



Review

Toward Greener Smart Cities: A Critical Review of Classic and Machine-Learning-Based Algorithms for Smart Bin Collection

Alice Gatti [†], Enrico Barbierato ^{*,†} and Andrea Pozzi [†]

Department of Mathematics and Physics, Catholic University of the Sacred Heart, via Garzetta 48, 25133 Brescia, Italy; alice.gatti@unicatt.it (A.G.); andrea.pozzi@unicatt.it (A.P.)

* Correspondence: enrico.barbierato@unicatt.it

[†] These authors contributed equally to this work.

Abstract: This study critically reviews the scientific literature regarding machine-learning approaches for optimizing smart bin collection in urban environments. Usually, the problem is modeled within a dynamic graph framework, where each smart bin's changing waste level is represented as a node. Algorithms incorporating Reinforcement Learning (RL), time-series forecasting, and Genetic Algorithms (GA) alongside Graph Neural Networks (GNNs) are analyzed to enhance collection efficiency. While individual methodologies present limitations in computational demand and adaptability, their synergistic application offers a holistic solution. From a theoretical point of view, we expect that the GNN-RL model dynamically adapts to real-time data, the GNN-time series predicts future bin statuses, and the GNN-GA hybrid optimizes network configurations for accurate predictions, collectively enhancing waste management efficiency in smart cities.

Keywords: smart bins; routing; graph neural networks; hybrid models



Citation: Gatti, A.; Barbierato, E.; Pozzi, A. Toward Greener Smart Cities: A Critical Review of Classic and Machine-Learning-Based Algorithms for Smart Bin Collection. *Electronics* **2024**, *13*, 836. <https://doi.org/10.3390/electronics13050836>

Academic Editors: Jesús Ángel Román Gallego, María-Luisa Pérez-Delgado, María Concepción Vega Hernández, Alfonso Jose Lopez Rivero and Daniel Hernández De la Iglesia

Received: 9 January 2024
Revised: 13 February 2024
Accepted: 16 February 2024
Published: 21 February 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

The role of AI in the domain of smart cities [1–4], especially in garbage collection, has recently emerged in the landscape of urban development and sustainable practices. In the contemporary landscape, the exploration of intelligent technologies—encompassing smart bins, robotic systems, predictive modeling, and optimized routing algorithms—and their pivotal role in optimizing waste collection processes has become essential. Thinking about city administration and garbage management, collecting, removing, and re-utilizing the produced garbage is an arduous task. The rapid accumulation of garbage necessitates a well-organized and efficient system, with a focus on minimizing the environmental impact wherever possible. The conventional garbage collection operation, with a rigid routine that encompasses the continuous reiteration of the combination of manual collection and removal with segregation and recycling tasks, can be both inefficient and resource-consuming. Consequently, integrating advanced technologies such as AI and smart waste management solutions is paramount. Pivotal components contributing to this transformation are smart bins, smart routing, smart segregation, and smart prediction. Hence, a novel approach to waste management is a characteristic of smart cities.

Smart bins, also called intelligent dumpsters, are the starting point towards smartness in smart cities from a garbage collection perspective. Their functionality varies between the alternative proposed implementations, but the core principle lies in the automatic detection of fill level and smart notifications. The main feature of the bins lies in their integration with both Internet of Things (IoT) sensors and Artificial Intelligence (AI) software. Despite various practical alternatives that have been proposed, the core concept remains unchanged.

Collecting garbage from smart bins rationally and efficiently is a critical aspect of modern waste management, significantly contributing to environmental sustainability, operational efficiency, and cost reduction. The evolution of garbage collection methodologies,

especially with the integration of technology and smart algorithms, showcases an interesting journey from traditional methods to advanced, AI-driven approaches. The initial phase in the evolution of garbage collection routes relied heavily on manual planning and simple heuristics. These methods, while straightforward, often led to suboptimal routes, increased fuel consumption, and excessive time spent on collection. The need for more efficient systems led to the adoption of classical algorithms such as the Traveling Salesman Problem (TSP) and the Vehicle Routing Problem (VRP). TSP, which aims to find the shortest possible route that visits each location once and returns to the origin point, and VRP, which extends this concept to multiple vehicles, were foundational in developing more efficient garbage collection routes. However, these models often struggled with real-world complexities such as varying bin capacities, traffic conditions, and dynamic scheduling. The integration of Geographical Information Systems (GIS) into route planning marked a significant advancement. GIS allowed for the incorporation of real-time geographic data, traffic patterns, and road networks, enabling more realistic and adaptable route planning. This integration significantly enhanced the efficiency of routes but still relied heavily on predefined algorithms and lacked real-time adaptability. Algorithms based on AI, such as Artificial Neural Networks (ANNs) and Genetic Algorithms (GAs), began to address the limitations of classical methods. These AI-based models can learn from past data, adapt to changing scenarios, and even predict future waste generation patterns, leading to more dynamic and efficient route planning. ANNs, for instance, can analyze vast amounts of data, learning from various factors such as historical waste levels, seasonal variations, and public events that might influence waste production. This analysis allows for the prediction of waste generation patterns and the optimization of collection routes accordingly. GAs, inspired by the process of natural selection, provide another robust method for route optimization. These algorithms generate multiple potential solutions and iteratively refine them, mimicking the evolutionary process to arrive at the most efficient route. Smart bins equipped with sensors can relay real-time data on their fill levels, allowing for dynamic route planning based on actual waste levels rather than fixed schedules. This integration not only optimizes routes but also ensures that bins are collected at the right time, reducing overflow and associated environmental impacts.

The focus of this work is anchored in a comprehensive literature review, scrutinizing a spectrum of routing algorithms pivotal for enhancing waste collection efficiency within an urban framework, equipped with an interconnected array of intelligent waste bins. This investigation is not merely descriptive but is also analytical, dissecting the operational framework and efficacy of these algorithms in the context of smart urban waste management. Specifically, the contributions of this study are as follows:

- An in-depth exploration and critical analysis of traditional algorithms, primarily focusing on their application and performance in static graph environments, laying a foundation for understanding their suitability and limitations in the context of waste management routing;
- A thorough examination and assessment of Machine-Learning (ML) methodologies tailored for dynamic graph scenarios, effectively encapsulating the complexities and real-time dynamics inherent in urban waste collection systems;
- A comparative critique through the lens of a hypothetical case study, methodically evaluating the efficacy and applicability of both standalone traditional and ML-based routing strategies, followed by an examination of the synergistic potential of hybrid ML models in addressing the complexity of waste collection optimization.

This work not only reviews the current landscape of routing algorithms in smart city waste management but also paves the way for future innovations and methodological enhancements in this domain. It also concludes by supporting the adoption of hybrid ML models as a transformative strategy for optimizing the smart bin collection process, paving the way for the empirical realization of a real case study set against the backdrop of a smart city in Italy.

The intended readership of this article encompasses a diverse spectrum of stakeholders, primarily from academic communities and industry sectors associated with environmental engineering, urban development, and the burgeoning domain of smart city technologies. This includes, but is not limited to, academicians and industry experts specializing in AI and ML, with a particular emphasis on those engaged in the application and advancement of Graph Neural Networks (GNNs), time-series forecast models, GAs, and sophisticated optimization algorithms. Furthermore, the insights emerging from this study hold substantial relevance for municipal waste management entities and urban policy strategists looking for cutting-edge and sustainable solutions to streamline resource management and operational efficiency in urban settings.

The structure of the work is as follows. After the introductory part, Section 2 reviews the related work. Section 3 delineates an array of smart city paradigms. An exploration of predominant routing algorithms, encompassing both conventional approaches and methodologies rooted in ML, is articulated in Section 4. Evaluative metrics form the basis of Section 5, wherein a critical assessment of the performance metrics associated with each model is undertaken. Section 6 engages in a comparative analysis, offering a case study on a dynamic graph while elucidating the advantages and limitations inherent in each paradigm. Finally, Section 7 encapsulates the findings and proposes trajectories for future research endeavors.

Figure 1 presents a visual representation of the work’s structure.

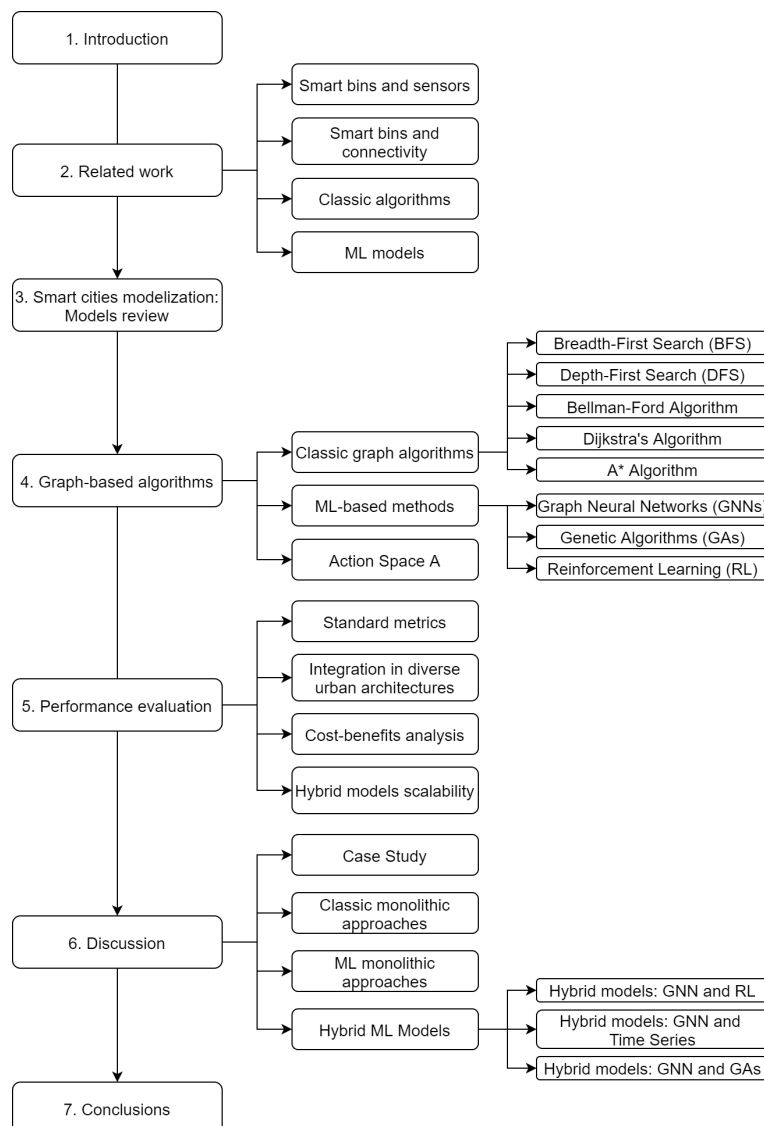


Figure 1. Visual representation of the work’s structure.

2. Related Work

2.1. Smart Bins and Sensors

Automatic monitoring can be accomplished through the use of sensors mounted on the bins, thus fulfilling the need for a way to accurately measure the state of each container. The detection can be visual, where thresholds on fullness are registered, or practical, where the weight of the content is constantly measured. The available sensors are connected to a network and can communicate their status on a real-time basis. Hisham et al. [5] used an ultrasonic sensor mounted on the cover of the dumpster to monitor fullness through the measurement of distance propagation of emitted sound waves: the elapsed time between emission and reception of the waves returns the distance between the amount of accumulated waste and cover of the bin, and thus the fullness. Alternatively, Catarinucci et al. [6] implemented a system based on a weight sensor installed under a double-bottom inside of their bins.

Internet of Things (IoT) consists of the deployment of wireless communication systems, spanning from close-range NFC to the extended reach of Sigfox. Therefore, according to the desired characteristics of the infrastructure under development, different wireless IoT systems are available, as shown in Table 1. A trade-off between the cost of implementation, the required distance of transmission, desired maximum bitrate, and maximum frequency band is needed.

Table 1. Characteristics of different available wireless IoT systems.

IoT Wireless System	Distance	Max Bitrate	Main Freq. Bands
NFC	<0.1 m	848 kbps	13.56 MHz
RFID	1–10 m	100 kbps to 4 Mbps	125 kHz, 134 kHz, 13.56 MHz, 2.4 GHz, 5.8 GHz
Bluetooth	10–100 m	3 Mbps	2.4 GHz
Zigbee	10–100 m	250 kbps	2.4 GHz
Thread	10–100 m	250 kbps	2.4 GHz
Wi-Fi	10–100 m	867 Mbps	2.4 GHz, 5 GHz
WSN	10 m–1 km	kbps to Mbps	433 MHz, 868 MHz, 915 MHz, 2.4 GHz, 5.8 GHz
eMTC	1–10 km	1 Mbps	800–900 MHz
NB-IoT	1–10 km	50 kbps	800–900 MHz
EC-GSM-IoT	1–10 km	240 kbps	900–1800 MHz
LoRaWAN	5–10 km	50 kbps	868 MHz
Sigfox	10–50 km	600 bps	868 MHz

Likotiko et al. [7] suggest the deployment of a smart bin integrated with an Arduino ultrasonic sensor powered by a battery, and configured to establish connectivity through a Wi-Fi Shield equipped with GSM/GPRS capabilities. Chowdhury et al. [8] underline the effectiveness of RFID sensors in the smart waste management field, as they can (i) adapt to resist both substances and conditions and are (ii) simple, (iii) not expensive, and (iv) versatile.

Kumari et al. [9] suggest that it is possible to integrate smart bins with small solar panels. Sigongan et al. [10] deployed a solar-panel-based system. Their smart bin is equipped with both a battery and a Solar Panel Mono-crystalline 200-watt with MC4 Photovoltaic Connector. Solar energy is converted through a 12V DC to a 220–230V AC Car Home Solar Power Inverter with Buzzer. Notifications are sent via SMS.

2.2. Smart Bins and Connectivity

A network of bins enables live communication, as the bins have a strong connection and the capability to send messages. In a cloud-based waste monitoring system each bin sends the registered data to the headquarters. Once the data are received through the system of smart notification, they are analyzed and evaluated through AI and ML models.

The prospect of devising integrated systems boasting inherent intelligence has become a reality due to recent advancements in microcontroller and microcomputer technologies, which are equipped with operating systems. Furthermore, with the recent prominence of the era of edge computing, systems can autonomously process information locally. Simultaneously, cloud-based systems, accessible over the Internet, offer an alternative paradigm by offloading computational tasks to connected machines. As a consequence, the effective redistribution of the computational load and a new cost-effective strategy led to the efficiency of information processing and management across various applications. Ghahramani et al. [11] concur on the advantage of using a microcontroller-based platform for the scope of smart routing.

In addition to cloud connectivity, smart bins often interface with external systems and applications through Application Programming Interfaces (APIs). These APIs serve as bridges that facilitate communication between the smart bin and other components of the broader smart city infrastructure. For instance, by exposing relevant data and functionalities through well-defined APIs, smart bins can seamlessly interact with municipal waste management systems, transportation networks, or even mobile applications. This interconnectedness not only enhances the overall efficiency of waste management but also fosters the development of a comprehensive and integrated smart city ecosystem. As different levels and thresholds of fullness are available, an API is used to query the state of each component connected to the network. Each bin outputs its state (e.g., full or not) and the action to take (e.g., leave it as is or empty it). Once a list of bins is created, it is input to sorting algorithms. Each bin has a specific IP address and geographic coordinates, as it is crucial to optimize the routing of the vehicles for garbage collection.

MQTT, or Message Queuing Telemetry Transport, excels in supporting efficient data exchange between devices operating in resource-limited environments, particularly within the burgeoning Internet of Things (IoT) landscape. Its publish–subscribe architecture streamlines communication, enabling devices to publish messages to specified topics while others subscribe to receive those messages. MQTT's lightweight design, characterized by its low overhead, renders it an ideal choice for applications demanding minimal bandwidth or processing power. It facilitates real-time data exchange, ensuring seamless communication between IoT devices and backend systems, enabling scalable and efficient data-driven operations. Nagesh et al. [12] analyze the integration of MQTT with Map APIs within a Smart Garbage Management scenario. The system described involves deploying sensor-equipped garbage bins with low-cost embedded communication devices to monitor waste levels. Each bin has a unique identifier, allowing easy tracking via a web interface and smartphone app. Integration with Google Maps via APIs enables real-time tracking. When a bin reaches a preset threshold, its status is transmitted via MQTT messages, allowing municipal authorities to take immediate action. Additional features include bin tracking, identifying the nearest bin, and remote garbage level indication. Python Big Data Analytics is exploited to visualize the waste collection route heuristics of smart bin data in [13], where the authors present a comprehensive set of waste management APIs, leveraging the power of data science libraries to transform waste management data into actionable insights for various stakeholders. These APIs effectively process and visualize waste management

data, providing a wealth of information for council administrators, waste management contractors, and the general public. They offer visualization options, including maps of smart bin locations and fullness levels, bar charts of fullness frequency, and line charts of fullness over time. Additionally, the APIs generate route optimization solutions for waste collection trucks, reducing operational costs. The code developed for these APIs exhibits modularity and extensibility, making it readily transferable to the analyses of data from other smart bin systems and other local government areas.

2.3. Classic Algorithms

Norhafezah et al. [14] utilize Dijkstra's algorithm to simulate and optimize municipal solid waste collection routes. It aims to address inefficiencies caused by unsystematic planning and multiple collection points by shortening travel distances, thereby enhancing cost-effectiveness and reducing the time and costs associated with waste management.

Priyadarshi et al. [15] discuss optimized waste collection strategies, mainly focusing on dynamic routing in resource-constrained societies. They highlight the importance of real-time data in calculating optimal routes and introduce models that consider various real-time considerations such as waste levels in bins and the location of collection vehicles. The models aim to maximize waste collection while minimizing travel distance, utilizing a mixed-integer linear programming approach for the solution.

Barth et al. [16] present the main highlights from the current scientific literature to optimize solid waste collection. The reviewed approaches encompass IoT, web GIS systems, tactical planning, discrete event simulation, and stochastic optimization.

2.4. ML Models

Liang et al. [17] offers a concise review of modern solutions, including various meta-heuristic algorithms such as ant colony optimization, simulated annealing, and GAs, among others. The study also explores Geographic Information Systems (GIS) as a tool for WCRP. It concludes with a performance analysis using real-world benchmarks and outlines potential areas for future research in the field of waste collection routing. Cha et al. [18] present a new approach to improve waste generation rate (WGR) prediction using hybrid ML models. Specifically, two primary ML algorithms were used: Artificial Neural Network (multi-layer perceptron) (MLP) and Support Vector Machine Regression (SVMR). Categorical Principal Component Analysis (CATPCA) was applied to these algorithms. Furthermore, four predictive models were developed: ANN (MLP), SVMR, CATPCA—ANN (MLP), and CATPCA—SVMR. According to the authors, the CATPCA—ANN (MLP) model showed improvements over the ANN (MLP) model in certain statistical metrics. The CATPCA—SVMR model significantly outperformed the SVMR model across all statistical metrics. Specifically, the best performance was observed in the CATPCA—SVMR model, and the mean Daily Waste Generation Rate (DWGR) was very close between the observed values and the predicted values (1161.52 kg/m²) for the CATPCA—SVMR model. As a result, the use of CATPCA allows for the effective utilization of ML algorithms that are typically less effective with categorical variables.

Lilhore et al. [19] discuss a study on improving waste management in the context of increasing industrialization and smart city development. The work emphasizes the importance of waste collection, classification, and planning, particularly for recycling processes that aim to minimize pollution and promote sustainability. The research introduces a smart waste classification system utilizing a hybrid model that combines Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) networks. The model incorporates transfer learning, leveraging the ImageNet database to improve its ability to classify and predict waste categories into recyclable and organic. It addresses overfitting and data sampling issues through an improved data augmentation process. The authors support the evidence that the hybrid model showed superior performance, achieving the highest precision compared to other models. It also demonstrated the best optimization and

accuracy with the least modeling loss during training, validation, and testing, attributed to the Adaptive Moment Estimation (AME) optimization algorithm.

Zhang et al. [20] focus on the challenges and advancements in using ML to improve waste management processes such as collection, sorting, recycling, and disposal. The paper introduces an optimized hybrid DL model specifically designed for waste classification. The model utilizes a multi-step approach including data collection and preprocessing, feature extraction using a Convolutional Neural Network (CNN) (specifically AlexNet), waste prediction using a Deep Belief Network (DBN), and finally hyperparameter optimization with Optuna to fine-tune the model.

Arunkumar et al. [21] review the development of an innovative waste management system for smart cities, leveraging the capabilities of the Internet of Things (IoT) and ML. In particular, the study utilizes a hybrid approach called Decision Tree with Extreme Learning Machine (DT-ELM) to analyze city waste data. Unlike single classifiers that require time-consuming iterative training, the proposed hybrid model is more efficient. Decision trees effectively classify based on selected features, and additional weights are calculated to enhance classification accuracy. Entropy theory is used to map the decision tree to ELM, aiming for accurate prediction results.

3. Smart City Modelization: Model Review

Modeling a smart city encompasses a diverse range of perspectives and methodologies, often employing a combination of technological tools and frameworks to capture the intricacies of urban life. While graphs hold immense value for visualizing and analyzing relationships within an urban center, there exist various alternative approaches to modeling a smart city.

Numerous alternatives exist for representing and modeling the road network of a city: (i) raster representation, (ii) vector representation, (iii) topology-based representation, (iv) hierarchical representation, (v) graphical models, (vi) 3D models, and (vii) graphs (static or dynamic).

In raster representation, the entire city is divided into cells, each of which represents a portion of the whole urban network. The presence or absence of road segments in a cell is indicated by an assigned value in pixels. In the obtained rendering, roads are depicted as lines of pixels. Depending on the size of each unit of the produced matrix-like structure, a certain level of detail in the representation is reached. The use of smaller cells brings the advantage of higher resolution and precision, enabling a more detailed depiction of the urban network, but this necessitates increased computational resources and memory allocation. Walter et al. [22] explain that shortest path analysis in raster maps can be done in four steps: (i) during the pre-processing phase the input map is converted into binary format, where 1 denotes a traversable trait of the map (i.e., a road) and 0 an inaccessible area of the map; (ii) the map is skeletonized (i.e., a process where a complex shape is reduced to a minimal form that represents its general structure; specifically, it involves converting the shape into a 'skeleton' by stripping away the majority of the shape's body while preserving its fundamental topology) and converted into a graph; (iii) the shortest path algorithm is applied; (iv) smoothing is performed and the resulting path is generated via visibility calculations. Taillandier et al. [23] acknowledge that the prevalence of roadmap representation via raster models is attributed to their simplicity of implementation and their demand for fewer data for the modeling. Raster representation is well suited for surface modeling or grid-based operations. The major drawback is that width, direction, and other geometric details may be lost.

Compared to raster representation, a higher degree of precision is achievable via vector representation. In the latter, each road of the city (i.e., a segment of the urban network) is rendered as a line with associated attributes related to its characteristics (e.g., length, width, direction). Vector representation excels in handling dynamic attributes such as real-time traffic data, road conditions, and speed limits, thus enhancing the ability to model and analyze the changing nature of the road network. Consequently, this leads

to the exploitation of connectivity, exploration of relationships, discovery of patterns, and holistic analysis of the available road network. Chen et al. [24] acknowledge the importance of vector representation for city road networks, as it enhances the discovery of the consequences of the combined effects of various origins on the network of the city, and propose a road vectorization mapping network framework. Chenjing et al. [25] achieve road vectorization starting from historical maps input as raster images through the following steps: (i) the map is converted into binary format and skeletonized; (ii) each trait of the skeletons is approximated using a set of connected straight segments; (iii) each segment is associated with symbols corresponding to the state of the road.

Topology-based representation exploits connectivity and relationships within the streets of the city, providing insights for network analysis works. In this representation, spatial coordinates and attributes take a secondary role, as the primary goal is to acquire knowledge about the structure and behavior of the road network. This type of representation often results in more efficient storage and processing compared to detailed geometric representations, especially for large and complex road networks. An advantage in the usage of the topology-based approach for road representation is that it can accommodate dynamic changes in the network, making it suitable for real-time applications, and particularly suitable for scenarios where the road network is subject to frequent updates or where real-time traffic information needs to be considered. Jiang et al. [26] observed that the topological representation of streets offers a better analytical means for the geographical knowledge of streets, as vehicle flows are correlated to the morphological properties of streets. Spadon et al. [27] developed a description of cities through the usage of vectors of topological features derived from the roads, modeling cities as a complex network and ultimately identifying groups of cities by extracting features from their topology.

Hierarchical representation networks are organized based on an importance criterion, as roads are considered with a different significance level according to factors such as capacity, speed, or a specific computed metric. This representation is valuable and essential in modeling scenarios where traffic flow and the subsequent speed of occurrence are critical factors as it enables the study of traffic volume and provides insights into such analyses. Song et al. [28] handled efficient routing by using a hierarchical model based on community structure on road networks.

Graphical models represent each road intersection as a node, and the relationships between each couple of nodes (i.e., the arc) can be modeled based on historical data, real-time traffic conditions, or other relevant time-dependent factors that reflect their probabilistic dependencies. This approach is a suitable means to capture the probabilistic relationships and variable interactions within the complex system of the network of a city's streets. Graphical models are a flexible means that allow for the representation of dynamic systems where conditions change over time through graphical structures. Examples of graphical models are Bayesian networks and Markov models. Jeong et al. [29] used a Bayesian network to develop a risk-adaptive roadmap for autonomous vehicles, while Alterovitz et al. [30] embraced Markov models for a roadmap maximizing the probability of avoiding collisions.

Three-dimensional models are a valuable method to represent cities by incorporating 3D structures, allowing for capturing both the horizontal layout of streets and buildings and the vertical dimension, thus including variations in terrain and elevation. This approach can be useful for simulating traffic flow in complex urban environments, including features such as hills, valleys, bridges, and tunnels, consequently allowing for a more accurate visualization of the city landscape and the spatial relationships between different elements. Modelling the urban network to include elevation data enhances the precision of the simulation by considering the impact of factors such as slope or surrounding changes on vehicle movement. Vitalis et al. [31] acknowledge the importance of 3D city models in applications such as evacuation scenarios and energy consumption estimation as both geometry and semantic information is considered.

Graph theory is fundamental for modeling transportation networks and utilities within the city. Nodes can represent intersections, buildings, or people, while edges can represent roads, pathways, or relationships. The following summarizes the basic formalization of this data structure.

Graphs excel in capturing the relationships and networks that define a smart city's infrastructure, from its roads and utilities to its communication networks. This intuitive representation allows for easy expansion and modification, adapting to the ever-changing landscape of urban environments. Specialized algorithms from the scientific literature enable the efficient calculation of shortest paths, network flows, and connectivity, critical aspects of transportation planning, logistics, and infrastructure management. Moreover, the scalability of modern graph databases and algorithms makes them well-suited for handling the vast datasets generated by a smart city.

However, the representation of spatial or hierarchical data may not be as straightforward as with other models, such as GIS or tree structures, potentially introducing unnecessary complexity. Certain graph algorithms, especially for large and dense graphs typical in smart city data, can be computationally intensive, posing performance challenges. The learning curve associated with graph databases and graph theory can be steep, potentially deterring teams or individuals without specialized knowledge. Visualizing and interpreting large graphs can also become increasingly difficult as their size and complexity grow, hindering effective insights and decision-making. Additionally, while graphs work very well at handling dynamic changes, real-time management of events such as temporary road closures or network topology shifts can be demanding, often requiring sophisticated algorithms or frequent graph updates.

Table 2 summarizes the main characteristics of the proposed road representation approaches, highlighting both the benefits and limitations of each.

Table 2. Conversion of a roadmap using different representation approaches.

Approach	Description of Roads	Benefits	Limitations
Raster	Pixel-based grids of cells	Simple; intuitive; little data demanding	Limited scalability, detail, and precision; Loss of geometric details
Vector	Vector geometry	More precise; efficient storage and processing; excellent handling of dynamic data	Computationally expensive for large networks
Topology-based	Connectivity; relationships	Exploits structure and behaviour	Requires accurate network dataset; geometry and attributes are secondary
Hierarchical	Levels of importance	Considers different significance levels; flexible; scalable	Difficult definition of hierarchical levels; requires accurate design; higher storage demand
Graphical Models	Probabilistic relationships	Considers time-dependent factors; suitable for simulations	Computationally complex and expensive; data-intensive
3D Models	Three-dimensional space	Accurate and realistic representation on more dimensions; suitable for simulation and planning; includes vertical data	Higher data, data storage, and computation requirements
Graphs	Nodes and edges	Complexity in representation; computational overhead; difficulty in visualizing large graphs	Intuitive representation of relationships and networks; flexibility; route and network optimization

4. Graph-Based Algorithms

This section reviews the main algorithms used in graph theory for searching and finding the shortest paths between nodes in a graph, grouped into classic and ML ap-

proaches. Firstly, the classic algorithms, such as BFS, DFS, Bellman–Ford, A*, and Dijkstra are reviewed, comparing the computational complexity of each. Secondly, the section compares algorithms based on ML, such as GNNs, GAs, and Reinforcement Learning (RL) adapted to work on graph theory.

4.1. Classic Graph Algorithms

Let $G = (V, E)$ be a directed graph with a cost function $w : E \rightarrow \mathbb{R}$. A (minimum-cost) path between a pair of vertices, x and y , connected in G is a path, p_{xy} , that has a cost less than or equal to that of any other path, p'_{xy} , between the same vertices: $w(p_{xy}) = \min_{p'_{xy} \in G} w(p'_{xy})$. An edge, $(u, v) \in E$, belongs to a minimum path from s to v if and only if (i) u is reachable from s and (ii) $d_{su} + w(u, v) = d_{sv}$.

As a mixed graph has both directed and undirected edges, it is suitable for representing a road map, as schematized in Table 3. In the context of a graph, G , representing a city road map, one can envisage the trajectory of a council worker collecting waste from dumpsters by traversing a path within G . As a result, each street is conceptualized as a node, while intersections serve as the connecting edges. This graph-based representation captures the spatial relationships and connectivity between different segments of the city, providing a structured framework for analyzing and optimizing transportation networks. Graphs allow urban planners and researchers to model various attributes of streets, such as traffic flow, accessibility, and connectivity, facilitating data-driven decision-making processes. Graph-based representations of city streets offer a versatile platform for the application of algorithms and analytics. Graph algorithms, such as Dijkstra’s algorithm or the A* search algorithm, can be employed to find optimal routes, identify critical transportation corridors, and optimize traffic flow. Moreover, the integration of additional data layers, such as land-use information or demographic data, enables a more holistic understanding of the urban landscape. This graph-centric approach not only aids in the planning of transportation infrastructure but also contributes to the development of smart city initiatives, supporting sustainable and efficient urban development.

Table 3. Conversion of a roadmap to a graph.

Street Component	Graph Representation
Intersection between 2+ roads	Vertex
Ending of a dead-end street	Vertex
Road segment without intersections	Edge
Two-way street	Undirected edge
One-way street	Directed edge
Length of the street	Weight of the edge
Route between 2+ locations	Subgraph of the given graph

4.1.1. Breadth-First Search (BFS)

Breadth-First Search (BFS) and Depth-First Search (DFS) are two fundamental algorithms used for graph traversal and searching [32]. The Breadth-First Search (BFS) algorithm is one of the simplest algorithms for graph search.

Given a graph, $G = (V, E)$, and a starting vertex, s , BFS systematically explores the edges of G to find every vertex reachable from s . It calculates the distance from s to every other reachable vertex. Its main feature is that it expands the frontier between visited and unvisited vertices uniformly: it visits all vertices at distance k from s before visiting any vertex at distance $k + 1$.

The overall computational cost of BFS is $O(V + E)$. The method is depicted in Algorithm 1.

Algorithm 1 Breadth-First Search

```

1: function BFS( $G, s$ )
2:   for each vertex  $u \in G.V - \{s\}$  do
3:      $u.status \leftarrow \text{notVisited}$ 
4:      $u.d \leftarrow \infty$ 
5:      $u.p \leftarrow \text{None}$ 
6:   end for
7:    $s.status \leftarrow \text{visited}$ 
8:    $s.d \leftarrow 0$ 
9:    $s.p \leftarrow \text{None}$ 
10:   $Q \leftarrow \emptyset$ 
11:  ENQUEUE( $Q, s$ )
12:  while  $Q \neq \emptyset$  do
13:     $u \leftarrow \text{DEQUEUE}(Q)$ 
14:    for each vertex  $v \in G.Adj[u]$  do
15:      if  $v.status == \text{notVisited}$  then
16:         $v.status \leftarrow \text{visited}$ 
17:         $v.d \leftarrow u.d + 1$ 
18:         $v.p \leftarrow u$ 
19:        ENQUEUE( $Q, v$ )
20:      end if
21:    end for
22:     $u.status \leftarrow \text{closed}$ 
23:  end while
24: end function

```

4.1.2. Depth-First Search (DFS)

The strategy of Depth-First Search (DFS) is based on exploring as deeply as possible in the graph. It traverses the edges from the most recent vertex, v , and its predecessors are temporarily left with unexplored outgoing edges. When all edges of v have been explored, the search backs up to explore other outgoing edges from the vertex from which v was discovered. This process continues until all vertices reachable from the source vertex have been discovered. If there are remaining unexplored vertices, one of them is selected as the new source, and the search is repeated from this new source. The algorithm repeats the entire process until every vertex has been discovered. The overall computational cost of BFS is $O(V + E)$. The method is depicted in Algorithm 2.

4.1.3. Bellman–Ford Algorithm

The Bellman–Ford algorithm [33–35] solves single-source shortest path problems in a general case where the edge weights may also be negative. Given a directed weighted graph, $G = (V, E)$, with a source, s , and a weight function, $w : E \rightarrow \mathbb{R}$, the Bellman–Ford algorithm returns a Boolean value indicating whether there is or is not a path with a negative weight cycle. If there is a cycle of that kind, then the algorithm indicates that there is no solution. In this case, the algorithm finds and returns the minimum paths and their respective costs. The method is depicted in Algorithm 3.

4.1.4. Dijkstra’s Algorithm

Dijkstra’s algorithm (see, for example, [36–38]) solves the single-source shortest-paths problem on a directed and weighted graph, $G = (V, E)$, in the case where all vertices are non-negative. It is assumed that $w(u, v) \geq 0$ for every edge $(u, v) \in E$. A set, S , of vertices is maintained, whose total costs of minimum paths from the source s have already been determined. The algorithm repeatedly chooses a vertex, $u \in V - S$, with the minimum estimated path, adds that vertex to S , and relaxes all edges outgoing from u . The method is depicted in Algorithm 4.

Algorithm 2 Depth-First Search

```

1: function DFS( $G$ )
2:   for each vertex  $u \in G.V$  do
3:      $u.status \leftarrow$  NOTVISITED
4:      $u.p \leftarrow$  None
5:   end for
6:    $time \leftarrow 0$ 
7:   for each vertex  $u \in G.V$  do
8:     if  $u.status =$  NOTVISITED then
9:       DFS-VISIT( $G, u$ )
10:    end if
11:  end for
12: end function

```

DFS-VISIT Procedure

```

1: function DFS-VISIT( $G, u$ )
2:    $time \leftarrow time + 1$ 
3:    $u.d \leftarrow time$ 
4:    $u.status \leftarrow$  VISITED
5:   for each vertex  $v \in G.Adj[u]$  do
6:     if  $v.status =$  NOTVISITED then
7:        $v.p \leftarrow u$ 
8:       DFS-VISIT( $G, v$ )
9:     end if
10:  end for
11:   $u.status \leftarrow$  CLOSED
12:   $time \leftarrow time + 1$ 
13:   $u.f \leftarrow time$ 
14: end function

```

Algorithm 3 Bellman–Ford Algorithm**Require:** Graph $G(V, E)$, source vertex s **Ensure:** Shortest distances from s to all other vertices, or detection of negative weight cycles

```

1: Initialize distance to all vertices as infinite and distance to source  $s$  as 0
2: for each vertex  $v \in V - 1$  do
3:   for each edge  $(u, v) \in E$  do
4:      $dist[v] \leftarrow \min(dist[v], dist[u] + weight(u, v))$ 
5:   end for
6: end for
7: for each edge  $(u, v) \in E$  do
8:   if  $dist[v] > dist[u] + weight(u, v)$  then
9:     Report negative weight cycle
10:  end if
11: end for

```

Algorithm 4 Dijkstra’s Algorithm

Require: Graph $G(V, E)$ with non-negative weights, source vertex s

Ensure: Shortest distances from s to all other vertices

- 1: Create vertex set Q
 - 2: Initialize distances: $dist[v] \leftarrow \infty$ for all $v \in V$, except $dist[s] \leftarrow 0$
 - 3: Add all vertices to Q
 - 4: **while** Q is not empty **do**
 - 5: $u \leftarrow$ vertex in Q with min $dist[u]$
 - 6: Remove u from Q
 - 7: **for** each neighbor v of u still in Q **do**
 - 8: $alt \leftarrow dist[u] + length(u, v)$
 - 9: **if** $alt < dist[v]$ **then**
 - 10: $dist[v] \leftarrow alt$
 - 11: **end if**
 - 12: **end for**
 - 13: **end while**
-

4.1.5. A* Algorithm

The A* algorithm (see, for example, [39,40]) is a search algorithm that finds a path from a given source node to a given goal node. It ranks each node based on an estimate of the best path passing through that node. It is complete, optimal, and efficient. In many cases, it is the best solution and can be seen as an extension of Dijkstra’s algorithm; it is also classified as a greedy algorithm.

The key idea is to define a heuristic function, $h(v)$, that estimates how far a given vertex, v , is from the destination vertex (the goal). The notion of distance to minimize is defined as the sum of the actual distance and the “heuristic distance”, no longer as the simple distance from the initial vertex. As a result, when a vertex is extracted from the queue, the algorithm chooses the vertex that minimizes the sum, $h(v) + g(v)$, where $g(v)$ is the cost to reach the node under consideration. The method is depicted in Algorithm 5.

Table 4 summarizes the different algorithms by comparing different metrics.

Table 4. Comparison of pathfinding algorithms.

Criteria	A*	DFS	BFS	Bellman–Ford	Dijkstra
Time Complexity	$O(b^d)$	$O(b^m)$	$O(b^d)$	$O(VE)$	$O(V^2)$
Space Complexity	$O(b^d)$	$O(bm)$	$O(b^d)$	$O(V + E)$	$O(V)$
Works with Negative Weights	No	No	No	Yes	No
Optimal	Yes	No	Yes	Yes	Yes
Use Cases	Shortest path in weighted graphs	Maze solving, Topological Sorting	Shortest path in unweighted graphs	Shortest paths with negative weights	Shortest path in weighted graphs without negative weights

Algorithm 5 A* Algorithm**Require:** Graph $G(V, E)$, start node $start$, goal node $goal$, heuristic function h **Ensure:** Shortest path from $start$ to $goal$ if it exists

```

1: function ASTAR ( $start, goal$ )
2:   Create open set  $openSet$  and initialize it with  $start$ 
3:   Create cameFrom map to store navigation paths
4:    $gScore[start] \leftarrow 0$  ▷ Cost from start along best known path
5:    $fScore[start] \leftarrow h(start)$  ▷ Estimated total cost from start to goal
6:   while  $openSet$  is not empty do
7:      $current \leftarrow$  the node in  $openSet$  with the lowest  $fScore[current]$ 
8:     if  $current = goal$  then return RECONSTRUCTPATH(cameFrom,  $current$ )
9:     end if
10:    Remove  $current$  from  $openSet$ 
11:    for each neighbor  $neighbor$  of  $current$  do
12:       $tentative_gScore \leftarrow gScore[current] + dist(current, neighbor)$ 
13:      if  $tentative_gScore < gScore[neighbor]$  then
14:         $cameFrom[neighbor] \leftarrow current$ 
15:         $gScore[neighbor] \leftarrow tentative_gScore$ 
16:         $fScore[neighbor] \leftarrow gScore[neighbor] + h(neighbor)$ 
17:        if  $neighbor$  not in  $openSet$  then
18:          Add  $neighbor$  to  $openSet$ 
19:        end if
20:      end if
21:    end for
22:  end while return Failure
23: end function
24: function RECONSTRUCTPATH( $cameFrom, current$ )
25:   Total path  $\leftarrow [current]$ 
26:   while  $current$  in  $cameFrom.Keys$  do
27:      $current \leftarrow cameFrom[current]$ 
28:     Prepend  $current$  to Total path
29:   end while return Total path
30: end function

```

4.2. ML-Based Methods

4.2.1. Graph Neural Networks (GNNs)

GNNs have emerged as a powerful tool for learning from data that is structured as graphs [41–44]. A GNN operates on a graph, $G = (V, E)$, where each node, $v \in V$, has an associated feature vector, x_v .

The goal of a GNN is to learn a representation vector, h_v , for each node, which captures not only its features but also the structure of its neighborhood.

The core idea of GNNs is to update the representation of a node by aggregating features from its neighbors:

$$h_v^{(l+1)} = \text{UPDATE}\left(h_v^{(l)}, \text{AGGREGATE}\left(\left\{h_u^{(l)} : u \in \mathcal{N}(v)\right\}\right)\right), \quad (1)$$

where $h_v^{(l)}$ is the representation of node v at layer l , $\mathcal{N}(v)$ is the set of neighbors of v , and AGGREGATE and UPDATE are differentiable functions, often implemented as neural networks.

Training a GNN involves learning the parameters of the AGGREGATE and UPDATE functions. This is typically done using a supervised learning approach, where the GNN is trained to minimize a loss function on a set of labeled examples.

For a node classification task, the loss function is often the cross-entropy loss between the predicted labels and the true labels of nodes. The parameters are optimized using gradient descent, where gradients are computed via backpropagation.

GNNs can determine the shortest path between two points in a graph by learning the underlying structure through node embedding and message passing. Initially, each node is given an embedding, which is iteratively updated as the network processes information from neighboring nodes. This helps the GNN understand the graph's structure. For path prediction between source and destination nodes, the GNN interprets the node embeddings to suggest a path. Training involves adjusting the network to minimize errors between predicted and actual shortest paths, enabling the GNN to predict shortest paths in similar, unseen graphs after training. The process is summarized in Algorithm 6.

Algorithm 6 Training GNN for Shortest Path Prediction

Require: Graphs with known shortest paths, Node features for each graph, Number of training epochs, Learning rate

```

1: Initialize GNN model with random weights
2: for each epoch do
3:   for each graph in the training dataset do
4:     for each pair of nodes (source, target) in the graph do
5:       Calculate node embeddings using GNN
6:       Predict path from source to target using embeddings
7:       Compute actual shortest path from source to target
8:       Calculate loss (e.g., path length difference)
9:       Update GNN weights using backpropagation and learning rate
10:    end for
11:  end for
12: end for

```

Scaling up GNNs to manage waste collection in larger urban areas with more complex infrastructures poses several challenges, but it can be effective with careful consideration and appropriate strategies.

For instance, GNNs rely on large volumes of data to learn meaningful representations of the urban environment. In larger urban areas, acquiring comprehensive data regarding waste collection routes, bin locations, traffic patterns, and other relevant factors becomes essential. Collaborations with local governments, waste management agencies, and other stakeholders are crucial to gather and maintain such data.

As urban areas grow larger and more complex, GNN models need to scale accordingly to capture the intricacies of the urban infrastructure. This involves designing GNN architectures that can handle larger graphs with more nodes and edges while ensuring computational efficiency and scalability. Techniques such as graph partitioning, parallel processing, and distributed computing can be employed to handle the increased complexity.

Waste collection in larger urban areas often involves dynamic spatial and temporal factors, such as varying traffic patterns, changing waste generation rates, and evolving infrastructure. GNNs must be capable of incorporating both spatial and temporal information to adaptively optimize waste collection routes in real-time. Time-aware GNN architectures and recurrent or temporal graph convolutional networks (TGCNs) can be employed to model temporal dynamics effectively.

Efficient resource allocation becomes critical when managing waste collection in larger urban areas. GNNs can assist in optimizing resource allocation by predicting optimal collection routes, scheduling collection activities, and allocating resources based on predicted waste generation rates and demand patterns. Multi-objective optimization techniques can be integrated into GNN-based models to balance various objectives, such as minimizing collection costs, reducing environmental impact, and maximizing service coverage.

At the same time, GNN models must be robust and capable of generalizing across diverse urban environments to effectively manage waste collection in larger urban areas with varying infrastructural characteristics. Transfer learning techniques, domain adaptation methods, and robust training strategies can help GNNs generalize well to unseen urban environments and adapt to changing conditions.

Finally, collaboration and engagement with local authorities, waste management agencies, urban planners, and residents are essential for the successful implementation and scaling of GNN-based waste collection management systems. Feedback loops and iterative improvements based on real-world observations and stakeholder input can help refine GNN models and optimize waste collection operations over time.

While scaling up GNNs to manage waste collection in larger urban areas presents challenges, it can be effectively achieved by leveraging comprehensive data, designing scalable and adaptable models, considering spatial and temporal dynamics, optimizing resource allocation, ensuring robustness and generalization, and fostering collaboration and stakeholder engagement.

4.2.2. Genetic Algorithms (GAs)

Evolutionary Algorithms (EAs) are a class of optimization algorithms that are inspired by natural biological evolution [45,46]. The idea of EAs is rooted to the inherent search and selection mechanisms found in nature that lead to the survival and reproduction of the most suitable individuals. Considering the continuous process of environmental change, adaptation is a fundamental mechanism for survival within a species, connected to the intricate interplay between an individual's unique traits and the underlying genetic content dictating these traits through gene regulation [47]. Indeed, genes play a crucial role, as they are responsible for individual characteristics and, consequently, for survival and proliferation in a competitive environment.

GAs belong to EAs and provide an alternative to classical algorithms for optimization. The basis of their functioning is a selection mechanism used for the generation of a sequence of populations, and crossover and mutation as search mechanisms. Two classes of problems can be simplified through GAs: (i) problems that involve the pruning of numerous possible solutions, (ii) problems demanding adaptability and high performance in a dynamic and evolving environment.

GAs are based on the creation of a population of chromosomes (i.e., the individuals). A chromosome consists of genes, each of which represents a specific parameter or variable in the solution space. First, the population is created randomly and, hypothetically, each individual could be accounted as a solution for the given problem, as stated in [48]. Mitchell [49] explains that the GAs involve at least three operations: (i) selection, (ii) crossover, (iii) mutation. Selection is operated according to a fitness function that rates each potential solution. Each chromosome receives a score based on its ability to fit and solve the given problem. Chromosomes scoring the highest values are selected for reproduction, as they will create the subsequent generation of individuals. Crossover (or recombination) combines the selected individuals, generating new chromosomes (i.e., descendants or offspring) with (ideally) better characteristics than the previous ones. Mutation slightly and randomly manipulates genes, introducing new features in chromosomes as genetic diversity helps to prevent local optima trapping. The operations are repeated until a certain criterion is verified.

Algorithm 7 summarizes the functioning of a generic GA in the shortest path-finding context.

Algorithm 7 GA for Shortest Path Problem

Require: Graph $G(V, E)$ with node features, Set of training source-target pairs with known shortest paths

Ensure: the Shortest path from start to goal if it exists

- 1: Randomly initialize chromosomes ▷ i.e., routes
- 2: **repeat**
- 3: Calculate fitness values ▷ Calculate total distance for each route
- 4: Evaluate objective function
- 5: Select parents for reproduction ▷ Select routes for crossover
- 6: Perform crossover to create offspring ▷ Create new routes via crossover
- 7: Apply mutation to the offspring ▷ Apply swap mutation to the offspring routes
- 8: Replace old population with the new one
- 9: **until** convergence or maximum number of iterations is reached

Fujdiak et al. [50] and Ikram et al. [51] use GAs applied to the smart waste management problem. Ochelska-Mierzejewska et al. [52] discuss the usage of GAs for solving the routing problem.

4.2.3. Reinforcement Learning (RL)

RL is a subset of ML where an agent learns to make decisions by taking actions in an environment to achieve some notion of cumulative reward. In the context of graph theory, RL can be applied to find the shortest path between two nodes. The agent learns to navigate through the graph, from a start node to a target node, by interacting with the environment (the graph) and receiving feedback in the form of rewards or penalties.

Consider a graph, $G = (V, E)$. The goal is to find the shortest path from a start node, $s \in V$, to a target node, $t \in V$. Then, the RL problem can be formulated as follows. The state space represents all possible situations the agent can be in. In a graph scenario, a state, $s \in S$, can be defined as the current node the agent is positioned at. Moreover, the action space consists of all possible actions the agent can take from a given state. For a node, v , the possible actions, $a \in A$, are moving to any adjacent nodes, i.e., $a = \{(v, u) | u \in \mathcal{N}(v)\}$, where $\mathcal{N}(v)$ represents the neighbors of node v . The reward function, $R(s, a, s')$, defines the immediate reward the agent receives after transitioning from state s to state s' by taking action a . A typical reward setting for the shortest path problem is to give a small negative reward for each step to encourage shorter paths and a large positive reward when the target node, t , is reached.

A policy, π , is a strategy that the agent employs to determine the next action based on the current state. It maps states to actions, i.e., $\pi : S \rightarrow A$. The goal of RL is to learn an optimal policy, π^* , that maximizes the cumulative reward. Finally, Q-Learning is a model-free RL algorithm used to find the optimal action-selection policy for any given finite Markov decision process. It works by learning an action-value function, $Q(s, a)$, which gives the expected utility of taking action a in state s and following the optimal policy thereafter. The Q-Learning update rule is as follows:

$$Q(s, a) \leftarrow Q(s, a) + \alpha [R(s, a, s') + \gamma \max_{a'} Q(s', a') - Q(s, a)] \quad (2)$$

where α is the learning rate, γ is the discount factor, and $R(s, a, s')$ is the reward received after transitioning from state s to state s' by taking action a .

Applying RL to the shortest path problem in graphs involves defining the states, actions, and rewards in the context of the graph structure. The agent learns to navigate the graph by interacting with it and receiving feedback based on the defined reward structure. With a well-defined reward function and an appropriate RL algorithm such as Q-Learning, the agent can efficiently learn to find the shortest path between two nodes. Algorithm 8 summarizes the described approach.

Algorithm 8 Q-Learning for Shortest Path in Graphs**Require:** Graph $G = (V, E)$, Start node s , Target node t , Learning rate α , Discount factor γ **Ensure:** Optimal policy π^* to reach from node s to node t

- 1: Initialize $Q(s, a)$ arbitrarily for all $s \in V$ and $a \in A(s)$
- 2: Initialize V (Visited set) as an empty set
- 3: **repeat**
- 4: Set $current_node \leftarrow s$
- 5: **while** $current_node \neq t$ **do**
- 6: Choose a from $A(current_node)$ using policy derived from Q (e.g., ϵ -greedy)
- 7: Take action a , observe reward R and next node s'
- 8: $Q(current_node, a) \leftarrow Q(current_node, a) + \alpha [R + \gamma \max_{a'} Q(s', a') - Q(current_node, a)]$
- 9: Add $current_node$ to V
- 10: $current_node \leftarrow s'$
- 11: **end while**
- 12: **if** all nodes in V are visited **then**
- 13: **break**
- 14: **end if**
- 15: **until** convergence or maximum number of iterations is reached
- 16: Extract policy π^* from Q

5. Performance Evaluation*5.1. Standard Metrics*

This subsection reviews the models previously discussed through a set of standard metrics, i.e., accuracy, response time, resources, efficiency, and energy consumption.

DFS has low accuracy due to its tendency to find paths quickly, but not necessarily the shortest or most optimal ones. Both BFS and GA have moderate accuracy. While BFS ensures finding the shortest path, it may not always be the most optimal; GA may find good solutions but not always the best path. Dijkstra's algorithm and GNNs have high accuracy; the former guarantees finding the shortest path and the latter can learn from data and provide accurate predictions for optimal paths. Similarly, time-series forecasting models can leverage historical data to predict optimal paths accurately.

Concerning response time, GAs perform poorly because they involve iterative processes that may take longer to converge. GNNs, BFS, time-series forecast, and Dijkstra's algorithm models would instead perform moderately: GNNs may require some computation to learn optimal paths but can respond in a reasonable time; BFS explores all possible paths, which can be time-consuming; time-series forecasting models require processing historical data but can provide timely predictions once trained; Dijkstra's algorithm explores the graph based on distance and is faster than BFS but slower than DFS. Moreover, DFS is fast because it explores one path until the end of a dead end, making it faster than BFS in some scenarios.

In the evaluation process, resource efficiency plays an important factor, which is low for BFS and DFS. BFS explores all possible paths, which can be resource-intensive, while DFS may backtrack and explore deep branches, consuming resources without guaranteeing optimality. On the other hand, Dijkstra's algorithm consumes moderate resources by exploring the graph based on distance, and similarly for GAs, as they may require significant computational resources for convergence. On the other hand, time-series and GNN models would exhibit high resource usage. In the first case, a model would leverage historical data efficiently once trained; in the second, it would be possible to learn representations from data and provide efficient predictions for optimal paths.

Energy consumption holds significant importance in smart cities, particularly concerning algorithmic operation, from different perspectives: (i) sustainability, as smart cities prioritize environmental concerns, necessitating energy-efficient algorithms to align with these goals; (ii) cost reduction, because high energy consumption translates to increased

operational costs, making energy-efficient algorithms essential for minimizing expenses; (iii) resource conservation, as energy-efficient algorithms contribute to preserving resources such as electricity and computing resources, supporting sustainable urban development; (iv) infrastructure resilience, because energy-efficient algorithms help ensure the resilience of smart city infrastructure by reducing energy demands, particularly during periods of high demand or shortages; (v) device longevity, since in many smart city applications algorithms run on battery-powered devices and energy-efficient algorithms extend device battery life, enhancing system reliability and reducing maintenance requirements; and lastly, (vi) community engagement, because demonstrating a commitment to sustainability through energy-efficient practices fosters community engagement and support for smart city initiatives.

In this sense, DFS requires low energy because it explores one path at a time, resulting in lower energy consumption. Instead, BFS and time-series models are moderate: the former may require moderate energy due to its exploration of all possible paths, while the latter may consume moderate energy for processing historical data and making predictions. Equivalently, Dijkstra’s algorithm consumes moderate energy by exploring the graph based on distance. Energy consumption is instead high for GNNs and GAs: the first may require significant energy for training and inference processes, and the second involves iterative processes that may consume significant energy resources. Table 5 summarizes the evaluation factors for each model.

Table 5. Comparison of pathfinding algorithms for smart bin collection.

Algorithm	Accuracy	Response Time	Resource Efficiency	Energy Consumption
BFS	Moderate	Moderate	Low	Moderate
DFS	Low	Fast	Low	Low
Dijkstra	High	Moderate	Moderate	Moderate
A*	High	Moderate to High	Moderate	Moderate
B.F.	High	Low to Moderate	Moderate	Moderate to High
GNNs	High	Moderate	High	High
RL	High	Variable	Moderate to High	Moderate to High
GA	Moderate	Slow	Moderate	High
T. S. Models	High	Moderate	High	Moderate

A*’s heuristic-driven search can provide highly accurate results for pathfinding, especially if the heuristic is well-tuned to the specific characteristics of the urban environment. Moreover, the algorithm is generally faster than uninformed search algorithms due to its heuristic, which guides the search towards the goal. However, its performance can vary based on the complexity of the heuristic calculation and the topology of the search space. While A* is more resource-efficient than some algorithms due to avoiding exploration of unlikely paths, it can still consume significant memory, especially in large or complex graphs, as it needs to store all explored nodes. In this sense, the energy consumption of A* correlates with its computational and memory usage. Efficient heuristics can reduce energy consumption by minimizing unnecessary computations and path explorations.

Bellman–Ford is very accurate, especially in graphs with negative edge weights where other algorithms such as Dijkstra’s might fail. It guarantees finding the shortest path if one exists, without restrictions on edge weights. However, the response time can be a limitation due to the algorithm’s higher time complexity, requiring multiple iterations over all edges in the graph, especially noticeable in large and dense networks. The algorithm’s resource usage is primarily dictated by the need to iteratively process all edges in the graph. While it does not require storing multiple paths, its iterative nature over all edges can lead to significant resource use in dense networks. Given its computational intensity and the need for repeated iterations over all graph edges, Bellman–Ford can consume more energy, particularly in complex or dynamically changing environments where frequent recalculations are necessary.

Regarding ML models, GNNs allow for accurate predictions and decision-making, especially when node features and edge attributes are significant in determining the optimal path for bin collection. The response time for GNNs can vary based on the network's complexity and the size of the graph. While GNNs can be efficient in processing localized node information, the aggregation and update steps across layers can become computationally intensive for large graphs. Moreover, GNNs require sufficient computational resources for training and inference, especially as the number of nodes and edges in the graph increases. The memory requirement for storing node features, edge attributes, and network parameters can also be substantial. The energy consumption of GNNs generally correlates with their computational intensity and resource usage. Training GNNs can be particularly energy-intensive due to the need for multiple forward and backward passes through the network during the optimization process.

GAs can provide highly accurate solutions to optimization problems, including routing for smart bin collection. However, the accuracy is heavily dependent on the design of the fitness function, genetic operators (crossover and mutation), and other parameters. The response time can be high, especially if the population size is large or if the problem space is complex. Furthermore, this approach requires memory to store the population of solutions and their respective fitness values. The resource efficiency depends on the population size and the complexity of the individuals (solutions). The energy consumption of GAs can be significant due to the need for multiple evaluations of the fitness function and the genetic operations applied over several generations.

In smart bin collection, RL can learn highly effective strategies for routing and scheduling by interacting with the environment and optimizing the cumulative reward, which might represent factors such as route efficiency, fuel consumption, or the timeliness of bin collection. The response time in RL depends on the complexity of the state space, the learning algorithm used, and the convergence criteria. However, in dynamic environments, the model might require continuous learning or frequent retraining, which can impact the response time. RL models, especially those with deep-learning architectures (Deep RL), can be resource-intensive during the training phase due to the need for numerous iterations and data storage.

In the considered case study, time-series models can predict waste generation patterns or bin fullness levels with high accuracy if historical data are representative and the models are well-tuned. Making predictions is usually fast, as it involves applying the model to the most recent data. However, training and tuning the model, especially in the presence of large datasets or complex models, can be time-consuming. These models are generally resource-efficient during the inference phase, as they require less computational power compared to complex ML models. However, resource efficiency during the training phase can vary depending on the model complexity and data volume. The energy consumption of time-series forecasting models is primarily associated with the training phase, where model selection and parameter tuning can be computationally intensive. However, the energy consumption during the forecasting (inference) phase is usually low.

Finally, RL can optimize routes and schedules by learning from interactions with the environment, leading to highly efficient strategies for bin collection, traffic navigation, and resource allocation. The response time for RL is contingent on the complexity of the state space and the architecture of the learning model. While the training phase can be time-consuming and requires numerous iterations, a well-trained RL agent can offer real-time decision-making capabilities. However, dynamic environments might necessitate continuous learning or periodic retraining, impacting the response time. Once an RL model is trained, the inference or decision-making process can be relatively resource-efficient. The energy consumption of RL models is considerable during the training phase, attributed to the computational demands for processing extensive data and updating model parameters. However, for a trained agent making decisions, the energy consumption can be significantly lower.

5.2. Integration in Diverse Urban Architectures

Integrating models with existing waste management systems in smart cities involves understanding the capabilities of each model and their applicability within diverse urban system architectures.

DFS could be used for route planning in waste collection by exploring one path until reaching the end of a dead end. Moreover, DFS could be implemented as part of a routing algorithm within a waste management system. It could be integrated with GIS (Geographic Information Systems) data to efficiently navigate through city streets. Similar to DFS, BFS could also be employed for route planning, exploring all possible paths from the starting point. In urban system architecture, BFS could complement DFS by providing alternative routes or exploring areas that may have been missed. It could be integrated into a waste management system as an additional route optimization algorithm.

In waste management systems, Dijkstra's algorithm could be utilized for optimal route planning, considering factors such as distance, traffic conditions, and waste bin fill levels. It could be integrated as a core component of route optimization modules within existing waste management software.

It should be noted that A* can be paired with GIS data to provide real-time, context-aware navigation, considering factors such as traffic conditions, road closures, and bin locations. Furthermore, A* can be tailored to reflect the priorities of the waste management system, such as minimizing travel time, reducing fuel consumption, or prioritizing bins that are known to fill up quickly. Notably, A* can complement DFS and BFS, offering a more directed search towards the goal while still considering multiple paths. In combination with time-series analysis, A* could utilize forecasts of waste generation rates or traffic patterns to inform its heuristic, enhancing the efficiency of the route planning process. Bellman–Ford's capability to handle negative weights makes it suitable for urban environments where certain paths may have attributes that make them less desirable, effectively acting as a 'cost' (e.g., congested areas, zones with higher waste levels). It can be used in waste management systems to plan routes that avoid these high-cost areas or dynamically adjust the routes based on changes in these weights, such as sudden increases in traffic or changes in bin fill levels.

The Bellman–Ford algorithm can be integrated into the optimization modules of waste management systems, particularly for scenarios where the graph's edge weights are not static and can change over time, providing robust route solutions. It can work alongside GAs or time-series analysis techniques to periodically update and optimize the routes based on historical and real-time data. In combination with GNNs, Bellman–Ford could provide initial route solutions that GNNs can further optimize based on the learned patterns from historical data. It can also complement Dijkstra's algorithm in systems where the graph might contain negative weight edges or dynamic weight calculations and need frequent updates.

GNNs can learn from graph-structured data and provide predictions for optimal paths. In waste management systems, GNNs could be trained on the historical data of waste collection routes, bin fill levels, and traffic patterns to predict efficient routes in real-time. They could be integrated into route planning modules to provide adaptive and context-aware route suggestions.

GAs could be employed for route optimization by evolving a population of candidate routes over successive generations. They could be integrated into optimization modules to continuously improve waste collection routes based on feedback and performance metrics.

Finally, time-series analysis techniques could be applied to the historical data of waste collection routes, bin fill levels, and traffic patterns to identify patterns and trends over time. In waste management systems, time-series analysis could help in forecasting future waste generation rates, optimizing collection schedules, and predicting optimal collection routes based on historical trends. It could be integrated into decision support systems to enhance route planning and resource allocation strategies. In diverse urban system architectures, these models could be implemented as standalone modules or integrated into

existing waste management software systems. They could leverage real-time data streams from IoT sensors, GPS devices, and traffic monitoring systems to adaptively optimize waste collection routes and schedules, leading to more efficient and sustainable waste management practices in smart cities.

5.3. Cost-Benefits Analysis

Performing a comprehensive cost-benefit analysis comparing the models proposed so far with traditional waste management approaches would require detailed data and context-specific considerations beyond the scope of this work. However, a theoretical analysis can be undertaken by considering the following aspects: (i) costs, (ii) benefits, (iii) risk and uncertainty, (iv) long-term viability, and (v) social and environmental sustainability. Each factor can be studied concerning classic algorithms (BFS, DFS, Dijkstra) and ML models (GNN, GA, time-series-based forecast models).

Regarding classic approaches, the costs can be decomposed into implementation, operation, infrastructure, and training/education. Implementation costs include the initial setup costs for deploying the waste management system and integrating the chosen algorithmic models. Ongoing operation expenses are related to system maintenance, software updates, and data management. The infrastructure costs refer to the investment required for hardware, sensors, and other IoT devices used in waste collection. Furthermore, it is necessary to account for costs associated with training personnel to use and maintain the waste management system effectively.

The benefits can vary. For example, efficiency gains may occur if algorithmic models lead to more efficient waste collection routes, reducing fuel consumption, vehicle wear and tear, and labor costs. By optimizing route planning and resource allocation, algorithmic models can minimize unnecessary trips and improve overall resource utilization, leading to an optimized resource allocation. The quality of the service can be improved by enhancing route optimization, leading to more timely waste collection, reducing the risk of overflowing bins, and improving overall cleanliness and sanitation. From a similar perspective, reduced fuel consumption and emissions resulting from optimized routes contribute to environmental sustainability and may lead to long-term cost savings and health benefits for residents. It is also worth noting that algorithmic models provide valuable insights through data analysis, enabling informed decision-making and proactive problem-solving in waste management operations. The risks associated with the implementation and integration of algorithmic models into existing waste management systems, including technical challenges, data quality issues, and stakeholder resistance, need to be evaluated. This process also includes the fact that the actual benefits of algorithmic models may vary depending on factors such as data accuracy and model performance, as well as external factors such as weather and traffic conditions. The long-term viability factor can be subdivided into scalability and adaptability. The former consists of the consideration of whether the chosen algorithmic models can scale effectively to meet the evolving needs of growing urban populations and changing waste management requirements. The latter describes the ability of algorithmic models to adapt to new challenges, emerging technologies, and regulatory changes over time. Moreover, social and environmental impacts can be quantified by different aspects. For example, community acceptance identifies the social acceptability of algorithmic waste management approaches and their perceived impact on residents' daily lives and communities. As a counterpart, environmental sustainability denotes the evaluation of the environmental impacts of algorithmic models compared to traditional waste management approaches, including their contribution to reducing greenhouse gas emissions and mitigating pollution.

Concerning ML-based methods, implementation costs are related to the initial setup costs for deploying and training the ML models, including data acquisition and preprocessing. Ongoing expenses related to the model maintenance, updates, and computational resources required for inference and analysis denote the operational costs. Investment

in computational resources and expertise for training the ML models on historical data compose the training effort.

Regarding the benefits, in terms of predictive accuracy ML models can offer improved predictive accuracy for waste collection route optimization, leading to more efficient resource allocation and service delivery. Similarly, ML models provide valuable insights through data analysis, enabling informed decision-making and proactive problem-solving in waste management operations, from the perspective of a data-driven approach. Within the context of ML models, adaptive optimization reflects on the ability of the models to adapt to changing environmental conditions, traffic patterns, and waste generation rates in real-time.

Risks can be associated with the performance and reliability of ML models in real-world applications, including overfitting, data biases, and model drift. Moreover, dependence on the quality and availability of historical data for training the ML models may be subject to inaccuracies and inconsistencies.

In terms of long-term viability, it is necessary to ensure the robustness and generalization capabilities of ML models across diverse urban environments and evolving waste management scenarios. Evaluating the scalability of ML models is important for handling larger datasets and more complex urban infrastructures over time.

Finally, ML models can be evaluated from the point of view of engagement with stakeholders to ensure the ethical and responsible deployment of ML models in waste management operations, addressing concerns related to privacy, fairness, and transparency. On the other hand, assessing the environmental impacts of ML models, including their energy consumption and carbon footprint, and exploring strategies for minimizing their ecological footprint is of paramount importance when considering smart cities and responsible AI.

5.4. Hybrid Models Scalability

The scalability of a hybrid model based on a GNN and another ML paradigm for waste collection in larger urban areas with complex infrastructures depends on various factors. While such models offer promising solutions, scaling them effectively involves addressing several challenges: (i) Data volume and diversity, as in larger urban areas there is typically a substantial increase in data volume due to a greater number of bins and more diverse environmental factors. The model must handle these larger and more diverse datasets efficiently: (ii) Model complexity, as waste collection logistics become more complex as urban areas grow. Models must consider factors such as traffic patterns, road conditions, and varying waste generation rates across neighborhoods. Scaling the model effectively requires addressing this complexity. (iii) Computational resources, because larger urban areas require more computational resources for real-time decision-making. Ensuring that the infrastructure can support the computational demands of the hybrid model is crucial. (iv) Latency and response time, since maintaining low latency and fast response times is essential for real-time waste collection optimization. Delays in decision-making can lead to inefficiencies. Scaling should not compromise response times. (v) Resource allocation, as efficiently allocating collection resources, such as vehicles and personnel, across a vast urban landscape is a complex optimization problem. Scaling up the model involves adapting resource allocation strategies to handle larger areas effectively. (vi) Environmental impact, as minimizing the environmental impact of waste collection is increasingly important. Scaling the model should consider sustainability objectives, aiming to reduce emissions and align with the goals of greener, more sustainable urban environments. (vii) Data quality and integration, as in larger urban areas data may come from various sources and formats. Ensuring data quality and effective data integration are essential for the model's accuracy and performance at scale. (viii) Infrastructure and connectivity, since the availability of infrastructure, including edge computing capabilities and high-speed connectivity, plays a significant role in scaling the model. Ensuring that the necessary infrastructure is in place is crucial. (ix) Adaptability, as the model should be adaptable to changing conditions and

evolving urban infrastructure. Regular updates and retraining with new data are essential to maintain effectiveness as the urban environment changes.

6. Discussion

This section proposes a hypothetical case study based on the collection of smart bins in an urban area. The bins are subjected to a constraint, i.e., they need to be emptied only if the amount of waste exceeds a certain threshold. As a consequence, if the area is modeled as a graph, different strategies to implement the optimal waste collection can be evaluated. The considered approaches can be monolithic (both classic and ML-based) or hybrid. Although the literature about hybrid models including GNNs and RL is not new (see Section 2), the presented algorithms target a specific problem and, in this sense, are novel. Each method is presented and discussed from the perspective of space and time complexity.

The second part of the section reviews monolithic models through standard performance metrics such as (i) accuracy, (ii) response time, (iii) resource efficiency, and (iv) energy consumption, which suits the context of greener smart cities. Furthermore, the debate is extended to an evaluation analysis through cost-benefits and integration in diverse urban architectures.

Performing a similar analysis for hybrid models requires a deeper investigation involving experiments on a real case study and is outside the scope of this work.

6.1. Case Study

Consider a town with a geographically defined layout and a network of smart bins strategically placed throughout the cityscape. To effectively manage waste collection and optimize resource allocation, a graph model, G , can be employed to represent the distribution of these smart bins and their connectivity within the town's infrastructure. Each smart bin is represented as a node in the graph. These nodes are uniquely identified and characterized by their physical location within the town. A node, $g \in G$, is denoted by the following attributes:

- **Smart Bin ID:** A unique identifier assigned to each smart bin for unambiguous identification.
- **Location:** The precise geographical coordinates of the smart bin, represented by latitude and longitude.
- **Current Waste Level:** A real-time indication of the amount of waste present in the bin, ranging from a minimum value to a maximum value.

Edges E of G are introduced to connect neighboring smart bins, representing the physical proximity between them. The existence of an edge between two nodes indicates that there is a direct route connecting the corresponding smart bins.

Edge attributes are characterized by a weight (i.e., a numerical value assigned to each edge to represent the distance between the connected smart bins) and a connectivity density (i.e., a measure of the relative density of smart bins in a given area. Higher values indicate a more densely distributed network of smart bins, while lower values suggest a more sparsely distributed network). For the sake of clarity, a simplified example of this scenario is displayed in Table 6.

Table 6. Example of a connected smart bin network.

Smart Bin ID	Location (Lat, Long)	Current Waste Level	Neighboring Bins
001	(35.6895, 139.6917)	75%	002, 003, 004
002	(35.6897, 139.6920)	50%	001, 003, 004
...	(..., ...), ..., ...
007	(35.6898, 139.6918)	40%	001, 002, 003

The last column in the table, “Neighboring Bins”, indicates the connections each smart bin has with other bins in the network. It lists the IDs of bins that are nearby or directly linked to each bin, reflecting the dense connectivity in the network. This information is crucial for planning efficient waste collection routes, as it shows which bins are adjacent and can be serviced together.

Figure 2 reports a sandbox example of the conversion from a roadmap to a graph. First, the roadmap is visualized, complete with both one-way streets (denoted by arrows showing the direction) and two-way streets. Then, the nodes (i.e., the bins, which are supposed to be located at road intersections and dead-ends) are individuated, together with the edges (i.e., streets) connecting them. Subsequently, weights (i.e., distances) are applied. The resulting graph is obtained as the composition of seven vertices and seven weighted edges (three oriented and four unoriented). Each vertex contains valuable information about its status: the ID of the smart bin, its location (latitude and longitude), its current waste level, and its neighboring bins. If a bin exceeds a certain previously set threshold of fullness, it is marked in red as it needs to be emptied. Last, a graph made of bins that need to be emptied is produced.

The depicted scenario can be modeled by using a dynamic graph, i.e., a type of graph where the structure changes over time [53]. This can involve the addition or removal of nodes and edges. More formally, a dynamic graph can be represented as $G(t) = (V(t), E(t))$, where $V(t)$ and $E(t)$ denote the sets of vertices and edges at time t , respectively. The changes in $V(t)$ and $E(t)$ across different time steps capture the dynamic nature of the graph. This representation allows for the modeling of real-world systems where relationships and entities evolve, such as social networks or transportation systems.

6.2. Classic Monolithic Approaches

In the dynamic scenario of smart bin collection, traditional graph traversal algorithms have unique applications and constraints.

BFS, exploring the network level by level, can efficiently find the shortest routes for nearby bins but might not effectively prioritize bins based on dynamic waste levels.

In contrast, DFS delves deeply into each path, offering thorough exploration at the cost of potentially longer, inefficient routes and not necessarily addressing high-priority bins first. Different urban layouts can significantly impact the performance of graph-based models employing traversal algorithms such as BFS and DFS. For instance, urban areas with a high density of nodes, such as downtown areas, can pose challenges for BFS as it may traverse a larger number of nodes due to increased connectivity. DFS might encounter difficulties with deep branches, potentially resulting in longer traversal times. Furthermore, sparse urban layouts, such as suburban areas, may present different challenges. BFS might need to navigate longer paths between nodes, while DFS could traverse fewer nodes but may encounter dead ends more frequently. The structure of the road network also plays a crucial role. BFS may perform better in grid-like networks with uniform connectivity, while DFS might excel in complex networks with multiple interconnected pathways. Moreover, obstacles or blocked paths within the urban environment can hinder traversal efficiency for both BFS and DFS. The former may adapt better to changes by exploring alternative paths, while the latter may get stuck in blocked routes. Furthermore, the choice between the two algorithms depends on the balance between finding the best path and speed, influenced by urban layout and task requirements. While BFS guarantees the shortest path, DFS is faster but does not guarantee optimality.

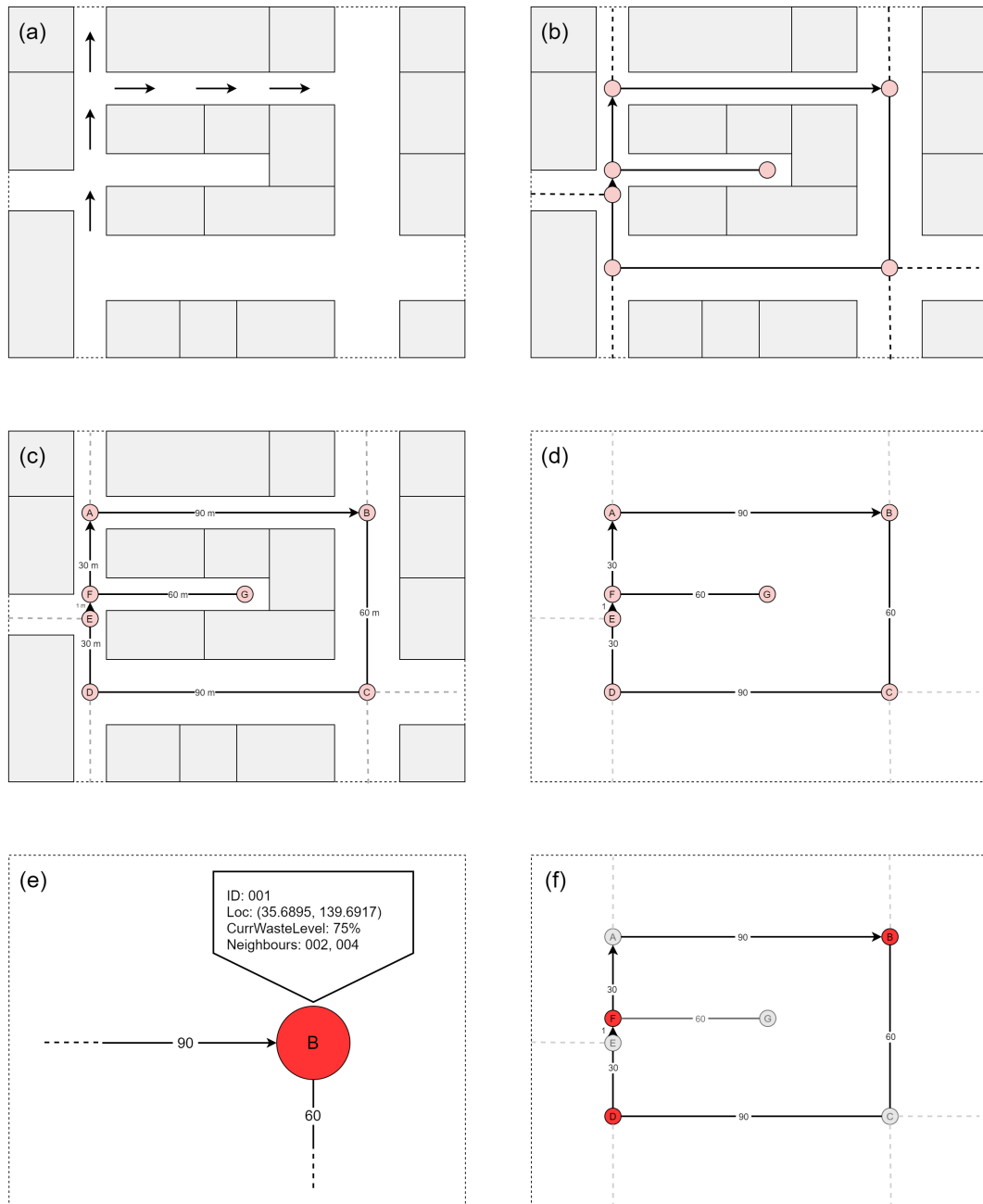


Figure 2. Conversion from a roadmap to a graph of full bins. The image consists of 6 sub-images, labeled from (a–f). From sub-image (a) to sub-image (c) grey rectangles represent buildings, white areas depict streets, and black arrows indicate one-direction-only streets. Starting from (b), pink circles with uppercase letters denote available nodes (i.e., bins). The lines and arrows connecting each node represent arcs of the resulting graph, with assigned costs (distances). Grey dotted lines depict the possibility of a scenario with additional nodes and arcs, meaning more roads are considered. Red circles in sub-images (e,f) mark the selected nodes (i.e., bins to be emptied) for the routing operation, while grey circles mark the discarded nodes.

Dijkstra’s algorithm, adept at navigating weighted graphs, could use waste levels as weights to find efficient routes to service high-priority bins. However, its computational complexity and the need for frequent recalculations in response to changing bin levels pose significant challenges. Different urban layouts can also impose penalties on classic algorithms such as Dijkstra’s algorithm, in several ways. For instance, irregular road networks or complex spatial configurations in urban layouts can complicate pathfinding

for Dijkstra's algorithm. This can lead to longer paths as the algorithm navigates detours or dead ends, resulting in suboptimal routes. Additionally, obstacles such as buildings, parks, closed roads, or traffic congestion can disrupt Dijkstra's algorithm by forcing it to explore alternative paths. Consequently, this can cause longer traversal times and suboptimal route choices.

A* uses a heuristic to estimate the cost to reach the target from a given node, which can be particularly beneficial in urban environments where certain paths may be predictably more efficient due to traffic patterns, road types, or bin locations. In high-density areas, A* can be effective by prioritizing paths that seem most promising, avoiding unnecessary exploration of dense node networks. In sparse suburban areas, A*'s heuristic can guide the search, reducing the time spent exploring less promising paths. The algorithm can adapt to dynamic changes in the environment if the heuristic and cost functions are defined to account for varying waste levels or traffic conditions, allowing real-time re-calibration of routes. However, the efficiency heavily depends on the accuracy of the heuristic function. An inaccurate heuristic might lead to suboptimal paths or increased computation time. A* might still face challenges due to computational load in highly dynamic environments where frequent recalculations are necessary.

The Bellman–Ford algorithm can handle graphs with negative weight edges, which could be useful if certain paths have attributes that make them less desirable, effectively acting as a 'cost' (e.g., high waste levels or areas prone to traffic jams). The algorithm can be beneficial in complex urban layouts, as it iteratively relaxes the distances to nodes, effectively adapting to intricate pathways and sudden changes in the graph. Moreover, the same algorithm could be used for dynamic route recalculations as it does not assume initially known or fixed distances, making it adaptable to changing conditions such as sudden road closures or waste bin status updates. However, the Bellman–Ford algorithm has a higher time complexity compared to algorithms such as Dijkstra's or A*. In large, dense urban graphs, the computational load might be significant. Finally, due to its nature of iterating over all edges, the performance might degrade in large, dense urban networks with many bins and connecting paths.

6.3. ML Monolithic Approaches

A GNN can learn from historical data, identifying patterns that indicate when bins are nearing full capacity. By training on these data, the model can predict which bins are likely to need emptying, as it would consider not just the fill level of each bin but also the context provided by the network of bins. This approach would lead to an optimized waste collection route, focusing on bins that are full while bypassing those that are not, thus improving efficiency in urban waste management. A generic proposal of an algorithm can be found in Algorithm 9, which depicts a new procedure employing a GNN for smart bin collection, articulated in the following steps:

1. Initialization: It starts by initializing the graph $G(V, E)$, representing the smart bins as nodes with features such as location and current waste level.
2. Training: The GNN model M is trained on historical data to learn patterns of bin fullness.
3. Real-time Monitoring: The algorithm continuously receives real-time waste level updates for each bin.
4. Update and Prediction: Node features in the graph are updated with new waste levels, and the GNN model predicts which bins are likely to be full.
5. Identification of Full Bins: For each bin, if its waste level exceeds the threshold, θ , it is marked for collection.
6. Route Determination: The algorithm determines an optimized collection route that covers all marked bins.
7. Waste Collection Execution: The waste is collected along the determined route.

Algorithm 9 Smart Bin Collection using a GNN

- 1: **Initialize** GNN to model smart bin network:
 - 2: Define GNN architecture and parameters
 - 3: Initialize GNN weights
 - 4:
 - 5: **Train GNN Model** on historical bin data
 - 6:
 - 7: **repeat**
 - 8: Receive real-time waste levels for bins
 - 9: Update node features in the graph G with real-time data
 - 10: Apply GNN model to predict full bins
 - 11: **for** each bin v in V **do**
 - 12: **if** waste level of $v \geq \theta$ **then**
 - 13: Mark v as a collection target
 - 14: **end if**
 - 15: **end for**
 - 16: Determine collection route covering marked bins
 - 17: Perform waste collection on the route
 - 18: **until** end of operation
-

Based on the existing scientific literature (see, for instance, Feng et al. [54], Hadou et al. [55], and Ding et al. [56]), it can be noticed that the space complexity for storing node features depends on the number of bins in the graph (n) and the number of features associated with each bin (k). Therefore, it can be expressed as $O(nk)$. Moreover, the space complexity of the GNN model primarily depends on its architecture and the number of nodes in the graph (N). Typically, GNNs have a space complexity of $O(N)$. Another point to be considered concerns storing the collection route, which is contingent on the number of bins marked as collection targets. In the worst case, if all bins are marked the space complexity would be $O(n)$. Overall, the total space complexity of the algorithm can be approximated as $O(nk + N)$. The time complexity of this algorithm comprises several key steps. The initialization of node features for each bin takes $O(nk)$ time. The time complexity for training the GNN model depends on the number of training iterations (m) and the complexity of forward and backward passes. If each iteration takes $O(N)$ time, then the training complexity is $O(mN)$. Furthermore, receiving real-time waste level data and updating node features for bins takes $O(nk)$ time in each iteration; similarly, applying the GNN model to predict full bins takes $O(N)$ time since it operates on the entire graph. Looping through each bin to check if its waste level exceeds the threshold takes $O(n)$ time while determining the optimized collection route may involve various algorithms, but it typically ranges from $O(N^2)$ or better, depending on the specific optimization techniques used. Finally, the actual waste collection along the route would depend on factors such as vehicle speed and the number of collection targets. It may involve traversing the entire route, which would take $O(N)$ time.

Overall, the time complexity is primarily determined by the training process ($O(mN)$), real-time updates ($O(nk)$), and route optimization (typically $O(N^2)$ or better). The training process is the most computationally intensive part of the algorithm, and its complexity can vary based on factors such as the GNN model’s architecture and the number of training iterations. Table 7 recaps the complexity for each step of the proposed method.

Table 7. Summary of time complexity for each process.

Process	Time Complexity
Initialization of node features for each bin	$O(nk)$
Training the GNN model (per iteration)	$O(N)$
Total training complexity	$O(mN)$

Table 7. Cont.

Process	Time Complexity
Receiving real-time waste level data	$O(nk)$
Updating node features for bins (per iteration)	$O(nk)$
Applying the GNN model to predict full bins	$O(N)$
Looping through bins to check waste level	$O(n)$
Determining the optimized collection route	$O(N^2)$ or better
Waste collection along the route	$O(N)$

Algorithm 9 faces several challenges. Its scalability is a concern, as performance may decrease with large-scale networks due to high computational and memory demands. The effectiveness of the model hinges on the availability of extensive, accurate historical data. Adapting swiftly to sudden changes in the network, such as adding new bins or shifts in waste patterns, can be difficult for GNNs. The model's generalization across different regions might require retraining, given the variability in waste generation habits. Real-time data updates are crucial but could be hampered by transmission delays or sensor errors. Additionally, the substantial computational resources needed for GNNs might pose a challenge for smaller organizations with limited IT infrastructure.

To address adaptability issues in a GNN, several strategies can be employed. One approach is to incorporate mechanisms that allow the GNN to adjust to changes in the graph structure over time. This can be achieved through techniques such as incremental learning [57] or transfer learning [58], where the network is trained not only on the initial data but also continuously adapts to new data. Additionally, using more flexible aggregation functions that can handle varying node degrees and changing graph dynamics can improve adaptability. Regularly updating the model with new data and employing architectures that can efficiently process dynamic graphs are also crucial for enhancing the adaptability of GNNs.

On the other hand, Algorithm 9 offers significant benefits. Firstly, GNNs excel in capturing complex relationships within data, allowing for a nuanced understanding of the spatial distribution and fill patterns of smart bins. This leads to more effective routing for waste collection, optimizing resource allocation. Additionally, GNNs can adapt to changes in the network, such as the addition of new bins or shifts in usage patterns and incorporating node features such as location, connectivity, and urban characteristics. They can process these changes without the need for complete model retraining, ensuring that the system remains efficient and up-to-date. Furthermore, the ability of GNNs to learn from historical data enables predictive insights, potentially improving waste management strategies. They can handle spatial heterogeneity, which makes them well-suited for diverse urban environments.

6.4. Hybrid ML Models

6.4.1. Hybrid Models: GNN and RL

Coupling the GNN approach with RL or other ML techniques can offer several benefits for the smart bin collection scenario. RL, in particular, could enable the system to learn optimal collection routes through trial and error, continuously improving efficiency based on feedback from real-world operations. It can also adapt to changing environmental conditions and bin usage patterns. Integrating other ML approaches might provide additional predictive capabilities, such as forecasting future fill rates of bins, further enhancing the effectiveness and efficiency of the waste collection system. This integration leads to a more robust and adaptive waste management solution.

To articulate a hybrid model combining a GNN with RL for smart bin collection, the implementation strategy would involve the steps described in Algorithm 10.

Algorithm 10 Hybrid GNN-RL Model for Smart Bin Collection

- 1: **Initialization:**
- 2: **Initialize GNN** to model the smart bin network:
- 3: Define GNN architecture and parameters
- 4: Initialize GNN weights
- 5: **Integrate RL Agent** for route optimization:
- 6: Define RL agent architecture and policy
- 7: Initialize RL agent parameters
- 8: **Define Actions and Rewards** for collection efficiency
- 9: **repeat**
- 10: **Update GNN** with current bin fill levels:
- 11: Receive real-time waste level data for bins
- 12: Update GNN node features with real-time data
- 13: **Predict Full Bins** using GNN:
- 14: Apply GNN model to predict which bins are full
- 15: **RL Agent** decides the collection route based on GNN output:
- 16: RL agent processes GNN's predictions and selects a collection route
- 17: Define actions for the RL agent (e.g., bin selection and order)
- 18: **Execute Collection** following RL agent's route:
- 19: Waste collection is performed according to the RL agent's selected route
- 20: **Receive Reward** based on route efficiency:
- 21: Evaluate the efficiency of the collection route based on predefined criteria
- 22: Calculate a reward signal indicating the route's performance
- 23: **RL Agent Updates** policy based on reward feedback:
- 24: The RL agent updates its policy through RL, learning from reward feedback
- 25: **Retrain GNN** periodically with new data:
- 26: Accumulate new data from waste collection operations
- 27: Periodically retrain the GNN model to adapt to changing conditions
- 28: **until** end of operation

The combination of GNN with RL in smart bin collection is discussed in the literature (see, for example, [59], where the authors focus on control policies such as offloading, routing, and resource allocation). Models of this sort can be employed to study complex spatial relationships and optimize dynamic collection routes. However, this approach has limitations, such as high computational demands, potential overfitting in GNN to specific bin patterns, and the challenge of RL in handling large state spaces. Overcoming these involves optimizing model architectures, incorporating regularization techniques, and applying efficient learning algorithms. Additionally, simplifying the RL problem or using hierarchical approaches can make the system more manageable and effective. GNNs have a space complexity of $O(N)$, where N is the number of nodes in the graph. Concerning the RL agent, the space complexity is determined by its architecture, including the number of states, actions, and policy parameters. Depending on the RL algorithm used, this complexity can vary. For example, Q-learning might require storing a Q-table with dimensions related to the number of states and actions, while policy gradient methods might involve parameter vectors. Fathinezhad et al. [60] explore the structural and computational intricacies involved in such a paradigm, underscoring its potential modeling of multi-agent and multi-task goals. This insight can substantiate a short discussion regarding the complexity of such a hybrid model. Storing the collection route's complexity depends on the number of bins marked as collection targets. In the worst case, if all bins are marked the space complexity for the collection route would be $O(n)$. In general, the space complexity of the hybrid algorithm can be approximated as $O(N + M)$, where N is the space complexity of the GNN model and M represents the space complexity of the RL agent and the collection route.

Regarding the time complexity, Munikoti et al. [61] provide a comprehensive review of the hybridization of DRL and GNN, highlighting how this combination can lead to increased generalizability and a reduction in computational complexity. Moreover, ref. [62]

discuss how models such as the Deep Q-learning (DQN) utilize deep neural networks to approximate Q-values, which represent the expected rewards for specific actions in given states. The study explores the trade-off between expressivity and computational cost, noting that achieving higher expressivity in models can exponentially increase computational demands. The work introduces a framework where DQN is adapted to optimize the identification of the most discriminative subgraphs, aiming to balance expressivity and computational efficiency.

Again, this insight can support some considerations for the specific case study. In particular, initializing the GNN model and the RL agent is typically done offline and does not significantly contribute to the runtime of the algorithm. However, the algorithm involves real-time updates, including updating node features with waste level data ($O(nk)$), applying the GNN model for predictions ($O(N)$), RL agent decision-making (depends on RL algorithm and model complexity), and route execution (depends on route length and vehicle speed). The time complexity for evaluating the efficiency of the collection route and calculating rewards depends on the specific criteria used and the size of the route. This complexity is application-dependent. On the other hand, the RL agent updates its policy based on reward feedback, which depends on the RL algorithm used. The update process can vary in complexity, but it is typically efficient. Retraining the GNN model periodically with new data is an offline process and does not impact the real-time operation of the algorithm.

In summary, the time complexity of the hybrid algorithm is mainly determined by the real-time updates (including GNN predictions, RL agent decision-making, and route execution) and reward calculation, which can be influenced by factors such as the size of the graph and the specific RL algorithm used. The initialization and periodic GNN retraining are typically one-time or periodic tasks that do not significantly affect the runtime performance.

6.4.2. Hybrid Models: GNN and Time Series

Another ML technique that could effectively complement a GNN in addressing the smart bin collection issue is time-series forecasting. This approach can predict future bin fill levels based on historical data trends. By integrating time-series forecasting as per Algorithm 11, the system can not only respond to current fill levels (as identified by the GNN) but also anticipate when bins will likely become full. This predictive capability can enhance the efficiency of collection routes, allowing for proactive management of waste collection before bins reach critical levels.

A hybrid approach combining GNN and time-series forecasting for smart bin collection has limitations, such as potential overfitting in GNN to specific data patterns, complexity in integrating time-series predictions with GNN outputs, and the challenge of forecasting accuracy for time series in rapidly changing environments. Moreover, the computational demands for both models can be significant, especially for large networks.

In addition to the GNN model complexity ($O(N)$), the space complexity of the time-series model depends on its specific architecture and the number of parameters used for forecasting. It can vary but is often manageable. The space complexity for storing historical bin data depends on the amount of data collected over time and the number of features associated with each data point. If m represents the number of historical data points and k represents the number of features per data point, the space complexity for historical data storage is $O(mk)$. Also in this scenario, in the worst case, if all bins are marked for collection the space complexity would be $O(n)$.

Overall, the space complexity of the hybrid algorithm can be approximated as $O(N + M + mk)$, where N is the space complexity of the GNN model, M represents the space complexity of the time-series model, m is the number of historical data points, k is the number of features per data point, and n is the number of bins.

Algorithm 11 Hybrid GNN-Time Series Model for Smart Bin Collection

- 1: **Initialization:**
 - 2: Initialize the GNN for analyzing the current bin status.
 - 3: Set up the time-series model for predicting future bin fill levels.
 - 4: **Input Historical Bin Data** into the time-series model.
 - 5: **repeat**
 - 6: **Update GNN** with real-time bin fill levels:
 - 7: Receive real-time data on bin fill levels.
 - 8: Update the GNN with current bin fill data.
 - 9: **Forecast Future Fill Levels** using the time-series model.
 - 10: **Combine GNN and time-series output** to identify bins for collection:
 - 11: Combine predictions from the GNN and the time-series model to assess which bins are likely to reach capacity.
 - 12: **Plan Collection Route** prioritizing bins likely to be full:
 - 13: Use the combined output to plan an efficient collection route, giving priority to bins expected to be full soon.
 - 14: **Execute Waste Collection** following the planned route:
 - 15: Perform waste collection based on the planned route to empty bins efficiently.
 - 16: **Collect New Data** on bin fill levels post-collection:
 - 17: After collection, record new data on bin fill levels to update the models.
 - 18: **Retrain GNN and time-series model** periodically with updated data:
 - 19: Periodically update and refine the GNN and time-series model using the newly collected data to improve accuracy.
 - 20: **until** end of operation.
-

Again, initializing the GNN model and setting up the time-series model is usually done offline and does not impact the runtime of the process. The algorithm involves real-time updates, including updating the GNN with real-time bin fill levels ($O(nk)$), forecasting future fill levels using the time-series model (depends on the forecasting method), combining GNN and time-series output ($O(n)$), planning the collection route (typically $O(N^2)$ or better, depending on optimization techniques), and executing waste collection (depends on route length and vehicle speed). Collecting new data on bin fill levels and periodically retraining the GNN and time-series model are offline processes and do not impact the real-time operation of the algorithm.

Overall, the time complexity of the hybrid algorithm is primarily determined by the real-time updates, including GNN updates, forecasting, route planning, and waste collection. The algorithm's efficiency can be influenced by the choice of GNN and time-series model architectures, as well as the specific optimization techniques used for route planning.

The initialization and periodic data collection and retraining are typically one-time or periodic tasks that do not significantly affect the runtime performance. The choice of specific forecasting methods and their optimizations can further refine the actual runtime characteristics of the algorithm in practice.

6.4.3. Hybrid Models: GNN and GAs

Pairing a GNN with GAs could offer a robust approach for optimizing smart bin collection. GAs, known for their ability to find solutions in complex search spaces, could refine and optimize the parameters or structure of the GNN. For instance, they could help in determining the most effective network topology or in tuning hyperparameters. This hybrid approach could enhance the GNN's performance in accurately predicting full bins and improving the overall efficiency of the waste collection routes. The evolutionary nature of GAs adds a layer of optimization that adapts to the dynamically changing scenarios of urban waste management. The method is described by Algorithm 12.

Algorithm 12 Hybrid GNN-GA Model for Smart Bin Collection

- 1: **Initialization:**
- 2: Initialize the GNN for predicting bin fullness. The GNN architecture, including the number of layers and parameters, is set up.
- 3: Define the GA population, consisting of various GNN configurations or individuals. Each individual represents a potential GNN configuration.
- 4: **GA Iterations:**
- 5: **for** each generation in the GA **do**
- 6: **for** each individual (GNN configuration) in the population **do**
- 7: **Evaluate GNN:** Execute the GNN with the given configuration and assess its performance using a fitness function. The fitness function measures how accurately the GNN predicts bin fullness.
- 8: **end for**
- 9: **Select Best-Performing Configurations:** Identify the best-performing GNN configurations based on their fitness scores. These configurations represent the most promising candidates for accurate bin fullness prediction.
- 10: **Apply Genetic Operators:** Utilize genetic operators such as crossover (combining attributes of two configurations) and mutation (introducing random changes) to create a new generation of GNN configurations. This mimics the natural selection and evolution process.
- 11: **end for**

The space complexity of the GA population depends on the number of individuals (GNN configurations) in each generation. If there are P individuals and each individual has a fixed space complexity related to GNN configuration, it is possible to denote the space complexity of the population as $O(P)$. As a result, the space complexity can be approximated by Algorithm 12 as $O(N + P)$, where N is the space complexity of the GNN model and P is the space complexity of the GA population.

The time complexity of running the GA primarily depends on the number of generations (iterations) and the number of individuals evaluated in each generation. Let G denote the number of generations and P the number of individuals. The time complexity of a single generation, including GNN evaluation, selection, and genetic operators, can be represented as $O(P)$. The time complexity of evaluating each GNN configuration involves executing the GNN with a particular set of hyperparameters and assessing its performance using a fitness function. If evaluating a single GNN configuration takes $O(M)$ time, where M represents the evaluation time, the total evaluation time for all P individuals in a generation is $O(PM)$. The time complexity for selecting the best-performing GNN configurations and applying genetic operators such as crossover and mutation depends on the specific methods used. Typically, this part of the algorithm is efficient and can be represented as $O(P)$. Finally, selecting the best GNN configuration for deployment as the optimized model for real-time smart bin collection optimization takes constant time and does not significantly impact the overall time complexity. In conclusion, the time complexity of the algorithm is determined by the number of generations, G , and the evaluation time for each GNN configuration, represented as $O(GPM)$. Table 8 summarizes the space and time complexity regarding the hybrid models.

Table 8. Space and time complexities for hybrid algorithms.

Hybrid Algorithm	Space Complexity	Time Complexity
GNN and RL	$O(N + P + mk)$	$O(GPM)$
GNN and Time Series	$O(N + M + mk)$	$O(nk + NM^2)$
GNN and GAs	$O(N + P)$	$O(GPM)$

7. Conclusions

This paper effectively highlights the potential of hybrid models, particularly those integrating GNNs with classic and ML routing algorithms, in enhancing the efficiency of smart bin collection in urban landscapes.

Specifically, it acknowledges the strengths of individual methodologies while also highlighting the synergistic benefits of combining these approaches to address the dynamic and complex nature of urban waste management. Limitations are acknowledged in the realms of computational demand and real-time adaptability, underscoring the need for further refinement and optimization.

Future work is promising, especially in the empirical part, as the authors plan to focus on a real case of smart bin deployment in a town located in northern Italy, aiming to translate theoretical advancements into practical, real-world solutions.

Author Contributions: Conceptualization, E.B. and A.G.; methodology, E.B.; software; validation, A.G., A.P. and E.B.; formal analysis, A.G.; investigation, A.G.; resources, A.G.; data curation; writing—original draft preparation, A.G.; writing—review and editing, E.B. and A.P.; visualization, A.G.; supervision, E.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. França, R.P.; Monteiro, A.C.B.; Arthur, R.; Iano, Y. An overview of the machine learning applied in smart cities. In *Smart Cities: A Data Analytics Perspective*; Springer: Berlin/Heidelberg, Germany, 2021; pp. 91–111.
2. Mahamuni, C.V.; Sayyed, Z.; Mishra, A. Machine Learning for Smart Cities: A Survey. In Proceedings of the 2022 IEEE International Power and Renewable Energy Conference (IPRECON), Kollam, India, 16–18 December 2022; pp. 1–8. [\[CrossRef\]](#)
3. Gupta, A.; Gupta, S.; Memoria, M.; Kumar, R.; Kumar, S.; Singh, D.; Tyagi, S.; Ansari, N. Artificial Intelligence And Smart Cities: A Bibliometric Analysis. In Proceedings of the 2022 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COM-IT-CON), Faridabad, India, 26–27 May 2022; Volume 1, pp. 540–544. [\[CrossRef\]](#)
4. Zamponi, M.E.; Barbierato, E. The Dual Role of Artificial Intelligence in Developing Smart Cities. *Smart Cities* **2022**, *5*, 728–755. [\[CrossRef\]](#)
5. Soh, Z.H.C.; Al-Hami Husa, M.A.; Abdullah, S.A.C.; Shafie, M.A. Smart Waste Collection Monitoring and Alert System via IoT. In Proceedings of the 2019 IEEE 9th Symposium on Computer Applications & Industrial Electronics (ISCAIE), Kota Kinabalu, Malaysia, 27–28 April 2019; pp. 50–54. [\[CrossRef\]](#)
6. Catarinucci, L.; Colella, R.; Consalvo, S.I.; Patrono, L.; Rollo, C.; Sergi, I. IoT-Aware Waste Management System Based on Cloud Services and Ultra-Low-Power RFID Sensor-Tags. *IEEE Sens. J.* **2020**, *20*, 14873–14881. [\[CrossRef\]](#)
7. Likotiko, E.D.; Nyambo, D.; Mwangoka, J. Multi-Agent Based IoT Smart Waste Monitoring and Collection Architecture. *Int. J. Comput. Sci. Eng. Inf. Technol.* **2017**, *7*, 1–14. [\[CrossRef\]](#)
8. Chowdhury, B.; Chowdhury, M.U. RFID-based real-time smart waste management system. In Proceedings of the 2007 Australasian Telecommunication Networks and Applications Conference, Christchurch, New Zealand, 2–5 December 2007; pp. 175–180. [\[CrossRef\]](#)
9. Kumari, N.; Pandey, S.; Pandey, A.K.; Banerjee, M. Role of Artificial Intelligence in Municipal Solid Waste Management. *Br. J. Multidiscip. Adv. Stud.* **2023**, *4*, 5–13. [\[CrossRef\]](#)
10. Sigongan, J.; Sinodlay, H.; Cuizon, S.X.; Redondo, J.; Macapulay, M.; Bulahan-Undag, C.; Gumonan, K.M.V. GULP: Solar-Powered Smart Garbage Segregation Bins with SMS Notification and Machine Learning Image Processing. *Int. J. Comput. Sci. Res.* **2023**, *7*, 2018–2036. [\[CrossRef\]](#)
11. Ghahramani, M.; Zhou, M.; Molter, A.; Pilla, F. IoT-Based Route Recommendation for an Intelligent Waste Management System. *IEEE Internet Things J.* **2022**, *9*, 11883–11892. [\[CrossRef\]](#)
12. Nagesh, U.B.; Kotari, M.; Chethan, S.C. Integration of MQTT Protocol with Map APIs for Smart Garbage Management. In Proceedings of the 2021 IEEE International Conference on Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER), Udipi, India, 19–20 November 2021; pp. 145–150. [\[CrossRef\]](#)
13. Watson, R.B.; Ryan, P.J. Visualization and Waste Collection Route Heuristics of Smart Bins Data using Python Big Data Analytics. In Proceedings of the 2021 4th International Conference on Software Engineering and Information Management, Yokohama, Japan, 16–18 January 2021; ACM: New York, NY, USA, 2021; pp. 124–130. [\[CrossRef\]](#)
14. Norhafezah, K.; Nurfadzliana, A.; Megawati, O. Simulation of municipal solid waste route optimization by Dijkstra’s algorithm. *J. Fundam. Appl. Sci.* **2017**, *9*, 732–747. [\[CrossRef\]](#)

15. Priyadarshi, M.; Maratha, M.; Anish, M.; Kumar, V. Dynamic routing for efficient waste collection in resource constrained societies. *Sci. Rep.* **2023**, *13*, 2365. [[CrossRef](#)] [[PubMed](#)]
16. Barth, L.; Schweiger, L.; Benedech, R.; Ehrat, M. From data to value in smart waste management: Optimizing solid waste collection with a digital twin-based decision support system. *Decis. Anal. J.* **2023**, *9*, 100347. [[CrossRef](#)]
17. Liang, Y.C.; Minanda, V.; Gunawan, A. Waste collection routing problem: A mini-review of recent heuristic approaches and applications. *Waste Manag. Res.* **2022**, *40*, 519–537. [[CrossRef](#)]
18. Cha, G.W.; Moon, H.J.; Kim, Y.C. A hybrid machine-learning model for predicting the waste generation rate of building demolition projects. *J. Clean. Prod.* **2022**, *375*, 134096. [[CrossRef](#)]
19. Lilhore, U.K.; Simaiya, S.; Dalal, S.; Damaševičius, R. A smart waste classification model using hybrid CNN-LSTM with transfer learning for sustainable environment. *Multimed. Tools Appl.* **2023**, 1–25. [[CrossRef](#)]
20. Zhang, H.; Cao, H.; Zhou, Y.; Gu, C.; Li, D. Hybrid deep learning model for accurate classification of solid waste in the society. *Urban Clim.* **2023**, *49*, 101485. [[CrossRef](#)]
21. Arunkumar, M.; Sathishkumar, P.; Suguna, R.; Deepa, S. An Internet of Things based Waste Management System using Hybrid Machine Learning Technique. In Proceedings of the 2022 6th International Conference on Electronics, Communication and Aerospace Technology, Coimbatore, India, 1–3 December 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 438–443.
22. Walter, V.; Kada, M.; Chen, H. Shortest path analyses in raster maps for pedestrian navigation in location based systems. In Proceedings of the International Symposium on “Geospatial Databases for Sustainable Development”, Goa, India, 27–30 September 2006.
23. Taillandier, P.; Banos, A.; Drogoul, A.; Gaudou, B.; Marilleau, N.; Truong, Q.C. Simulating Urban Growth with Raster and Vector Models: A Case Study for the City of Can Tho, Vietnam. In Proceedings of the Autonomous Agents and Multiagent Systems, Singapore, 9–13 May 2016; Osman, N., Sierra, C., Eds.; Springer: Cham, Switzerland, 2016; pp. 154–171.
24. Chen, D.; Zhong, Y.; Zheng, Z.; Ma, A.; Lu, X. Urban road mapping based on an end-to-end road vectorization mapping network framework. *ISPRS J. Photogramm. Remote Sens.* **2021**, *178*, 345–365. [[CrossRef](#)]
25. Jiao, C.; Heitzler, M.; Hurni, L. A novel framework for road vectorization and classification from historical maps based on deep learning and symbol painting. *Comput. Environ. Urban Syst.* **2024**, *108*, 102060. [[CrossRef](#)]
26. Jiang, B.; Liu, C. Street-based topological representations and analyses for predicting traffic flow in GIS. *Int. J. Geogr. Inf. Sci.* **2009**, *23*, 1119–1137. [[CrossRef](#)]
27. Spadon, G.; Gimenes, G.; Rodrigues, J.F. Topological Street-Network Characterization Through Feature-Vector and Cluster Analysis. In Proceedings of the Computational Science—ICCS 2018, Wuxi, China, 11–13 June 2018; Shi, Y., Fu, H., Tian, Y., Krzhizhanovskaya, V.V., Lees, M.H., Dongarra, J., Sloot, P.M.A., Eds.; Springer: Cham, Switzerland, 2018; pp. 274–287.
28. Song, Q.; Wang, X. Efficient Routing on Large Road Networks Using Hierarchical Communities. *IEEE Trans. Intell. Transp. Syst.* **2011**, *12*, 132–140. [[CrossRef](#)]
29. Jeong, Y.; Jang, H.; Yoon, B. Developing a risk-adaptive technology roadmap using a Bayesian network and topic modeling under deep uncertainty. *Scientometrics* **2021**, *126*, 3697–3722. [[CrossRef](#)]
30. Alterovitz, R.; Simeon, T.; Goldberg, K. The Stochastic Motion Roadmap: A Sampling Framework for Planning with Markov Motion Uncertainty. In Proceedings of the Robotics: Science and Systems, Atlanta, GA, USA, 27–30 June 2007.
31. Vitalis, S.; Arroyo Otori, K.; Stoter, J. Incorporating Topological Representation in 3D City Models. *ISPRS Int. J. Geo-Inf.* **2019**, *8*, 347. [[CrossRef](#)]
32. Banerjee, N.; Chakraborty, S.; Raman, V. Improved space efficient algorithms for BFS, DFS and applications. In Proceedings of the International Computing and Combinatorics Conference, Ho Chi Minh City, Vietnam, 2–4 August 2016; Springer: Berlin/Heidelberg, Germany, 2016; pp. 119–130.
33. Goldberg, A.V.; Radzik, T. *A Heuristic Improvement of the Bellman-Ford Algorithm*; Citeseer: Pennsylvania State University: State College, PA, USA, 1993.
34. AbuSalim, S.W.; Ibrahim, R.; Saringat, M.Z.; Jamel, S.; Wahab, J.A. Comparative analysis between dijkstra and bellman-ford algorithms in shortest path optimization. In Proceedings of the IOP Conference Series: Materials Science and Engineering, Chennai, India, 16–17 September 2020; IOP Publishing: Bristol, UK, 2020; Volume 917, p. 012077.
35. Awerbuch, B.; Bar-Noy, A.; Gopal, M. Approximate distributed bellman-ford algorithms. *IEEE Trans. Commun.* **1994**, *42*, 2515–2517. [[CrossRef](#)]
36. Banerjee, P.; Kumar, P.; Kumar, B.; Thakur, K. A New Proposed Modified Shortest Path Algorithm’s Using Dijkstra’s. In Proceedings of the 2023 International Conference on Advances in Computing, Communication and Applied Informatics (ACCAI), Chennai, India, 25–26 May 2023; IEEE: Piscataway, NJ, USA, 2023; pp. 1–8.
37. Qing, G.; Zheng, Z.; Yue, X. Path-planning of automated guided vehicle based on improved Dijkstra algorithm. In Proceedings of the 2017 29th Chinese Control and Decision Conference (CCDC), Chongqing, China, 28–30 May 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 7138–7143.
38. Deng, Y.; Chen, Y.; Zhang, Y.; Mahadevan, S. Fuzzy Dijkstra algorithm for shortest path problem under uncertain environment. *Appl. Soft Comput.* **2012**, *12*, 1231–1237. [[CrossRef](#)]
39. Wang, X.; Zhang, H.; Liu, S.; Wang, J.; Wang, Y.; Shangguan, D. Path planning of scenic spots based on improved A* algorithm. *Sci. Rep.* **2022**, *12*, 1320. [[CrossRef](#)]

40. Xiang, D.; Lin, H.; Ouyang, J.; Huang, D. Combined improved A* and greedy algorithm for path planning of multi-objective mobile robot. *Sci. Rep.* **2022**, *12*, 13273. [[CrossRef](#)]
41. Zhou, J.; Cui, G.; Hu, S.; Zhang, Z.; Yang, C.; Liu, Z.; Wang, L.; Li, C.; Sun, M. Graph neural networks: A review of methods and applications. *AI Open* **2020**, *1*, 57–81. [[CrossRef](#)]
42. Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; Bengio, Y. Graph Attention Networks. *arXiv* **2018**, arXiv:stat.ML/1710.10903.
43. Hamilton, W.; Ying, Z.; Leskovec, J. Inductive representation learning on large graphs. In Proceedings of the Advances in Neural Information Processing Systems 30 (NIPS 2017), Long Beach, CA, USA, 4–9 December 2017; Volume 30.
44. Scarselli, F.; Gori, M.; Tsoi, A.C.; Hagenbuchner, M.; Monfardini, G. The Graph Neural Network Model. *IEEE Trans. Neural Netw.* **2009**, *20*, 61–80. [[CrossRef](#)]
45. Holland, J.H. Genetic Algorithms. *Sci. Am.* **1992**, *267*, 66–73. [[CrossRef](#)]
46. Holland, J.H. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*; The MIT Press: Cambridge, MA, USA, 1992. [[CrossRef](#)]
47. Srinivas, M.; Patnaik, L. Genetic algorithms: A survey. *Computer* **1994**, *27*, 17–26. [[CrossRef](#)]
48. Forrest, S. Genetic algorithms. *ACM Comput. Surv.* **1996**, *28*, 77–80. [[CrossRef](#)]
49. Mitchell, M. Genetic algorithms: An overview. *Complexity* **1995**, *1*, 31–39. [[CrossRef](#)]
50. Fujdiak, R.; Masek, P.; Mlynek, P.; Misurec, J.; Olshannikova, E. Using genetic algorithm for advanced municipal waste collection in Smart City. In Proceedings of the 2016 10th International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP), Prague, Czech Republic, 20–22 July 2016; pp. 1–6. [[CrossRef](#)]
51. Ikram, S.T.; Mohanraj, V.; Ramachandran, S.; Balakrishnan, A. An Intelligent Waste Management Application Using IoT and a Genetic Algorithm—Fuzzy Inference System. *Appl. Sci.* **2023**, *13*, 3943. [[CrossRef](#)]
52. Ochelska-Mierzejewska, J.; Poniszewska-Marañda, A.; Marañda, W. Selected Genetic Algorithms for Vehicle Routing Problem Solving. *Electronics* **2021**, *10*, 3147. [[CrossRef](#)]
53. Šiljak, D. Dynamic graphs. *Nonlinear Anal. Hybrid Syst.* **2008**, *2*, 544–567. [[CrossRef](#)]
54. Feng, J.; Kong, L.; Liu, H.; Tao, D.; Li, F.; Zhang, M.; Chen, Y. Towards Arbitrarily Expressive GNNs in $O(n^2)$ Space by Rethinking Folklore Weisfeiler-Lehman. *arXiv* **2023**, arXiv:2306.03266.
55. Hadou, S.; Kanatsoulis, C.I.; Ribeiro, A. Space-time graph neural networks. *arXiv* **2021**, arXiv:2110.02880.
56. Ding, M.; Rabbani, T.; An, B.; Wang, E.; Huang, F. Sketch-GNN: Scalable Graph Neural Networks with Sublinear Training Complexity. *Adv. Neural Inf. Process. Syst.* **2022**, *35*, 2930–2943.
57. Wang, Z.; Zhang, Z.; Lee, C.Y.; Zhang, H.; Sun, R.; Ren, X.; Su, G.; Perot, V.; Dy, J.; Pfister, T. Learning to prompt for continual learning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, New Orleans, LA, USA, 18–24 June 2022; pp. 139–149.
58. Iman, M.; Arabnia, H.R.; Rasheed, K. A Review of Deep Transfer Learning and Recent Advancements. *Technologies* **2023**, *11*, 40. [[CrossRef](#)]
59. Tam, P.; Song, I.; Kang, S.; Ros, S.; Kim, S. Graph Neural Networks for Intelligent Modelling in Network Management and Orchestration: A Survey on Communications. *Electronics* **2022**, *11*, 3371. [[CrossRef](#)]
60. Fathinezhad, F.; Adibi, P.; Shoushtarian, B.; Chanussot, J. Graph Neural Networks and Reinforcement Learning: A Survey. In *Deep Learning and Reinforcement Learning*; Yang, J., Chen, Y., Zhao, T., Wang, Y., Pan, X., Eds.; IntechOpen: Rijeka, Croatia, 2023; Chapter 2. [[CrossRef](#)]
61. Munikoti, S.; Agarwal, D.; Das, L.; Halappanavar, M.; Natarajan, B. Challenges and opportunities in deep reinforcement learning with graph neural networks: A comprehensive review of algorithms and applications. *IEEE Trans. Neural Netw. Learn. Syst.* **2023**. [[CrossRef](#)]
62. Kong, L.; Feng, J.; Liu, H.; Tao, D.; Chen, Y.; Zhang, M. MAG-GNN: Reinforcement Learning Boosted Graph Neural Network. *arXiv* **2023**, arXiv:2310.19142.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.