Adaptive Stop-Skipping Scheduling Approach Using Reinforcement Learning

Perla Hajjar[1,2(✉)], Leïla Kloul[2], and Dominique Barth[2]

[1] Communauté d'Agglomération de Saint Quentin en Yvelines, Trappes, France
[2] Laboratoire DAVID/Université Versailles SQY/Université Paris Saclay, Versailles, France
{perla.hajjar,leila.kloul,dominique.barth}@uvsq.fr

Abstract. To adapt to real-time variations, stop-skipping control has been widely adopted in public transport systems. However, solving the optimal stopping pattern for buses in static bus scheduling problems is challenging due to its combinatorial nature. The complexity of this problem increases exponentially as the number of stations increases, making it difficult to find the best solution in real time. To overcome such a limitation, this paper proposes an adaptive scheduling game model approach using the stop skipping control strategy. The adaptive game model aims to minimize passenger delay by adjusting bus stops, demonstrating the effectiveness of continuous schedule adaptation against a fixed, pre-determined schedule. This game is then solved with Reinforcement Learning (RL) to optimize the bus scheduling sequence based the current system's state. We compare this approach against the Simulated Annealing metaheuristic algorithm in finding a near-optimal schedule. Our results show that the RL-based adaptive scheduling outperforms the schedule found statically and all-stop schedules, reducing waiting, ride, and total trip times.

Keywords: Stop-skipping · Adaptive bus scheduling · Reinforcement Learning · Simulated Annealing

1 Introduction

With the expansion of cities, traditional models of public transport systems struggle to find a balance between frequency of services, time spent in traveling, and operational costs, especially at peak hours when the demand and congestion are highest [1]. Traditional approaches, such as static schedules, struggle to adapt to real-time changes often leading to inefficiencies like increased passenger waiting time or bus bunching.

Solutions have been proposed in the literature such as the increase of the frequency of the bus lines and bus control strategies, such as dedicated bus lanes, traffic signal priority, and vehicle holding [2,7]. However, these solutions require additional infrastructure and operational costs. In addition, while these

A. Kocian et al. (Eds.): INTSYS 2024, LNICST 608, pp. 162–176, 2025.
https://doi.org/10.1007/978-3-031-86370-7_10

strategies are effective in reducing passengers traveling times, they do not address the problem of adapting dynamically to real-time changes in passengers' demand.

The stop-skipping control strategy has gained attraction in recent years. It allows buses to bypass certain stops based on the current demand to improve travel time efficiency and reduce operational costs [4]. However, determining optimal stop-skipping schedules in real-time is a complex combinatorial problem due to fluctuating and unpredictable passenger demand.

In our previous work [13], we adopted the stop-skipping control strategy to solve the optimal scheduling problem in a fixed demand scenario, with the objective of finding a unique global optimal schedule that minimizes passenger delay. Once this schedule was identified, it remained fixed and was applied consistently until all the passengers are served. Our static model successfully learns the optimal bus schedule. However, in real-world scenarios, passengers continuously board at bus stations, which results in changes in the demand matrix. As a result, a pre-calculated static schedule may no longer be the optimal solution as the state of the system changes.

To address this limitation, we propose an adaptive scheduling game model capable of adapting to the current system's state. In this model, we aim to investigate if the generation of a sequence of schedules as the system's state changes may result in improved minimization of passenger delay compared to maintaining a fixed, pre-calculated schedule. Given the demand matrix, our objective is to obtain an optimal bus scheduling sequence that minimizes passengers' delay. In order to enable adaptive stop-skipping decisions based on evolving passenger demand, we suggest the use of Reinforcement Learning (RL) approach that has showed its ability to find a global optimal bus schedule in the static model. The RL allows the system to learn optimal actions through trial and error with the goal of minimizing passenger delay by optimizing the bus schedules. We compare the performance of the RL approach with a traditional metaheuristic approach, specifically Simulated Annealing (SA).

The rest of this paper is structured as follows: Sect. 2 reviews the related literature on stop-skipping control strategies and their limitations. Section 3 defines the problem, providing details on the constraints and objectives of the proposed adaptive scheduling solution. Section 4 introduces the Adaptive Scheduling game model, followed by the proposed learning processes including the RL and SA approaches in Sect. 5. Section 6 provides numerical results and comparative analysis, and Sect. 7 concludes the paper with future directions.

2 Related Works

In recent years, numerous studies have focused on optimizing transit operations by implementing stop-skipping control and on-demand services to improve efficiency and service quality.

The authors in [1] proposed a real-time stop-skipping control strategy in their work that allows passengers to get off at skipped stations. They aim to minimize passenger waiting and in-vehicle travel time using an exhaustive search

method. Their approach succeeds in reducing travel time, however, it is unable to adapt to dynamic demand fluctuations. Similarly, [3] developed a bi-level genetic algorithm to find the optimal stooping patterns of buses. Although his proposed model succeeds in reducing the total travel time by 7%, it is difficult to adapt to real time changes as it was based on static demand assumptions.

Autonomous and on-demand transport services have also implemented the stop-skipping control strategy. The authors in [5] proposed a stop skipping autonomous shuttle bus service that adapts to real-time demand. They proposed a deficit function based algorithm to calculate the optimal stop skipping schedules by considering the number of available buses and their capacities. Their results shows a reduction of 1.8% in passenger travel time and 8.1% in the number of used vehicles. However, the deficit function algorithm complexity increases as the network increases which poses challengers in terms of real-time adaptations to demand.

The authors in [6] aims to optimize both the headways and the stopping patterns in a bus rapid transit system. They adopted the genetic algorithm to minimize the costs of the operator and the passengers. Their results shows that the performance of the algorithm depends on accuracy of the demand predictions which may not be reliable during peak hours. The authors in [9] also employed the genetic algorithm to design a limited-stop service that aims to determine the bus's stopping patterns under unbalanced passenger demand. The algorithm aims to minimize both the waiting and in-vehicle times of the passengers while taking into consideration the capacity constraint. Their model succeeds to adapt to demand variations but is limited when it comes to sudden demand fluctuations. The authors in [8] combined the genetic algorithm with Monte Carlo simulations to develop an optimization model to determine the stop-skipping patterns. The Monte Carlo simulation is used to model varying passenger demands and the genetic algorithm is used to adjust the bus's headways and minimize the passengers' waiting times. Although this approach improves the reliability of stop skipping and succeeds in adjusting the headways, but it may be computationally expensive.

Despite the promising results of stop-skipping control and on-demand services, real-time adaptability, scalability and reliance on accurate demand forecasting remain challenging. Therefore, this study proposes an adaptive scheduling game model with Reinforcement Learning method to determine the sequence of buses' stop skipping patterns that can minimize the total travel time for passengers.

3 Problem Definition

In our previous work [13], we introduced a static model where the passenger demand was given at time step $t = 0$. The objective was to find the optimal bus schedule that best serves the passengers with minimum time until the last passenger reaches his/her destination, denoted as delay. We proved the problem to be NP-Hard by considering a polynomial transformation from the Set Partitioning Problem. Thus we introduced a new notion, the schedule *load-delay*, denoted

as $LD(Sch)$, that calculates the time, expressed in time steps, needed by the bus having the maximum load to serve its passengers. We have also showed that a correlation exists between the real schedule delay and the schedule *load-delay*. Thus, we focus our optimization on the *load-delay*. However, this static model does not take into consideration the current system's state.

In this paper, we extend our static model by introducing an adaptive scheduling approach. Our aim is to find the optimal bus scheduling sequence while considering the current system's state, such as the number of waiting passengers at stations and the number of passengers onboard of each bus. The main objective is to find the best schedule at each time step t, denoted Sch^t, that minimizes the load-delay $LD(Sch^t)$.

Formally the load-delay of schedule Sch^t, at time step t, is defined as follows:

Definition 1. *Given a transportation system with N slots, K stations, B buses, and the origin-destination demand matrix $M[O,D]^t$, the* load-delay *of schedule Sch^t, noted as $LD(Sch^t)$, is defined as:*

$$LD(Sch^t) = \max_{b_j \in B} \left(\left\lceil \frac{MAXload_j^t}{Cap} \right\rceil \times (N + Stp_j) \right) \tag{1}$$

where

- $MAXload_j^t$ represents the maximum load of bus b_j on station s_i at time step t, for $i \in \{1...K\}$ *and* $j \in \{1...B\}$, calculated as:

$$MAXload_j^t = \max_{1 \le i \le K} \left(\sum_{\substack{x \neq y \ crossing \ s_i \\ s.t. \ D_j[s_x]=D_j[s_y]=1}} \frac{M[s_x, s_y]^t}{deg(s_x, s_y)} \right) \tag{2}$$

  It depends on the number of passengers that can board bus b_j at time step t based on its serving vector D_j that determines which stations this bus will stop at, divided by the number of buses serving each origin-destination station pair (s_x, s_y), denoted by $deg(s_x, s_y)$, such as $D_j[s_x] = D_j[s_y] = 1$, for $x, y \in \{1, ..., K\}$ and $x \neq y$.
- $\lceil \frac{MAXload_j^t}{Cap} \rceil$ represents the number of trips needed for bus b_j, having maximum load, to serve its passengers.
- $(N + Stp_j)$ represents the total time bus b_j spends traveling around the ring (N) and stopping at stations (Stp_j), expressed in time steps.

The main objective is to determine the optimal bus scheduling sequence that minimizes the load-delay while taking into consideration the current passenger demand. This requires an adaptive solution that continuously updates the bus schedules based on the current system state.

4 Adaptive Scheduling Game Model

A fixed schedule will not always be the best schedule to serve the passengers over time. To solve this problem, we define an adaptive scheduling game model

which takes into consideration the current demand to dynamically calculate a scheduling sequence that best serves the demand.

Similar to the static scheduling game model defined in [13], in this approach we suppose that the stations are the players and the choices of bus stops at each station are the strategies.

4.1 Game Components

The components of the dynamic game include:

- **The environment:**
  - A discrete ring R of N slots, labeled from 0 to N–1. They are used to represent discrete events. Each slot can be occupied only by one bus at a time and it represents the possible position of the bus in the system.
  - The set of B buses serving K stations. We denote by Pos_j^t, $1 \leq j \leq B$, the position of bus b_j at time step t. Each bus b_j serves stations according to its serving vector D_j, where $D_j[s_i] = 1$, $1 \leq i \leq K$, means that bus b_j will stop at station s_i, 0 otherwise.
  - The demand matrix $M[O, D]^t$ which represents the number of passengers at stations with their respective destinations at time step t.
- **The players:** These are the K stations in the network. Each station s_i, $i \in \{1...K\}$, is associated with a slot $Slot_i$ with at most one station per slot.
- **The actions:** The action set A_i^t of each player s_i, $i \in \{1...K\}$, consists of all possible bus stopping patterns, with $|A_i^t| = 2^B$. Each action $a_i^t \in A_i^t$ is a vector of length B, where each element is either 1 or 0. A value of 1 indicates that station s_i has chosen bus $b_j, 1 \leq j \leq B$, to stop at it, $a_i^t[j] = 1$, while 0 means bus b_j will not stop.
- **The strategy profile:** A strategy profile $\pi^t = \{a_1^t, ..., a_K^t\}$ defines a unique schedule Sch_π^t.

4.2 Game Model Parameters

Given the ring R, the number of stations K, the bus fleet B, and the demand matrix $M[O, D]^t$, each station s_i chooses an action $a_i^t \in A_i^t$, for $1 \leq i \leq K$, resulting in schedule Sch_π^t.

Load Calculation: For each schedule Sch_π^t, we define a local station load calculated at time step t for any $s_i, 1 \leq i \leq K$, denoted by SJN_i^t in Eq. 3. This parameter helps to calculate the required time steps to serve the passengers waiting at station s_i at time step t based on the given schedule Sch_π^t. This station load is related to the maximum load of a bus b_j, $1 \leq j \leq B$, stopping at s_i multiplied by the number of slots (N) and stations to stop at (Stp_j), which represents the time steps needed to serve the current demand.

$$SJN_i^t = \max_{j \ s.t \ D_j[s_i]=1} \left(\left\lceil \frac{\sum\limits_{\substack{x \neq y \ crossing \ s_i \\ s.t. \ D_j[s_x]=D_j[s_y]=1}} \frac{M[s_x,s_y]^t}{deg(s_x,s_y)}}{Cap} \right\rceil \times (N + Stp_j) \right) \quad (3)$$

An upper bound of the *load-delay* for any schedule, denoted as WST, is defined in Eq. 4. This upper bound represents the worst-case scenario and is calculated under the assumption that only one bus serves all the demand. By penalizing schedules that rely heavily on one bus to serve the demand, the model encourages stations to learn and adopt a more balanced and efficient bus stopping patterns by distributing the demand more evenly across available buses.

$$WST = \left\lceil \frac{\max\limits_{\substack{s_i \\ 1 \leq i \leq K}} \left(\frac{\sum\limits_{x \neq y \ \ crossing \ \ s_i} M[s_x, s_y]^t}{deg(s_x, s_y)} \right)}{Cap} \right\rceil \times (N + K + 1) \tag{4}$$

Cost Minimization: At each time step t, station s_i selects an action $a_i^t \in A_i^t$ that minimizes its cost $C_i^t(a_i^t)$. This cost is defined as:

$$C_i^t(a_i^t) = \gamma \times TOT - \alpha \times Loc_i \tag{5}$$

where $\gamma \geq 1$ and $\alpha \geq 0$ are two tuning parameters of the game indicating the weight of the global (TOT) and the local (Loc_i) loads, respectively. These loads are:

- $TOT = WST$ if Sch_π^t is not feasible, else $TOT = LD(Sch_\pi^t)$
- $Loc_i =$
$$\begin{cases} 0 & \text{if } \exists\ i', 1 \leq i' \leq K \text{ such that } M[s_i, s_i']^t > 0 \text{ and } deg(s_i, s_i') = 0 \\ SJN_i^t & \text{if } \exists\ i', 1 \leq i' \leq K \text{ such that } M[s_i, s_i']^t > 0 \text{ and } deg(s_i, s_i') > 0 \\ TOT & \text{if } \forall\ i', 1 \leq i' \leq K, M[s_i, s_i']^t = 0 \text{ and } M[s_i', s_i]^t = 0 \end{cases}$$

We will use this game model as a distributed algorithmic approach to determine a global minimum schedule, at every time step t, that optimally serves passengers by minimizing the schedule *load-delay*, $LD(Sch^t)$.

5 Learning Process

To avoid unnecessary learning while ensuring adaptability to changes in the system, the learning process is initiated whenever a bus is in front of a station whether it is said to stop at it or not. This strategy ensures that learning occurs sufficiently to address system changes without being overly exhaustive at every time step. Once a bus is announced to stop at a station, this decision remains fixed until the bus reaches that station the next turn. This prevents frequent changes and ensures that on board passengers reaches their destination.

More specifically, consider that at time step t, bus b_j is in front of station s_i for $1 \leq j \leq B, 1 \leq i \leq K$. We define a slot to be an (i, j)-*milestone* if a bus b_j is in front of station s_i at this slot whether it is stopping at it or not. At every (i, j)-*milestone*, the learning process then runs for T learning steps (or until

convergence), taking into consideration the current demand matrix $M[O, D]^t$ and the set of available actions A_i^t for each player s_i.

For the learning process, we propose and evaluate two different strategies for defining the action set of each station:

- The action subset strategy: this means that the stations are only allowed to choose from the subset of actions which allows it to modify its action concerning buses that have not yet been scheduled to stop at it. This strategy implements the learned actions in real-time while limiting the action set to those that are feasible given the current state of the system.
- The full action set strategy: this strategy allows all the stations to choose actions regarding all the buses. However, the action is only applied by the station that has a bus in front of it to ensure that on board passengers reaches their destinations with a minimum delay. This means that all stations learn simultaneously, but their learned actions are applied with a delay.

These strategies define how stations make decisions regarding which bus will stop and how are they applied in real time.

In the following, we present the two learning processes that are used to solve our adaptive scheduling game model: Linear Reward Inaction (LRI) and Simulated Annealing (SA).

5.1 Linear Reward Inaction

Distributed LRI [10] is a reinforcement learning method where stations (players) aim to optimize a common cost through a reward system. Each player s_i, for $1 \leq i \leq K$, has a strategy vector q_i, which is a stochastic vector of potential actions such as $|q_i^t| = |A_i^t|$. Each action $a_i^t \in A_i^t$ for player s_i has an initial probability $q_i^t(a_i^t)$ of being selected. At each time step t, each player s_i randomly selects an action, noted a_i^t, from its strategy vector. The cost of choosing action $C_i^t(a_i^t)$ by station s_i at time step t is calculated based on Eq. 5. After calculating the $C_i^t(a_i^t)$, the next step is to calculate the utility of player s_i. It helps each player to evaluate the effectiveness of choosing action a_i^t and guides it in learning the optimal strategy over time. Hence, we define the utility function for player s_i at time step t as:

$$U_i^t = \frac{C_i^{max}(a_i^t) - C_i^t(a_i^t)}{C_i^{max}(a_i^t) - C_i^{min}(a_i^t)} \tag{6}$$

with $C_i^{max}(a_i^t)$ (resp. $C_i^{min}(a_i^t)$) being the maximum (resp. minimum) cost impacted to s_i when choosing action a_i^t at time step t. The design of the utility function is critical for the player's learning of optimal stopping pattern. It should be both broad enough to capture the impact of the chosen actions, but specific enough to not cause noise during learning.

After each learning step, the strategy vector is updated using the LRI update rule:

$$\begin{cases} q_i^{t+1}(a) = q_i^t(a) + \eta \times U_i^t \times (1 - q_i^t(a)) & \text{If } a = a_i^t \\ q_i^{t+1}(a') = q_i^t(a') - \left(\dfrac{q_i^t(a')}{1 - q_i^{t+1}(a')} \times \eta \times U_i^t \times (1 - q_i^t(a)) \right) & \forall\, a' \neq a_i^t \;\&\; a = a_i^t \end{cases} \tag{7}$$

where:

- η is the learning parameter such that $0 < \eta < 1$.
- $q_i^t(a)$ is the probability that player s_i selects action a at iteration t.
- U_i^t is the utility function.

For the LRI learning process, the definitions of the full action set approach and the action subset strategies are detailed below.

The Full Action Set Strategy: Consider that at time step t, bus b_j reaches station s_i. All stations are allowed to choose simultaneously an action from all the possible available actions. To maintain stability by preserving the previous learned probabilities, the strategy vector q_i^t at the beginning of each learning process for each station s_i is calculated as follows:

$$q_i^t(a_i^t) = q_i^{t-1}(a_i^{t-1}) \tag{8}$$

At the beginning of each learning process, we take into consideration the current demand matrix $M[O, D]^t$ and the schedule found at time step $t-1$, denoted as Sch^{t-1}. For the bus b_j that is in front of station s_i, the action concerning whether this bus will stop at this station or not is initially set to zero. During the learning process, at each learning step, all stations are allowed to choose actions based on their respective stochastic vectors q_i^t, and after calculating the *load-delay*, these values are updated as in Eq. 7.

At the end of the learning process, only the action a_i^t learned by station s_i having the highest probability is considered. A logical OR operation is performed between the previous action a_i^{t-1} and the newly learned a_i^t, yielding the final action for station s_i. For all other stations, their previously learned actions remain unchanged, indicating that the learned actions for these stations are not directly applied until their respective turns.

The Action Subset Strategy: Consider that at time step t, bus b_j reaches station s_i, we set the action considering if b_j will stop at s_i to 0. Then, the LRI algorithm runs for T learning steps (an input parameter), taking into consideration the current demand matrix $M[O, D]^t$, the schedule Sch^{t-1} found at time step $t-1$, and the set of available actions $\overline{A_i^t} \subseteq A_i^t$ for each player such that:

$$\overline{A_i^t} = \{a_i^t \in A_i^t \mid a_i^t[j] = 0 \;\forall j \in \{1, ..., B\}\}$$

This means that station s_i is only allowed to choose from the subset of actions $\overline{A_i}^t$ which only allows it to modify its action concerning buses that have not yet been scheduled to stop at it based on the previously found schedule at time step $t-1$, Sch^{t-1}. This ensures that once a bus is designated to stop at a station, this decision cannot be reversed until the bus reaches the station.

Since the station is only allowed to choose actions from $\overline{A_i^t}$, then $\overline{q_i^t}$ is a normalized sub-vector of q_i^t where $|\overline{q_i^t}| = |\overline{A_i^t}|$ and the values correspond to respective probability of choosing these actions in the action subset. After all the stations choose simultaneously the appropriate action from $\overline{A_i^t}$, the expected schedule delay is calculated based on previously defined schedule load-delay in Eq. 1. The schedule load-delay in this case determines how many time steps are needed to serve all the passengers if each station sticks to its chosen action until the system is empty. At each learning step, the station's stochastic vectors are updated based on Eq. 7. At the end of each learning process, each station picks the action with the highest probability based on q_i^t, for $1 \leq i \leq K$. All selected actions determines the schedule Sch^t found by LRI at time step t.

5.2 Simulated Annealing

A centralized Simulated Annealing (SA) algorithm is a probabilistic method proposed in [11]. It is used to estimate the global minimum for a function with many variables. This algorithm can produce a good local though not necessarily global optimal solution within a reasonable computing time. Essentially speaking, simulated annealing can be seen as a "randomized variation" of the local search method [12]. The SA algorithm begins with a high temperature, which gradually reduces with time. The temperature is used to control the probability of accepting worse solutions as the algorithm explores the solution space. Higher temperatures permit more exploration, while lower temperatures focus on exploitation.

Consider that at time step t, bus b_j reaches station s_i, for $1 \leq i \leq K$ and $1 \leq j \leq B$. The SA algorithm runs until the temperature reaches 0 or until the number of generated neighbor schedules is equal to the T learning steps of the LRI algorithm. At the beginning of each learning process, the SA runs taking into consideration the demand matrix available at time step t, $M[O,D]^t$, and the schedule Sch^{t-1} found at time step $t-1$, the previous learning step. At each step of the algorithm, we generate a random neighbor schedule by altering the choice of a bus b_j stopping at s_i. This neighbor schedule generation can be based on either a full action set or an action subset strategy. The current schedule Sch^t found at the (i,j)-milestone, is denoted by V_{Sch^t}, and is the concatenation of the buses binary vectors D_j.

The Full Action Set Strategy: We randomly select a value of i, for $1 \leq i \leq K$, and a value of j, for $1 \leq j \leq B$. If the previous schedule value $V_{Sch^{t-1}}[i][j]$ is 1, it is changed to 0, and if it is 0, it is changed to 1. Then, the schedule *load-delay* is recalculated, and the acceptance criteria are verified. If the new schedule results

in a lower load-delay, it is accepted. However, if the new schedule produces a higher load-delay, it can still be accepted with a probability based on the current temperature. As the temperature decreases, the likelihood of accepting a worse schedule decreases, promoting convergence towards the optimal solution. At the end of the learning process, the final schedule at time step t is determined by combining the previous and current schedule decisions using a logical OR operation, denoted as $V_{\mathrm{Sch}^t} = V_{\mathrm{Sch}^{t-1}}[i][j] \vee V_{\mathrm{Sch}^t}[i][j]$. For all other stations, if the previous decision is that bus b_j is to stop at it, then this value can't be changed until b_j reaches s_i. This ensures that only the action related to the bus currently in front of the station is allowed to change its decision regarding the bus stopping at it.

The Action Subset Strategy: We begin by also randomly selecting values for i and j. If the previous schedule value $V_{\mathrm{sch}^{t-1}}[i][j]$ is 0, it is set to 1; otherwise, no changes are made. Then, the schedule *load-delay* is computed, and the resulting neighbor schedule is either accepted or rejected according to the same acceptance criteria defined in the full action set approach. Once the SA learning process is complete, a logical OR operation is performed between $V_{\mathrm{sch}^{t-1}}$ and the updated V_{sch^t}, resulting in the final V_{sch^t}. In this strategy, all the actions that have been learned by the stations are applied collectively.

6 Numerical Results

In this section, we present the results of the proposed adaptive scheduling model.

6.1 The Input Data

To test our proposed adaptive scheduling model, we use real varying data of the one-direction public transport line 414 of the urban community of Saint-Quentin-en-Yvelines, a Paris suburban area. The road is considered as a ring R consisting of $N = 47$ slots numbered from 0 to 46 as shown in Fig. 1. For this line, there are $B = 3$ buses serving $K = 8$ stations.

Passenger Demand: The data set represents a one-direction bus line with the passenger demand defined at the beginning of the simulation. In this scenario, no new passengers enter the system; so the demand matrix decreases based on the passengers boarding the stopping buses. This will allow us to investigate the ability of the learning processes to find the optimal bus scheduling sequence that minimizes the passengers' waiting and traveling time. This scenario considers that there are no new passengers arriving at the stations and the demand $(M[O, D]^t)$ is defined at time step $t = 0$. The demand matrix $M[O, D]^t$ representing the varying demand for every origin destination station, at time step $t = 0$, is provided in Table 1 .

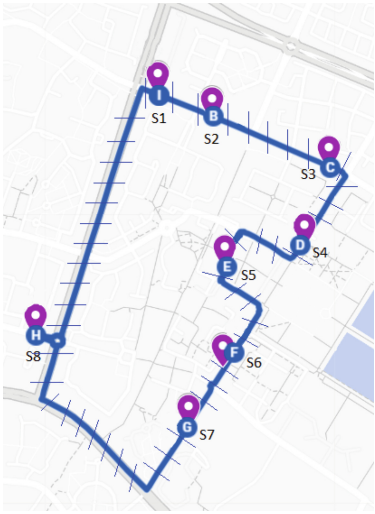

Fig. 1. Bus line 414 in the urban community of Saint-Quentin-en-Yvelines, France

Table 1. Origin-destination demand matrix $M[O, D]^{t=0}$ of line 414

O/D	1	2	3	4	5	6	7	8
1	0	77	34	6	14	3	3	2
2	0	0	55	43	89	17	31	6
3	0	0	0	22	53	20	57	7
4	0	0	0	0	5	4	8	2
5	0	0	0	0	0	6	20	8
6	0	0	0	0	0	0	43	9
7	0	0	0	0	0	0	0	40
8	0	0	0	0	0	0	0	0

LRI Algorithm Parameters: For the LRI algorithm, simulations are executed for $T = 2.5$ million iterations using parameter values that were identified as optimal in the static environment in previous work [13]. For the tuning parameters γ and α in Eq. 5, we set $\gamma = 2$ and $\alpha = 1$ as these values have shown a balance between global load-delay and local station loads. For the learning rate η, the chosen value is 0.00625. Additionally, for θ, the trade-off parameter, we test the value of $\theta = 0$ indicating that we consider the stochastic vectors at the previous learning step.

SA Algorithm Parameters: For the SA algorithm, the initial temperature was set to a high value of 150000, which allows the algorithm to explore a wide range of possible solutions in the early stages as shown in the results of the static

system [13]. A cooling rate of 0.98 which ensures slow cooling process was used to gradually decrease the temperature as the algorithm progresses. The algorithm stops when the number of generated neighbor schedules is equal to the number of learning steps (T) used in the Linear Reward Inaction (LRI) algorithm.

6.2 Performance Evaluation of the Adaptive Scheduling Model

We run several simulations of the defined learning processes (LRI, SA) using the SUMO traffic simulator [14] and the TraCI library, a Python library that allows for real-time communication with the SUMO simulator [15]. This setup enables us to model real-time passenger arrival rates and dynamically control bus movements during the simulation.

We compare the performance of LRI and SA with a baseline scenario where buses stop at all stations. This baseline is appropriate since our demand matrix includes passengers traveling from every origin to every destination, thus representing a scenario with maximum service coverage. Additionally, we compare these results with the results of our previous static scheduling model [13], where the optimal bus schedule is well known.

The performance of the LRI and SA algorithms are evaluated based on three key metrics: waiting time, ride time and total trip time of passengers measured in minutes. Tables 2, 3, and 4 provide a summary of the results obtained.

Table 2. Statistics Summary for Waiting Times (in minutes)

Statistics	Static scheduling		Adaptive scheduling				All stops
	SA	LRI	SA Full	SA Sub	LRI Full	LRI Sub	All stops
Min	1.13	1.65	1.73	1.40	1.17	1.32	1.54
Max	13.90	13.05	20.87	14.93	14.75	14.83	14.57
Mean	4.53	5.84	5.62	4.65	4.31	4.46	4.60
Std Dev	3.12	3.00	5.32	3.46	3.29	3.35	3.80

Table 3. Statistics Summary for Ride Times (in minutes)

Statistics	Static scheduling		Adaptive scheduling				All stops
	SA	LRI	SA Full	SA Sub	LRI Full	LRI Sub	All stops
Min	0.25	0.25	0.25	0.25	0.25	0.25	0.25
Max	2.46	2.23	2.67	2.61	2.37	2.45	2.50
Mean	0.77	0.65	0.84	0.75	0.70	0.72	0.80
Std Dev	0.48	0.41	1.55	0.47	0.39	0.42	0.50

Table 4. Summary Statistics for Total Trip Duration (in minutes)

Statistics	Static scheduling		Adaptive scheduling				All stops
	SA	LRI	SA Full	SA Sub	LRI Full	LRI Sub	All stops
Min	1.90	1.90	1.98	1.75	1.42	1.56	1.80
Max	16.36	15.28	23.54	17.12	17.12	17.28	17.00
Mean	5.24	6.49	6.46	5.47	4.96	5.10	5.40
Std Dev	3.40	3.27	5.51	3.81	3.51	3.60	4.23

Results of LRI: Results in the tables show that the LRI full action set strategy outperforms the action subset strategy across all key metrics. These results show also that this strategy can adapt to changes in the system, whereas the action subset strategy converges to a sub-optimal schedule where buses stop at every station.

For most of the metrics, the LRI full action set strategy outperforms the LRI in the static scheduling scenario, except when considering the maximum values, we observe that the performance varies. Indeed, the LRI full adaptive strategy shows an increase of approximately 13.0% in maximum waiting times, an increase of 6.3% in maximum ride times, and an increase of 12.0% in maximum total trip duration compared to the LRI in static scheduling. These differences are due to the difference in how actions are applied in each approach. In static scheduling, all actions are directly applied without constraints, while in adaptive scheduling, even though all actions are available for all players, they are not directly applied which can affect the efficiency.

When comparing the LRI full action set approach with the all stops approach, we can see that this one performs better in minimizing both the maximum waiting time and the maximum total trip time. However, it leads to a slight increase of 5.3% in maximum ride times. This is due to the buses stopping at every stop in the all stops approach which increases the total travel times for on board passengers.

Results of SA. The SA action subset strategy performs better compared to the SA full action strategy in all key metrics. In particular, the SA action subset strategy achieves a 28.5% improvement in maximum waiting times, a 2.2% reduction in maximum ride times, and a 27.3% decrease in maximum total trip duration. This is due to the fact that the SA full action strategy gets stuck on a local minimum with a high *load-delay* value which prevents it from converging to a good local minimum scheduling sequence.

On the other hand, the SA static scheduling strategy succeeds in minimizing all the metrics compared to the SA action subset strategy in adaptive scheduling. This is due to the fact that the SA action subset strategy has a limited action set available after each learning step, which causes it to always converge to a schedule where buses stop at every station.

In addition, when comparing the SA action subset strategy to the all stops approach, we observe that the latter performs better in reducing the maximum of all metrics.

6.3 Discussion Summary

The results show that the SA algorithm performance can vary and its ability to adapt to changes is not always consistent. They also show that the LRI-based adaptive scheduling outperforms the SA-based approaches in both adaptability and minimizing passenger delay.

The LRI full action set strategy shows advantages over the static and the all-stops scenarios, by reducing waiting, ride, and total trip times with lower variability, making it a more reliable choice for real-time bus scheduling. However, while the SA action subset strategy offers some improvements over the full action set, it lacks the consistency and efficiency demonstrated by LRI in adapting to the system's state. Thus, the LRI full action strategy emerges as the more effective algorithm for adaptive bus scheduling in dynamic public transport systems.

In summary, the SA algorithm struggles to find optimal solutions and encounters difficulties during the last stages of the simulation, particularly when the demand begins to decline and certain stations either have no waiting passengers or no passengers needing to travel to them. However, the LRI algorithm with the full action set strategy shows ability to escape local minimums and produces a scheduling sequence that reduces passengers' times while avoiding unnecessary stops.

7 Conclusion

We propose in this research work an adaptive scheduling model game to solve the sequential bus stop-skipping decision problem in a static demand scenario. The goal is to minimize passenger travel time by adjusting bus stops decisions based on current demand. We compared the performance of two proposed learning processes: the Linear Reward Inaction (LRI) and the Simulated Annealing (SA) in adapting to the changes in the system. We also propose two strategies to determine the action sets of the players for each learning process in our game model. The first is the full action set strategy, which allows all players to simultaneously choose an action from all possible actions available in the action set; however, the learned action is applied with a delay. The second is the action subset strategy, which limits the action set available to the players but applies the learned actions in real time.

Results show that the LRI full action set strategy outperforms the SA algorithm approaches by succeeding in minimizing the passengers delays across all key metrics. The LRI full action set strategy also shows better adaptability and ability to find a scheduling sequence that avoids unnecessary stops. In contrast, the SA algorithm frequently converges to a sub optimal schedule.

In future work, we will focus on extending this adaptive game model to handle real time varying passenger demands and more complex networks.

References

1. Sun, A., Hickman, M.: The real-time stop-skipping problem. J. Intell. Transp. Syst. **9**, 91–109 (2005). https://doi.org/10.1080/15472450590934642
2. Liu, Z., Yan, Y., Qu, X., Zhang, Y.: Bus stop-skipping scheme with random travel time. J. Transp. Res. Part C: Emerg. Technol. **35**, 46–56 (2013)
3. Niu, H.: A matheuristic for optimizing skip-stop operation strategies in rail transit lines. Int. J. Transp. Dev. Integr. **3**(4), 306–316 (2011)
4. Black, A.: A method for determining the optimal division of express and local rail transit service. Chic. Area Transp. Stud. **347**, 120–106 (1962)
5. Cao, Z., Ceder, A.: Autonomous shuttle bus service timetabling and vehicle scheduling using skip-stop tactic. J. Transp. Res. Part C: Emerg. Technol. Elsevier (2019)
6. Chen, X., Hellinga, B., Chang, C., Fu, L.: Optimization of headways with stop-skipping control: a case study of bus rapid transit system. J. Adv. Transp. **49**, 385–401. Wiley (2015)
7. Gkiotsalitis, K., Cats, O.: At-stop control measures in public transport: literature review and research agenda. Transp. Res. Part E: Logist. Transp. Rev. **145** (2021)
8. Mou, Z., Zhang, H., Liang, S.: Reliability optimization model of stop-skipping bus operation with capacity constraints. J. Adv. Transp. (2020)
9. Zhang, H., Zhao, S., Liu, H., Liang, S.: Design of limited-stop service based on the degree of unbalance of passenger demand. PLoS ONE **13**(3) (2018)
10. Sastry, P.S., Phansalkar, V.V., Thathachar, M.A.L.: Decentralized learning of Nash equilibria in multi-person stochastic games with incomplete information. IEEE Trans. Syst. Man Cybern. **24**(5), 777–769 (1994)
11. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. J. Sci. **220**, 671–680 (1983)
12. FanRandy, W., Machemehl B.: Using a simulated annealing algorithm to solve the transit route network design problem. J. Transp. Eng. **132**(2) (2006)
13. Hajjar, P., Kloul, L., Barth, D.: Optimal bus scheduling using a distributed game model approach. In: IEEE 26th International Conference on Intelligent Transportation Systems (ITSC), pp. 4571–4576 (2023)
14. Behrisch, M., Bieker-Walz, L., Erdmann, J., Krajzewicz, D.: SUMO - simulation of urban mobility: an overview. In: Proceedings of SIMUL (2011)
15. Wegener, A., Piorkowski, M., Raya, M., Hellbrück, H., Fischer, S., Hubaux, J.P.: TraCI: an interface for coupling road traffic and network simulators. In: Proceedings of the 11th Communications and Networking Simulation Symposium (2008)