ISSN: 2664-9640

# A Data-driven Scheduling Scheme for Shared Bicycles based on Spatio-temporal Nested Line Segment Tree Modeling

Beichen Wang*, Xianggui Tan, and Dongpo Wu

Luoyang NO.1 Senior High School, Luoyang 471023, China

*Corresponding Author: beyondchrist@163.com

## Abstract

**The scheduling of shared bicycles often relies on manual experience, but frequent scheduling still cannot address the coexistence of "overcrowding" and "vacancies"; If artificial intelligence algorithms and big data frameworks are used, the cost is high and the operation is complex, making it difficult for small and medium-sized scheduling operators to undertake. The project constructs a mathematical model using a spatio-temporal dual dimensional nested line segment tree, breaking through the limitations of traditional single dimensional line segment trees and enabling rapid acquisition of bicycle supply and demand data at any time period and region. The designed data-driven bidirectional scheduling scheme can combine the stock and supply and demand data of parking points with the optimal quantity of bicycles across hourly intervals, introduce quantitative supply-demand differences and dynamic thresholds, and calculate and filter the scheduling volume of "small supply-demand fluctuations" based on spatio-temporal dimensions, thereby avoiding ineffective scheduling and achieving the transformation of scheduling operations from experience driven to data-driven. The implementation of the solution does not require complex big data frameworks or databases, is lightweight and easy to deploy, and adapts to the technical capabilities of small and medium-sized shared bicycle dispatch operators. It can also provide reference for span data processing in areas such as library partition book allocation and scarce water and electricity supply allocation.**

## Keywords

**Shared Bicycles; Bidirectional Scheduling; Segment Tree; Data-driven.**

## 1. Introduction

In the field of resource search and scheduling, data-driven intelligent decision-making technology is currently a hot research topic. Its core is to explore the spatio-temporal patterns in historical and real-time data, construct prediction and optimization models, and achieve efficient matching and dynamic scheduling of resources. Usually, data analysis techniques such as clustering, trend analysis, and anomaly detection are first used to identify the spatio-temporal distribution characteristics of resource supply and demand (such as peak areas and periods of demand for ride hailing services in cities); Then, a predictive model is constructed to estimate future demand, or an optimization model based on integer programming, reinforcement learning, etc. is constructed to solve the optimal scheduling strategy. For example, bike sharing platforms analyze user cycling trajectory data, establish demand forecasting models, and combine bicycle location information to schedule the transfer of bicycles from low demand areas to high demand areas, achieving balanced resource allocation [1]. In addition, intelligent optimization methods such as reinforcement learning have been used to solve dynamic resource scheduling problems in urban logistics, continuously optimizing scheduling strategies through interaction with the environment, and improving

ISSN: 2664-9640

response efficiency and cost control capabilities [2]. These data-driven intelligent decision-making technologies integrate methods such as statistics, machine learning, and operations optimization, and are widely used in logistics distribution, intelligent transportation, cloud computing resource management, and other fields, significantly improving resource utilization and service response efficiency. However, the models involve many factors, complex algorithms, and high deployment costs.

For small and medium-sized enterprises that only undertake bicycle scheduling tasks in the supply-demand imbalance of shared bicycles, or those who do not have the ability to deploy big data frameworks, lightweight modeling techniques and simple and efficient problem-solving are important factors to consider. Reasonable selection of data structures and algorithms is the key to efficiently managing spatio-temporal dynamic resources and building data-driven scheduling systems. The data structures commonly used for resource search and scheduling include line segment trees, heaps, graphs, etc. Line segment trees can efficiently handle span queries and dynamic updates, and are suitable for fast retrieval of spatio-temporal resource distribution [3]; The heap can maintain real-time resource priority and quickly obtain the optimal resource items in task scheduling [4]; The use of graph structure can establish a model for resource flow paths, combined with adjacency table storage to improve the traversal efficiency of large-scale networks [5]. When implementing scheduling strategies, greedy algorithms can achieve approximate optimal scheduling through local optimal decisions, such as the shortest path priority strategy in bicycle routing problems [6]; Dynamic programming is suitable for multi-stage resource allocation, optimizing long-term scheduling benefits through state transition equations [7]; Heuristic algorithms, such as genetic algorithms and simulated annealing, can solve approximate optimal solutions under complex constraints and are commonly used in large-scale resource scheduling scenarios [8]. However, both heap and hash tables are good at maintaining global minima and have the advantage of query complexity O (1). They are suitable for single point queries but cannot efficiently query local minima or sums within intervals; The graph structure is more suitable for path planning problems, and its core is to solve the problems of "accessibility between nodes" and "path cost" such as distance and time. If a relatively simple data structure is used, such as a two-dimensional array to store spatio-temporal data, the spatial complexity is O (M × N) (M is the temporal dimension, N is the spatial dimension), and the scale is difficult to dynamically adjust, which is not suitable for scenarios where data changes dynamically. Therefore, among the above data structures, line segment trees are more suitable for supporting fine-grained queries of "certain time span and certain area" in order to quickly respond to the scheduling between parking stations.

## 2. A Mathematical Model for the Spatio-temporal Stock of Parking Spots

### 2.1. Double Layered Nested Tree Structure

As shown in Figure 1., the dual dimensional nested line segment tree constructs a nested structure of "outer time line segment tree+inner space line segment tree" with "time axis" and "space axis" as the two major dimensions:

① Outer time segment tree: Divide a day and 24 hours into non overlapping time intervals (such as the root node being [0,23] hours, the next layer being split into [0,11] and [12,23], and recursively split into leaf nodes of individual hours [h, h], called "hourly interval nodes"), with each hourly interval node corresponding to a complete "inner space segment tree".

② Inner space segment tree: Taking all parking points in the city as objects (such as 1000 parking point numbers from 0 to 999, the adjacent numbers correspond to the parking spots near the spatial locations), it is also split into non overlapping parking point intervals (such as the root node being [0,999], the next layer is split into [0,499] and [500,999], recursively split into a single parking point [p, p], this type of leaf node is called a "parking point node").

ISSN: 2664-9640

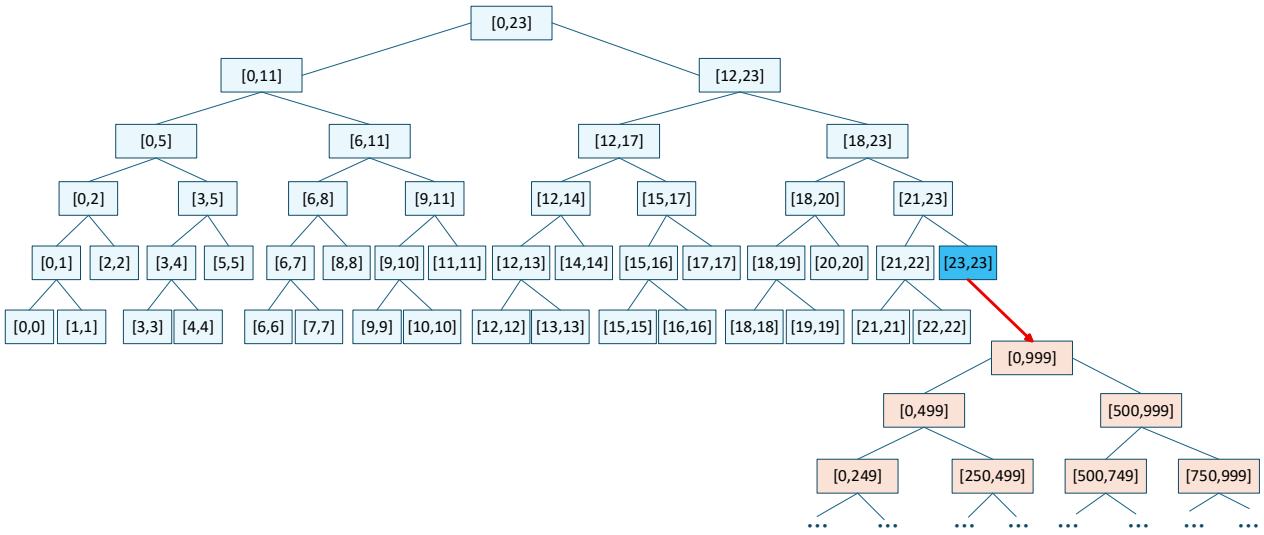


**Figure 1.** Logical structure of spatio-temporal dual dimensional line segment tree

## 2.2. Definition and Storage Rules of Node Data in Different Hourly Interval

Based on the relationship between hourly interval and "current time" (non current, current), set differentiated data storage content for each parking point node to ensure that the data meets actual scheduling needs:

① Non current hourly interval (hours before the current time, such as 14:00, then 0-13:00 and 15-23:00 on the same day are non current hourly intervals)

- Storage data: The "historical average storage capacity (Avg_hist)" and "historical optimal quantity of bicycles stored (O_hist)" of the parking spot in that hourly interval.
- Data source: Take the average inventory of bicycles at the same parking spot during the same time period in the past 7 days (such as calculating the average quantity of bicycles from 10 to 11 o'clock in the past 7 days as the Avg.hist for today's 10 to 11 o'clock); O_hist is set based on the historical average storage capacity (such as Avg_hist × 1.2, to ensure it can meet historical peak demand).
- Function: To provide data support for analyzing the "historical flow pattern", such as comparing the actual stock that has passed the hourly interval today with the historical average, and determining whether there is abnormal accumulation/shortage.

② Current hourly interval (the hour that coincides with the current time, such as 14:30, the current hourly interval is 14, indicating 14-15 pm)

- Storage data: The "real-time bicycle storage capacity (N_current)" and "current optimal quantity of bicycles stored (O_current)" of the parking spot in that hourly interval.
- Data source: N_current is collected and integrated in real-time through parking point sensors or shared bike server side (updated every 5 minutes to ensure data timeliness); The combination of "historical average storage capacity O_hist" and "current date passenger flow characteristics" is used to set the O_current (such as the business district parking spot O_current=O_hist × 1.3 during peak weekend travel periods).
- Function: As the core basis for real-time scheduling, by calculating the "supply-demand difference D=N_current - O_current", it can directly determine whether the parking spot needs to be adjusted in (D<0) or out (D>0).

## 2.3. Range Aggregation and Query

The dual dimensional nested segment tree achieves efficient range queries by "merging sub node data layer by layer", supporting operators to quickly obtain bicycle inventory information in any spatio-temporal range:

ISSN: 2664-9640

① Spatial dimension aggregation: Within the same hourly interval, non leaf nodes of the spatial line segment tree (such as parking point intervals [50,200]) converge and merge the core data of sub nodes - "maximum real-time supply-demand difference of interval equals maximum D of sub range (N_current - O_current, including both positive and negative maximum supply-demand differences and corresponding node indexes of parking points)" and "average historical stock of interval equals sum of Avg_hist of each parking point in sub interval", making it convenient to quickly query the overall situation of a certain area.

② Time dimension aggregation: Within different hourly intervals, non leaf nodes of the time segment tree (such as the morning rush hour in time intervals [7,9]) converge and merge spatial tree data of child nodes (hourly intervals of each hour) - for example, the total supply and demand difference between 7 and 9 o'clock in the morning rush hour (including positive and negative values, i.e., the total sum of stacked bicycles and the total sum of required bicycles), and support trend analysis across time periods.

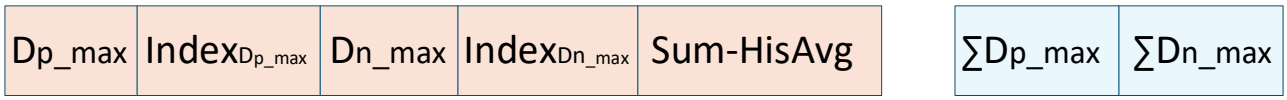


**Figure 2.** Logical structure of the intermediate node between the spatial segment tree (left) and the temporal segment tree (right)

As shown in Figure 2, the middle node of the spatial line segment tree stores the maximum stack bicycle supply and demand difference Dp_max (D>0) and its corresponding parking point node index $Index_{Dp\_max}$, the maximum shortage bicycle supply and demand difference Dn_max (D ≤ 0) and its corresponding parking point node index $Index_{Dn\_max}$, as well as the sum of the historical average parking volume of each parking point in the index range during the current period, Sum-HisAvg; The middle node of the time segment tree holds two values: the total amount of bicycles piled up at each parking point within the current time interval, ∑Dp_max, and the total amount of missing bicycles, ∑Dn_max. If the two values are close, it indicates that the supply and demand of bicycles in the region during that time period are basically balanced. Otherwise, it belongs to the overall state of piled up or required bicycles, providing reference for the company to release new bicycles or reduce the investment in operating bicycles.

Query example: If you need to query the maximum supply-demand difference and corresponding parking points for the evening rush hour (time range) from 17pm to 19pm and the parking point (spatial range) from NO.50-200, the model will first locate the [17,19] range of the time segment tree, find the [17,17], [18,18], [19,19] leaf nodes, and then call the corresponding nodes of the spatial tree under the three nodes [50,200] to directly extract the aggregated "maximum pile bicycle supply-demand difference Dp_max" and "maximum shortage bicycle supply-demand difference Dn_max" and their corresponding parking points, and then perform a simple cross area maximum value extraction without traversing the original data of each parking point, greatly improving query efficiency.

This model retains the advantages of efficient line segment tree queries through "spatio-temporal binding+time-based data storage", while also fitting the operational characteristics of shared bicycles, which require real-time changes and a combination of historical and predictive data, providing a solid mathematical data foundation for subsequent precise scheduling.

## 3. Line Segment Tree Search/Update Optimization

The core idea of traditional line segment sloth marking technology is: when querying or updating intervals, if the current node range completely covers the target interval, directly operate on the current node; If it is not fully covered, first drop the Lazy Propagation to the

ISSN: 2664-9640

child node, and then recursively process the child node. By delaying the update of child nodes, unnecessary operations are reduced and the efficiency of segment trees during range operations is improved. Due to the need to place Lazy Propagation before recursion to ensure data consistency, although this process is correct, it brings frequent memory access and recursion overhead.

In response to the high query efficiency requirements of the shared bicycle scheduling system, Permanent Lazy Propagation technology is introduced to optimize the line segment tree search/update algorithm. This technology significantly improves query performance by avoiding actual delegation operations, especially in scenarios where high-frequency updates and queries are handled.

## 3.1. The Core Idea of Making Lazy Propagation Permanent

Lazy Propagation permanence is an innovative line segment tree optimization strategy, whose core lies in breaking the constraints that traditional Lazy Propagation must be placed. In traditional methods, Lazy Propagation need to be dropped to child nodes before recursive updates or queries to ensure data consistency, but this can result in multiple memory writes and recursive calls. And Lazy Propagation permanence is optimized through the following methods:

Update operation: When modifying a certain interval, only set Lazy Propagation on fully covered nodes and update node values , without immediately recursively dropping them to child nodes.

Query operation: During the query process, the Lazy Propagation values of all ancestor nodes on the cumulative path are passed through parameters, and the total impact of the Lazy Propagation is calculated at once when reaching the target node, thus avoiding actual deployment.

This method reduces unnecessary memory access and recursion depth, improves cache efficiency, while maintaining the correctness of query results.

The operation flowchart for permanently marking after improvement is shown in Figure 3.:

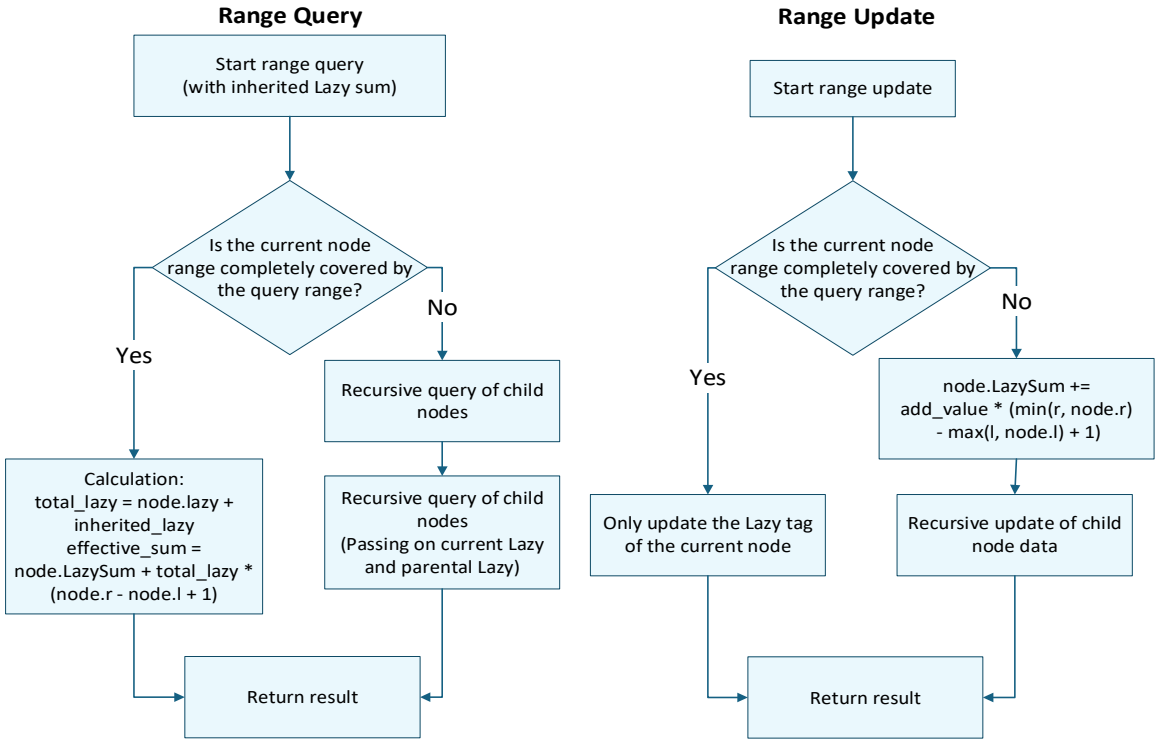


**Figure 3.** Optimized query/update process by Lazy Propagation permanent

ISSN: 2664-9640

The efficiency of making Lazy Propagation permanent is mainly reflected in reducing the times of deployment operations. In traditional Lazy Propagation, each time an update or query encounters a partially covered interval, it is necessary to recursively drop the Lazy Propagation, resulting in a linear correlation between the times of operations and the depth of the tree. If the tree depth is large or the operation is more frequent, the times of drops will increase sharply. In a tree with a depth of h, a partial coverage update may trigger O (h) drops.

However, Lazy Propagation permanence completely avoids delegation operations and only dynamically calculates cumulative values during queries, significantly reducing time complexity. This optimization method, which reduces memory writing and recursive calls, improves cache locality and overall performance.

### 3.2. Improvement Effect

In the shared bicycle scheduling system, high-frequency queries and updates are achieved through double recursion, which maintains O (logT × logP) time complexity compared to traditional line segment trees, T and P represent the quantity of hourly intervals and parking spots respectively. However, the smaller constant factor enables this optimization to significantly reduce response time. Table 1. shows the improvement effect before and after optimization.

**Table 1.** line segment tree query/update efficiency improvement by Lazy Propagation permanent

| | Code length (KB) | Time (s) | Memory usage (MB) |
|---|---|---|---|
| traditional algorithm | 1.48 | 1.97 | 12.55 |
| Lazy Propagation permanent | 1.43 | 1.79 | 12.54 |

According to Table 1, compared to traditional line segment trees, the improvement efficiency of the Lazy Propagation permanent optimization algorithm in terms of code length, time, and memory usage has increased by 4.73%, 9.14%, and 0.08%, respectively, and the improvement effect is very significant.

## 4. Scheduling Quantity Calculation and Bidirectional Scheduling Strategy

### 4.1. Scheduling Threshold Setting

The core function of the scheduling threshold is to filter out "small supply and demand fluctuations" (such as when the current quantity of bicycles stored at a parking point is one more than the optimal value, there is no need for frequent scheduling), while avoiding "supply and demand imbalance caused by excessively high thresholds". The threshold cannot be fixed and needs to be dynamically set in combination with the "current area/parking demand" and "future time demand" (the optimal quantity of bicycles stored in the next hourly interval).

If scheduling is based on regions, the sum of the inventory and optimal quantity of parking points within the region is first aggregated, and then bidirectional scheduling between regions is achieved. If scheduling is based on parking points, the inventory and optimal quantity of parking points within the scheduling region are directly calculated to achieve bidirectional scheduling within the region. The following illustrates the bidirectional scheduling strategy using the most basic parking point scheduling as an example. The specific design is as follows:

① Threshold calculation formula

$$T = \max(\alpha \times O\_current, \beta \times |O\_current - O\_next|) \quad (1)$$

ISSN: 2664-9640

T is the triggering threshold for scheduling, and scheduling is only triggered when the absolute value of the pre-adjustment degree of a single parking point | D |>T; If | D | ≤ T, do not schedule temporarily (considered as an acceptable fluctuation range). Where D is the supply-demand difference between "actual stock - optimal stock":

$$D = N\_current - O\_current \quad (2)$$

The supply and demand status of parking spots can be directly determined by the positive, negative, and absolute values of D:

- If D>0: The actual quantity of bicycles exceeds the optimal value, and the parking point is "bicycle accumulation", it needs to be adjusted (the adjustment amount is the value of D);
- If D<0: the actual quantity of bicycles is less than the optimal value, and the parking point is "short of bicycles", it needs to be adjusted (the adjustment amount is the absolute value of D);
- If D=0: The actual stock perfectly matches the optimal value, no scheduling is required.

② The meaning and design reasons of parameters in the threshold calculation formula (as shown in Table 2.)

**Table 2.** The parameter meanings in the threshold calculation formula

| | Parameter definition | Value suggestion | Design rationale |
|---|---|---|---|
| O_current | Optimal bicycle quantity of the parking point in current hourly interval | Based on historical passenger flow and current date passenger flow characteristics | Reflecting the core demand of the parking spot during the current time period, the larger the optimal value, the larger the acceptable fluctuation range (threshold) should be |
| O_next | Optimal bicycle quantity of the parking point in next hourly interval | | Anticipate future demand changes: If there is a sudden increase in demand in the next hourly interval (such as O_next being much greater than O_current), even if the current D fluctuation is small, the threshold needs to be relaxed to replenish bicycles in advance |
| α | Proportional coefficient of the current optimal value | 0.1-0.2 | Control the 'allowable fluctuation of current demand': for example, with O_current=30 and α=0.1, then the product of α and O_current equals 3 (i.e. ± 3 bicycles are not scheduled) |
| β | Proportional coefficient of the optimal difference between the front and rear hourly intervals quantity | 0.3-0.5 | Control the fluctuation of future demand impact: determined by the average passenger flow in hourly intervals before and after historical periods |

## 4.2. Design of Dispatch Quantity between Two Parking Spots in an Area

If only selecting the "parking point with the highest amount of transfer in (denoted as S_in, D_in<0, taking the absolute value as |D_in|)" and "parking point with the highest amount of transfer out (denoted as S_out, D_out>0)" within the area for bidirectional scheduling, it is necessary to take into account the "maximum amount of transfer out parking point", "maximum amount of transfer in parking point", and "effective demand after threshold filtering". The specific design is as follows:

ISSN: 2664-9640

① Core principles

Retrieve the parking point S_out: At most, only the "effective retrieval amount after threshold filtering" can be retrieved (i.e. D_out - T_out, where T_out is the scheduling threshold of S_out);

Entering the parking point S_in: At most, only the "threshold filtered effective entry amount" (i.e. | D_in | - T_in, where T_in is the scheduling threshold of S_in) can be entered;

The final scheduling quantity is taken as the smaller of the two (to avoid "transfer out quantity exceeding transfer in demand" or "transfer in quantity exceeding transfer out capacity").

② Calculate the adjustment degree C for two parking spots

$$C = \min (D\_out - T\_out, |D\_in| - T\_in) \quad (3)$$

## 4.3. Solution Verification

Through nearly two weeks of research, a partial hourly interval data of 40 shared bicycle parking spots of a certain brand in Luolong District, Luoyang City was compiled, and predicted (30%) to generate storage data and optimal storage quantity for other hourly interval parking spots. The data is shown in Table 3, where parking spots 1-20 are located near residential areas (including surrounding subway stations/bus stations), and parking spots 21-40 are located near business districts (including surrounding subway stations/bus stations).

**Table 3.** The parking quantity (N) and optimal storage quantity (O) data for 40 parking points and 24 hourly intervals (partial)

| Hourly interval | **1_N** | **2_N** | **...** | **40_N** | **1_O** | **2_O** | **...** | **40_O** |
|---|---|---|---|---|---|---|---|---|
| 0 | 18 | 16 | ... | 17 | 3 | 2 | ... | 4 |
| 1 | 17 | 15 | ... | 16 | 2 | 1 | ... | 3 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 11 | 8 | 7 | ... | 9 | 15 | 14 | ... | 17 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |

Using C++programming to write code, input the start and end numbers of the hourly interval span and parking point span to be queried, in order to achieve dynamic scheduling threshold calculation for each parking point within the parking point span of the hourly interval span, fast querying of spatial line segment trees, and data-driven intelligent scheduling decision-making and cross hourly interval span data synchronization. Thus, two types of results can be achieved correctly. The first type is to find parking spots that urgently need bidirectional scheduling and implement scheduling. The second type is to filter out scheduling volumes with "minor supply and demand fluctuations" and avoid frequent and inefficient scheduling. The data-driven bidirectional scheduling scheme is feasible and effective.

# 5. Conclusion

The paper focuses on the problem of imbalanced supply and demand of shared bicycles, and constructs a scheduling system based on a spatio-temporal dual dimensional nested line segment tree. It does not require a complex big data framework and can quickly query the supply and demand situation of bicycles in any time period within 24 hours through lightweight algorithms. It dynamically calculates and adjusts the scale, helping operators shift from empirical scheduling to data-driven scheduling, effectively solving the problem of full or shortage of bicycles. The system breaks through the single dimensional limitations of traditional line segment trees, optimizes search algorithms, improves code efficiency and

ISSN: 2664-9640

memory usage, and also designs a bidirectional scheduling strategy to reduce ineffective scheduling, thereby achieving low-cost data-driven scheduling capabilities.

## References

[1] J. Zhang, et al.: Deep Spatio-Temporal Neural Networks for Ride-Hailing Demand Forecasting, IEEE Transactions on Knowledge and Data Engineering, Vol. 34 (2022) No. 6, p. 2789-2802.

[2] Y. Li, et al.: Reinforcement Learning for Dynamic Resource Scheduling in Urban Logistics, Transportation Research Part C: Emerging Technologies, Vol. 146 (2023) No. 1, p. 103987.

[3] W.M. Yan, W.M. Wu: Data Structures (C Language Edition) (Tsinghua University Press, China 2007).

[4] T.H. Cormen, C.E. Leiserson, R.L. Rivest, et al.: Introduction to Algorithms (MIT Press, USA 2009).

[5] S.S. Wang, et al.: Graph Theory and Its Applications (Science Press, China 2019).

[6] J. Kleinberg, E. Tardos: Algorithm Design (Addison-Wesley, USA 2005).

[7] R. Bellman: Dynamic Programming, Science, Vol. 128 (1957) No. 3331, p. 983-989.

[8] D.E. Goldberg: Genetic Algorithms in Search, Optimization, and Machine Learning (Addison-Wesley, USA 1989).