Article

A Blockchain-Based Shared Bus Service Scheduling and Management System

Tengfei Li [1], Xuanrui Xiong [1,*], Guifeng Zheng [1], Ying Li [2] and Amr Tolba [3]

1 School of Communications and Information Engineering, Chongqing University of Posts and Telecommunications, Chongqing 400065, China; s220132085@stu.cqupt.edu.cn (T.L.); s220132230@stu.cqupt.edu.cn (G.Z.)
2 Information and Communication Branch of State Grid Inner Mongolia East Electric Power Co., Ltd., Huhehaote 010020, China; 32017098@mail.dlut.edu.cn
3 Department of Computer Science, Community College, King Saud University, Riyadh 11437, Saudi Arabia; atolba@ksu.edu.sa
* Correspondence: xiongxr@cqupt.edu.cn

Abstract: With the continuous development of urbanization, it has become an important issue to effectively alleviate urban road traffic congestion and improve traffic efficiency. By combining blockchain technology and shared buses, this paper builds an intelligent traffic-service scheduling management system based on blockchain. The system effectively solves the core problems of shared buses, improves data security and privacy protection, realizes intelligent scheduling and route planning, and simplifies cross-organization cooperation and settlement processes. The research shows that the system can reduce the distance and number of buses, and improve the service quality and operation efficiency while ensuring the waiting time of passengers. The results of this paper verify the feasibility and advantages of the system, bring innovation and improvement to the field of traffic management, and promote the sustainable development of urban intelligent traffic management system. Future research could further explore the application of blockchain technology in traffic management to achieve more intelligent and sustainable urban traffic management.

Keywords: blockchain; data security; intelligent transportation; service scheduling

check for updates

Citation: Li, T.; Xiong, X.; Zheng, G.; Li, Y.; Tolba, A. A Blockchain-Based Shared Bus Service Scheduling and Management System. *Sustainability* **2023**, *15*, 12516. https://doi.org/10.3390/su151612516

Academic Editor: Assunta Di Vaio

Received: 4 July 2023
Revised: 15 August 2023
Accepted: 15 August 2023
Published: 17 August 2023

1. Introduction

Transportation is a complex issue that encompasses various aspects such as politics, the economy, and society and plays a vital role in the development of countries worldwide. For the transportation industry, it is crucial to expedite the adoption of green and low-carbon transportation methods, strengthen the development of green infrastructure, and promote the use of new energy and intelligent, digital, and lightweight transportation equipment to foster environmentally friendly transportation and facilitate low-carbon travel. According to statistics from the Ministry of Public Security, as of the end of November 2022 [1], the total number of motor vehicles in China reached 415 million, representing a significant increase compared to the 395 million recorded at the end of 2021. This translates to an approximate 1.05-fold rise in the number of motor vehicles within just one year. The rapid growth in the number of motor vehicles has exacerbated the supply–demand imbalance between existing road traffic resources and urban transportation demands, leading to increasingly complex challenges in traffic dispatching and escalating concerns such as data leakage.

To align with the ever-growing demands for transportation capacity driven by the rapidly evolving economic landscape and alleviate urban traffic congestion, the establishment of a robust and intelligent transportation system [2] has emerged as a crucial factor in urban development and the creation of smart cities. With the launch and widespread adoption of 5G technology, its high reliability and low-latency communication capabilities have greatly propelled the rapid growth of the connected vehicle industry. As a significant application area within the 5G infrastructure, connected vehicles are set to become a core component of future intelligent transportation systems [3]. Present-day vehicles are equipped with a multitude of communication systems and sensing devices, transforming them into intelligent vehicles. During their journeys, intelligent vehicles are tasked with a wide range of responsibilities and the need to communicate with various entities within the traffic environment. However, the latency requirements, throughput demands, and reliability expectations vary significantly depending on the specific tasks at hand. As connected vehicle technology and autonomous driving capabilities continue to advance, intelligent vehicles are equipped with more powerful sensors, advanced signal processing techniques, increased computational resources, and enhanced communication capabilities. The capacity of vehicle-mounted sensors to capture extensive environmental information is continuously strengthening. The massive interaction and fusion of perceptual information from diverse vehicles form a network rich in sensing data. Nevertheless, as the volume of perceptual information and data expands, the demand for information processing power continues to rise. Efficiently handling vast amounts of perceptual data, performing real-time analysis, and making prompt decisions require robust computational capabilities and efficient algorithms.

In this regard, the introduction of blockchain technology [4] has presented a unique solution for intelligent transportation. Shared buses, as a novel mode of transportation in line with the sharing economy trend, have gradually entered the public's consciousness, garnering attention and favor. Serving as a link between surrounding communities and pick-up locations, shared buses cater to users with commuting needs between transfer points and residential areas. Passengers can conveniently book their trips in advance, while shared bus companies flexibly adjust bus routes, schedules, and stops based on passenger demand [5] to provide tailored services. The shared bus covers the blind spot of public transportation in space, is more flexible than public transportation in time, meets the personalized travel needs of passengers from the perspective of passengers, takes into account the advantages of public transportation economy and the advantages of traditional taxi flexibility, and can improve the service quality of urban transportation under the premise of energy saving and environmental protection.

Various approaches to traffic management exist in the current research field, but there is limited research on combining blockchain with shared bus services to address specific challenges in the transportation sector. With this unique combination, we are committed to revolutionizing the way transport services are operated and managed. A comprehensive review of the key literature in the field has been conducted, and it has been found that there is a research gap in applying blockchain technology to the improvement of shared bus services. This study aims to fill this gap and provide a comprehensive solution for improving the efficiency and reliability of urban transportation.

1.1. Motivation

However, based on the practical operation of shared buses, there is still a lack of sufficient integration between theoretical research and practical application. Various issues persist, such as low passenger occupancy rates, excessively long waiting times for passengers, and underutilization of bus resources. To address these challenges, shared bus companies are urgently seeking a solution that can simultaneously reduce passenger waiting times to enhance the travel experience and improve bus resource utilization to lower operational costs.

Based on the above research background, in order to promote the development of the shared bus and its related applications, this paper focuses on the contradiction between the operating cost of shared bus companies and the travel experience of passengers. This paper proposes a dynamic shared bus service scheduling traffic management system based on blockchain technology, and sets up smart contracts that can automatically execute preset rules, such as assigning suitable buses according to route and vehicle and passenger demand, to achieve efficient scheduling and route planning. The combination of blockchain technology can ensure the security and privacy protection of traffic data, increase the trust and transparency of the shared bus system by recording route planning and scheduling schemes and related order data on the blockchain, and achieve a decentralized shared bus service scheduling system [6]. At the same time, a blockchain-based shared bus system can simplify the interaction and settlement process between participants, improving the efficiency and sustainability of the system. In addition, it also helps to innovate the sharing economy model and provide better experience and convenience for users.

1.2. Research Challenge

Traditional intelligent transportation systems rely on motor vehicle traffic data for road condition monitoring, but they often overlook the potential hazards caused by data leaks. While conventional encryption algorithms are used to secure traffic data, they have clear limitations as they cannot directly process and analyze encrypted data. To address these limitations, a blockchain-based encryption approach is introduced in the service scheduling management system [7]. In this system, each bus trip and its associated order information are recorded in a block on the blockchain. These blocks are then linked together to form a chain, where each block contains the hash value of the previous block, ensuring the integrity and continuity of the data. All participants, including shared bus companies, bus drivers, and passengers, can access and verify the information stored on the blockchain, thereby enhancing transparency and trust. This novel encryption approach offers distinct advantages over traditional methods as it eliminates the need for decryption, which would otherwise expose plaintext data to the data processing center and prolong the time required for traffic monitoring.

Although blockchain technology holds significant importance for the transformation and upgrading of the real economy, it still faces challenges during its practical implementation in industries. These challenges include information barriers, difficulties in onboarding physical assets onto the blockchain, and the need for advancements in underlying technologies. Incompatibility and integration issues arise from the diverse data systems of different organizations and entities, making data sharing and interoperability difficult. Blockchain applications associated with physical assets encounter technical hurdles in ensuring authenticity, traceability, and verification. Furthermore, there are technical challenges in the performance, scalability, and privacy protection of blockchain technology [8]. Public chains often suffer from slow transaction speeds, private chains face concerns regarding centralization and trust, and finding the right balance between privacy protection and effective data sharing is an ongoing challenge. The development of blockchain technology also raises legal, regulatory, and compliance concerns. The legal framework and regulatory policies have yet to keep pace with technological advancements, resulting in legal uncertainties and compliance risks.

To address these challenges, comprehensive solutions are particularly important. The government and relevant agencies should promote the establishment of unified data standards and formats to promote data consistency and interoperability. At the same time, IoT technology and sensors are used to bind physical information to digital assets on the blockchain to ensure physical authenticity and traceability. We continue to invest resources in the research and development of the underlying blockchain technology to improve performance and scalability while adopting privacy protection technology to protect personal privacy and data autonomy. Cross-industry cooperation and consensus is also a key part of the solution, and parties such as government, enterprise, and academia should strengthen cooperation to jointly research and develop standards and specifications for blockchain technology and promote the coordinated development of the blockchain ecosystem. In addition, resources have been continuously invested in the promotion and popularization of blockchain technology, improving public awareness and understanding of blockchain technology, and promoting the wide application of blockchain technology in various industries. Through the above comprehensive solutions, we are expected to overcome the challenges faced by blockchain technology in the real economy, realize the effective application of blockchain technology in transformation and upgrading, and make positive contributions to the sustainable development of society and economy.

In conclusion, further research and solutions are necessary to address the challenges of data sharing, interoperability [9], the association of physical assets with blockchain, performance, privacy protection, as well as legal and regulatory considerations in practical applications. With technology constantly evolving and practical implementation advancing, we anticipate the broad application of blockchain technology in intelligent traffic management and other sectors of the real economy. This will contribute to the realization of intelligent and sustainable urban transportation systems.

1.3. Contributions

The dynamic shared bus service scheduling and traffic management system based on blockchain technology can provide a safer, more transparent, and efficient shared bus service [10], effectively addressing the conflicts between the operational costs of shared bus companies and the passenger experience. This system has the potential to drive the development of shared bus services and serves as a technological reference for future intelligent traffic management. This article contributes in three main aspects:

- The relevant data, including the total distance traveled by buses, the average number of serviced passengers, and the required number of shared buses, will be recorded on the blockchain to validate the effectiveness of the bus scheduling and routing algorithm (BSA), along with the ant colony optimization (ACO) algorithm [11] and genetic algorithm (GA) [12];
- Using blockchain technology, a global bus scheduling and route planning algorithm is proposed. The arrival-data-based passenger assignment algorithm (ADPT) is implemented to dynamically schedule vehicles, aiming to obtain an optimal bus scheduling plan with the minimum number of buses required and the shortest distance traveled. The global bus scheduling and route planning algorithm is then encapsulated and integrated with blockchain technology to form a module within the system's algorithm, which can be called upon during the development of specific functional modules within the system;
- Based on the Spring Boot, Vue, and Flask frameworks, this blockchain-based intelligent transportation service scheduling and management system is designed and developed using communication technologies such as WebSocket. Its purpose is to provide a comprehensive solution for shared bus companies, integrating intelligent route planning and vehicle scheduling to optimize both the passenger experience and operational costs.

The remainder of this paper is organized as follows: Section 2 provides a review of related work, followed by Section 3, which presents the system's design. In Section 4, the implementation of the system is described, while Section 5 focuses on the testing of the system, which is discussed in Section 6. Finally, Section 7 concludes the paper.

2. Related Work

With the continuous development of the economy and advancements in technology, progress in information and communication technology has brought forth new possibilities across various fields, leading us into the era of big data. In this era, challenges such as traffic congestion have emerged as pressing issues that require immediate solutions. Traffic congestion is a widespread problem in urban areas. As the population grows and the number of vehicles increases, the burden on road networks continues to escalate, resulting in higher frequencies and greater severity of traffic congestion. The impact of traffic congestion goes beyond inconveniences in people's daily lives; it also has negative repercussions on the economy, environment, and social sustainability.

IoT has made substantial contributions in the field of intelligent transportation, encompassing real-time data collection and monitoring, vehicle tracking and safety, intelligent navigation, and smart traffic management [13–15]. The transmission, storage, and processing of large volumes of data in intelligent transportation systems can impose significant pressures on network bandwidth, storage capacity, and computing resources. To address the challenges faced by intelligent transportation systems, such as effectively allocating diverse vehicle applications and managing dynamically changing network resource states [16], the emergence of edge computing [17,18] networks has provided a promising solution for optimizing network resource consumption and improving performance. By establishing a distributed cooperative service framework [6], edge devices take on specific computing and data processing tasks, thereby reducing reliance on central cloud servers and alleviating network loads. This collaborative relationship can be viewed as a game scenario [19], where network operators and edge devices mutually depend on and influence each other, ultimately striving to achieve a Nash equilibrium [20]. Due to the uneven distribution of traffic flows, network operators must design intelligent offloading strategies to improve network performance [21]. By dynamically coordinating edge computing devices and content caching, lightweight computing services can be provided to users, ensuring high-quality service delivery [22–25]. On the other hand, fog computing [26] enables data processing at edge nodes, reducing the frequency and scope of data transmission to cloud servers, thereby enhancing data privacy and security [27]. For applications with higher requirements for data privacy and security, fog computing offers a more reliable solution [28]. The integration of fog computing with smart cities brings numerous benefits and innovations. By deploying edge computing nodes in transportation facilities and traffic nodes, real-time traffic monitoring and management can be achieved [29]. This allows for a more accurate understanding of traffic conditions and enables real-time traffic monitoring, optimization of traffic signals, and intelligent navigation in congested areas, thereby improving traffic efficiency, reducing congestion, and optimizing road network layouts [30]. Fog computing provides powerful computing and data processing capabilities for realizing smart cities. As an integration of fog computing and vehicular networks, vehicle fog computing holds the potential to achieve real-time and location-aware network responses.

Blockchain [31,32], as a decentralized and trustworthy storage technology, has attracted significant attention from governments, capital markets, and various industries. Unlike traditional centralized approaches, blockchain ensures decentralized, immutable, and secure transactions among nodes without the need for third-party institutions. It adopts distributed verification to achieve decentralization, immutability, and security [6]. In the context of intelligent transportation systems [33], secure and efficient data transmission holds great significance. Therefore, constructing a blockchain-based distributed traffic management framework [34] has emerged as a solution to support intelligent traffic management systems and decision-making processes. This framework utilizes the unique features of blockchain to facilitate fast data sharing and verification, thereby enhancing security [35–37] and preserving data privacy in vehicular mobile edge computing scenarios. It contributes to establishing a trusted, secure, and decentralized vehicular edge computing system [38], providing valuable support for intelligent traffic management and driving the development and innovation of intelligent transportation systems.

3. System Design

3.1. System Requirement Analysis

The blockchain-based shared bus platform [39] utilizes smart contracts and decentralized technology to ensure the immutability and transparency of transportation data, enabling the viewing and verification of optimal solutions. In this paper, a thorough analysis of requirements was conducted to identify the core functionalities, including bus scheduling, route planning [40], and dataset management. The system must address the management needs of users, orders, and bus information, with a strong focus on reducing operational costs for bus companies and enhancing the passenger experience. Specific functional requirements encompass bus management as well as bus scheduling and route planning. Users should be able to select a specific day's order to obtain the optimal bus scheduling solution, considering criteria such as minimizing passenger wait time and the number of buses required. Based on these requirements, a comprehensive system module diagram was designed, as depicted in Figure 1.

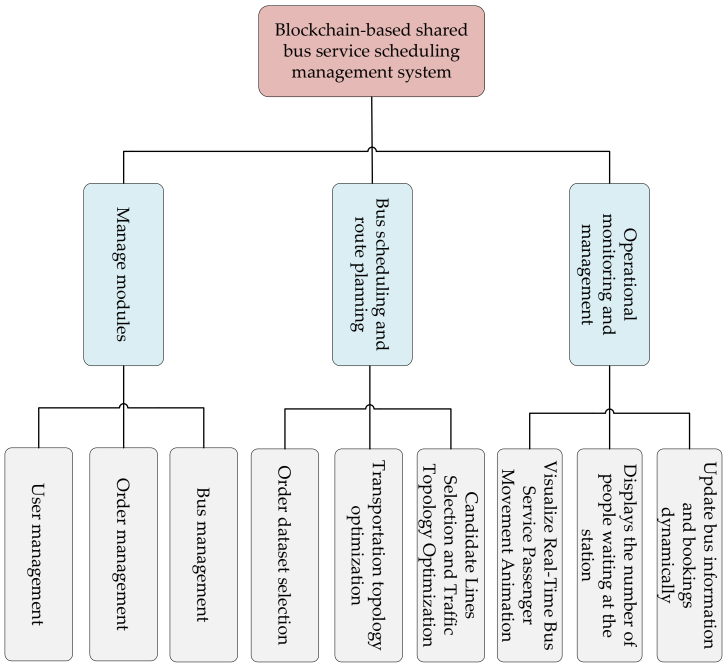

Blockchain-based shared bus service scheduling management system
Manage modules
Bus scheduling and route planning
Operational monitoring and management
User management
Order management
Bus management
Order dataset selection
Transportation topology optimization
Candidate Lines Selection and Traffic Topology Optimization
Visualize Real-Time Bus Service Passenger Movement Animation
Displays the number of people waiting at the station
Update bus information and bookings dynamically

Figure 1. Architecture Distribution of Blockchain-Based Shared Bus Service Scheduling and Management System.

3.2. Design of System Architecture

In order to ensure the security of business functions and data and consider the maintenance cost and development efficiency of the system, the system adopts the B/S (browser/server) architecture design with frontend and backend separation. This allows the system user to enter the system directly through the browser and easily use the system functions. At the same time, the database storage is used as a data storage mode, and the interface is not exposed to the outside world to ensure data security. In fact, the backend server provides services such as interfaces and algorithms for the frontend, making the data interaction and function invocation between the frontend and the backend smooth. The database storage [41] focuses on providing data access and data storage functions for the entire system for the backend, ensuring reliable storage and efficient management of data. This architecture is designed to provide an efficient, secure, and easy-to-maintain system environment that enables users to use the system smoothly and ensures data integrity and availability. The overall architecture of the system is depicted in Figure 2.

It can be mainly divided into three layers: frontend interface display layer, backend service support layer, and database storage layer.

(1) Front-end interface display layer

The frontend interface utilizes the lightweight Vue framework and incorporates improvements using blockchain technology. The Vue framework is flexible and lightweight, adopting an incremental development approach that reduces the learning curve and improves development efficiency. By leveraging blockchain technology, the system achieves decentralized [42] data storage and ensures data security and auditability. Additionally, blockchain is used for user identity authentication and access control, providing a secure user experience. The combination of the Vue framework and blockchain technology in the frontend interface ensures a balance between agility, usability, and data security, ultimately enhancing the overall user experience.

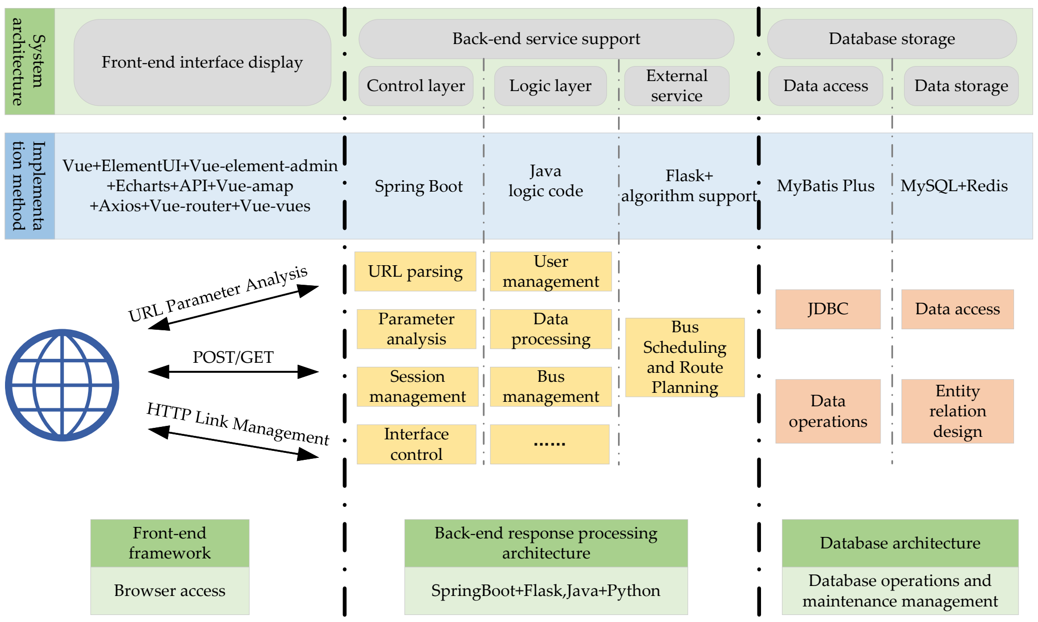


Figure 2. System overall architecture diagram.

(2) Back-end service support layer

The backend service is responsible for data management, algorithm support, and functional implementation, which is improved in conjunction with blockchain technology. Java and Spring Boot framework are used to develop the server side, and Maven is used for dependency management. SprinBoot and Flask are the technical frameworks used to build the backend server of the system. SprinBoot is a Java-based development framework for building powerful backend services and RESTful apis for communication and data transfer with the frontend. Flask is a Python lightweight Web application framework, mainly used to handle some simple backend logic and processing requests, to support the rapid implementation of some system functions. Java and Python play different roles in the system. Java is primarily used to handle complex algorithms and data processing, including core functions such as bus scheduling and route planning. Java's computing power and rich data processing libraries make it ideal for dealing with these complex tasks. Python is mainly used for some simple data processing and auxiliary functions, such as data visualization and animation display. Python's simplicity, ease of use, and rich graphics library make it great in these areas. These parts are tightly integrated in the system. SprinBoot and Flask, as backend server frameworks, jointly provide interfaces and services to the frontend system. Through such integration, the system can give full play to the advantages of each part and provide an efficient and stable intelligent traffic service scheduling management system. In terms of data management, blockchain is introduced as a distributed database to ensure that data are immutable and traceable. Algorithms support the execution of trusted algorithms via blockchain smart contracts. In terms of function implementation, the use of blockchain identity authentication technology to achieve user identity management and access control. The architecture simplifies configuration, provides out-of-the-box features and a good development experience. Blockchain technology enhances data security and trustworthiness, making systems more reliable and transparent.

(3) Database Storage Layer

This system utilizes the MySQL database and Redis storage system, combined with blockchain technology for further improvements. The MyBatis Plus framework is used for common CRUD (create, read, update, delete) operations. Blockchain is introduced as a trusted and distributed database, ensuring data immutability and traceability. Key data and operation records are stored on the blockchain, guaranteeing data security and reliability. The data are distributed across nodes in the blockchain network, with consensus algorithms maintaining data consistency. MySQL is responsible for storing conventional data, while blockchain access tools are employed for querying and updating blockchain data. JWT (JSON Web Tokens) technology is integrated with blockchain identity authentication to enhance security. Users register on the blockchain [43] and obtain a unique identity identifier for authentication and access control.

By introducing blockchain technology to enhance the credibility and transparency of data, the transparency of blockchain technology allows the source and historical changes of data to be traced, and users can better understand the source and processing of data. A distributed database provides a more secure data storage and management environment. Data are distributed across multiple nodes and hash encryption technology is used to protect data privacy and security. This makes the data less vulnerable to the risk of hacking or data breaches, providing better protection for users' data. Smart contract technology allows automated contract rules to be defined and enforced in advance. In this way, the system can automatically perform corresponding operations when certain conditions are met, so as to provide more efficient and fast services. Identity authentication technology can ensure the authenticity and uniqueness of user identities and avoid the abuse of false identities. At the same time, the adoption of identity authentication technology can also better protect the user's personal privacy and ensure that the user's personal information will not be abused or leaked.

3.3. System Detailed Design

3.3.1. Bus Management Module Design

The bus management module is mainly responsible for providing system users with management functions, such as query and modification of company bus data. The development of this function uses the Vue3.2.js framework, which is based on the idea of componentization so that developers can pay more attention to the development of frontend style modularization. In addition, the Vue3.2 framework also has the advantages of quick hand and low threshold for developers, which can effectively improve the development efficiency of developers. The Vue3.2 framework is also easy to integrate with third-party libraries or other completed projects. Reactive bidirectional data binding is a core of Vue3.2.js, and its implementation can reject frequent DOM operations on the frontend.

This module follows a defined workflow where users log in to the system, navigate to the bus management page, and interact with the backend to retrieve and display bus information from the database. The frontend presents the bus list to the user. Users can add new bus information or select existing buses for modification. The frontend provides a dedicated page for users to input new data, performs format validation, and offers prompts for accurate user input. Once the modifications are confirmed, the frontend submits the form information to the backend. The backend then interacts with the database to update the bus information and notifies the user of the successful modification.

Additionally, users have the ability to select and delete bus information. The frontend ensures multiple confirmation prompts before proceeding with the deletion. Upon user confirmation, the frontend sends the selected bus ID to the backend, which carries out the necessary database operations to delete the corresponding bus entry. Considering the significant impact of bus quantity on the accuracy of the scheduling module, the frontend ensures precise and error-free user interactions. Overall, the bus management module plays a crucial role in facilitating efficient management and control of bus data within the system, empowering users to make accurate modifications and deletions with confidence.

3.3.2. Design of the Bus Scheduling and Route Planning Module

The application of the Spring Boot v3.1 framework and Flask v2.3.3 framework to bus scheduling and route planning module design can provide a comprehensive and efficient solution. These two frameworks are suitable for Java and Python languages, respectively, for backend development and frontend development, and they can work together to build a complete bus scheduling and route planning system. In the bus scheduling system, Spring Boot v3.1 can be used to build backend apis to deal with bus scheduling logic, route planning algorithms and other factors. The powerful features of Spring Boot v3.1 include dependency management, automatic configuration, RESTful API support, and more, which help improve development efficiency and system performance. Flask v2.3.3 is a lightweight Python Web framework for quickly building Web applications. In the bus scheduling system, Flask v2.3.3 can be used to build a frontend interface to display bus routes, passenger information, scheduling results, etc. Flask provides easy-to-use routing and template functions to facilitate page rendering and user interaction. Spring Boot v3.1 is responsible for handling complex scheduling algorithms and business logic, while Flask v2.3.3 is responsible for frontend display and user interaction, which cooperate with each other to form a complete bus scheduling and route planning system.

The bus scheduling and route planning module is a crucial component of the system that provides optimal bus scheduling solutions based on the user-selected date, order dataset, existing buses, and parameters. The main objective is to minimize the number of buses and travel distance while ensuring the maximum waiting time for passengers. This section focuses on the detailed design of this module, specifically the algorithmic strategies employed. The following is an explanation of the algorithms used.

After logging into the system, users can select the "Algorithm Execution" page. They are required to input parameters such as the date, number of bus seats, and maximum waiting time for passengers. The frontend then passes this parameter list to the backend. The backend is responsible for executing the algorithms based on the provided parameters. After each algorithm execution, the resulting bus scheduling solution is stored in the Redis cache to improve query efficiency. The description above provides an overview of the module's functionality and design from a macro perspective. The core aspect of this module lies in the implementation of the algorithms. To achieve the system's objectives, this paper adopts the ADPT algorithm as the primary algorithm. The following section will provide a detailed explanation of the algorithm, its utilization, and its design within the system.

For the route planning problem in the context of this system [44], the first step is to perform preprocessing operations based on the site map information. This step aims to obtain a bus stop route optimization map that is suitable for solving the problem.

(1) Preprocessing of Station Routes

The original abstraction method is utilized to abstract the existing real-world transportation station map. The original abstraction method is a method to simplify and abstract the complex traffic station diagram in the real world. Its purpose is to describe the road network form between stations more intuitively by transforming the spatial position relationships, such as stations, sections, and intersections, into topological structures. In this process, stations can be abstracted as critical network nodes and intersections as critical nodes, and sections of the road between nodes can be represented by connecting edges [45]. The latitude and longitude coordinates of the intersection nodes are extracted and stored for future use. Taking the UK shared bus dataset used in this study as an example, the process is illustrated in Figure 3. The stations on the left side of the figure are abstracted as key network nodes ($V_1, V_2, V_3, \ldots, V_9$), while the intersections are represented as black key nodes on the right side of the figure. The street segments connecting the nodes are abstracted as edges in the station topology graph. Notably, stations V_8 and V_9 serve as departure points, and V_2 functions as the destination station. By transforming the map of the real world into an abstract topology and applying the information of these key nodes and edges to the blockchain-based intelligent traffic service scheduling management system, this abstract way enables the system to better deal with the complex traffic network and quickly and accurately find the optimal vehicle scheduling and route planning scheme, which brings many benefits to the field of traffic management. This includes reducing operating costs, improving vehicle utilization, ensuring the passenger travel experience, and making an important contribution to improving the quality and efficiency of transportation services.

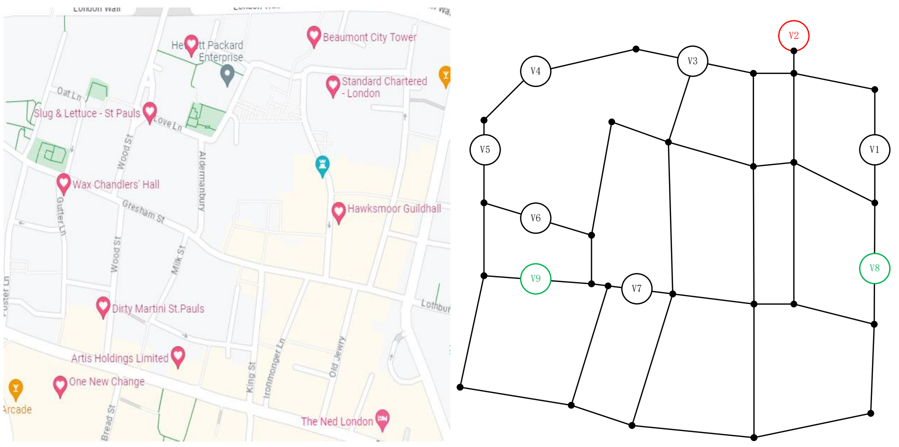


Figure 3. Visualization of Station Road Network Using the Original Abstraction Method.

Based on the existing topology graph of stations, as well as the latitude and longitude information of stations and intersection nodes, the distances between each edge of the topology graph are calculated using the Python programming language and then saved. The distance calculation formulas from point V to point U are represented by Equation (1):

$$\|V-U\| = 2\times 6371.004\times \sin^{-1}\sqrt{\sin\left(\frac{Y_V-Y_U}{2}\times\frac{\pi}{180}\right)^2+\cos\left(\frac{Y_U\times\pi}{180}\right)\times\cos\left(\frac{Y_V\times\pi}{180}\right)\times\sin\left(\frac{X_V-X_U}{2}\times\frac{\pi}{180}\right)^2} \tag{1}$$

where $\|V-U\|$ represents the straight-line distance between points U and V, $P_{t,e}$ denotes the geographic coordinates of point V , and (X_U, Y_U) corresponds to the geographic coordinates of point U. In the existing bus stop topology diagram, any edges that have excessively long distances between two stops and can be replaced by shorter routes are eliminated. This is because such cases are not suitable as bus route options, and there are shorter alternatives available. Furthermore, any edges in the stop network diagram that represent routes where vehicles cannot directly travel between two stops are also removed. After this process of filtering edges, the bus stop topology diagram undergoes preliminary processing.

Based on the initial processing of the station's topological structure diagram, a consolidation was performed specifically for the scenario where V_8 and V_9 serve as departure stations and V_2 as the destination station. This consolidation resulted in a directed distance table between stations, indicating the existence of feasible paths between two stations in the data entries. Subsequently, the station's topological structure diagram was abstracted based on the directed distances, yielding a directed graph representing the stations. In further optimizing the station's topological structure diagram, factors such as passenger waiting conditions based on historical order data and distances between stations in the dataset are taken into consideration. Through a comprehensive analysis of these factors, the station's topological structure diagram is further refined. The algorithm introduces the concepts of time network maximum flow and passenger network maximum flow. The following formulas, as shown in Equations (2) and (3), provide the definitions for these concepts:

$$T_{f,e} = \frac{\gamma \overline{u}_t + \zeta U_t^{\max}}{e < u, v > d} \tag{2}$$

$$P_{f,e} = \frac{\overline{N}_u}{e < u, v > d} \tag{3}$$

where $T_{f,e}$ represents the time network flow, $P_{t,e}$ represents the passenger network flow, $e < u$ and $v > d$ represent the distance lengths calculated between station u and station

v in the previous step, $\overline{U}_t$ represents the average waiting time of passengers at station u in the historical order dataset, $\overline{N}_u$ represents the average number of daily passenger orders at station u in the historical order dataset, $U_t^{\max}$ represents the maximum waiting time of passengers at station u in the historical order dataset, and γ and ζ are coefficients associated with $\overline{U}_t$ and $U_t^{\max}$, respectively. These coefficients can be calculated using the entropy method.

Next, based on the existing historical order dataset, the average passenger waiting time $waitTime$ and the maximum passenger waiting time $U_t^{\max}$ for each station are calculated, along with the average number of daily passenger orders at each station. Using the distance lengths between stations obtained in the previous step, the $Ford - Fulkerson$ algorithm is implemented to calculate the maximum flow for the defined time network flow $T_{f,e}$ and passenger network flow $P_{t,e}$. Once the maximum flow values are obtained, the minimum value between the two flows is determined. Through analysis based on the historical order dataset, the definitions and calculations for the time network flow and passenger network flow are further clarified. It is then established that the edges in the topological structure graph that are lower than the minimum values of the time network flow and passenger network flow are considered insignificant in the station routes. Therefore, these corresponding edges in the obtained station topological structure graph are removed. Specifically, the edges from station V_9 to V_4, from V_4 to V_6 and from V_4 to V_7 are deleted, and the corresponding directed edge distance table between stations is updated accordingly.

Based on the previously processed directed graph, the depth-first search (DFS) algorithm is applied to identify all cycles. These cycles are subsequently broken within the graph, and the edges that are incapable of accommodating passengers are removed. This step aims to further mitigate operational costs from the standpoint of the bus company.

First, the removal of cycles is performed by deleting the edges corresponding to the nodes with the highest outdegree or indegree in the cycles that do not have a starting point, as indicated by the algorithm. In this dataset, using the depth-first search algorithm, two non-starting point cycles $V_6 \rightarrow V_7 \rightarrow V_4 \rightarrow V_5 \rightarrow V_6$ and $V_6 \rightarrow V_4 \rightarrow V_5 \rightarrow V_6$ are identified. Since node V_4 has the highest indegree of 4, the edge corresponding to $V_7 \rightarrow V_4$ is deleted to break the cycle. Similarly, since node V_4 has the highest indegree of 4, the edge corresponding to $V_6 \rightarrow V_4$ is deleted to break the cycle.

Furthermore, the next step involves removing edges without passenger-carrying capacity. This is achieved by checking if there is a connection between all the stations that point towards the destination station. If a station is found to have a connection with another station and also points back to the starting station, it is considered that the edge between these two stations, as well as the edge pointing back to the starting station, lacks passenger-carrying capacity. Therefore, these two edges need to be deleted. In the given dataset, following the aforementioned criteria, the stations that point towards the destination station are V_1, V_3, and V_7. Notably, there exists an edge $V_3 \rightarrow V_1$, and the station V_1 points back to the starting station V_8.

Lastly, the four edges mentioned above, namely $V_7 \rightarrow V_4$, $V_6 \rightarrow V_4$, $V_3 \rightarrow V_1$, and $V_1 \rightarrow V_8$, are removed. This results in an optimized directed graph of traffic stations suitable for bus operations, as illustrated in Figure 4 below.

After obtaining the directed graph of traffic stations suitable for bus operations, we move on to the algorithmic solution in the following section. In this study, we adopt a combination of the integrated tabu table search local search algorithm and ADPT algorithm, proposing the BSA algorithm. When invoking the algorithm, the first step is to check if there is a TXT file containing the candidate route set for the specific date and parameters in the backend. If available, it is used as input for the ADPT algorithm. If not, the local search algorithm is utilized to generate the candidate route set for the given date and situation, which is then saved in a TXT file. Subsequently, the ADPT algorithm is executed to obtain the bus scheduling plan. In the following sections, we will provide detailed explanations of the algorithms involved.

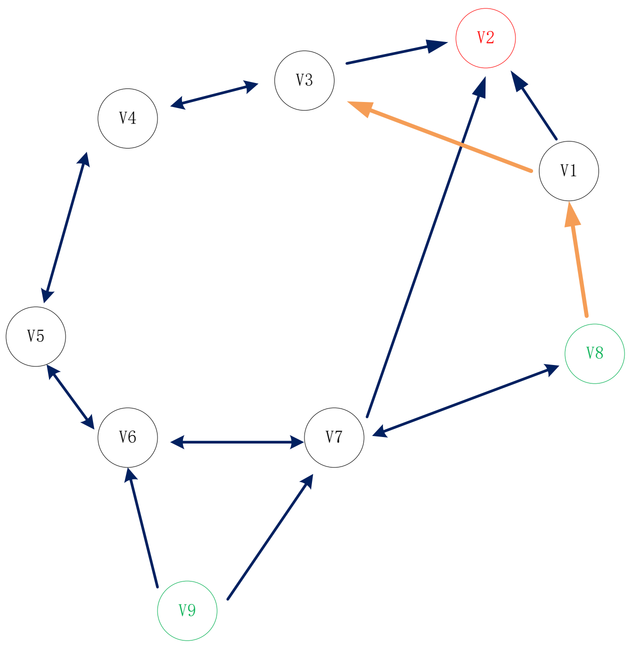
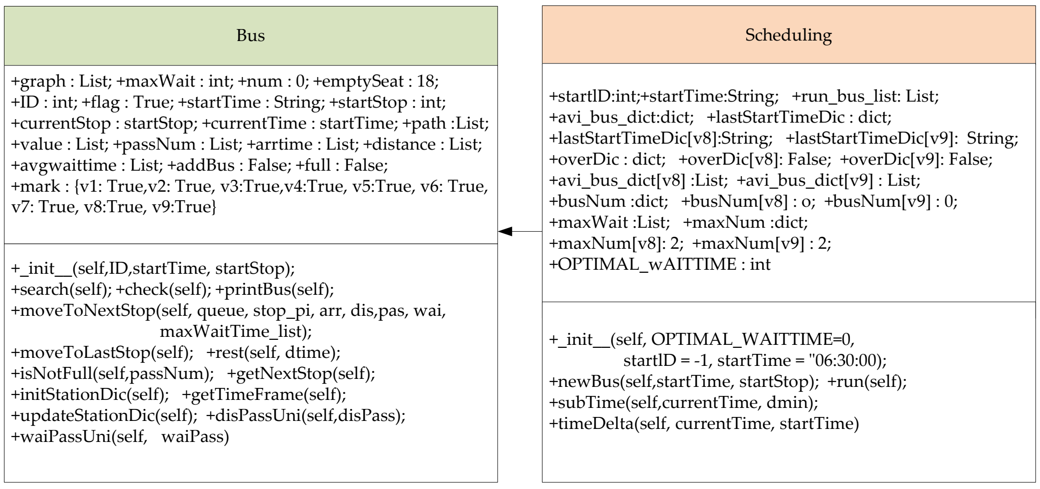


Figure 4. Directed map of transportation stops suitable for bus operation.

After users select the relevant parameters, the algorithmic part of the system's backend needs to determine the feasible candidate route list for subsequent bus vehicle scheduling, based on the existing directed graph of traffic stations suitable for bus operations and the order dataset for the selected date. To ensure the maximum waiting time for passengers, this study implements the tabu table search local search algorithm for solving and determining the candidate route set. Now, let us proceed with an introduction to this process.

(2) Design of Local Search Algorithm for Solution Implementation

Firstly, this algorithm module requires inputs such as the order dataset, the maximum waiting time for passengers T_max, and the number of seats on the bus $Seat\_num$. It defines the Bus class, representing the bus vehicle, and the $Scheduling$ class, responsible for managing the bus operations, as shown in Figure 5 of the following class. Additionally, it initializes the global best waiting time as infinity and the global tabu list and local tabu list in the format $(startStop, currentStop, currenTime, nextStop, waitTime)$, where $waitTime$ represents the maximum waiting time for passengers at that station.



```
Bus

+graph : List; +maxWait : int; +num : 0; +emptySeat : 18;
+ID : int; +flag : True; +startTime : String; +startStop : int;
+currentStop : startStop; +currentTime : startTime; +path :List;
+value : List; +passNum : List; +arrtime : List; +distance : List;
+avgwaittime : List; +addBus : False; +full : False;
+mark : {v1: True,v2: True, v3:True,v4:True, v5:True, v6: True,
v7: True, v8:True, v9:True}

+_init__(self,ID,startTime, startStop);
+search(self); +check(self); +printBus(self);
+moveToNextStop(self, queue, stop_pi, arr, dis,pas, wai,
                    maxWaitTime_list);
+moveToLastStop(self);   +rest(self, dtime);
+isNotFull(self,passNum);   +getNextStop(self);
+initStationDic(self);   +getTimeFrame(self);
+updateStationDic(self);  +disPassUni(self,disPass);
+waiPassUni(self,   waiPass)
```

```
Scheduling

+startlD:int;+startTime:String;   +run_bus_list: List;
+avi_bus_dict:dict;   +lastStartTimeDic : dict;
+lastStartTimeDic[v8]:String;   +lastStartTimeDic[v9]:  String;
+overDic : dict;   +overDic[v8]: False;  +overDic[v9]: False;
+avi_bus_dict[v8] :List;  +avi_bus_dict[v9] : List;
+busNum :dict;   +busNum[v8] : o;  +busNum[v9] : 0;
+maxWait :List;   +maxNum :dict;
+maxNum[v8]: 2;  +maxNum[v9] : 2;
+OPTIMAL_wAITTIME : int

+_init__(self, OPTIMAL_WAITTIME=0,
            startlD = -1, startTime = "06:30:00);
+newBus(self,startTime, startStop);  +run(self);
+subTime(self,currentTime, dmin);
+timeDelta(self, currentTime, startTime)
```

Figure 5. Class diagram for implementing the local search algorithm.

The *Bus* class abstracts the real-world bus vehicle and its core function is to search for the next station. The *Scheduling* class represents the real-world dispatcher and is responsible for scheduling buses and serving passengers. By utilizing the *Scheduling* class, which depends on the *Bus* class and incorporates the tabu list search algorithm, it is possible to iteratively determine a feasible candidate route list that meets the maximum waiting time for passengers. During the algorithm execution, the subsequent candidate station list for a given station is obtained, and their respective objective function values are calculated. The objective function value of the algorithm is defined by Equation (4) as follows:

$$T_{vx} = \left(\alpha\left(\frac{D_{vx}}{P_{vx}}\right)' + \beta\left(\frac{W_{vx}}{P_{vx}}\right)'\right) \cdot 2^{(T_{wx}^{\max} - waitime\_avg_{vx})/\lambda} \tag{4}$$

where α and β represent the weights calculated using the entropy method based on all historical order information. D_{vx} represents the distance between the current station and the subsequent candidate stations. P_{vx} is the total number of passengers boarding at the subsequent candidate stations, and W_{vx} is the sum of the waiting times for all passengers at the subsequent candidate stations. To calculate the ratio of the sum of distances and the sum of waiting times to the total number of passengers, the $min - max$ normalization function is applied. The parameters required for the $min - max$ normalization function are also obtained through entropy method calculations. $T_w^{\max}$ represents the maximum waiting time for passengers, which is the input value T_max . $self.OPTIMAL\_WAITTIME$ is a parameter, and the value of λ can adjust the strictness of the objective function in selecting candidate stations. The average waiting time for subsequent candidate stations is defined by Equation (5) as follows:

$$\sum_{p \in P_v(\phi)} \frac{(p_t(v) - p_\alpha(v))}{P_v(\phi)} \tag{5}$$

By formulating and optimizing this objective function, the proposed algorithm can strike a balance between minimizing passenger waiting time and reducing bus travel distance. The objective function is the key criterion for the system to determine the efficient bus scheduling scheme and optimize the passenger convenience and operation efficiency. The tradeoff achieved by this objective function ensures that the proposed algorithm can effectively manage the passenger waiting time while minimizing the overall travel distance of the bus fleet, thus achieving the improvement and optimization of traffic services.

Next, a probability value p is generated randomly, where $p < P$ means that the search will not be performed in the global tabu list, and $p \geq P$ means that the search will be performed in the global tabu list. Here, P represents the critical probability threshold. If $p < P$ and all candidate sites have no passengers, the algorithm selects the closest candidate site. If $p < P$ and there are passengers at the candidate sites, the algorithm selects the site with the minimum value of the objective function. If $p \geq P$, the algorithm searches the global tabu list. If a corresponding subsequent site is found based on $(startStop, currentStop, currentTime)$, the algorithm calculates whether the maximum waiting time at that site is less than the input maximum waiting time, T_max. If it is less, the algorithm selects that site and stores its maximum waiting time in the global tabu list. If the maximum waiting time at the subsequent site is greater than T_max, it is removed from the global tabu list. If a corresponding subsequent site cannot be found in the global tabu list, the algorithm randomly selects a subsequent site and calculates its maximum waiting time. If this value is less than T_max, the algorithm stores this information in the local tabu list. When the maximum average waiting time of passengers at each site among all buses is smaller than the optimal average waiting time, $self.OPTIMAL\_WAITTIME$, of the scheduling class, it signifies the end of the current iteration. The algorithm then compares the local tabu list and updates the global tabu list based on the maximum passenger waiting time as the fitness value and clears the local tabu list.

There are two types of bus queues: the *run_bus_list* queue in the *Scheduling* class, which contains the buses currently in operation, and the *avi_bus_dict* queue, which contains the available buses at the starting stations. Bus scheduling involves three core aspects.

(1) In the $run()$ method of the *Scheduling* class, if the maximum waiting time for the subsequent station exceeds half of the maximum passenger waiting time $T\_max/2$, additional buses are dispatched. The dispatch function selects the best vehicle from the available bus list *avi_bus_dict*. If no vehicles are available, new buses are dispatched.
(2) In the $run()$ method of the *Scheduling* class, if it is determined that the bus is full and unable to serve passengers, it proceeds directly to the destination station and dispatches a new bus. Similarly, priority is given to selecting from the available bus list *avi_bus_dict* at the starting station. If no vehicles are available, new buses are dispatched.
(3) When the bus reaches the destination station at the end of its operation, the effectiveness of the bus operation is evaluated by checking if it has served any passengers. If it is effective, relevant information is saved, and the maximum average waiting time at each bus stop is stored in the *maxWait* list of the *Scheduling* class. It is then checked if it meets the criteria for stopping the departure from the starting station (i.e., if the latest departure time is greater than 9:30). If the criteria are met, the bus is directly removed from the list of running buses. If the criteria are not met, the bus is removed from the list of running buses and added to the available bus list.

If the bus operation is not effective, it is removed from the list of running buses, and it is checked if there are any buses currently running from the starting station. If there are none, the bus is added back to the list of running buses. If there are buses running from the starting station, the bus is removed from the list of running buses and added to the available bus list.

When the maximum average waiting time of passengers at each bus stop in all buses is less than $OPTIMAL\_WAITTIME$, the solution is obtained for the next iteration. The update process for the relevant data involves the following three steps:

(1) Set the value of $OPTIMAL\_WAITTIME$ in the *Scheduling* class as the maximum average waiting time of passengers at each bus stop in all buses. Store this value in a text document.
(2) Compare it with the local tabu table and update the global tabu table based on the maximum passenger waiting time.
(3) Clear the local tabu table. Then, return the $OPTIMAL\_WAITTIME$ from the *Scheduling* class and assign it to the global $OPTIMAL\_WAITTIME$ for the next iteration loop.

After obtaining the feasible candidate route list that satisfies the maximum passenger waiting time, the next step involves implementing ADPT algorithm for bus scheduling. The goal is to achieve a bus scheduling plan that minimizes the number of buses used and the total distance traveled. In the following sections, we will provide a detailed explanation of the ADPT algorithm.

(3) Design and Implementation of ADPT Algorithm for Solution Solving

First of all, this part of the algorithm module needs to input the order data collection, the maximum passenger waiting time T_max, and the number of seats of the bus $Seat\_num$ and instantiate the list used to store the bus scheduling results, the System class responsible for bus scheduling, which needs to instantiate the list of available buses $busNewcompany[]$ in the System bus number instantiation class, which relies on the *Bus* class to implement. The *Bus* class abstracts the real bus vehicles, and the core is used for driving to the next stop to record the driving information; the System class abstracts the scheduling class, and the core goal is to schedule as few buses as possible while satisfying the passenger waiting, and the System class relies on the Bus class to reproduce the ADPT algorithm to obtain the optimal bus scheduling solution. Figure 6 shows the design flowchart of ADPT algorithm in the case of morning peak (6:30–10:00) in the data set.

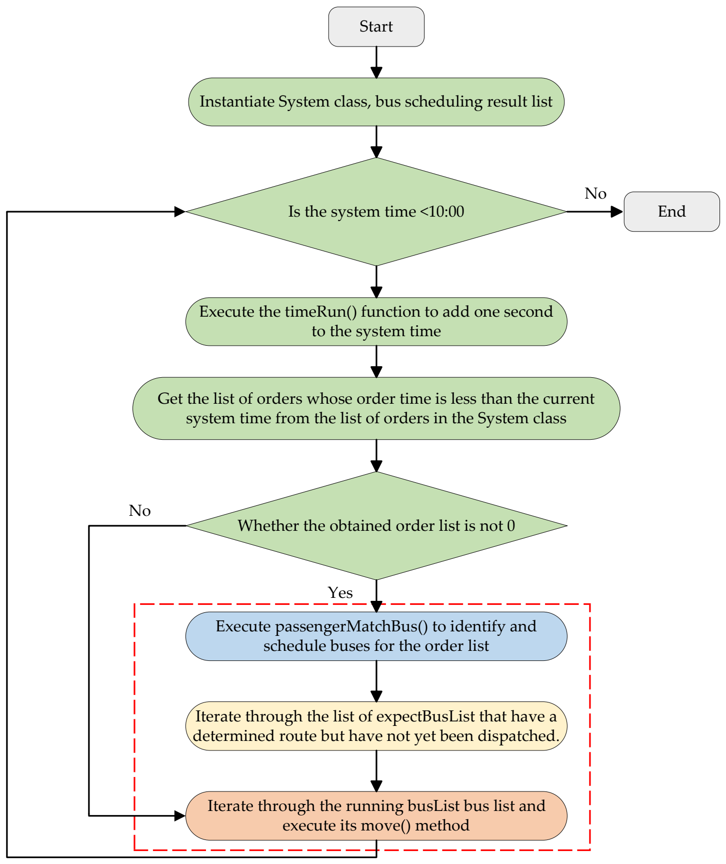

Start
Instantiate System class, bus scheduling result list
Is the system time <10:00
No
End
Execute the timeRun() function to add one second to the system time
Get the list of orders whose order time is less than the current system time from the list of orders in the System class
Whether the obtained order list is not 0
No
Yes
Execute passengerMatchBus() to identify and schedule buses for the order list
Iterate through the list of expectBusList that have a determined route but have not yet been dispatched.
Iterate through the running busList bus list and execute its move() method

Figure 6. Class diagram for implementing the local search algorithm.

In the process of obtaining the optimal bus scheduling solution, the most crucial part is the dashed box shown in Figure 6. Let us discuss it in detail. In this algorithm, there are three types of bus queues: the *busList* in the System class represents the buses that are currently running, the *expectBusList* represents the buses that have been assigned routes and departure times but have not yet departed, and the *busNewCompany* represents the list of available buses. These queues play a significant role in the algorithm and are essential for efficient bus scheduling.

During the bus assignment phase of the order list, the system iterates through each order. For each order, it records the boarding point and order time and sets the local variables $matchFlag = False$ (indicating no match yet) and *minarrtime* to ‘12:00:00’, which is used to track the earliest arrival time for buses. The system checks two types of bus queues: the *expectBusList*, which contains scheduled routes that have not departed yet, and the *busList*, which contains currently running buses. It looks for the earliest arriving bus and records its arrival time in *minarrtime*. If this arrival time satisfies the order’s waiting time requirement (i.e., it is less than the input maximum waiting time T_max), matchFlag is set to True, indicating that the order will be serviced by a bus.

If no suitable bus is found for the order, a new bus is dispatched. The dispatching process is as follows: “First, based on the order time and the maximum waiting time, calculate the latest departure time. Then, identify the set of routes that include the boarding point. Iterate through this set, calculating the departure time for each route using the $back()$ function. If the departure time is later than the order time, it is considered feasible and added to a list. Finally, select the route with the minimum cost (route distance divided by the number of passengers at that stop) as the final choice. Use the $addBus(starttime, line)$ method to add the departure time and route to the *expectBusList* of buses waiting to operate". This process ensures that the newly dispatched bus selects a candidate route that minimizes the distance traveled while serving more passengers. After completing this process, the $flag$ of the order is set to 1, indicating that a bus has been assigned to it.

Next, iterate through the $expectBusList$ to check if the system time matches the departure time. If there is a match, use the $addRunBus()$ method to return a Bus instance. Priority is given to selecting a bus from the $busNewcompany$ list. If no bus is available, create a new Bus instance and add it to the list of running buses. Finally, remove the bus instance from the $expectBusList$.

Finally, iterate through each bus in the running $busList$ and store the return value of $move\_\_(self.timeDic, self.time)$ of that bus in end. If end is $True$, then the bus will be removed from the running $busList$ and added to the $busNewcompany$ list.

The bus class $move\_\_(self.timeDic, self.time)$ is used to execute the bus moving process; its execution process is as follows: determine whether the system time is equal to the departure time of the bus, and if so, the bus needs to depart. Find the order list of the Bus instance where the order is placed at that stop, the order time is less than the departure time, and the $flag$ is 0. If the number of the list is not zero, then iterate through the orders to change the $flag$ bit to 1 and calculate the waiting time of each order to obtain the maximum waiting time and the average waiting time, add the number of the list to the $passNum$ of the bus, and store the bus information in bus_info to indicate that the passenger is on board. The $time$ property of the bus is changed to the current departure time. If it is not equal, the arrival time of the bus to the next stop is calculated. If the present time is exactly equal to the arrival time, it means that the bus is running to the next stop, and similar to the departure stop, it is necessary to complete the passenger boarding operation at that stop and modify the relevant information. Modify the $time$ property of the bus to the arrival time. If the bus reaches the terminal and the number of passengers on board is not zero, the bus information will be saved in the bus scheduling result list and a True value will be returned, indicating the end of the bus operation. If the bus has not reached the terminal, then a $False$ value is returned, indicating that the bus has not yet finished its run.

By implementing the aforementioned algorithm, we can ultimately achieve a bus scheduling solution that minimizes the distance traveled by buses while satisfying the maximum passenger waiting time. This solution ensures the least number of buses required for the scheduling process.

3.4. Database System Design

Database Entity Design

The section on database entity design primarily focuses on designing based on the requirements analysis conducted in Section 3.1. Following an engineering approach, the unified modeling language is utilized to establish the database model that fulfills user needs and illustrates the relationships among different entities within the system. This section employs the structure of an entity–relationship (E-R) diagram, which comprises three key elements: entities (representing objects in the data model, typically depicted as rectangles), attributes (representing the characteristics and properties of objects, typically depicted as ovals), and relationships (depicting associations between objects, categorized as many-to-many, one-to-many, or one-to-one, typically depicted as diamonds). By utilizing the E-R diagram, developers can conveniently engage in system development and implementation. This section presents the E-R diagram for the system, tailored to the specific requirements, as depicted in Figure 7.

The system consists of several entities, including System User, Bus, Order, Station, and the routes between two stations. The relationships between these entities are as follows:

- Users have a one-to-many relationship with buses, orders, and stations. A user can manage multiple bus information, multiple order information, and view multiple station information.

- The relationship between stations and the routes between them is a many-to-many relationship. A station can have multiple routes between two stations, and a route is associated with two stations.
- The relationship between orders and stations is also a many-to-many relationship. An order includes information about two stations, and a station can have multiple order information.

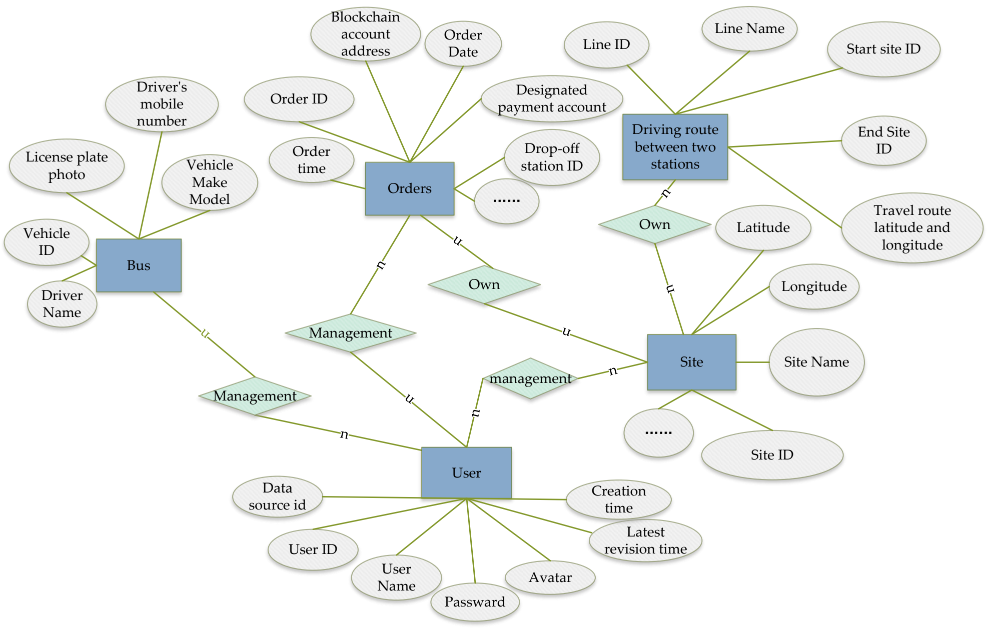


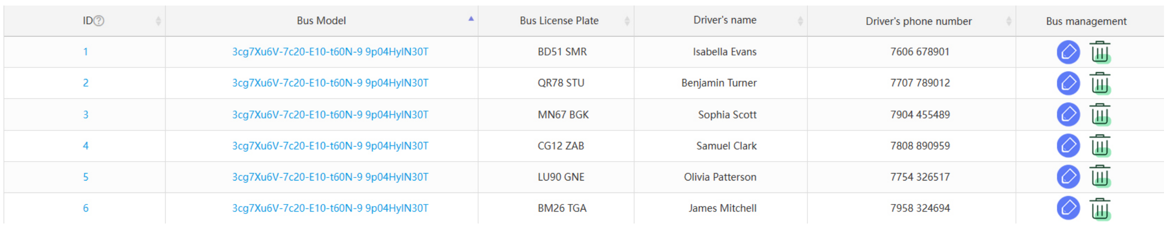
Figure 7. Class diagram for implementing the local search algorithm.

4. System Implementation

This chapter completes the implementation of the system according to the above requirements' analysis and system design. This part includes the introduction of system development tools and environment, as well as the functional coding implementation of each functional module, and displays the interface of each functional module in the form of screenshots.

4.1. Bus Management Module

Upon successfully logging into the system, users can access the "Bus Management" section from the left-side navigation bar, which will display the Bus Management page as shown in Figure 8. On this page, users are presented with a comprehensive list of all bus information stored in the database. Each row in the list has corresponding buttons on the right-hand side, allowing users to modify or delete bus information. Additionally, users can add new bus information by clicking the blue button located at the top right corner of the table. When users submit the information, the system will process the data, interact with the backend, and store it in the database, providing relevant prompts and feedback to the user.



ID	Bus Model	Bus License Plate	Driver's name	Driver's phone number	Bus management
1	3cg7Xu6V-7c20-E10-t60N-9 9p04HyIN30T	BD51 SMR	Isabella Evans	7606 678901	
2	3cg7Xu6V-7c20-E10-t60N-9 9p04HyIN30T	QR78 STU	Benjamin Turner	7707 789012	
3	3cg7Xu6V-7c20-E10-t60N-9 9p04HyIN30T	MN67 BGK	Sophia Scott	7904 455489	
4	3cg7Xu6V-7c20-E10-t60N-9 9p04HyIN30T	CG12 ZAB	Samuel Clark	7808 890959	
5	3cg7Xu6V-7c20-E10-t60N-9 9p04HyIN30T	LU90 GNE	Olivia Patterson	7754 326517	
6	3cg7Xu6V-7c20-E10-t60N-9 9p04HyIN30T	BM26 TGA	James Mitchell	7958 324694	

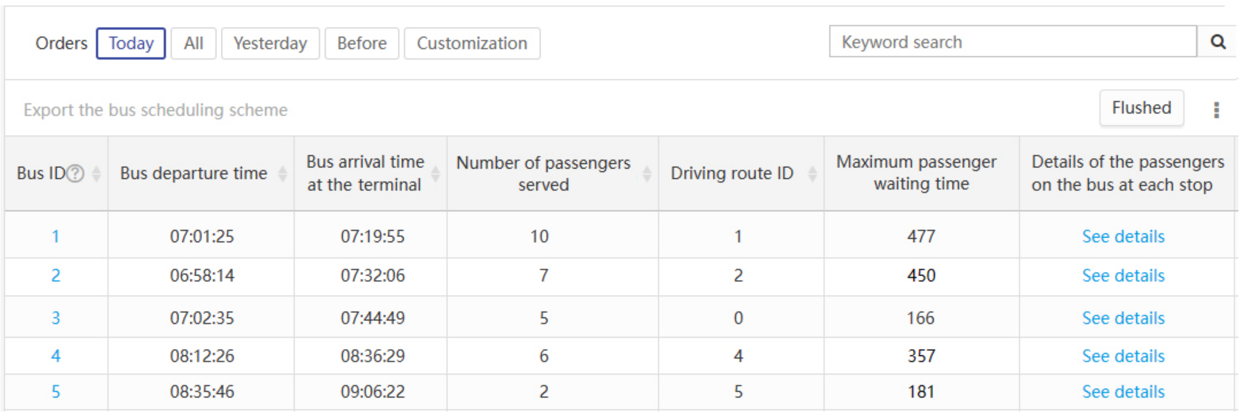
Figure 8. Bus management page.

4.2. Bus Dispatching and Route Planning Module

This module is the core functionality of the system, providing users with the optimal bus scheduling solution. During the design phase, the "Station Route Preprocessing" is conducted to obtain an optimized directed graph of stations. The module implements the iterative algorithm for generating candidate route sets and the ADPT algorithm, both of which play a crucial role in achieving the bus scheduling solution.

The system offers two options for obtaining the bus scheduling solution: using the default parameters or customizing the parameters. The customizable parameters include the number of bus seats and the maximum passenger waiting time. In the system, the default parameter values are 18 seats for the bus and a maximum waiting time of 10 min for passengers. Users can select the "Algorithm Execution Page" to choose the desired method for obtaining the bus scheduling solution.

When users select the method and parameters and click the "Execute Algorithm" button, the frontend passes the parameter list to the backend. The backend first checks the Redis cache to see if there is a scheduling solution available for that specific date and total number of buses under the given parameters. If a solution is found in the cache, it is directly displayed on the page along with the "Export Bus Schedule" button. If there is no cached solution, the backend retrieves the order list and total number of buses for that date from the database. This information is then passed to the algorithm module. A popup window informs the user to wait briefly while the algorithm runs. Once the algorithm completes, the backend stores the obtained bus scheduling solution for that date and parameters in the Redis cache for future use. The solution is also returned to the frontend and displayed on the page along with the "Export Bus Schedule" button, as shown in Figure 9.



Bus ID	Bus departure time	Bus arrival time at the terminal	Number of passengers served	Driving route ID	Maximum passenger waiting time	Details of the passengers on the bus at each stop
1	07:01:25	07:19:55	10	1	477	See details
2	06:58:14	07:32:06	7	2	450	See details
3	07:02:35	07:44:49	5	0	166	See details
4	08:12:26	08:36:29	6	4	357	See details
5	08:35:46	09:06:22	2	5	181	See details

Figure 9. The page on which the algorithm runs.

In the bus scheduling solution table, users can click the "View Details" button for each boarding passenger to see specific information about passengers at each boarding stop. If the user clicks the "Export Bus Schedule" button, a download request page pops up, allowing the user to confirm or cancel the download. After downloading, the user receives the bus scheduling solution displayed in the form of an Excel spreadsheet in the frontend. Once the user obtains the bus scheduling solution for the selected date, a prompt dialog box appears, suggesting the user click the "Start" button to initiate operational monitoring.

5. Algorithm Validation and System Testing

In this section, we will first introduce the algorithm validation part and experiment with the BSA algorithm using the real dataset from Panda Travel to prove its performance. Secondly, we will conduct system testing by performing relevant operations under the set conditions to identify any issues in the system.

5.1. Algorithm Validation

In this section, we will validate the effectiveness of the BSA algorithm using a real dataset of one-week orders from Panda Travel. The validation will focus on three key performance indicators: the total distance traveled by buses, the average number of serviced passengers, and the number of shared buses required. Additionally, we will compare the BSA algorithm with the ACO algorithm and the GA algorithm and analyze the complexity of these algorithms to assess their effectiveness in achieving optimal results. The analysis is as follows:

ACO algorithm is an algorithm inspired by the behavior of ants in the process of finding food. In the ant colony algorithm, the process of ants searching for food is simulated, and the ants release pheromone to guide other ants to a better path. The ants choose which direction to move next based on the pheromone concentration along the path. When an ant chooses a path, it releases pheromones along the path, and the pheromones evaporate over time. Through the accumulation and evaporation of pheromones, the ant colony gradually develops a preference for the path and thus finds a shorter path. The ACO algorithm has the advantages of global search and distributed computing and is suitable for complex combinatorial optimization problems. In the path planning and scheduling problem, it can help to find the optimal or near-optimal path and scheduling scheme and optimize the utilization efficiency of resources and operating costs. At the same time, its complex analysis is as follows:

(1) Time complexity

The ACO algorithm involves the movement of ants and the updating process of pheromones, and each ant needs to choose a path according to the pheromone concentration and heuristic function. In each iteration, all ants need to make path selection and pheromone updates, so the time complexity is usually expressed as O($h(k)$), where k is the number of iterations and h is the complexity of the ant's path selection.

(2) Spatial complexity

The ACO algorithm needs to maintain data structures such as ant location and pheromone concentration matrix, and its spatial complexity mainly depends on the size of the data structure, which can usually be expressed as O(l), where l is the size of the data structure.

The GA algorithm is a kind of optimization algorithm developed by a natural evolution process. By simulating the genetic and evolutionary process in nature, the GA algorithm evaluates the adaptability of individuals through fitness functions, and then selects, crosses, and mutates to eliminate the fittest of individuals, so as to continuously evolve better solutions. Its advantage is that it has strong global search ability in the problems with large and complex search space. In problems such as resource scheduling and path planning, it can help find the optimal or near-optimal solution, optimize resource allocation and path selection, and improve operational efficiency and cost effectiveness. At the same time, its complex analysis is as follows:

(1) Time complexity

The GA algorithm involves the process of population selection, crossover, and mutation, and each iteration needs to select and update the population. In each iteration, all individuals need to be selected and updated, so the time complexity is usually expressed as O($i(j)$), where j is the population size and i is the complexity of the genetic algorithm.

(2) Spatial complexity

The GA algorithm needs to maintain data structures such as new individuals generated in the operation process of population, crossover, and mutation, and its spatial complexity mainly depends on the size of the data structure, which can usually be expressed as O(k), where k is the size of the data structure.

The BSA algorithm is a global bus scheduling and route planning algorithm proposed in this paper. Its flow involves calling algorithm part and local search algorithm. The ant colony algorithm and genetic algorithm, as optimization algorithms, are closely related to the bus scheduling and route planning goals of the BSA algorithm. At the same time, its complex analysis is as follows:

(1) Time complexity

The BSA algorithm needs to select and optimize the candidate line set in the process of calling the algorithm part and the local search algorithm, so a certain calculation needs to be carried out each time the algorithm is called. The time complexity of the algorithm depends mainly on the size of the set of candidate lines and the complexity of the local search algorithm, which can usually be expressed as O($f(n)$), where n is the number of candidate lines.

(2) Spatial complexity

The BSA algorithm needs to maintain data structures such as candidate route sets, traffic station directed graphs, and order data sets and use some auxiliary data structures in the local search algorithm. Therefore, its spatial complexity depends mainly on the size of the data structure, which can usually be expressed as O($g(m)$), where m is the size of the data structure.

The ACO algorithm and the GA algorithm were chosen to compare with the proposed algorithm BSA because both are common optimization algorithms that are widely used to solve problems such as path planning and resource scheduling. By comparing with these two algorithms, we can better evaluate the advantages and disadvantages of BSA algorithm in terms of performance and effect so as to provide guidance and inspiration for further improvement and optimization of the BSA algorithm. Therefore, the selection of the ACO and GA algorithms to compare with the BSA algorithms helps to better understand the performance and application range of BSA algorithms and to promote its development and application in practical applications.

At the same time, we conducted a sensitivity analysis on the experimental results, and the upper limit of passenger waiting time was changed when the number of shared bus seats was fixed at 18, and the data set within one week was tested. Table 1 of the experimental results shows the total distance traveled by buses, the number of shared buses required, and the average number of passengers served under different algorithms.

Table 1. Experimental results of different schemes when the number of seats is 18.

Maximum Wait Time	**Total Distance Traveled**			**Number of Buses Required**			**Evaluate the Number of Passengers in the Service**		
	BSA	**ACO**	**GA**	**BSA**	**ACO**	**GA**	**BSA**	**ACO**	**GA**
5	233	424	465	13	16	14	3.6	2.8	3.5
7	215	345	431	11	14	13	3.9	3.4	3.7
9	197	293	395	11	11	11	4.3	4.1	4.2
11	195	221	360	9	11	11	4.5	4.4	4.4

From the perspective of the total distance traveled, the BSA algorithm is superior to the other two algorithms, mainly because the BSA algorithm will select the appropriate line from the candidate line set, and preferentially select the short distance line for vehicle scheduling. The ACO and GA algorithms only select routes according to the station situation, resulting in some routes being too long to be suitable for the operation of shared buses. From the analysis of the required number of buses, the BSA algorithm requires fewer buses than the other two algorithms. The core lies in the fact that the ADPT algorithm in the BSA algorithm gives priority to the vehicles in operation to provide services for passengers and can dynamically determine the appropriate departure time and route for buses according to the needs of passengers so as to make full use of shared bus resources. However, other algorithms lack scheduling optimization, such as the optimization of departure time, resulting in the need for more buses. From the perspective of the average number of service passengers, the value of the BSA algorithm is higher than the other two algorithms. The ACO and GA algorithms need to increase buses or increase frequency to ensure the maximum waiting time of passengers, while the BSA algorithm can optimize the average number of service passengers to reduce operating costs. Through the above analysis, it can be shown that the BSA algorithm can effectively reduce the distance of buses and the number of buses required, improve the average number of passengers, and reduce the operating costs of bus sharing companies on the premise of ensuring the maximum waiting time of passengers.

Secondly, when the maximum waiting time of passengers is 10 min, the data set within a week is tested under the condition that the number of shared bus seats is changed. Table 2 of the experimental results shows the total distance traveled by buses under different algorithms, the number of shared buses required, and the average number of passengers served.

Table 2. Experimental results of different schemes when the maximum waiting time is 10 min.

Number of Seats	**Total Distance Traveled**			**Number of Buses Required**			**Evaluate the Number of Passengers in the Service**		
	BSA	**ACO**	**GA**	**BSA**	**ACO**	**GA**	**BSA**	**ACO**	**GA**
10	240	311	586	10	10	16	5.2	5.2	4.3
12	225	280	563	9	9	15	5.6	5.6	4.5
14	224	275	522	8	9	14	5.9	5.7	4.8
16	218	275	463	8	9	13	6	5.7	5.3

From the perspective of a different number of seats, the BSA algorithm can obtain shorter total distance, higher average number of passengers, and fewer buses. When the number of seats reaches 12, 14, and 16, the number of buses required by BSA algorithm and ACO algorithm no longer increases, indicating that the number of buses has met the travel needs of passengers at this time. Through the above analysis, it can be shown that the experimental results obtained by BSA algorithm under different number of seats are still better than the experimental results of the ACO algorithm and GA algorithm, indicating that the BSA algorithm is effective and feasible for changes in the number of seats and maximum waiting time.

In order to evaluate the validity and universality of the results, this paper conducted multiple runs under the condition of different number of seats and maximum waiting time of passengers. As shown in Table 3 below, the experimental results of the total distance traveled by buses, the number of shared buses required, and the average number of passengers served were obtained.

As can be seen from Table 3, the total distance traveled by the bus obtained by BSA algorithm is the smallest in the case of the number of seats and the maximum waiting time of passengers. This shows that the BSA algorithm is able to plan the route of the bus more efficiently, resulting in shorter travel distances, which reduces traffic congestion and carbon emissions. When both the number of seats and the maximum waiting time of passengers change, the number of shared buses required by the BSA algorithm is less than that of ACO algorithm and GA algorithm. This shows that the BSA algorithm can schedule bus resources more reasonably, make full use of bus seats, and improve operation efficiency. When the number of seats and the maximum waiting time of passengers change, the BSA algorithm can serve more passengers than the ACO algorithm and GA algorithm. This shows that the BSA algorithm can better meet the needs of passengers, reduce the waiting time of passengers, and improve the user experience.

Table 3. Comparison of Experimental Results for ACO, GA, and BSA Algorithms.

Algorithm	Number of Seats	Maximum Wait Time	Total Distance Traveled	Number of Buses Required	Evaluate the Number of Passengers in the Service
ACO	18	10	120.2	7	40.5
ACO	18	15	150.8	9	37.2
GA	20	10	118.7	6	43.6
GA	20	15	146.5	8	40.8
BSA	22	10	116.3	5	45.2
BSA	22	15	143.7	7	42.1

In summary, through the detailed analysis and interpretation of the experimental results, the experimental conclusion is drawn: compared with the ACO and GA algorithms, the BSA algorithm has advantages in bus scheduling and route planning, which can effectively reduce the distance of buses and the number of buses required, improve the average number of service passengers, and realize the improvement and optimization of traffic services.

5.2. System Testing

5.2.1. Functional Testing

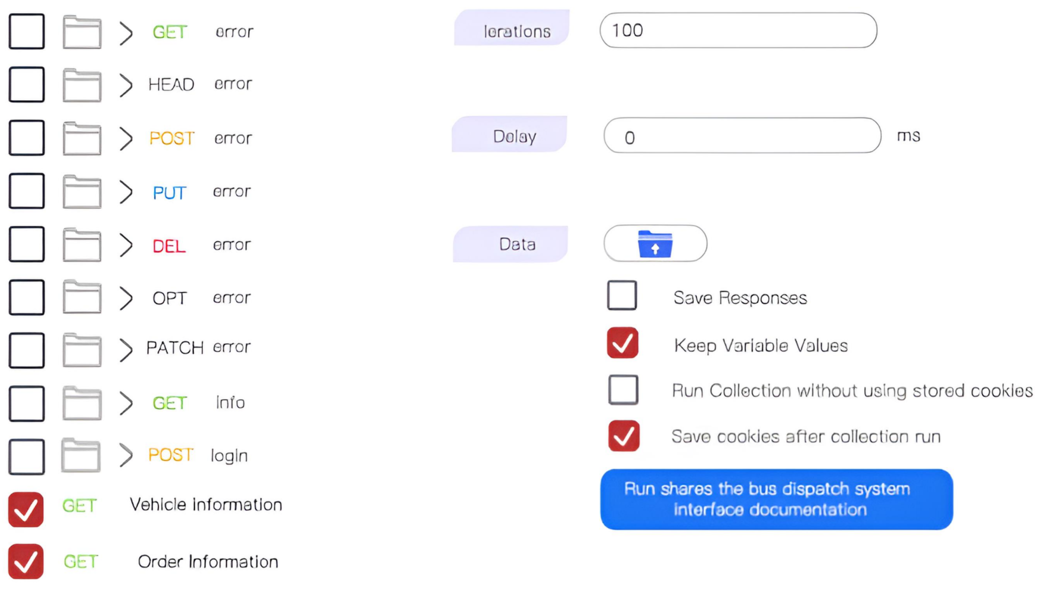
This paper adopts the black-box testing method to test the key functional modules of the system by creating test cases. Due to space constraints, particular emphasis is placed on conducting comprehensive system functionality testing for the bus scheduling and route planning modules. The test cases for this module are presented in Table 4.

Table 4. Bus scheduling and route planning function module test cases.

Use Case Number	Test Steps	Expected Results	Test Results
1	Click Customize on the "Algorithm Run Page" to get the bus scheduling scheme.	The page is successfully redirected.	Consistent with expected results.
2	Select the date of Default Parameter to confirm the date that there is a candidate line set for that day in Redis.	The collection of candidate routes for the day is displayed on this page with the "Export Bus Proposal" button.	Consistent with expected results.
3	Confirm the date after selecting the Default Parameter date; there is no candidate line set for that day in Redis.	A pop-up window prompts the user to wait for the algorithm to run, and the page is displayed after the algorithm is run.	Consistent with expected results.
4	Click the date picker for "Custom parameters".	And show the "Export Bus Plan" button, only the date that the order dataset has can be selected.	Consistent with expected results.
5	Select the Export Bus Scheme button for "Algorithm Run".	View the order details.	Consistent with expected results.
6	Click on the details of the passengers boarding at each stop in the bus dispatch plan to view the details.	Check for success.	Consistent with expected results.

5.2.2. Performance Testing

Regarding performance testing, the system underwent concurrency testing using the Postman testing tool. Since the system's usage scenario does not involve a high level of concurrent access, the concurrency level was set to 100 for testing purposes, with zero delay. Figure 10 below depicts the performance testing and its results. The displayed image demonstrates that even under a concurrency level of 100, the system maintains stable performance, thereby meeting the users' requirements in terms of system performance.



- ☐ GET error
- ☐ HEAD error
- ☐ POST error
- ☐ PUT error
- ☐ DEL error
- ☐ OPT error
- ☐ PATCH error
- ☐ GET info
- ☐ POST login
- ✓ GET Vehicle information
- ✓ GET Order Information

Iterations: 100

Delay: 0 ms

Data

- ☐ Save Responses
- ✓ Keep Variable Values
- ☐ Run Collection without using stored cookies
- ✓ Save cookies after collection run

Figure 10. The page on which the algorithm runs.

6. Discussion

6.1. Research Significance

This paper is dedicated to explore and implement a blockchain-based shared bus service scheduling management system. The emergence of blockchain technology brings new opportunities and challenges for traffic management and shared bus services. By combining the decentralization, security, and transparency features of blockchain, this research aims to improve the operational efficiency, safety and user experience of shared bus services.

First, blockchain technology brings greater data trust and transparency to shared bus services. Traditional scheduling management systems may have the risk of data tampering, leading to unfair resource allocation and decision-making. The immutability of the blockchain ensures the integrity of the data, and every data modification will be recorded on the blockchain, achieving permanent storage and tracking of the data. This will provide traffic authorities and users with a reliable data base to improve the accuracy and fairness of scheduling decisions.

Second, blockchain's smart contract capabilities make payment and settlement of shared bus services more efficient and secure. Traditional payment methods may have the risk of payment disputes or payment information leakage. With smart contracts, users can make payments using cryptocurrencies or digital payment methods, and transactions will be automatically executed on the blockchain and will only be confirmed if certain conditions are met. This will reduce uncertainty and transaction risk in the payment process and lead to a better payment experience for users and service providers.

Finally, this study will help promote the sustainable development of shared bus services. By optimizing resource allocation and route planning, blockchain technology can help reduce the operating costs of bus services and improve operational efficiency, thus promoting the position of shared bus services in urban transportation. The promotion and popularization of shared bus services can help reduce urban traffic congestion, reduce environmental pollution, and improve the travel experience of urban residents.

6.2. Future Direction

In this study, our assumption of blockchain data integrity is based on the nature of blockchain technology that data cannot be modified once written to the blockchain and requires consensus from multiple nodes in the network. However, in order to more specifically ensure the integrity of the data, we will consider the following aspects in future work:

(1) Encryption technology application

Encryption technology plays a key role in ensuring the privacy and integrity of data. We will explain how encryption can be used to sign, verify, and secure data to prevent unauthorized access and tampering.

(2) Audit and monitoring

We will examine how blockchain data can be audited and monitored on a regular basis to detect potential anomalies or attempts at tampering and consider doing so by checking the transaction history of the blockchain and the execution of smart contracts.

(3) On-chain data verification

Explore how the immutable nature of blockchain can be used to verify the integrity of data. After the data are written to the blockchain, it is verified that the data have not been tampered with by checking whether the corresponding hash value matches the hash value stored on the blockchain.

In future work, we will integrate the above factors to establish a more comprehensive data integrity guarantee mechanism to ensure the effectiveness and credibility of blockchain technology in intelligent transportation systems. However, the system still has limitations, and as a preliminary study, there are some possible further development directions worth further research and exploration:

(1) Privacy protection

In the shared bus service, the user's location and personal information are sensitive data, and in future research, we can further explore how to ensure user privacy and personal data security while ensuring data transparency. At the same time, blockchain-based authentication and access control mechanisms can also be explored to ensure that only authorized users have access to specific data, thereby protecting users' personal information from unauthorized access.

(2) Scalability and performance optimization

As the number of users and transaction volume increases, blockchain systems may face scalability and performance challenges. How to improve the throughput of the system and reduce the transaction delay is a problem worth paying attention to.

(3) Environmentally friendly

The mining process of blockchain technology can involve a lot of energy consumption, so studying how to optimize the energy efficiency and environmental friendliness of shared bus service systems is also an important research direction.

This paper discusses the application and potential value of blockchain technology in the field of traffic management by implementing a blockchain-based shared bus service scheduling management system. The decentralization, security, and transparency features of blockchain bring many advantages to shared bus services, helping to improve operational efficiency, safety, and user experience. However, there are still some aspects that need further improvement and in-depth study in the future to achieve a more comprehensive shared bus service system.

7. Conclusions

This paper analyzes the difficulties in traffic management through demand analysis and focuses on the contradiction between the operating cost of the shared bus company and the passenger travel experience. Based on this background, we developed a blockchain-based intelligent traffic service scheduling management system, which not only integrates vehicle scheduling and route planning algorithms but also has the advantages of data transparency, trust, security, and decentralization. Although we did not explore the smart contract aspect in detail in this article, in future work we will focus on the development of smart contracts. In future work, we will further focus on the development of smart contracts to strengthen our proposed blockchain-based intelligent traffic-service scheduling management system. Specifically, we will focus on deploying smart contract trading using solidity programming in the EVM. Our goal is to ensure that smart contracts are written without any malicious attacks during execution by exploring best practices. To this end, we will use methods such as rigorous code audits and vulnerability testing to improve the security of smart contracts. These smart contracts will act as a link between our shared bus dispatch management system and blockchain technology, ensuring that transactions are secure, transparent, and immutable. Through this in-depth research and practice, we will be able to fully apply blockchain technology and smart contracts in the system to further enhance the efficiency, safety, and trust of traffic management. We look forward to more innovations in the field of smart contracts in future work and to making more significant contributions to the development of intelligent transportation systems.

Author Contributions: Conceptualization, T.L. and X.X.; methodology, T.L. and G.Z.; software, Y.L. and A.T.; validation, G.Z. and Y.L.; formal analysis, T.L. and Y.L.; investigation, X.X. and Y.L.; resources, X.X. and A.T.; data curation, G.Z. and Y.L.; writing—original draft preparation, T.L.; writing—review and editing, X.X.; visualization, G.Z. and Y.L.; supervision, X.X. and A.T.; project administration, T.L. and X.X.; funding acquisition, A.T. All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded by the Researchers Supporting Project number (RSPD2023R681), King Saud University, Riyadh, Saudi Arabia.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

References

1. Sovacool, B.K.; Upham, P.; Martiskainen, M.; Jenkins, K.E.H.; Torres Contreras, G.A.; Simcock, N. Policy prescriptions to address energy and transport poverty in the United Kingdom. *Nat. Energy* **2023**, *8*, 2058–7546. [CrossRef]
2. Chen, J.; Zhang, Y.; Teng, S.; Chen, Y.; Zhang, H.; Wang, F.Y. ACP-Based Energy-Efficient Schemes for Sustainable Intelligent Transportation Systems. *IEEE Trans. Intell. Veh.* **2023**, *8*, 3224–3227. [CrossRef]
3. Ribeiro, D.A.; Melgarejo, D.C.; Saadi, M.; Rosa, R.L.; Rodríguez, D.Z. A novel deep deterministic policy gradient model applied to intelligent transportation system security problems in 5G and 6G network scenarios. *Phys. Commun.* **2023**, *56*, 101938. [CrossRef]
4. Wang, X.; Garg, S.; Lin, H.; Kaddoum, G.; Hu, J.; Hassan, M.M. Heterogeneous Blockchain and AI-Driven Hierarchical Trust Evaluation for 5G-Enabled Intelligent Transportation Systems. *IEEE Trans. Intell. Transp. Syst.* **2023**, *24*, 2074–2083. [CrossRef]
5. Kong, X.; Li, M.; Tang, T.; Tian, K.; Moreira-Matias, L.; Xia, F. Shared Subway Shuttle Bus Route Planning Based on Transport Data Analytics. *IEEE Trans. Autom. Sci. Eng.* **2018**, *15*, 1507–1520. [CrossRef]
6. Ning, Z.; Sun, S.; Wang, X.; Guo, L.; Guo, S.; Hu, X.; Hu, B.; Kwok, R.Y.K. Blockchain-Enabled Intelligent Transportation Systems: A Distributed Crowdsensing Framework. *IEEE Trans. Mob. Comput.* **2022**, *21*, 4201–4217. [CrossRef]
7. Srivastava, V.; Debnath, S.K.; Bera, B.; Das, A.K.; Park, Y.; Lorenz, P. Blockchain-Envisioned Provably Secure Multivariate Identity-Based Multi-Signature Scheme for Internet of Vehicles Environment. *IEEE Trans. Veh. Technol.* **2022**, *71*, 9853–9867. [CrossRef]
8. Garcia, R.D.; Ramachandran, G.S.; Jurdak, R.; Ueyama, J. Blockchain-Aided and Privacy-Preserving Data Governance in Multi-Stakeholder Applications. *IEEE Trans. Netw. Serv. Manag.* **2022**, *19*, 3781–3793. [CrossRef]
9. Ning, Z.; Chen, H.; Ngai, E.C.H.; Wang, X.; Guo, L.; Liu, J. Lightweight Imitation Learning for Real-Time Cooperative Service Migration. *IEEE Trans. Mob. Comput.* **2023**, 1–18.
10. Wang, X.; Ning, Z.; Guo, S.; Wang, L. Imitation Learning Enabled Task Scheduling for Online Vehicular Edge Computing. *IEEE Trans. Mob. Comput.* **2022**, *21*, 598–611. [CrossRef]
11. Liu, J.; Weng, H.; Ge, Y.; Li, S.; Cui, X. A Self-Healing Routing Strategy Based on Ant Colony Optimization for Vehicular Ad Hoc Networks. *IEEE Internet Things J.* **2022**, *9*, 22695–22708. [CrossRef]
12. Liu, S.C.; Chen, Z.G.; Zhan, Z.H.; Jeon, S.W.; Kwong, S.; Zhang, J. Many-Objective Job-Shop Scheduling: A Multiple Populations for Multiple Objectives-Based Genetic Algorithm Approach. *IEEE Trans. Cybern.* **2023**, *53*, 1460–1474. [CrossRef] [PubMed]

13. Ning, Z.; Xia, F.; Ullah, N.; Kong, X.; Hu, X. Vehicular Social Networks: Enabling Smart Mobility. *IEEE Commun. Mag.* **2017**, *55*, 16–55. [CrossRef]
14. Wang, X.; Ning, Z.; Wang, L. Offloading in Internet of Vehicles: A Fog-Enabled Real-Time Traffic Management System. *IEEE Trans. Ind. Inform.* **2018**, *14*, 4568–4578. [CrossRef]
15. Zhu, F.; Lv, Y.; Chen, Y.; Wang, X.; Xiong, G.; Wang, F.Y. Parallel Transportation Systems: Toward IoT-Enabled Smart Urban Traffic Control and Management. *IEEE Trans. Intell. Transp. Syst.* **2020**, *21*, 4063–4071. [CrossRef]
16. Ning, Z.; Zhang, K.; Wang, X.; Obaidat, M.S.; Guo, L.; Hu, X.; Hu, B.; Guo, Y.; Sadoun, B.; Kwok, R.Y.K. Joint Computing and Caching in 5G-Envisioned Internet of Vehicles: A Deep Reinforcement Learning-Based Traffic Control System. *IEEE Trans. Intell. Transp. Syst.* **2021**, *22*, 5201–5212. [CrossRef]
17. Ning, Z.; Yang, Y.; Wang, X.; Guo, L.; Gao, X.; Guo, S.; Wang, G. Dynamic Computation Offloading and Server Deployment for UAV-Enabled Multi-Access Edge Computing. *IEEE Trans. Mob. Comput.* **2023**, *22*, 2628–2644. [CrossRef]
18. Ning, Z.; Sun, S.; Zhou, M.; Hu, X.; Wang, X.; Guo, L.; Hu, B.; Kwok, R.Y.K. Online Scheduling and Route Planning for Shared Buses in Urban Traffic Networks. *IEEE Trans. Intell. Transp. Syst.* **2022**, *23*, 3430–3444. [CrossRef]
19. Ning, Z.; Dong, P.; Wang, X.; Hu, X.; Guo, L.; Hu, B.; Guo, Y.; Qiu, T.; Kwok, R.Y.K. Mobile Edge Computing Enabled 5G Health Monitoring for Internet of Medical Things: A Decentralized Game Theoretic Approach. *IEEE J. Sel. Areas Commun.* **2021**, *39*, 463–478. [CrossRef]
20. Wang, X.; Ning, Z.; Guo, L.; Guo, S.; Gao, X.; Wang, G. Mean-Field Learning for Edge Computing in Mobile Blockchain Networks. *IEEE Trans. Mob. Comput.* **2022**, 1–17.
21. Ning, Z.; Zhang, K.; Wang, X.; Guo, L.; Hu, X.; Huang, J.; Hu, B.; Kwok, R.Y.K. Intelligent Edge Computing in Internet of Vehicles: A Joint Computation Offloading and Caching Solution. *IEEE Trans. Intell. Transp. Syst.* **2021**, *22*, 2212–2225. [CrossRef]
22. Feng, J.; Richard Yu, F.; Pei, Q.; Chu, X.; Du, J.; Zhu, L. Cooperative Computation Offloading and Resource Allocation for Blockchain-Enabled Mobile-Edge Computing: A Deep Reinforcement Learning Approach. *IEEE Internet Things J.* **2020**, *7*, 6214–6228. [CrossRef]
23. Ning, Z.; Dong, P.; Kong, X.; Xia, F. A Cooperative Partial Computation Offloading Scheme for Mobile Edge Computing Enabled Internet of Things. *IEEE Internet Things J.* **2019**, *6*, 4804–4814. [CrossRef]
24. Ning, Z.; Huang, J.; Wang, X.; Rodrigues, J.J.P.C.; Guo, L. Mobile Edge Computing-Enabled Internet of Vehicles: Toward Energy-Efficient Scheduling. *IEEE Netw.* **2019**, *33*, 198–205. [CrossRef]
25. Ning, Z.; Hu, H.; Wang, X.; Guo, L.; Guo, S.; Wang, G.; Gao, X. Mobile Edge Computing and Machine Learning in The Internet of Unmanned Aerial Vehicles: A Survey. *ACM Comput. Surv.* **2023**. [CrossRef]
26. Das, R.; Inuwa, M.M. A review on fog computing: Issues, characteristics, challenges, and potential applications. *Telemat. Inform. Rep.* **2023**, *10*, 100049. [CrossRef]
27. Zhang, Y.; Li, Y.; Wang, R.; Hossain, M.S.; Lu, H. Multi-Aspect Aware Session-Based Recommendation for Intelligent Transportation Services. *IEEE Trans. Intell. Transp. Syst.* **2021**, *22*, 4696–4705. [CrossRef]
28. Desikan, K.E.S.; Kotagi, V.J.; Murthy, C.S.R. Decoding the Interplay Between Latency, Reliability, Cost, and Energy While Provisioning Resources in Fog-Computing-Enabled IoT Networks. *IEEE Internet Things J.* **2023**, *10*, 2404–2416. [CrossRef]
29. Ning, Z.; Huang, J.; Wang, X. Vehicular Fog Computing: Enabling Real-Time Traffic Management for Smart Cities. *IEEE Wirel. Commun.* **2019**, *26*, 87–93. [CrossRef]
30. Chen, C.; Liu, B.; Wan, S.; Qiao, P.; Pei, Q. An Edge Traffic Flow Detection Scheme Based on Deep Learning in an Intelligent Transportation System. *IEEE Trans. Intell. Transp. Syst.* **2021**, *22*, 1840–1852. [CrossRef]
31. Ning, Z.; Chen, H.; Wang, X.; Wang, S.; Guo, L. Blockchain-Enabled Electrical Fault Inspection and Secure Transmission in 5G Smart Grids. *IEEE J. Sel. Top. Signal Process.* **2022**, *16*, 82–96. [CrossRef]
32. Di Vaio, A.; Hassan, R.; Palladino, R. Blockchain technology and gender equality: A systematic literature review. *Int. J. Inf. Manag.* **2023**, *68*, 102517. [CrossRef]
33. Verma, S.; Zeadally, S.; Kaur, S.; Sharma, A.K. Intelligent and Secure Clustering in Wireless Sensor Network (WSN)-Based Intelligent Transportation Systems. *IEEE Trans. Intell. Transp. Syst.* **2022**, *23*, 13473–13481. [CrossRef]
34. Shen, M.; Tang, X.; Zhu, L.; Du, X.; Guizani, M. Privacy-Preserving Support Vector Machine Training Over Blockchain-Based Encrypted IoT Data in Smart Cities. *IEEE Internet Things J.* **2019**, *6*, 7702–7712. [CrossRef]
35. Zhao, Y.; Zhao, J.; Jiang, L.; Tan, R.; Niyato, D.; Li, Z.; Lyu, L.; Liu, Y. Privacy-Preserving Blockchain-Based Federated Learning for IoT Devices. *IEEE Internet Things J.* **2021**, *8*, 1817–1829. [CrossRef]
36. Wang, X.; Li, J.; Ning, Z.; Song, Q.; Guo, L.; Guo, S.; Obaidat, M.S. Wireless Powered Mobile Edge Computing Networks: A Survey. *ACM Comput. Surv.* **2023**, *55*, 1–37. [CrossRef]
37. Xiao, L.; Ding, Y.; Jiang, D.; Huang, J.; Wang, D.; Li, J.; Vincent Poor, H. A Reinforcement Learning and Blockchain-Based Trust Mechanism for Edge Networks. *IEEE Trans. Commun.* **2020**, *68*, 5460–5470. [CrossRef]
38. Song, Z.; Liu, Y.; Sun, X. Joint Task Offloading and Resource Allocation for NOMA-Enabled Multi-Access Mobile Edge Computing. *IEEE Trans. Commun.* **2021**, *69*, 1548–1564. [CrossRef]
39. Teng, Y.; Cao, Y.; Liu, M.; Yu, F.R.; Leung, V.C. Efficient Blockchain-Enabled Large Scale Parked Vehicular Computing With Green Energy Supply. *IEEE Trans. Veh. Technol.* **2021**, *70*, 9423–9436. [CrossRef]
40. Boysen, N.; Emde, S.; Stephan, K. Crane scheduling for end-of-aisle picking: Complexity and efficient solutions based on the vehicle routing problem. *EURO J. Transp. Logist.* **2022**, *11*, 100085. [CrossRef]

41. Samaraweera, G.D.; Chang, J.M. Security and Privacy Implications on Database Systems in Big Data Era: A Survey. *IEEE Trans. Knowl. Data Eng.* **2021**, *33*, 239–258. [CrossRef]
42. Wamba, S.F.; Queiroz, M.M. Blockchain in the operations and supply chain management: Benefits, challenges and future research opportunities. *Int. J. Inf. Manag.* **2020**, *52*, 102064. [CrossRef]
43. Di Vaio, A.; Varriale, L. Blockchain technology in supply chain management for sustainable performance: Evidence from the airport industry. *Int. J. Inf. Manag.* **2020**, *52*, 102014. [CrossRef]
44. Huang, F.; Xu, J.; Weng, J. Multi-Task Travel Route Planning With a Flexible Deep Learning Framework. *IEEE Trans. Intell. Transp. Syst.* **2021**, *22*, 3907–3918. [CrossRef]
45. Wang, W.; Xia, F.; Nie, H.; Chen, Z.; Gong, Z.; Kong, X.; Wei, W. Vehicle Trajectory Clustering Based on Dynamic Representation Learning of Internet of Vehicles. *IEEE Trans. Intell. Transp. Syst.* **2021**, *22*, 3567–3576. [CrossRef]