A Data-driven Spatiotemporal Simulator for Reinforcement Learning Methods

Dingyuan Shi
Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing, Beihang University
chnsdy@buaa.edu.cn

Bingchen Song
Beijing Institute of Astronautical Systems Engineering
songbch@foxmail.com

Yuanyuan Zhang
North China Institute of Computing Technology
zyy-buaa@buaa.edu.cn

Haolong Yang
Beihang University
yanghaolong@buaa.edu.cn

Ke Xu
Beihang University
kexu@buaa.edu.cn

ABSTRACT

Spatiotemporal applications such as taxi order dispatching and warehouse task scheduling depend critically on the algorithms for operational efficiency. However, the inherent dynamic nature of these applications presents challenges in algorithm design. The growth of mobility services has facilitated the collection of extensive spatiotemporal data, which in turn prompted algorithm designers to use data-driven methods. Reinforcement learning (RL), recognized for its strong performance and suitability for spatiotemporal contexts, has garnered considerable research interest. Despite their potential, RL algorithms necessitate the use of a simulator for both training and validation purposes. However, no specific simulation system has been developed for spatiotemporal algorithm design. This vacancy hinders the progress of spatiotemporal algorithm designers. In this demo, we build a system called Data-driven Spatiotemporal Simulator (DSS), hoping to bring convenience for spatiotemporal algorithm designers. DSS is adept at handling problems related to taxi order dispatching and warehouse task scheduling and possesses the versatility to be expanded for other user-defined scenarios. The system includes visualization modules that offer insightful panels, alongside developer tools designed to streamline the development process. This enables designers to efficiently craft, evaluate, and refine their algorithms, potentially accelerating innovation in spatiotemporal application development.

PVLDB Reference Format:
Dingyuan Shi, Bingchen Song, Yuanyuan Zhang, Haolong Yang, and Ke Xu. A Data-driven Spatiotemporal Simulator for Reinforcement Learning Methods. PVLDB, 17(12): 4257 - 4260, 2024.
doi:10.14778/3685800.3685849

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/dingyuan-shi/spatiotemporal-simulator-rl.

1 INTRODUCTION

Recent decades have seen mobile services give rise to numerous spatiotemporal applications, such as route planning[9], taxi dispatching [11], logistics [10], and so on. Algorithms determine the operational efficiency in these domains; for example, sophisticated taxi dispatch algorithms enhance driver efficiency [4], while advanced scheduling in smart warehouses maximizes throughput by optimizing robot-assisted rack transportation [8]. The inherently dynamic spatial and temporal aspects of these challenges pose significant difficulties for algorithm designers. The growing body of spatiotemporal data has spurred a surge in reinforcement learning (RL), which excels at navigating the complexities of agent-environment interactions within such dynamic systems. RL's efficacy in adapting to these complex spatiotemporal fluctuations underscores its notable successes in diverse applications [4].

The design and validation of reinforcement learning (RL) algorithms are heavily reliant on the use of simulators. Platforms like Gym provide an extensive suite of integrated simulators for RL development, including environments modeled after Atari games. These simulation tools have been instrumental in driving forward the evolution of reinforcement learning techniques. Despite these advancements, there remains a conspicuous absence of specialized systems tailored for the design of spatiotemporal algorithms. This gap represents a significant bottleneck impeding the advancement of researchers and practitioners in the field of spatiotemporal algorithm development.

In this demonstration, we introduce a system named Data-driven Spatiotemporal Simulator (DSS), designed to facilitate the work of spatiotemporal algorithm designers. DSS is composed of three principal modules: the Environment module, the Visualization module, and the Developer Tools module.

Environment module. Within this module, we have developed two simulators: one for taxi order dispatching and another for warehouse task scheduling. The taxi order dispatching simulator replicates the dynamic interactions between drivers and incoming orders, where it consistently scans for new orders using the provided dataset, identifies available drivers, and generates a bipartite graph that incorporates additional data about the drivers and orders. Subsequently, the simulator invokes the dispatching algorithm to determine an effective order allocation scheme and executes the process of delivering passengers. This simulator also accommodates driver repositioning strategies in line with prevalent research [7, 12].

The warehouse task scheduling simulator, emulates the operational workflow within a smart warehouse, where robots are utilized to transport racks to pickers. It commences by constructing the warehouse layout according to predefined configuration parameters. Following that, it assigns picking tasks to each rack at discrete time intervals and supplies the algorithm with pertinent information including robots' positions, the queue lengths of pickers' tasks, and the anticipated time for task completion. Based on this input, the algorithm devises a plan for directing robots to efficiently distribute racks to pickers.

Both simulators are designed to operate in two modes: simulator-centric and agent-centric. In simulator-centric mode, the simulation is automatically propelled by an internal clock. Conversely, in agent-centric mode, the simulation behaves akin to a Gym-style environment, where progression is controlled through the `env.step()` method invocation. Our framework is intentionally crafted to be user-friendly, allowing algorithm developers to seamlessly integrate their own custom environments. This flexibility underscores the framework's capacity to accommodate a diverse range of spatiotemporal application scenarios.

Visualization module. This module provides interactive panels that supply algorithm designers with comprehensive insights into the performance of their algorithms. These panels present a combination of simulation visuals and pertinent statistical data. For the taxi order dispatching simulator, a central map display provides a visualization of the distribution of orders and drivers, while side panels offer various performance metrics such as cumulative rewards, order response rates, and algorithm execution times, among others. In the context of the warehouse task scheduling simulator, the main panel depicts the entire operational flow of the warehouse, including an aisle heat map that highlights areas prone to congestion. Complementing this, the side panels furnish additional information like the ratio of working time and other relevant performance indicators. Together, these visual and statistical tools empower algorithm designers with both an intuitive understanding and a granular analysis of their algorithms, laying the groundwork for more informed and effective enhancements.

Developer Tool module. This module comes equipped with an array of pre-packaged algorithms, extending beyond the realm of reinforcement learning (*e.g.*, REINFORCE, Deep Q-Learning, Proximal Policy Optimization) to include various combinatorial algorithms commonly employed in spatiotemporal applications such as Kuhn-Munkres matching algorithm. While this suite of built-in offerings is robust, algorithm designers also retain the flexibility to incorporate third-party libraries into their design process, ensuring they can tailor solutions to their specific needs.

Through this demonstration, we hope to streamline the algorithm design process, making it more straightforward to identify and address potential issues and bottlenecks within their algorithms, which will expedite the iterative refinement of their solutions, leading to faster advancements and optimizations.

2 SYSTEM DESIGN

2.1 System Overview

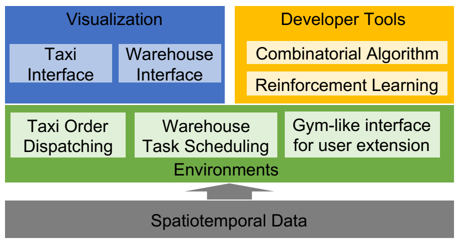


Figure 1: System Overview

DSS is a user-friendly, comprehensive framework tailored for the creation and assessment of data-driven spatiotemporal algorithms. It comprises three core modules: Environments (`Env`), Visualization (`Visual`), and Developer Tool (`DevTool`), as illustrated in Fig. 1. The `Env` module includes two classic spatiotemporal scenarios—taxi order dispatching and warehouse task scheduling—to serve as foundational environments. `Visual` offers panels that display simulation processes and statistical analyses, enabling algorithm developers to gain a better understanding of their work. `DevTool` equips users with a variety of algorithms, both from combinatorial optimization and reinforcement learning domains, to facilitate algorithm development. Next, we will elaborate respectively as below.

2.2 Environments

We implement two simulation environments: taxi order dispatching and warehouse task scheduling.

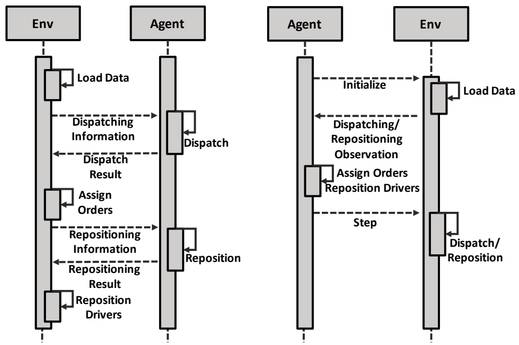
Taxi order dispatching. This simulator is designed for evaluating order dispatching algorithms.

The simulator can be triggered independently. Upon activation, the simulator initializes by loading data pertaining to taxis and orders, with drivers and orders appearing at their designated times. In bi-secundal intervals, it aggregates all available drivers and pending orders, identifying potential matches within a set distance parameter to construct a bipartite graph. This graph, coupled with statistics such as order value, driver-to-order proximity, and destination data, is provided to the implemented dispatch algorithm. The result is an expected dispatch plan that informs the assignment of orders to drivers. Engaged drivers then fulfill their tasks, reverting to availability post-estimated arrival, while idle ones relocate randomly in line with historical trends.

Additionally, the simulator incorporates a driver repositioning feature which, at predetermined intervals (*e.g.*,, every five minutes), presents idle drivers' details and earnings to the algorithm for strategizing positional adjustments. During repositioning, drivers remain eligible for order allocations.

It can also be used like a Gym environment, the simulation enables the order dispatching algorithm, cast as an agent, to guide the process via `env.step(Action)`. This function call yields a tuple with observation (a comprehensive dictionary for dispatching decisions), accumulated reward (total value of dispatched orders), finish status (boolean indicating completion), and customizable user information.

Throughout the simulation, pertinent metrics such as algorithm execution time, cumulative rewards, ratios of idle drivers, order fulfillment rates, and spatial distributions are logged for analysis. These insights are crucial for developers to refine their algorithms and can be visually represented through the Visualization module as detailed in Sec. 2.3.



(a) Environment-Centric (b) Agent-Centric

Figure 2: Workflows of two simulation modes.

Warehouse task scheduling. This simulator is designed for warehouse task scheduling. In such smart warehouses, items are stored on mobile racks that robots transport throughout the facility. When an order arrives, a robot performs a multi-step task: it retrieves a rack with the required item, delivers it to a picker, waits during the item-picking process, and finally returns the rack to its storage location. The scheduling challenge comprises two key sub-problems: task assignment [8] and collision-aware route planning [10]. The former requires an algorithm to determine efficient robot-to-rack assignments, while the latter demands path-planning for robots that concurrently avoid collisions.

Our simulator replicates the intricate operations of a smart warehouse. Analogous to the taxi dispatching simulator, it offers two operational modes: environment-centric, where simulations operate autonomously, and agent-centric, mimicking a Gym environment where actions are agent-driven. The distinct phases of both modes are depicted in Fig. 2.

Metrics recorded during simulation include ratios of rack, picker, and robot utilization times, alongside algorithm execution time. These statistics aid in optimizing task efficiency and are essential for developers to understand algorithm performance within a dynamic warehouse context.

2.3 Visualization

For the above two environments, we implemented two visualization modules, respectively.

Visualization for Taxi. In this module, we provide visualizations for many statistics which vary along with the simulation process. It includes an interactive progress bar, enabling algorithm designers to scrutinize specific moments within the simulation. The progress bar allows users to pause, resume, adjust the replay speed and navigate seamlessly through the simulation timeline in both directions.

Implementing bidirectional scrubbing on the progress bar is technically challenging due to the need for instantaneous updates to drivers' states, as they may be engaged in time-spanning tasks such as passenger delivery. So we need to quickly update all drivers' states to respond user's drag action. To achieve this, we build an R-tree based index. Specifically, an order (denoted as o) has a start time $o.s$ and a finish time $o.f$. When dragging the progress bar to time t, we need to find orders whose time span can cover this query time t (*i.e.*, $o.s \leq t \leq o.f$). In other words, the order is being served at time t. We put all start and end times of all orders in an R-tree. *e.g.*, for any order, we put its start and finish time in an R-tree using coordinates $(o.s, o.f)$. For a query time t, we query the R-tree using the box with diagonal coordinate pair $(0, t)$, (t, t_m), where t_m is the largest time on the progress bar. Orders falling in this box satisfy $o.s \leq t \leq o.f$. Utilizing the R-tree index, active orders can be retrieved efficiently in $O(\log(N))$ time, where N is the order numbers within one day.

Visualization for Warehouse. The warehouse visualization module displays the task scheduling process. In the central panel, it displays how the racks are being delivered between pickers and storage areas. Please refer to Sec. 3 for a more detailed introduction.

2.4 Developer Tool

Our `DevTool` equips algorithmic designers with a comprehensive suite of developer utilities, streamlining the design process by obviating the need for basic operator implementation and allowing for an enhanced focus on problem-solving. The toolkit bifurcates into two principal sub-modules: the Combinatorial Algorithms Library (`CombLib`) and the Reinforcement Learning Library (`RLLib`).

`CombLib` encompasses essential matching algorithms such as the Kuhn-Munkres Algorithm [1], along with graph planning algorithms like Breadth-First Search and Depth-First Search. These are critical for applications in spatial-temporal contexts, including order dispatching and route planning in industries like ride-hailing. Optimizing for performance, traditional algorithms are crafted in C++ and interfaced with Python for user accessibility.

`RLLib` features an array of reinforcement learning strategies, including REINFORCE [13], Deep Q-learning [3], Deep Deterministic Policy Gradient [2], and Proximal Policy Optimization [6]. Additionally, it integrates the prioritized experience replay mechanism [5] to enhance sampling efficiency.

3 DEMONSTRATION SCENARIO

3.1 Taxi Order Dispatching Algorithm Analysis

After the algorithm design, we can launch the simulation process either in environment-centric or agent-centric mode. Aster that, the results will be stored under *DSS_DIR/workdir*. Subsequent to simulation, the visualization module is activated, with the interface depicted in Figure 3, comprising six distinct panels for comprehensive analysis.

Panel A presents a geographic map with a heat map overlay that delineates the distribution of orders and drivers. A toggle in the lower-left corner permits the user to alternate between visualizing idle drivers and incoming orders. Additionally, it is feasible to trace the routes of active drivers, thus offering a spatially encompassing

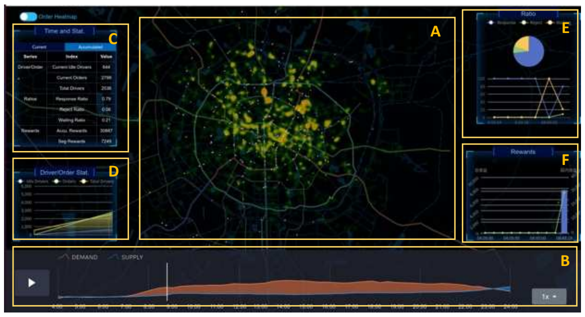


Figure 3: Illustration for taxi order dispatching.

perspective on driver availability, engagement, and service requests. Panel B integrates a dynamic scrollbar that portrays fluctuations in driver and order counts over time, assisting the designer in comprehending temporal distributions. Buttons adjacent to this scrollbar enable pausing and resuming the process, along with adjusting simulation velocity up to quintuple the standard rate. Moreover, as detailed in Section 2.3, the scrollbar facilitates temporal navigation by dragging. Panels C through F deliver exhaustive statistical data, including cumulative rewards, response and rejection ratios, execution times, and so on, to furnish algorithm designers with multifaceted and informed insights into order dispatch algorithms. These statistics are represented both graphically, with bars, and quantitatively, via polylines, to enhance interpretability.

3.2 Warehouse Task Scheduling Algorithm Analysis

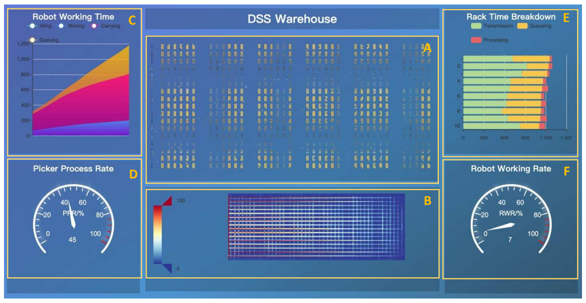


Figure 4: Illustration for warehouse task dispatching.

Fig. 4 illustrates simulation process for warehouse, which is composed of six panels.

Panel A graphically displays the operational state, where robots transport racks to pickers, offering algorithm designers an immediate visual comprehension of their algorithm's functionality. Panel B projects a heat map of warehouse traffic, with warmer hues representing higher visitation frequencies. The visualization suggests that the heated horizontal aisles, which traverse the entire warehouse, are prime candidates for expansion to alleviate congestion and enhance throughput. Alternatively, algorithmic adjustments could be made to distribute usage more evenly across these aisles. In Panel C, the temporal segmentation of robot activity is depicted, categorizing time into: *(i)* idle periods, *(ii)* locomotion, *(iii)* transit with rack-loaded, and *(iv)* queuing while laden with racks for processing. This granular breakdown assists designers in pinpointing efficiency bottlenecks and optimizing robot scheduling. Panel E delves into the temporal dynamics associated with rack movement, particularly focusing on the top 10 most frequently moved racks. It highlights the distribution of time spent, facilitating the identification of potential congestion points. For instance, excessive queuing times suggest that algorithms should address both path selection and task timing to expedite operations.

Panels D and F display the processing rates for pickers and working rates for robots, respectively, indicators of warehouse resource utilization. Suboptimal ratios may signal opportunities for algorithmic refinement to maximize resource deployment. Moreover, if low ratios persist despite operational efficiency, it may indicate that the quantities of robots or pickers exceed current requirements.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their suggestions. This work was partially supported by National Key Research and Development Program of China under Grant No. 2023YFF0725103, National Science Foundation of China (NSFC) (Grant Nos. U21A20516, 62336003) and Beijing Natural Science Foundation (Z230001), the Basic Research Funding in Beihang University No.YWF-22-L-531, and Didi Collaborative Research Program NO2231122-00047. Yuanyuan Zhang is the corresponding author in this paper.

REFERENCES

[1] Harold W Kuhn. 1955. The Hungarian method for the assignment problem. *Naval research logistics quarterly* 2, 1-2 (1955), 83–97.
[2] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2016. Continuous control with deep reinforcement learning. In *ICLR*.
[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. *CoRR* abs/1312.5602 (2013).
[4] Zhiwei Tony Qin, Hongtu Zhu, and Jieping Ye. 2022. Reinforcement learning for ridesharing: An extended survey. *Transportation Research Part C: Emerging Technologies* 144 (2022), 103852.
[5] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. 2016. Prioritized Experience Replay. In *ICLR*.
[6] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *CoRR* abs/1707.06347 (2017).
[7] Dingyuan Shi, Yongxin Tong, Zimu Zhou, Bingchen Song, Weifeng Lv, and Qiang Yang. 2021. Learning to Assign: Towards Fair Task Assignment in Large-Scale Ride Hailing. In *KDD*. ACM, 3549–3557.
[8] Dingyuan Shi, Yongxin Tong, Zimu Zhou, Ke Xu, Wenzhe Tan, and Hongbo Li. 2022. Adaptive Task Planning for Large-Scale Robotized Warehouses. In *ICDE*. IEEE, 3327–3339.
[9] Dingyuan Shi, Yongxin Tong, Zimu Zhou, Ke Xu, Zheng Wang, and Jieping Ye. 2024. Graph-Constrained Diffusion for End-to-end Path Planning. In *ICLR*.
[10] Dingyuan Shi, Nan Zhou, Yongxin Tong, Zimu Zhou, Yi Xu, and Ke Xu. 2023. Collision-Aware Route Planning in Warehouses Made Efficient: A Strip-based Framework. In *ICDE*. IEEE, 869–881.
[11] Xiaocheng Tang, Fan Zhang, Zhiwei (Tony) Qin, Yansheng Wang, Dingyuan Shi, Bingchen Song, Yongxin Tong, Hongtu Zhu, and Jieping Ye. 2021. Value Function is All You Need: A Unified Learning Framework for Ride Hailing Platforms. In *KDD*. ACM, 3605–3615.
[12] Yongxin Tong, Dingyuan Shi, Yi Xu, Weifeng Lv, Zhiwei Qin, and Xiaocheng Tang. 2023. Combinatorial Optimization Meets Reinforcement Learning: Effective Taxi Order Dispatching at Large-Scale. *TKDE* 35, 10 (2023), 9812–9823.
[13] Ronald J. Williams. 1992. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Mach. Learn.* 8 (1992), 229–256.