Z = \{1, 2, ..., zones\}$ ▷ Set of zones
$B = \{1, 2, ..., beautificators\}$ ▷ Set of beautificators
$T = \{0, 1, ..., t\_max\}$ ▷ Set of time slots
$N = \{(z, t) | z \in Z, t \in T\}$ ▷ Set of nodes
$A_{BEAU}, A_{HOT}, A_{MOVE}, A_{WAIT}$ ▷ Sets of arcs for each action type
$A = A_{BEAU} \cup A_{HOT} \cup A_{MOVE} \cup A_{WAIT}$ ▷ Set of all arcs
Variables:
$x_{b,a} \in \{0, 1\}$ ▷ Binary variable, 1 if beautificator b takes action a, 0 otherwise
Objective Function: Maximize $\sum_{b \in B} \sum_{a \in A} \pi_a \cdot x_{b,a}$ ▷ Maximize total profit
Constraints:
Starting Zone ▷ Each beautificator starts at their designated zone
Flow Conservation ▷ Flow in = Flow out of each node
Hotspot Capacity ▷ Limit on "bring-to-hotspot" actions
Beautification Limit ▷ Limit on beautification actions
Output:
Solve the optimization problem using CPLEX
Write the objective value and optimal solution ($x_{b,a}$) to output files

The philosophy behind the CPLEX algorithm 2 is the beautification, in other words, the modeling of the problem as a math programming problem. It first identifies, in an orderly fashion, the zones, beautifiers, time slots, nodes, and actions. Binary variables are being used for every beautifier each taking or not taking a certain action while taking maximum over all such an action. Constraints make sure that beautificators start in the designated zones, maintain consistency of flow at every node, and follow the action limits such as hotspot capacity and beautification quotas. The algorithm solves the problem using CPLEX and gives an optimum solution with detailed outputs for beautificator actions and maximum profit.

3. ANALYSIS OF THE RESULTS OF ACO AND CPLEX IMPLEMENTATIONS

In this chapter, we will examine parameters of ACO algorithm to achieve better total profit as result. We will examine each parameters and check the spread of results. Also we will compare ACO results against CPLEX results.

3.1. Effect of Ant Number in ACO

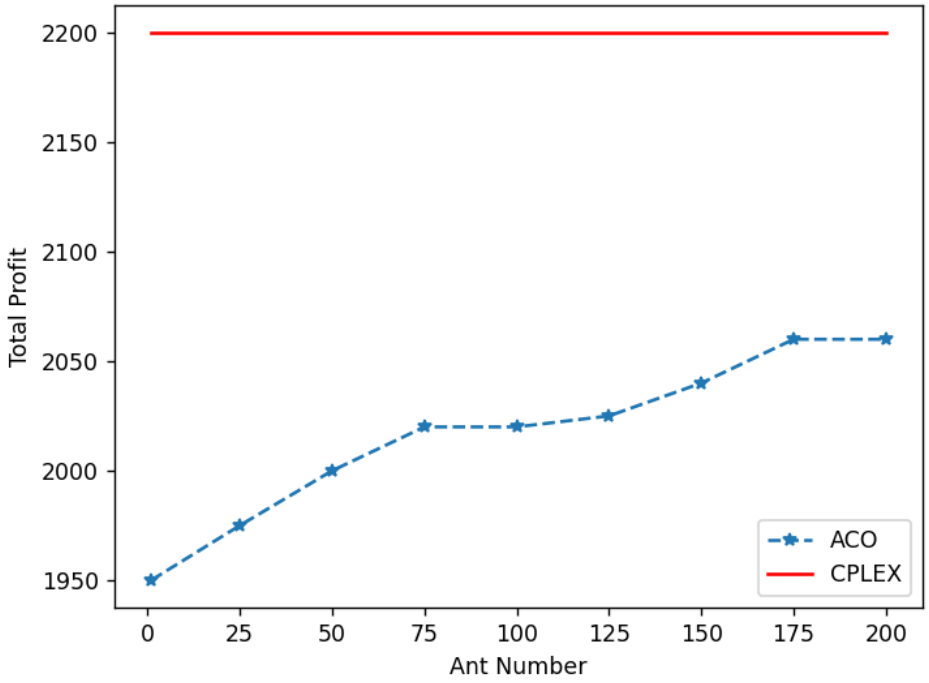

Figure 3.1: Effect Of Changing Ant Number (20 iterations)

In the figure 3.1, we see the effect of changing the number of ants on the result on the current dataset by keeping the iteration number constant at 20. As expected, increasing the number of ants has a positive effect on the result found. But since the number of ants directly affects time complexity, the running time of the program increases. We also added the CPLEX test result performed with the same dataset to the figure for comparison purposes.

3.2. Effect of Iteration Number in ACO

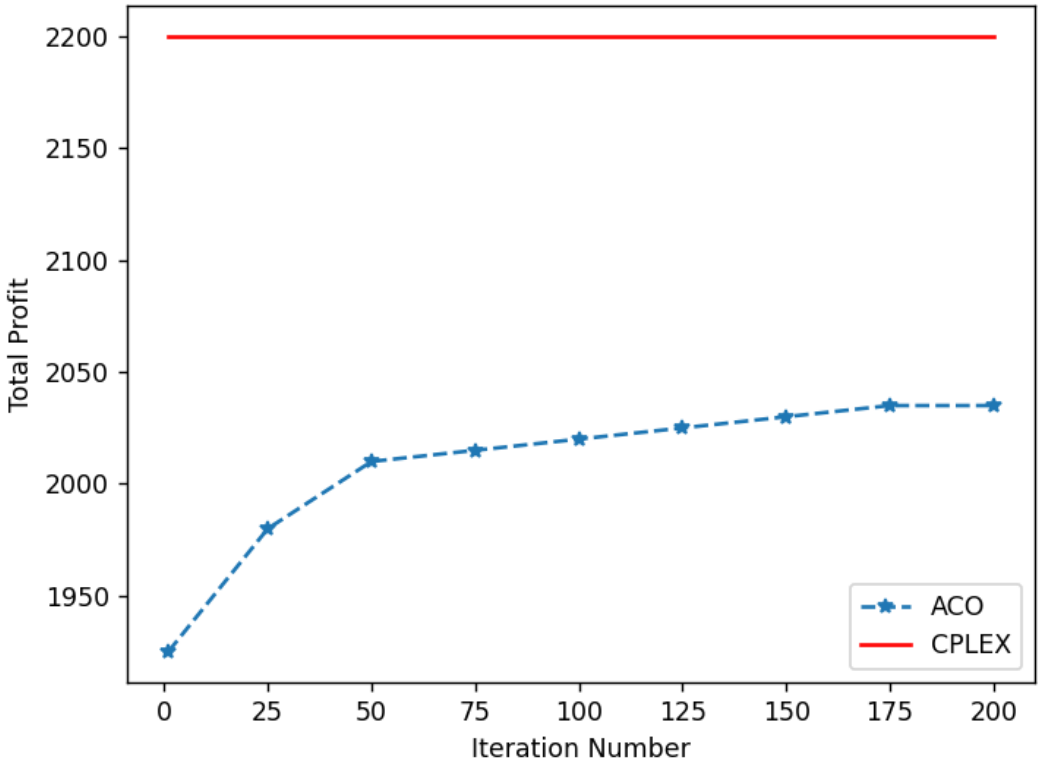

Figure 3.2: Effect Of Changing Iteration Number (20 ants)

In the figure 3.2, we see the effect of changing the number of iterations on the result on the current dataset by keeping the ant number constant at 20. Since iteration number effects time complexity exactly same as ant number, increasing the number of iterations has a positive effect on the result found. But since the number of iterations directly effects time complexity, the running time of the program increases. We also added the CPLEX test result performed with the same dataset to the figure for comparison purposes.

3.3. Effect of Evaporation Rate in ACO

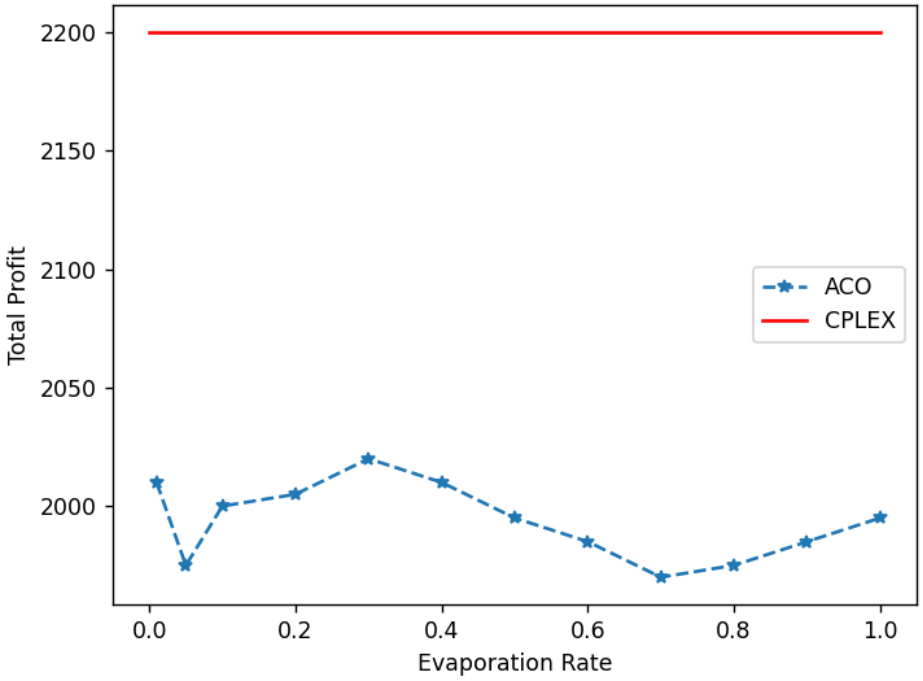

Figure 3.3: Effect Of Changing Evaporation Rate (20 ants, 20 iterations)

As you can see in the figure 3.3, when all parameters were the same, the evaporation rate change did not have a linear effect on the graph. Even if we increased or decreased the evaporation rate, we could not make a definitive judgment because the ants still made a probabilistic choice. The only striking point is that when we set the value to one, the selection possibilities decrease as the pheromone level of the unused edges drops significantly. The edges chosen in the first cycle affect almost the entire result. We also added the CPLEX test result performed with the same dataset to the figure for comparison purposes.

3.4. Execution Times of Algorithms

As a test case, the center area of the city of Izmit were chosen and decomposed into 16 zones. Tests are performed with variable number of beautificators (5 to 8) and scooters (150 to 250).

Table 3.1 shows how many seconds it takes for ACO and CPLEX to reach maximum profit on certain datasets. Since it can take hours or even days for CPLEX to

find the best result, we set an upper limit of 1800 seconds in our tests. ACO tests were performed on 100 ants and 50 iterations. According to the time complexity analysis we conducted in the previous sections, increasing or decreasing these values will, of course, change the running time of the ACO algorithm.

Dataset naming is done according to the following order:"l-b-n_z", where "l" is the location of the dataset, "b" is the number of beautificators and "n_z" is the total number of scooters. For example, for "izmit-5-150":

- "izmit": Location of the dataset
- "5": five beautificators
- "150": a total of one hundred and fifty scooters in all zones

Table 3.1: Total Execution Times

Dataset	ACO Execution Time (sec)	CPLEX Threshold (sec)
izmit-5-150	27.00	1800
izmit-5-175	28.79	1800
izmit-5-200	20.75	1800
izmit-5-225	25.67	1800
izmit-5-250	29.91	1800
izmit-6-150	31.81	1800
izmit-6-175	39.13	1800
izmit-6-200	37.98	1800
izmit-6-225	32.73	1800
izmit-6-250	37.13	1800
izmit-7-150	33.23	1800
izmit-7-175	28.72	1800
izmit-7-200	27.49	1800
izmit-7-225	28.14	1800
izmit-7-250	29.04	1800
izmit-8-150	37.49	1800
izmit-8-175	36.33	1800
izmit-8-200	38.96	1800
izmit-8-225	32.50	1800
izmit-8-250	31.37	1800

3.5. Total Profit Tests

The maximum profit numbers found as a result of ACO and CPLEX are given in table 3.2.

When we compare ACO and CPLEX, we can say that the quality of the result produced by CPLEX decreases as the complexity increases within the given limited time, whereas there is no significant decrease in the result quality of the ACO algorithm and the difference in results with CPLEX decreases clearly and visibly.

Table 3.2: Total Profits

Dataset	ACO Profit	CPLEX Profit	Difference (%)	CPLEX Optimality Gap (%)
izmit-5-150	940	1105	16.13%	8.86%
izmit-5-175	1070	1215	12.69%	4.02%
izmit-5-200	1165	1290	10.18%	3.34%
izmit-5-225	1185	1295	8.87%	7.91%
izmit-5-250	1220	1370	11.58%	6.67%
izmit-6-150	1030	1170	12.73%	6.98%
izmit-6-175	1140	1285	11.95%	11.00%
izmit-6-200	1320	1475	11.09%	2.57%
izmit-6-225	1350	1490	9.85%	6.05%
izmit-6-250	1425	1560	9.04%	5.48%
izmit-7-150	1015	1235	19.55%	1.46%
izmit-7-175	1175	1325	12.00%	8.06%
izmit-7-200	1335	1485	10.63%	11.48%
izmit-7-225	1540	1695	9.58%	3.86%
izmit-7-250	1640	1725	5.05%	5.82%
izmit-8-150	990	1220	20.81%	2.44%
izmit-8-175	1160	1365	16.23%	5.53%
izmit-8-200	1375	1605	15.44%	3.68%
izmit-8-225	1580	1770	11.34%	5.93%
izmit-8-250	1735	1910	9.60%	5.33%

As a result, we can say that it would be more logical to use the ACO algorithm in problems that are dynamic and require continuous updates at short intervals. In our problem, especially since scooter positions can change continuously and we may need to make new decisions dynamically after each change, we can say that the ACO algorithm would be a more logical choice due to its speed, even if it achieves five to ten percent worse results against CPLEX in complex datasets in our current scenario.

4. GUI

The GUI only asks the user for the name of the dataset to be used. Then, the user finds the results with one of the two solvers by using the ”Start ACO” or ”Start CPLEX” buttons. On the left side of the screen, the route for each beautificator can be visually examined.

To get a better visuality, we have prepared special icons for some actions in the overleaf map. These are:

-  : Represents start point in route of a beautificator
-  : Represents beautifying points in route of a beautificator
-  : Represents bring-to-hotspot points in route of a beautificator

To get a better understanding of our GUI, we've created a short demo video that shows about visulization and funcionality about GUI. You can access this video by clicking on the link provided in the reference section below.[5]

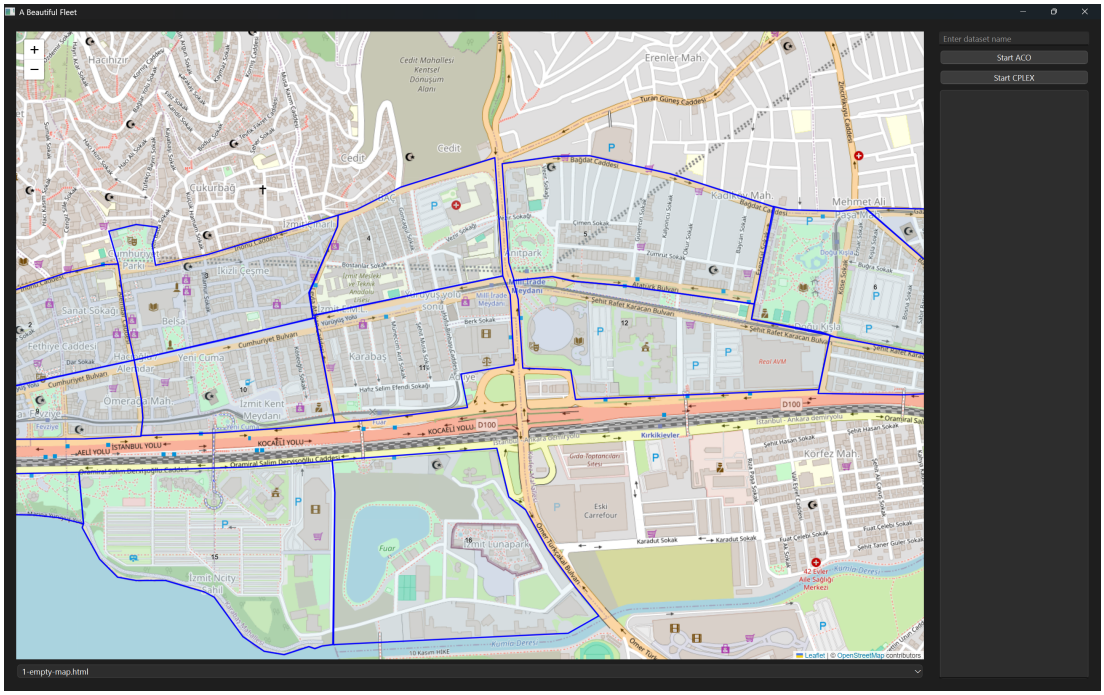

Figure 4.1: GUI

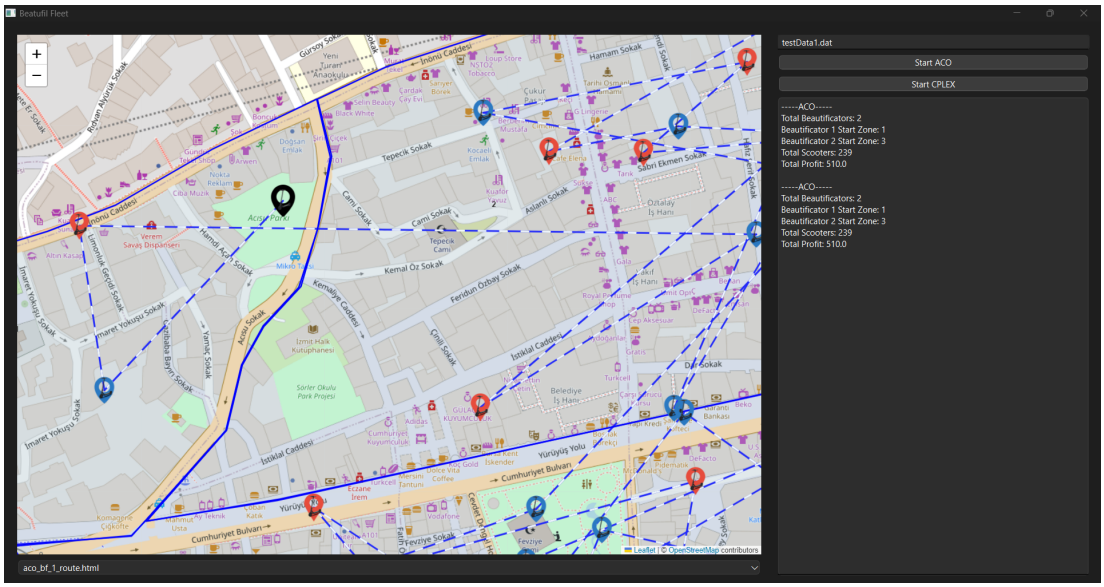

Figure 4.2: GUI with Results

5. CONCLUSIONS

To sum up, this project is directed towards solving the aftermath of the lack of urban decorum in the form of e-scooters unjustly parked in overcrowded cities. The contribution of the article "A Beautiful Fleet: Optimal Repositioning in E-Scooter Sharing Systems for Urban Decorum" is also studied and implemented for addressing decorum issues and for increasing scooting companies' profits. Such measures as minimization of the adverse effects on the urban environment decor while ensuring a sufficient supply of scooters at high traffic areas for maximum users' convenience all add to their effectiveness.

To achieve these goals, we implemented and compared two optimization approaches: Ant Colony Optimization (ACO) and CPLEX. ACO, in the same way is a heuristic algorithm based on the behavior of ants in nature, where ant searches for better paths are conducted through sequential exploration and pheromone deposition. CPLEX, on the other hand, pertains to a class of optimization approach as it gives a definite solution to a given probabilistic scenario using a set of constraints while maximizing profit through mathematical modeling and simulation. The comparison made in terms of those two approaches allows us to draw conclusions on their effectiveness, scalability and potential applications in practical settings.

As a result, we implemented these two technologies for our current problem in our project, compared their results, and obtained a user-friendly interface by creating a GUI.

BIBLIOGRAPHY

[1] S. Carrese, F. D'Andreagiovanni, T. Giacchetti, A. Nardin, and L. Zamberlan, "A beautiful fleet: Optimal repositioning in e-scooter sharing systems for urban decorum," *Transportation Research Procedia*, vol. 52, pp. 581–588, 2021. DOI: 10.1016/j.trpro.2021.01.069.

[2] "IBM ILOG CPLEX Optimization Studio." (2025), [Online]. Available: https://www.ibm.com/products/ilog-cplex-optimization-studio (visited on 01/2025).

[3] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *IEEE Computational Intelligence Magazine*, vol. 1, no. 4, pp. 28–39, 2006. DOI: 10.1109/MCI.2006.329691.

[4] M. Kurtcebe and U. Erdem, *Mehdikurtcebe/a_beautiful_fleet: First release*, version 0.1.0, Jul. 2025. DOI: 10.5281/zenodo.15833729. [Online]. Available: https://doi.org/10.5281/zenodo.15833729.

[5] "Demo Video about GUI." (2025), [Online]. Available: https://youtu.be/WTEDXfJxrIk (visited on 01/2025).