UNIVERSITÄT BONN

Conflict Resolution in Train Timetabling Using Alternative Graphs and Deep Q-Networks

Master Thesis

Shruthi Devaraj

Matriculation Number

3288918

This work was submitted to the

Institute of Computer Science IV

University of Bonn, Germany

Adviser(s):

Dr. Paulo Henrique Rettore Lopes
Dr. Clayson Celes
Dr. Bruno Santos
Mr. Philipp Zißner

Examiners:

Prof. Dr. Michael Meier Dr. Paulo Henrique Lopes Rettore

Registration date: 18-07-2025
Submission date: 19-01-2026

In collaboration with the Fraunhofer Institute for Communication, Information Processing and Ergonomics (FKIE), Bonn, Germany

Rheinische
Friedrich-Wilhelms-
Universität Bonn

Prüfungsausschuss Informatik

universität **bonn** · Institut für Informatik · 53012 Bonn

Vorsitzende des Prüfungsausschusses Prof. Dr. Anne Driemel

Stellvertretender Vorsitzender Prof. Dr. Thomas Kesselheim

Prüfungsamt:
Judith König
Tel.: 0228/73-4418
pa@informatik.uni-bonn.de
Postanschrift
Friedrich-Hirzebruch-Allee 5
Besucheranschrift:
Friedrich-Hirzebruch-Allee 8
53115 Bonn

www.informatik.uni-bonn.de

Erklärung über das selbständige Verfassen einer Abschlussarbeit
Declaration of Authorship

Titel der Arbeit/Title:

Conflict Resolution in Train Timetabling Using Alternative Graphs and Deep Q-Networks

Hiermit versichere ich
I hereby certify Devaraj, Shruthi
Name/name Vorname / first name

dass ich die oben genannte Arbeit – bei einer Gruppenarbeit meinen entsprechend gekennzeichneten Anteil der Arbeit – selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

that I have written the above-mentioned work – in the case of group work, my correspondingly marked part of the work – independently and that I have not used any sources or resources other than those indicated and that I have identified all quotations.

Bonn, 15/01/2026

D. Shruthi

Unterschrift (im Original einzureichen im Prüfungsamt Informatik)

Abstract

Railway systems are required to deliver reliable and frequent service on infrastructure operating close to capacity. In dense corridors, even small disturbances—such as delayed arrivals, extended dwell times, or minor faults—can propagate through the timetable and cause widespread disruption. Traditional rescheduling approaches based on Mixed-Integer Linear Programming (MILP) or handcrafted heuristics can produce high-quality solutions, but they often struggle to react within strict time limits, scale to many simultaneous conflicts, or adapt efficiently to changing traffic conditions.

These limitations motivate learning-based approaches that can reuse prior experience and provide rapid decision support. Deep Reinforcement Learning (DRL) offers the ability to learn dispatching policies that map operational states directly to conflict-resolution actions, enabling near real-time response. However, for such methods to be operationally viable, they must accurately capture microscopic railway constraints, restrict decisions to feasible actions, and optimize delay measures relevant to practice.

This thesis investigates the applicability of DRL for microscopic train timetable conflict resolution using the Alternative Graph (AG) representation, which is widely used in railway planning to model event timing, precedence relations, and train–block conflicts. A data-to-environment pipeline is developed that transforms operational exports from the Leistungsuntersuchung Knoten und Strecke (LUKS) system into AG instances and a Markov Decision Process. Within this environment, a Deep Q-Network (DQN) is trained to resolve conflicts by selecting local precedence decisions that minimize network-wide delay.

The proposed approach is evaluated on realistic railway scenarios and benchmarked against OptDis, an industry-standard MILP-based solver, under matched disturbance scenarios and time constraints. The results show that the trained DRL agent consistently produces feasible rescheduling decisions with negligible inference latency on fixed infrastructure topologies. Within a given corridor, the learned policy generalizes across disturbance magnitudes, train orderings, and traffic densities, but exhibits limited generalization to previously unseen infrastructure layouts and higher cumulative delay at very high traffic densities.

Overall, the findings demonstrate that DRL can complement, rather than replace, optimization-based rescheduling methods. When applied within a fixed operational context, a trained policy can provide fast, feasible decisions that support dispatchers under time pressure and potentially accelerate or guide exact optimization, contributing to more responsive and scalable railway timetable rescheduling.

Acknowledgement

To my supervisors Dr. Paulo Henrique Lopes Rettore, Dr. Clayson Celes, Dr. Bruno P. Santos, Mr. Philipp Zißner from University of Bonn and Dipl. Inform. Carsten Gester and Dr Frédéric Weymann from Quattron GmbH. Thank you for your guidance, support and unwavering confidence. I appreciate your mentorship, insightful feedback and time throughout my masters and thesis. To my family and friends, thank you for your support.

Contents

Nomenclature

AG	Alternative Graph
AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
GNN	Graph Neural Network
LP	Linear Programming
LUKS	Leistungsuntersuchung Knoten und Strecken
MDP	Markov Decision Process
MILP	Mixed-Integer Linear Programming

1

Introduction

Railway systems play a central role in modern societies by enabling large-scale, energy-efficient mobility for passengers and freight. They support economic activity, regional connectivity, and sustainable transportation policies by offering high-capacity transport with comparatively low environmental impact. As demand for rail transport continues to grow—driven by urbanization, climate goals, and shifts away from road and air traffic—railway infrastructure is increasingly operated close to its theoretical capacity limits.

Operating near capacity makes railway systems particularly sensitive to disturbances. Even small deviations from the planned timetable, such as delayed arrivals, extended dwell times, rolling stock irregularities, or minor infrastructure faults, can quickly propagate through the network. Because trains share scarce resources such as track sections, switches, and platforms under strict safety and signaling constraints, a local disruption may trigger chains of conflicts that affect many subsequent train movements. Managing these disruptions is therefore a critical operational task with direct consequences for service reliability, passenger satisfaction, and network efficiency.

A central challenge in this context is microscopic train timetable conflict resolution—the problem of deciding, at the level of individual block sections and switches, which train should be granted precedence when competing movements arise. These decisions must be made under severe time pressure, while respecting detailed operational constraints and minimizing delay propagation. In practice, dispatchers are required to continuously adapt the timetable in response to evolving conditions, often within seconds.

Existing approaches and their limitations

State-of-practice methods for railway timetable rescheduling are predominantly based on Mixed-Integer Linear Programming (MILP) formulations or carefully engineered heuristics. MILP-based approaches can model detailed operational constraints and

often produce high-quality or near-optimal solutions. However, their computational complexity grows rapidly with traffic density, network size, and the number of simultaneous conflicts. As a result, they may struggle to provide timely solutions in highly disturbed or large-scale scenarios, or when repeated re-optimization is required.

Heuristic and rule-based approaches offer faster response times, but typically rely on manually designed strategies that may be sensitive to scenario structure and difficult to adapt to new operating conditions. Both classes of methods face challenges when rapid, repeated decisions are required under tight real-time constraints.

These limitations have motivated growing interest in learning-based approaches, which aim to reuse experience across scenarios and produce decisions with very low inference latency. In particular, reinforcement learning (RL) has emerged as a promising paradigm for sequential decision-making problems under uncertainty. By learning policies that map operational states directly to control actions, RL methods offer the potential to support dispatchers with fast, adaptive decision suggestions once trained.

Alternative Graphs as a modeling foundation

For learning-based approaches to be operationally viable in railway applications, they must be grounded in a representation that accurately captures microscopic railway constraints and remains interpretable for practitioners. A widely used model in railway planning and dispatching is the Alternative Graph (AG).

At a high level, an Alternative Graph represents a timetable as a directed graph in which:

- **Nodes** correspond to train–resource events (e.g., a train occupying a block).
- **Fixed arcs** encode mandatory precedence relations within a train's route, such as running and dwell time constraints.
- **Alternative arc pairs** represent conflicts on shared infrastructure, capturing the two possible precedence orders between competing trains.

Resolving a conflict corresponds to selecting one arc from an alternative pair, thereby fixing the order in which trains use a shared resource. This structure makes conflicts explicit, supports incremental decision-making, and allows feasibility checking via well-defined graph properties. Because of these characteristics, the AG has been successfully used as a backbone for both exact optimization and heuristic dispatching methods and provides a natural interface for sequential decision-making.

Industrial context and motivation

This thesis is conducted in collaboration with Quattron GmbH, a company operating in the railway consulting and software segment. Quattron develops tools for railway infrastructure analysis, capacity assessment, and timetable planning, and supports infrastructure managers and railway undertakings in operational and strategic decision-making.

In this context, Quattron's proprietary software LUKS (Leistungsuntersuchung Knoten und Strecken) is used in practice to analyze network performance and resolve timetable conflicts under disturbances. LUKS includes an optimization-based rescheduling component (OptDis) that formulates conflict resolution as a MILP problem. While this approach is effective and delivers high-quality solutions, its computational effort can increase significantly as the number of conflicts grows, making large-scale or real-time rescheduling challenging in heavily disrupted scenarios.

Importantly, LUKS also provides access to realistic, industrial-grade planning data, including infrastructure layouts, block occupation times, and disturbed timetables. This makes it possible to evaluate alternative approaches under conditions that closely reflect real-world railway operations.

Thesis focus and approach

This thesis addresses the problem of microscopic train timetable conflict resolution under disturbances by investigating whether reinforcement learning can complement existing optimization-based approaches. Rather than recomputing complete schedules from scratch, the problem is formulated as a sequence of local precedence decisions on shared infrastructure, whose cumulative effect determines delay propagation across the network.

To this end, the thesis proposes a learning-based rescheduling framework that combines:

- The *Alternative Graph* as an interpretable and constraint-faithful representation of railway operations;
- *Reinforcement learning*, enabling the automatic acquisition of conflict-resolution policies from experience.

Once trained, such a policy can generate feasible conflict-resolution actions with very low latency, making it suitable for time-critical dispatching scenarios. The framework operates directly on data exported from the proprietary LUKS tool, which are converted into consistent graph-based representations suitable for learning-based decision-making.

Overall, this work does not aim to replace exact optimization methods. Instead, it investigates whether reinforcement learning can provide fast, feasible decisions that support dispatchers under time pressure and integrate naturally into hybrid workflows combining learning-based policies and optimization-based refinement.

1.1 Problem Statement

Railway networks are complex, tightly coupled systems in which even minor disturbances can cascade into significant delays due to interdependencies among trains, scarce infrastructure resources, and strict safety and headway constraints [4]. In dense operating conditions, such disturbances give rise to precedence conflicts on

shared blocks and junctions that must be resolved within tight operational time limits.

At Quattron GmbH, train scheduling conflicts are currently resolved using the proprietary LUKS software, which employs an optimization-based rescheduling approach. While this method can deliver high-quality solutions, its computational cost increases with the number of simultaneous conflicts and traffic density, limiting its effectiveness in real-time or large-scale rescheduling scenarios where rapid, repeated decisions are required [2].

Recent advances in artificial intelligence (AI), and in particular reinforcement learning, offer an alternative paradigm for addressing such problems. Learning-based methods can exploit experience from previous scenarios, adapt to evolving conditions, and produce decisions with predictable and low inference times [13]. However, their applicability in railway operations depends critically on the ability to preserve microscopic feasibility constraints and to operate on realistic industrial data.

The objective of this thesis is therefore to design and evaluate a reinforcement-learning-based framework for microscopic train timetable conflict resolution that:

- operates on an Alternative Graph representation derived from real operational data;
- produces feasible conflict-resolution decisions;
- limits delay propagation under disturbances; and
- can be fairly compared to an industrial optimization-based solver under matched conditions.

Key Research Questions:

1. How can Alternative Graph–based representations be leveraged to support efficient, learning-based conflict resolution in railway timetabling while preserving the microscopic constraints required in practice?
2. Can reinforcement learning methods learn effective conflict-resolution strategies that generalize across different disturbance scenarios on the same corridor and closely related topologies?
3. How does the performance of the proposed learning-based method compare to the existing LUKS optimization-based solver in practical railway network instances, with respect to total and maximum delay, time to solution, and scalability as conflicts and traffic density increase?

1.2 Proposed Approach and Contributions

This thesis proposes a learning-based framework for microscopic train timetable conflict resolution under disturbances. The approach combines an established graph-based representation used in railway planning with reinforcement learning to support

fast and feasible precedence decisions on shared infrastructure. Rather than recomputing complete schedules from scratch, the framework addresses rescheduling as a sequence of local conflict-resolution decisions whose cumulative effect determines delay propagation across the network.

The proposed approach is designed to operate on realistic operational data and to produce decisions with low inference latency, making it suitable for time-critical dispatching scenarios. Its effectiveness is assessed through systematic benchmarking against an industrial optimization-based solver under matched disturbance scenarios and comparable time budgets, focusing on solution quality, computational behavior, and scalability.

The main contributions of this thesis are:

- A practical data-to-decision pipeline that transforms industrial railway planning data into a form suitable for learning-based timetable rescheduling.
- A conflict-resolution formulation that integrates an interpretable graph-based railway model with reinforcement learning while preserving operational feasibility.
- An empirical evaluation of the proposed approach on realistic railway scenarios, including a comparative benchmark against an industrial optimization-based solver.
- An analysis of scalability, generalization across disturbance scenarios, and the potential of learning-based methods to complement optimization-based rescheduling in operational settings.

1.3 Hypothesis

This section presents the hypotheses guiding the empirical evaluation of the proposed framework, focusing on solution quality, generalization, and computational performance under realistic operational conditions.

H1. A reinforcement-learning-based agent operating on an Alternative Graph representation can learn conflict-resolution policies that reduce total and maximum delay compared to baseline heuristic approaches in disturbed scenarios. *Test:* Compare the learned policy against rule-based or greedy heuristics on matched disturbance scenarios, reporting total delay, maximum delay, and solution feasibility under identical time budgets.

H2. The learned conflict-resolution policy can generalize to unseen disturbance scenarios on the same network and to new traffic patterns on closely related infrastructure topologies. *Test:* Evaluate the policy on held-out disturbance scenarios and topology-preserving network variations, and assess performance degradation across increasing traffic densities using delay-based metrics.

H3. Under practical time constraints and disturbance scales, the proposed learning-based approach can achieve solution quality comparable to an industrial optimization-based solver while exhibiting more predictable or lower computation times. *Test:*

Conduct head-to-head comparisons with the OptDis solver under matched time budgets, measuring solution quality, time to first feasible solution, and scalability as the number of conflicts increases.

1.4 Goals

This section summarizes the main goals of the thesis, focusing on practical applicability, rigorous evaluation, and reproducible research.

- **Develop and validate** a microscopic Alternative-Graph-driven reinforcement learning framework for train timetable rescheduling that produces *feasible* solutions with *low total and maximum delay* and *latency compatible with real-time operation.*
- **Benchmark rigorously** against a state-of-practice optimization-based tool (OptDis) under matched scenarios and time budgets, comparing solution quality, runtime behavior, and scalability.
- **Ensure extensibility and reproducibility** through a modular framework design that supports exact reruns and future extensions.

Those goals are further specified in the following subsection.

1.4.1 Specific Objectives

This subsection specifies the concrete technical objectives that guide the design, implementation, and evaluation of the proposed framework.

Data and modeling.

- Prepare and harmonize realistic railway infrastructure and timetable data to support microscopic conflict-resolution experiments.
- Construct graph-based timetable representations capable of modeling multiple trains and shared infrastructure resources.

Decision framework.

- Formulate timetable rescheduling as a sequential decision problem that preserves operational feasibility while limiting delay propagation.
- Ensure that only valid conflict-resolution decisions are considered during rescheduling.

Learning approach.

- Train a reinforcement-learning-based policy to support fast conflict-resolution decisions suitable for time-critical dispatching.
- Investigate the impact of different state representations and learning objectives on solution quality and robustness.

Evaluation.

- Design realistic disturbance scenarios to assess solution quality, generalization, and robustness.
- Compare the proposed approach against an industrial optimization-based solver using delay- and runtime-oriented metrics.

Deliverables.

- A reusable experimental framework and an empirical evaluation of learning-based timetable rescheduling under realistic conditions.

1.5 Thesis structure

This thesis is structured as follows. Chapter 2 provides background on railway timetabling, the Alternative Graph model, and deep reinforcement learning, and reviews related work. In the third chapter the design and methodology of the proposed framework, including the data pipeline based on LUKS exports, the Alternative Graph construction, and the formulation of the rescheduling problem as a reinforcement learning environment are discussed.

Next chapter describes the learning approach in detail, including the DQN agent architecture, action masking, training procedure, and implementation aspects. The following chapter 5, reports the experimental evaluation and benchmarking against the industrial OptDis solver, analyzing solution quality, computational performance, scalability, and ablation studies. Chapter 6 discusses the results in a broader context, highlights limitations, and concludes the thesis with directions for future research.

2

Background

This chapter provides the background required for the proposed approach. It first introduces concepts of microscopic railway timetabling and conflict modeling using the Alternative Graph. It then reviews reinforcement-learning formulations relevant to sequential conflict resolution. Finally, related academic and industrial approaches are discussed to position the contribution of this thesis.

2.1 Railway Timetabling and Conflict Modeling

Railway timetabling coordinates trains that share scarce infrastructure resources such as block sections, platforms, and junctions under signaling, safety, and headway constraints. A timetable fixes the order and timing of movements so that each train has exclusive access to infrastructure for defined time intervals. Disturbances such as delays or extended dwell times can cause these intervals to overlap, creating conflicts that require resolution.

Operational constraints depend on the signaling regime. In fixed-block systems, blocks must be cleared before a following train may enter, whereas in moving-block systems separation is determined dynamically based on braking constraints. In both cases, rescheduling aims to restore safe separation while limiting delay propagation.

A widely used conceptual framework for understanding these constraints is the *blocking-time* view (*Sperrzeiten*). For each train and route, the time during which infrastructure is occupied can be decomposed into several phases, including route setup, approach to the protecting signal, traversal of the block, track clearance (including train length), and route release. When plotted over successive infrastructure elements, these phases form a characteristic *blocking-time stairway* that visualizes when a resource is unavailable to other trains.

From this perspective, the minimum headway between two trains on a shared section is the smallest temporal separation that prevents their blocking-time stairways

from overlapping at the first common element. Any remaining separation constitutes buffer time, which provides robustness against small disturbances. Two types of conflicts are typically distinguished in practice: *buffer-time conflicts*, where the buffer is reduced but remains positive, and *occupancy conflicts*, where blocking times overlap and a clear precedence decision is required. This distinction explains how minor schedule changes can propagate delays along a corridor and how local spacing decisions affect overall timetable stability.

At the microscopic level, these interactions can be represented using an *Alternative Graph (AG)* model. In this representation, nodes correspond to train events on infrastructure elements and carry timing information, while arcs encode precedence relations. Fixed arcs represent constraints that must always be satisfied, such as running and dwell times within a train's schedule. Conflicts on shared resources are modeled using pairs of mutually exclusive alternative arcs, each corresponding to a different ordering of the involved trains. Selecting one alternative resolves the conflict by fixing the precedence on that resource.

The AG provides a compact and interpretable representation of timetable conflicts and their resolution. It supports stepwise decision-making, where individual precedence choices can be analyzed in relation to physical infrastructure constraints and blocking-time interactions. Detailed algorithmic formulations and decision-making methods based on this representation are introduced in the following chapters.

2.2 Related Work

Research on timetable rescheduling spans exact optimization, heuristic and metaheuristic approaches, and learning-based methods [16]. Mixed Integer Linear Programming (MILP) formulations encode microscopic precedence constraints, headways, and capacity limits, and solve for conflict-free schedules that minimize total delay or makespan. These models are strong in terms of solution quality and optimality guarantees, but their computational cost can become prohibitive on dense networks or under strict real-time decision deadlines [2]. Heuristics and metaheuristics [19], including genetic algorithms, tabu search, particle swarm optimization, and simulated annealing, trade optimality guarantees for faster response times and improved scalability, but are often sensitive to parameter tuning and scenario structure. Survey and integrated studies further highlight the complexity of disruption management, where timetable, rolling stock, and crew rescheduling interact, and where uncertainty handling and decision timing play a critical role [23]. A structured comparison of representative timetable rescheduling approaches discussed in this section is provided in 2.1

[3] presents a comprehensive survey of railway disruption management, covering timetable, rolling stock, and crew rescheduling problems. The work classifies exact optimization approaches, heuristics, and integrated recovery frameworks, and highlights the trade-off between modeling fidelity and computational tractability. While it provides a broad overview of existing methods and challenges, it does not propose a concrete rescheduling algorithm and therefore serves primarily as contextual background for this thesis.

An exact Mixed-Integer Linear Programming formulation for microscopic train timetable rescheduling under disturbances is proposed in [5]. The model captures detailed precedence constraints, headways, and block occupancy conflicts, and is capable of producing high-quality feasible schedules. However, the reported computational effort increases significantly with network size and disturbance complexity, limiting its applicability in real-time or highly congested scenarios.

The Alternative Graph (AG) has long been used as a microscopic backbone for train dispatching because it cleanly separates fixed intra-train precedence constraints from resource conflicts through the use of alternative arcs [4, 20]. This structure supports incremental decision making—commit a precedence choice, propagate times, and check feasibility—and reduces feasibility verification to the absence of negative cycles. As such, the AG aligns closely with dispatcher workflows and has been successfully employed in both exact optimization and heuristic control frameworks. Classical studies demonstrate AG-based branching strategies, area-wise coordination between adjacent dispatching regions, and fast feasibility propagation, often combined with pruning rules that restrict the decision space around stations or junctions to meet real-time requirements.

Building on these foundations, more recent work has explored the integration of AG-based models with deep reinforcement learning [11]. In these approaches, each unresolved conflict—represented as a pair of alternative arcs—is treated as a discrete action, the state encodes current timing and resource availability, and the reward reflects delay reduction. Deep Q-Networks (DQN) are a natural choice in this setting due to the discrete action space. Feasibility studies on small or simplified networks [7], as well as experiments on selected real-world areas, indicate that trained policies can approach the solution quality of MILP-based methods while producing decisions orders of magnitude faster at inference time. Across these studies, several practical design principles recur: state representations must remain compact as traffic density grows, action masking is essential to prevent infeasible decisions and accelerate learning, and reward definitions should correlate directly with operational delay metrics. Some authors also report that combining stepwise delay reduction with a small terminal reward can improve training stability without compromising convergence speed.

Other learning-based directions investigate centralized versus decentralized control and multi-agent formulations. Decentralized approaches assign local agents to individual trains, which can improve scalability but may lead to myopic behavior if coordination mechanisms are weak. Centralized policies operate on a global view of the system and can capture delay propagation effects more accurately, at the cost of larger state and action spaces [25]. Graph Neural Networks (GNNs) have also been proposed as state encoders to exploit railway network structure explicitly [12]. While promising in terms of generalization, such models increase architectural complexity and reduce transparency. Across learning-based studies, common operational lessons emerge: unsafe actions should be masked rather than penalized post hoc, rewards should remain simple and interpretable, and detailed logging of constraints and timing is necessary to diagnose failures.

A hybrid rescheduling framework combining data-driven components with Mixed-Integer Linear Programming is presented in [17]. Machine-learning models are used to estimate delay propagation and constraint tightness, while final conflict resolution

is performed through an optimization solver. Although this integration improves robustness compared to purely optimization-based methods, the reliance on repeated MILP solves results in substantial computational overhead, which can hinder real-time applicability under heavy traffic conditions.

[9] investigates a reinforcement-learning-based approach to support real-time dispatching decisions. Conflict resolution is modeled as a sequential decision process, and learned policies are evaluated as fast decision-support tools rather than full timetable optimizers. The approach emphasizes practical usability but is demonstrated primarily on limited prototype scenarios and does not incorporate a full microscopic Alternative Graph representation or large-scale industrial data.

On the optimization side, robust and stochastic rescheduling models explicitly address uncertainty in running and dwell times, improving worst-case performance and service reliability at the expense of increased computational effort. Decomposition techniques and rolling-horizon approaches mitigate this cost by optimizing over limited time windows, applying decisions, and re-optimizing as the system evolves [5]. Corridor-level and station-centric models further reduce problem size by focusing on local bottlenecks. These strategies are not inherently opposed to learning-based methods; rather, they motivate hybrid approaches in which fast learned policies generate initial precedence orderings or prune poor choices, while exact solvers refine timing decisions.

A growing body of work explores tighter integration between learning and optimization. Imitation learning uses high-quality MILP solutions to train policies that mimic expert decisions, reducing exploration overhead. Learning-to-rank and pointer-network architectures score conflicts or candidate orderings before a short optimization "polish" step. Offline reinforcement learning and dataset aggregation leverage historical dispatch data or solver traces, which is attractive in safety-critical settings where online exploration is undesirable. Transfer learning and domain randomization further aim to improve robustness across varying traffic densities and disturbance patterns [14].

Industrial-grade solvers such as OptDis remain strong baselines due to their consistent solution quality under bounded computation times. As a result, recent studies emphasize the importance of fair and operationally meaningful comparisons, including matched disturbance scenarios, matched time budgets, and common metrics such as total delay, maximum delay, and time to first feasible solution. Rather than framing the problem as a competition between learning and optimization, this perspective highlights complementary strengths, where learning-based policies provide rapid, good-enough decisions and optimization ensures high-quality final solutions [10].

Finally, data realism remains a key differentiator between research prototypes and deployable systems. While many learning-based studies rely on synthetic or heavily simplified inputs, this thesis operates directly on industrial data exported from the LUKS system, including infrastructure elements, running times, and occupancy information. Block and segment extraction are performed directly from LUKS outputs, while preserving compatibility with standard AG construction pipelines. This design choice allows the proposed learning framework to be evaluated under realistic operational conditions and enables each decision to be traced back to concrete infrastructure elements and timing data familiar to practitioners.

Paper	Micro	AG	MILP	Heur.	RL	Seq. Dec.	Feasible	Real Data	Main limitation
Cacchiani et al. (survey) [3]	–	–	✓	✓	–	–	–	–	Survey: breadth, not method-specific.
Fischetti et al. (EJOR 2017) [6]	✓	–	✓	–	–	✓	✓	–	Scalability / solve times for large disturbed instances.
Aydın et al. (2023) [2]	✓	✓	✓	–	–	✓	✓	✓	Time limits under heavy conflicts.
Agasucci et al. (2023) [1]	✓	–	–	–	✓	✓	–	–	Scalability challenges; limited network sizes.
Zhu (2020) [27]	✓	–	–	–	✓	–	–	–	Early RL work; synthetic/limited scenarios.
Kim et al. (2023) [11]	✓	✓	–	–	✓	✓	–	–	Small-scale tests; limited generalization results.
Liu (2024) [13]	✓	–	–	–	✓	✓	–	–	Focus on single-track; limited topologies.
Yue et al. (2024) [24]	✓	✓	–	–	✓	✓	–	–	Advances scalability; needs broader industrial validation.
Zhang et al. (2024) [26]	✓	✓	✓	–	✓	✓	✓	–	Hybrid RL–optimization; early-stage evaluation.
Veelenturf (2016) [18]	–	–	✓	–	–	–	✓	–	Macroscopic focus; not microscopic.
Josyula (2024) [9]	✓	–	–	–	✓	–	–	–	Practical decision-support focus; prototyping.
Su (2024) [17]	✓	–	✓	–	–	✓	✓	–	Integrated MILP + data-driven; still compute-heavy.
This thesis	✓	✓	–	–	✓	✓	✓	✓	Aims to combine AG realism, RL speed and benchmark against OptDis.

Table 2.1 Comparison of representative railway timetable rescheduling approaches

In summary, the literature suggests three guiding principles that shape this work: the Alternative Graph provides a robust microscopic foundation for both optimization and learning; DQN with feasibility-aware action masking is a practical reinforcement learning choice for discrete conflict resolution; and rigorous, scenario-matched comparisons against industrial solvers are essential to assess real-time value. Following these principles, the present thesis contributes a LUKS-based data pipeline, an AG-driven rescheduling environment with explicit feasibility checks, a masked-action DQN agent, and a systematic benchmark against OptDis under realistic time constraints.

While recent studies explore practical decision support and hybrid learning–optimization frameworks, many either rely on simplified settings or remain computationally heavy. This thesis differs by combining an industrial-grade Alternative Graph model with reinforcement learning, enabling fast, feasible conflict resolution and systematic benchmarking against a state-of-practice solver using real operational data.

2.3 Industrial Railway Rescheduling Tools

In this thesis, the primary industrial reference and benchmarking baseline is the LUKS (Leistungsuntersuchung Knoten und Strecken) tool [8] developed and used at Quattron GmbH. LUKS is employed in operational and planning contexts to analyze railway node and corridor performance and to resolve timetable conflicts under disturbances. Its rescheduling component, OptDis, formulates conflict resolution as a Mixed-Integer Linear Programming problem and is capable of producing high-quality feasible schedules under a wide range of operational constraints. LUKS and OptDis [22] were selected as the reference tools for this work for several reasons. First, they are actively used in an industrial setting, making them a relevant and credible baseline for evaluating practical rescheduling approaches. Second, the availability of realistic infrastructure, timetable, and disturbance data exported from LUKS enables a consistent and reproducible experimental setup. Finally, OptDis [21] represents a state-of-practice optimization-based approach, providing a strong benchmark against which alternative methods can be fairly assessed under matched time and scenario constraints.

Several other tools and frameworks exist for railway timetable analysis and rescheduling. Commercial systems such as RailSys, OpenTrack, and TPS provide microscopic simulation and timetable analysis capabilities and are widely used for planning and capacity studies. However, these tools primarily focus on simulation-based evaluation rather than automated conflict resolution under tight operational time constraints.

In the academic domain, various optimization- and heuristic-based rescheduling frameworks have been proposed, often tailored to specific networks or problem variants. While these approaches provide valuable insights, they are typically not integrated with industrial data pipelines or deployed in real operational environments.

Despite their strengths, these alternative tools are not used as primary baselines in this thesis. Many commercial systems do not expose internal decision mechanisms or provide programmatic interfaces suitable for systematic benchmarking under controlled time budgets. Similarly, academic prototypes often rely on simplified or synthetic datasets, limiting their suitability for direct comparison with industrial-grade rescheduling approaches. In contrast, LUKS and OptDis provide both realistic data access and an established optimization-based rescheduling method, making them well suited for the objectives of this work.

2.4 Railway Infrastructure Model

The conflict resolution problem addressed in this thesis is defined on a microscopic representation of railway infrastructure derived from operational planning data exported from the LUKS system shown in Figure 2.1. The infrastructure model provides the physical and logical constraints within which train movements must be scheduled and rescheduled.

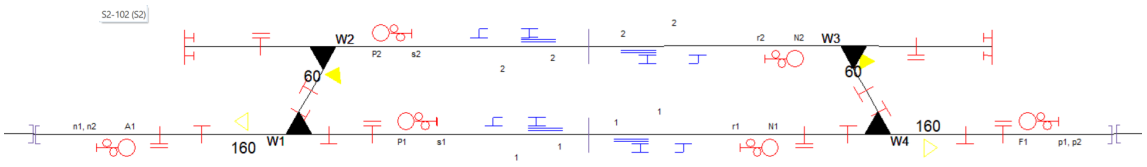

Figure 2.1 Infrastructure Element Export

2.4.1 Infrastructure Elements

The railway network is decomposed into a finite set of infrastructure elements, each of which represents a safety-critical resource that can be occupied by at most one train at a time. The primary element types considered are:

- **Main signal blocks**, representing fixed-block track sections protected by main signals;
- **Switch blocks**, representing turnouts and junctions where multiple routes intersect;
- **Terminal blocks (Gleisende)**, representing track ends and buffer stops.

Each element is uniquely identified and associated with a physical position (kilometer reference), track context, and direction of travel. For modeling and experimentation purposes, an Example Railway Network and trains are taken as shown in Figure 2.2, and elements are treated as discrete blocks that enforce exclusive occupancy constraints between trains.

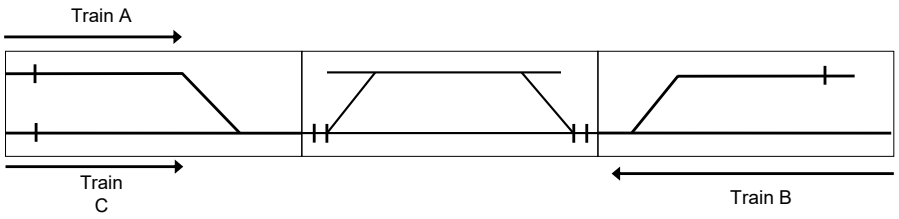

Figure 2.2 Example Railway Network, trains

2.4.2 Train Routes and Block Occupation

Each train follows a predefined route through the infrastructure, represented as an ordered sequence of blocks as shown in Figure 2.3. For every train–block pair, the

planned timetable specifies an entry time and an exit time, defining the temporal occupation of that block by the train. These times implicitly capture running, approach, clearance, and release intervals, consistent with the blocking-time stairway concept used in railway capacity analysis.

Multiple trains may traverse the same block, potentially in opposite directions or via different routes through switches. Whenever two trains request the same block during overlapping time intervals, an infrastructure conflict arises and must be resolved by imposing a temporal ordering.

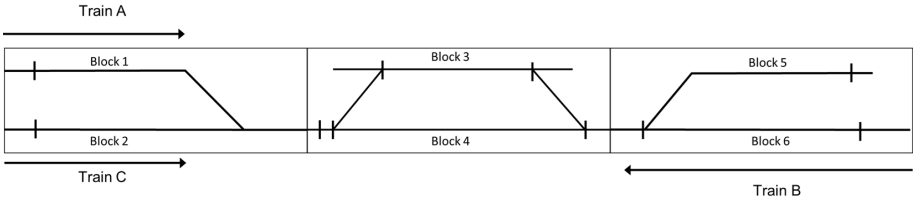

Figure 2.3 Example Railway Network, trains with blocks Notation

2.4.3 Types of Conflicts

Two principal conflict patterns occur in the considered infrastructure:

- **Switch conflicts**, where trains approach a shared switch block from different directions or routes and cannot occupy it simultaneously;
- **Signal block conflicts**, where trains compete for the same main signal block, typically in opposite directions or with insufficient headway separation.

In both cases, safety regulations require that one train fully clears the block before the other is allowed to enter, enforcing a strict precedence relation.

2.4.4 Mapping Infrastructure to the Alternative Graph

The infrastructure model forms the basis for the construction of the Alternative Graph (AG). Each visit of a train to a block is represented as a node in the AG, encoding the planned and actual timing of that event as shown in Figure 2.4. Fixed arcs connect successive nodes of the same train and represent mandatory intra-train precedence constraints derived from the route structure and minimum running times.

The Alternative Graph represents train–block events and precedence constraints, enabling conflict resolution through iterative precedence selection and timing propagation.

In the example Railway Network, considering Train A moving from block 1 to block 5 through block 4 and Train C moving from block 2 to block 5 through block 4 then the alternative graph looks like as shown in Figure 2.4

Conflicts arising from shared infrastructure blocks are represented by pairs of alternative arcs [11]. Each pair corresponds to a binary precedence decision between two

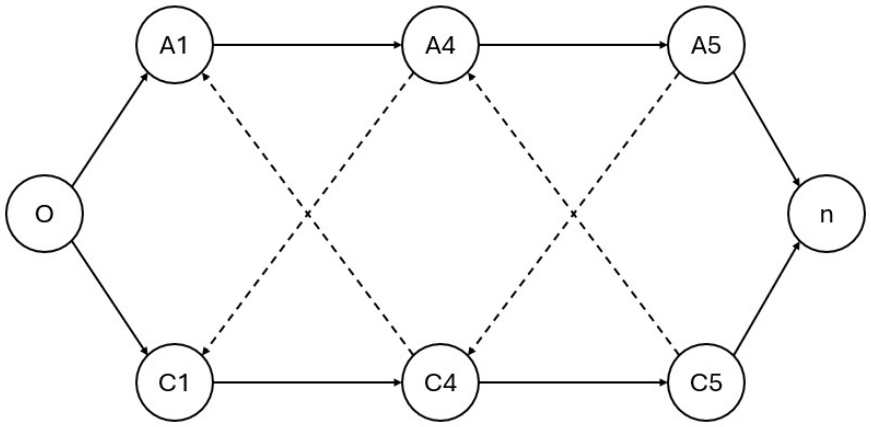

Figure 2.4 Alternative graph of Example Railway Network, trains

trains competing for the same block, expressing the two feasible orderings in which the block can be allocated. Selecting one alternative arc enforces a specific train ordering and removes the opposite option from consideration.

This representation preserves a direct correspondence between physical infrastructure constraints and graph-based scheduling decisions, enabling both exact optimization and reinforcement learning methods to operate on a common, interpretable model of railway operations [15][20].

Role in the Reinforcement Learning Environment

Within the reinforcement learning environment, unresolved conflicts in the Alternative Graph correspond to discrete precedence decisions on shared infrastructure, whose timing effects are propagated through the graph.

3

Design

This chapter describes the design and implementation of a reinforcement learning–based framework to resolve train timetable conflicts using the Alternative Graph (AG) model and Deep Q-Network (DQN) techniques. The chapter focuses on how the rescheduling problem is formulated, how operational railway data are transformed into a graph-based representation, and how this representation is embedded into a reinforcement learning environment suitable for sequential decision-making.

The framework centers on the design and implementation of a flexible reinforcement learning environment for train timetable rescheduling, closely integrated with data and operational models exported from Quattron's LUKS software. The process begins with the extraction, cleaning, and structuring of railway network and timetable data, producing consistent datasets that capture detailed block occupancy and timing information. These data are then used to construct the Alternative Graph, in which nodes represent train–block events and arcs encode both fixed intra-train precedence constraints and alternative conflict relations. Edge weights and feasibility checks are defined to ensure that all generated schedules satisfy operational timing constraints. An overview of the proposed design framework is shown in Figure 3.1.

Figure 3.1 illustrates the end-to-end workflow of the proposed framework, starting from a disturbed railway timetable, followed by data extraction and Alternative Graph construction, environment modeling, and DQN training, and concluding with the evaluation and comparison against OptDis to produce a conflict-resolved timetable. The individual phases of this pipeline and their implementation details are described in depth throughout this chapter.

The reinforcement learning environment is implemented as a modular Python framework and modeled as a Markov Decision Process (MDP) to support step-by-step conflict resolution. The state representation encodes the current configuration of the Alternative Graph, including planned and propagated event times and unresolved conflicts. The action space consists of feasible conflict-resolution decisions, where each action assigns precedence to one train on a shared block. Applying an

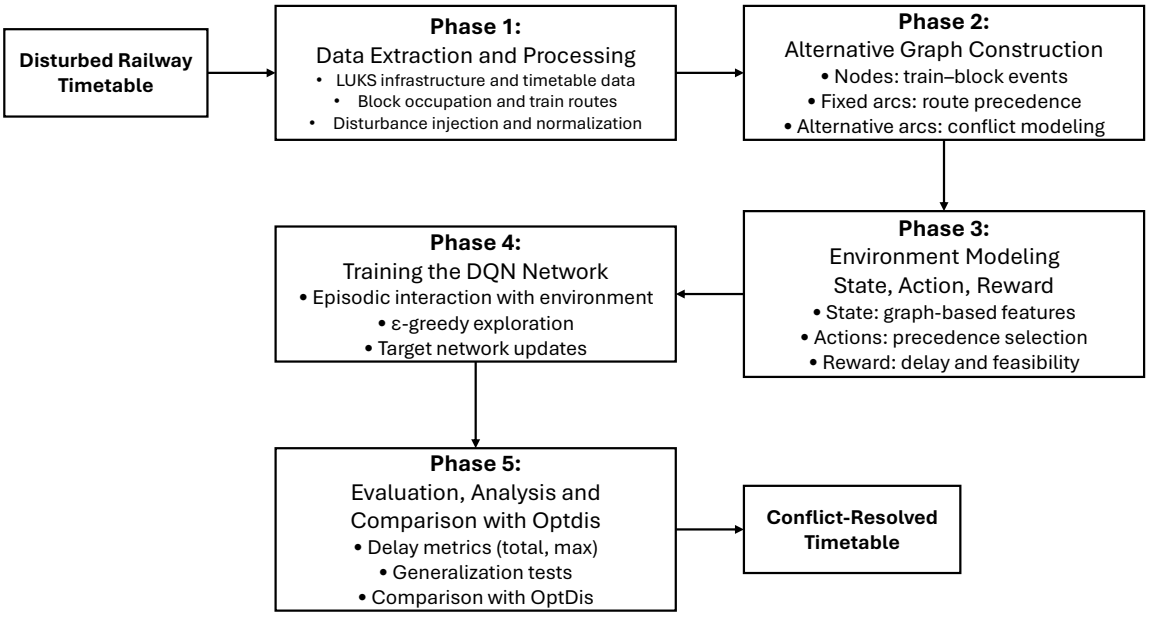

Figure 3.1 Pipeline of the proposed design framework

action updates the graph structure, propagates timing constraints, and updates delay measures, while the reward function provides feedback based on changes in a delay-based objective.

All experiments were conducted with a set of fixed random seeds, and intermediate artifacts including processed datasets, Alternative Graph snapshots, trained models, and logs were maintained to allow exact reruns and reproducibility of results.

Deep Q-Network Architecture

The reinforcement learning agent employed in this work is based on a Deep Q-Network (DQN), which uses an artificial neural network to approximate the action-value function. The neural network maps the current environment state to Q-values for all admissible actions, representing the expected cumulative reward associated with each conflict-resolution decision.

In the context of train timetable rescheduling, the network input consists of a compact state representation derived from the Alternative Graph, encoding delay information and conflict characteristics. The network output corresponds to precedence decisions between conflicting train movements. By selecting actions that maximize the estimated Q-values, the agent learns a policy that resolves conflicts while minimizing delay propagation.

A target network and experience replay are employed during training to stabilize learning and improve convergence. These techniques allow the agent to learn effective conflict-resolution strategies from repeated interaction with the environment.

MDP and Learning Formulation

The sequential resolution of conflicts in the Alternative Graph (AG) is modeled as a Markov Decision Process (MDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$. A state $s \in \mathcal{S}$ encodes the current configuration of the AG, including propagated event times, timing slack information, and the set of unresolved conflicts. The action space $\mathcal{A}(s)$ consists of feasible conflict-resolution decisions, where each action commits a precedence choice between two trains competing for the same infrastructure resource. Action masking is applied so that only unresolved and feasible conflicts are available at each decision step.

The transition function $P(s' \mid s, a)$ fixes the selected alternative arc in the AG, removes its opposing arc, and propagates timing constraints through the graph. If this propagation results in an infeasible configuration, such as the creation of a negative cycle, the action is rejected or penalized.

The reward function is defined to directly reflect the objective of delay minimization. The step reward is computed as the reduction in total delay,

$$r(s, a, s') \;=\; \Delta\mathrm{TD} \;=\; \mathrm{TD}(s) - \mathrm{TD}(s'), \tag{3.1}$$

where the total delay is given by

$$\mathrm{TD}(s) \;=\; \sum_{k \in \mathcal{T}} \sum_{m \in \mathcal{M}_k} \max\bigl\{0,\; t^{\mathrm{act}}_{k,m}(s) - t^{\mathrm{plan}}_{k,m}\bigr\}. \tag{3.2}$$

Here, $\mathcal{T}$ denotes the set of trains, $\mathcal{M}_k$ the set of relevant milestones for train k, $t^{\mathrm{plan}}_{k,m}$ the planned time, and $t^{\mathrm{act}}_{k,m}(s)$ the current propagated feasible time. The discount factor γ balances short- and long-term effects of precedence decisions.

To obtain fast conflict-resolution decisions, the optimal action-value function $Q^\star(s, a)$ is approximated using a Deep Q-Network (DQN). The optimal value function satisfies the Bellman optimality equation

$$Q^\star(s, a) \;=\; \mathbb{E}\Bigl[r + \gamma \max_{a'} Q^\star(s', a') \,\Big|\, s, a\Bigr]. \tag{3.3}$$

In practice, a neural network $Q_\theta(s, a)$ is trained by minimizing the temporal-difference loss

$$\mathcal{L}(\theta) \;=\; \mathbb{E}_{(s,a,r,s')}\Bigl[\bigl(r + \gamma \max_{a'} Q_{\bar{\theta}}(s', a') - Q_\theta(s, a)\bigr)^2\Bigr], \tag{3.4}$$

where $Q_{\bar{\theta}}$ denotes a slowly updated target network used to stabilize training. During both training and inference, action masking is applied to the Q-values so that only feasible conflict-resolution actions participate in the maximization operator.

3.1 Data Extraction and Processing

This project focuses on extracting, cleaning, and structuring railway network and train schedule data from various LUKS software exports (Excel and XML). The resulting datasets are suitable for analysis, timetable modeling, and reinforcement learning experiments.

A modular data pipeline converts data that is initially exported from Leistungsuntersuchung Knoten und Strecke (LUKS) into consistent infrastructure and timetable datasets, normalizes element names and IDs, reconstructs per-train paths with kilometer positions, and builds time-ordered segment traversals from occupancy files. These aligned artifacts are used to construct AG instances that reflect both network structure and planned operations.

The result is a reproducible pipeline: data processing with `pandas`, graph maintenance with `networkx`, structured spreadsheet outputs for inspection, and an environment API with `reset/step` semantics. The framework is designed to generalize beyond a single case study, allowing support for new networks and timetables with minimal changes and enabling hybrid use—standalone decision support or warm starts for MILP.

3.1.1 Data Sources

In an initial step the pipeline starts from structured exports produced in LUKS. From the infrastructure side, we use the element list that includes switches, main signals, and end-of-track markers (*Gleisende*), together with their line/track context and kilometer positions. From the timetable side, we use the running-time export that lists, for each train, the sequence of visited infrastructure elements and the corresponding times; when available, we also use occupancy/timing sheets to obtain explicit entry (*Anf*) and exit (*Ende*) times on segments. All names are normalized to a canonical form (uppercasing, whitespace trimming, standard type tags) so that the timetable text can be matched back to the infrastructure dictionary. Kilometer values are parsed from the same fields and stored as numeric references, which we later use to sort events and to validate running/clearance times.

A small but important convention is that physical segments are identified once, independent of direction, while the travel direction itself is carried as a separate attribute in the timetable layer; this keeps combining stable when multiple trains traverse the same track in opposite directions. Finally, the sources are read into data frames and persisted as intermediate tables so that each transformation step can be audited.

Three kinds of LUKS exports are utilized: i) an *infrastructure export* copied from the “Infrastructure” tab in LUKS provides the authoritative list of track elements (main signals, switches, and end-of-track markers) together with their attributes (line/track context, kilometer positions, and identifiers); ii) a *timetable export* supplies, for each train, the planned sequence of visited elements and scheduled times; this is our basis for reconstructing direction and ordering along the corridor. iii) a set of *per-train occupation sheets* gives block-entry and block-exit timestamps at a finer resolution, which we use to attach precise arrival and departure times to train–segment traversals and to validate headways and clearances during graph construction. Throughout the pipeline, element names are normalized and mapped to stable global IDs so that these sources can be joined reliably without depending on specific file names.

3.1.2 Network generation overview.

The infrastructure network is transformed into a direction-agnostic segment representation that serves as a stable backbone for timetable alignment. Each physical connection is assigned a unique segment identifier independent of travel direction, while direction is stored as an attribute at the timetable level. From this representation, admissible ordered segment pairs are enumerated to capture local continuity constraints around signals, switches, and terminals. The process is described in Alg. 1, iterating ...

Input: Infrastructure element list exported from LUKS
Output: Segment inventory and admissible segment-pair grammar
foreach *infrastructure connection* (u, v) **do**
  normalize element names and types assign a unique `Segment_ID` to the unordered pair $\{u, v\}$
foreach *segment* **do**
  enumerate admissible same-direction successor segments
add terminal connections (*Gleisende*) where applicable

Algorithm 1 Infrastructure Network Generation

3.1.3 Outputs and Structure

The previously described process produces two main spreadsheet outputs that act as a clean interface to the scheduling model. The infrastructure side is captured, which lists the canonical segment inventory (each with a direction-agnostic `Segment_ID`) and all admissible ordered segment pairs that define legal continuity in the network. This file is independent of any timetable and can be reused across scenarios. The timetable side is also captured and it contains, for each train, the time-ordered sequence of realized segment pairs with resolved IDs, names, and numeric `from_km`/`to_km`; when available, arrival and departure times are attached to each pair so that running and clearance can be checked later. Because both outputs share the same segment IDs, multiple trains naturally reference the same physical resource, and opposing movements remain consistent by construction.

These tables are then lifted into the Alternative Graph: nodes are created as train–block (segment) events with planned and mutable times; fixed arcs come from the intra-train order implied by the per-train list; and alternative arcs are generated wherever two or more trains request the same segment (or segment pair) within overlapping windows. With stable IDs, explicit terminal handling via *Gleisende*, and kilometer-based ordering, the AG remains easy to audit, and the resulting environment exposes a compact, reproducible state to the DQN agent without special-case logic downstream. For each segment/block traversed by every train, the train schedule input file includes:

- **trainname**: Train identifier
- **direction**: Direction of travel
- **order**: Sequential order in the route

- **network_segment_id**: Unique segment identifier
- **Anf**: Entry time for the segment
- **Ende**: Exit time for the segment

This forms a time-ordered mapping of train movements across the network. Here are the technical details used in the data extraction process:

- Utilizes `pandas` for data manipulation and cleaning.
- Employs regular expressions for robust parsing of text fields.
- Ensures consistency by globally mapping elements and segments to unique IDs.

3.2 Data and Alternative Graph Construction

This section describes how the processed timetable and infrastructure data are converted into a consistent *Alternative Graph (AG)* together with a set of explicit conflict decisions suitable for reinforcement learning or search-based methods.

The role of this generator is strictly constructive: it produces the graph structure, timing constraints, and conflict sets. The formal definition of the Markov Decision Process (state, action, reward, and transition dynamics) is deferred to Section 3.4.

3.2.1 Input Signatures and Invariants

The generator consumes two tab-separated input files derived from the outputs of Section 3.3:

- A **network file** describing directed physical connections between blocks as tuples $\langle \text{from\_block}, \text{to\_block}, \text{direction}, \text{distance} \rangle$. This file defines the physical backbone of the corridor.
- A **train file** listing time-stamped visits of blocks by trains as $\langle \text{train\_id}, \text{direction}, \text{order}, \text{block}, t_{\text{arr}}, t_{\text{dep}} \rangle$, where times are given in `HH:MM:SS` format.

The network parser builds a directed graph $G_P = (V_P, E_P)$ where edges carry direction and distance attributes. All blocks encountered are indexed exactly once and stored in a stable list used later for compact encodings and reproducibility.

The train parser converts times into seconds since midnight and constructs a `nodeInfo` object for each train–block visit. Two sentinel nodes are added: a dummy *start* node (index 0) and a dummy *end* node (index $N - 1$). Each node stores forward and backward pointers (`next_fixed_node`, `prev_fixed_node`) that will later encode intra-train precedence.

3.2.2 Construction of Fixed Arcs

Fixed arcs encode mandatory intra-train precedence constraints derived from the planned route order and minimum running or dwell times. Dummy start and end nodes anchor all trains to a common temporal reference. The process is described in Alg.2,

Input: Train routes with ordered block visits
Output: Fixed precedence arcs in the Alternative Graph
foreach *train* k **do**
  foreach *consecutive block visit* (i, j) *of train* k **do**
    add fixed arc $i \to j$ with weight $-d_{ij}$
  connect start node to the first block visit of train k connect the last block visit of train k to the end node

Algorithm 2 Fixed Arc Construction

3.2.3 Alternative arcs and conflict filtering

Conflicts occur when multiple trains request the same physical block. Each conflict is encoded as a pair of mutually exclusive alternative arcs representing the two possible precedence orders. Boundary conflicts near terminals are resolved deterministically to reduce the action space. The process is described in Alg.3,

Input: Alternative Graph with shared block usage
Output: Filtered set of feasible alternative arcs
foreach *shared block* b **do**
  foreach *train pair* (i, j) *using block* b **do**
    create alternative arc pair $(i \prec j)$ and $(j \prec i)$
foreach *alternative arc* **do**
  if *boundary or deterministic case* **then**
    fix precedence and remove the alternative arc

Algorithm 3 Alternative Arc Enumeration and Filtering

3.2.4 Feasibility Checking and Objective Evaluation

Feasibility and schedule quality are evaluated using single-source Bellman–Ford relaxations on G_A. The helper function `cost(`G_A`)` first checks for negative cycles. If one is detected, the graph is infeasible and a sentinel value (`BigM`) is returned. Otherwise, the weight of the path from the dummy start to the dummy end node is returned.

Because all constraints are encoded with negative weights, minimizing this path weight corresponds to maximizing feasibility. In the learning environment, the objective is defined as the negated cost, so lower delay yields higher return.

For diagnostics and reward shaping, additional routines extract per-train delay profiles from the Bellman–Ford distances. For a node n, the realized departure time is obtained as

$$t_n^{\text{act}} = \begin{cases} -\ell\big(\text{next\_fixed\_node}(n)\big), & \text{if successor exists,} \\ -\ell(n), & \text{otherwise,} \end{cases}$$

and the instantaneous delay is $\delta_n = t_n^{\text{act}} - t_n^{\text{plan}}$. These quantities support the computation of first/last delays per train and the largest delay increments acquired between consecutive blocks.

3.2.5 Outputs and Interface to the Environment

The generator produces three artifacts:

- a fully constructed Alternative Graph with fixed and unresolved alternative arcs;
- a list of unresolved conflict records corresponding to feasible decisions;
- node metadata required for timing propagation and delay analysis.

These artifacts define the initial state of the reinforcement learning environment. No MDP semantics are imposed at this stage: the generator is agnostic to how conflicts are selected or rewards are computed. This separation keeps data construction, feasibility logic, and learning dynamics modular and auditable.

3.3 Cost, Delay, and Reward Formulation

The rescheduling problem is encoded as an AG with a dummy start node (index 0) and a dummy end node (index $N-1$). Fixed arcs carry run/dwell constraints; alternative arcs represent unresolved conflicts. The *graph cost* is computed as the shortest (most negative) path length from start to end using Bellman–Ford; if a negative cycle is detected, a large penalty is returned. Precisely, the implementation checks for a negative cycle and, if none exists, obtains the path $0 \to N-1$ and its weight; otherwise it returns `BigM` (sentinel). In the environment, we maximize $-$`cost` so that *lower* schedule delay corresponds to *higher* return.

$$\text{cost}(\mathcal{G}) \;=\; \begin{cases} \text{path\_weight}\big(0 \to N-1\big), & \text{if no negative cycle,} \\ \texttt{BigM}, & \text{otherwise,} \end{cases} \qquad \text{objective } \mathcal{J} \;=\; -\text{cost}(\mathcal{G}). \tag{3.5}$$

Per-train delays are derived from single-source Bellman–Ford distances ($0 \rightsquigarrow v$). For a node n with realized departure $\widehat{d}_n$ and scheduled departure d_n, the instantaneous delay is $\delta_n = \widehat{d}_n - d_n$. The code computes per-train first/last delays and the largest *increment* in delay acquired between consecutive blocks (as well as the largest earliness decrease), which are later used for reward shaping and constraint caps. Let

$\Delta^{+}_{\max}$ denote the episode maximum over trains of positive per-block delay increments; let $\Delta^{-}_{\max}$ denote the maximum magnitude of negative (earliness) increments.

The one-step reward combines an *objective improvement* term with soft penalties for monotonicity and capped increments. With $J_t = -\text{cost}_t$, the base dense reward is a normalized improvement $r_{\text{obj}} = \frac{J_t - J_{t+1}}{\max(1, |J_t|)}$. Monotonicity discourages finishing a train with more delay than at its first block, i.e., $\sum_i \max(0,\, \delta_i^{\text{end}} - \delta_i^{\text{start}})$. Conflict-induced lateness increments and earliness increments are softly capped by user-set thresholds `conflict_delay_cap` and `early_cap`. The environment aggregates these with weights $w_{\text{obj}}, w_{\text{mono}}, w_{\text{cap}}, w_{\text{early}}$, and adds a tiny step penalty to encourage shorter episodes:

$$
\begin{aligned}
r_t = \; & w_{\text{obj}} \frac{J_t - J_{t+1}}{\max(1, |J_t|)} \; - \; w_{\text{mono}} \sum_i \frac{\max(0,\, \delta_i^{\text{end}} - \delta_i^{\text{start}})}{C} \\
& - \; w_{\text{cap}} \cdot \phi\big(\Delta^{+}_{\max}, \texttt{conflict\_delay\_cap}\big) \; - \; w_{\text{early}} \cdot \phi\big(\Delta^{-}_{\max}, \texttt{early\_cap}\big) \\
& - \; \texttt{step\_pen}
\end{aligned}
\tag{3.6}
$$

where C is a delay normalization constant and $\phi(x, \tau) = \max(0,\, x - \tau)/\tau$ is a soft hinge. Terminal reward equals $-$cost at "Finish," with hard penalty for infeasibility (negative cycle). The environment also tracks episode metrics such as the maximum final delay across trains for logging and analysis.

A DQN agent is trained within this environment, utilizing techniques such as epsilon-greedy exploration, experience replay, and a target network for stability. The reward function is designed to incentivize the agent to minimize delays and efficiently resolve conflicts.

3.4 Modeling the Environment

This chapter describes the design of the reinforcement learning environment for train timetable conflict resolution. The environment builds on the Alternative Graph representation of the railway infrastructure and timetable, and is formulated as a Markov Decision Process that enables sequential resolution of conflicts. The current AG encodes the state; an action selects one unresolved alternative arc; and the transition fixes precedence, removes the opposite arc, and propagates earliest feasible times. Feasibility is maintained through longest-path updates and negative-cycle checks; masking ensures only valid conflicts are presented.

3.4.1 Overview

The environment operationalizes the timetable rescheduling of trains as a finite-horizon MDP layered on top of an AG. Each episode starts from a feasible (or near–feasible) AG built from the current timetable and network, and proceeds by repeatedly selecting a contested block and exposing/committing precedence relations until (i) all conflicts are resolved, (ii) the agent declares *Finish*, or (iii) feasibility is

violated (negative cycle). All timing quantities, rewards, and feasibility checks derive from a single Bellman–Ford pass on the current AG, ensuring consistent semantics across state, transition, and objective. *Curriculum* schedules for caps on per–block delay increments stabilize training, and a compact state encoder keeps dimensions fixed while the underlying graph evolves.

3.4.2 MDP Components in the Environment

The rescheduling problem is formulated as a Markov Decision Process (MDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ as shown in Figure 3.2, where the environment encapsulates the Alternative Graph (AG) and enforces all feasibility constraints [16].

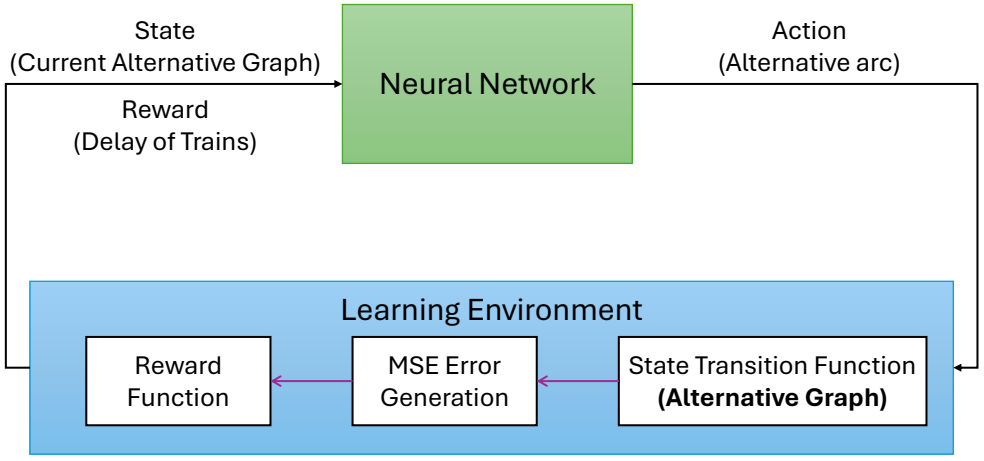

Figure 3.2 Reinforcement Learning Framework

The state space $\mathcal{S}$ consists of graph-derived features representing the current scheduling situation, including both static infrastructure information and dynamic timing and conflict-related attributes, as detailed in Section 3.4.6. The state is fully determined by the current configuration of the AG.

The action space $\mathcal{A}$ corresponds to unresolved conflicts in the AG. Each action selects one precedence relation by fixing an alternative arc and removing its opposing counterpart. Actions that would violate feasibility by introducing negative cycles are excluded from the admissible action set.

The transition function $\mathcal{P}$ is deterministic. Given a state and an action, the environment updates the AG accordingly and propagates timing constraints through longest-path computations.

The reward function $\mathcal{R}$ evaluates the effect of an action based on resulting train delays and feasibility, as described in Section 3.4.7. The discount factor $\gamma \in (0, 1]$ controls the trade-off between immediate and future delay reduction.

Value-based formulation rationale.

The rescheduling problem is modeled using a value-based formulation because each decision commits a single, irreversible precedence choice whose impact unfolds over subsequent conflict resolutions. Estimating action values therefore provides a natural mechanism to compare competing precedence decisions based on their long-term effect on delay propagation. In contrast, policy-gradient methods learn stochastic action distributions, which is less suitable in a feasibility-critical setting where decisions are deterministic and must be evaluated sequentially.

3.4.3 Core Class Structure and Methods

The class `Env` encapsulates the lifecycle: `__init__()` loads the physical network and schedule, builds the initial AG with dummy start/end, enumerates conflict records (alternative arc pairs), applies boundary fixes, and initializes per–train queues. `reset()` reconstructs an episode (including optional random entry delays), reapplies conflict enumeration/fixes, sets curriculum caps, caches start/end delays for monotonicity checks, and returns the initial state. `step(a)` executes the transition described above, recomputes objective and delay summaries, emits the shaped reward, updates episode statistics (max final delay; max late/early increments; return), and returns $(s', r, \texttt{done})$. Utility methods generate state features (`ag_to_state()`), derive current block occupancy and next–block lookahead, and compute direction flags and AG degrees at frontier nodes. All plotting/labels are cached for optional visualization.

3.4.4 Environment Initialization

The initialization of the environment ties together data and graph builders. The physical block graph G_P and canonical block index list `tc_list` come from the generator; time–stamped train events are parsed into sequential node records with planned arrival/departure; the AG is created by linking within–train fixed arcs (weights as negative required durations) and dummy connectors; and conflict records are enumerated per block as symmetric alter pairs, then *materialized on demand*. A first pass fixes trivial alter pairs at boundaries (those pointing to the dummy end) so the action space presented early in an episode reflects only genuine choices. Caps (τ_ℓ, τ_e), weights $(w_{\text{obj}}, w_{\text{cap}}, w_{\text{early}})$, and the tiny `step_pen` are set here, along with counters for feasible/optimal runs. All subsequent resets reuse these persistent objects to avoid I/O overhead.

3.4.5 Step Function and State Transitions

At each decision step, the environment receives a valid action corresponding to the resolution of a single conflict. Applying an action fixes the selected alternative arc in the AG and removes the opposing arc, thereby enforcing a precedence constraint between two trains on a shared block.

Features	Dimension	Type
Number of total trains (N_T)	1	Static
Physical network (PN)	$\|\mathcal{B}\| \times \|\mathcal{B}\|$	Static
Junction indicator (JI)	$\|\mathcal{B}\|$	Static
Occupation indicator (OI)	$\|\mathcal{B}\|$	Dynamic
Next occupation indicator (NI)	$\|\mathcal{B}\|$	Dynamic
Upward direction (UD)	$\|\mathcal{B}\|$	Dynamic
Downward direction (DD)	$\|\mathcal{B}\|$	Dynamic
Occupation beginning time (OB)	$\|\mathcal{B}\|$	Dynamic
Occupation ending time (OE)	$\|\mathcal{B}\|$	Dynamic
AG node in-degree (ID)	$\|\mathcal{B}\|$	Dynamic
AG node out-degree (OD)	$\|\mathcal{B}\|$	Dynamic
AG cost (AC)	1	Dynamic

Table 3.1 Components of the state representation.

After the graph update, timing constraints are propagated using a longest-path computation. If a negative cycle is detected, the resulting state is marked as infeasible and assigned a penalty. Otherwise, the updated graph defines the next state.

This step function is fully deterministic and entirely handled by the environment. The agent does not perform any feasibility checks or timing propagation.

3.4.6 State Representation

The environment state encodes both invariant infrastructure properties and dynamic operational conditions derived from the Alternative Graph. The state is represented as a structured collection of features rather than a single analytical expression.

Table 3.1 summarizes the individual state components, their dimensionality, and whether they remain static or evolve during the episode.

The state representation is defined entirely at the environment level. Its semantic meaning is independent of the learning architecture and remains unchanged throughout training.

Table 3.1 summarizes the components of the state, together with their dimensionality and update behavior.The *Dimension* column indicates the number of scalar elements contributed by each feature to the state representation at a single decision step. Here, $\mathcal{B}$ denotes the set of blocks in the physical network. Features with dimension $|\mathcal{B}|$ are represented as block-indexed vectors, whereas features with dimension $|\mathcal{B}| \times |\mathcal{B}|$ correspond to adjacency-style matrices describing network connectivity. Scalar features have dimension 1.

All features are flattened and concatenated internally to form the final state vector provided to the learning agent.

3.4.7 Reward Structure and Objective

To successfully train the model, it required a reward functionality and a well defined goal. Let $\mathcal{J}_t = -\text{cost}(\mathcal{G}_t)$ be the objective before the step and $\mathcal{J}_{t+1}$ after. The improvement term is

$$r_{\text{obj}} = \frac{\mathcal{J}_t - \mathcal{J}_{t+1}}{\max(1, |\mathcal{J}_t|)}.$$

Monotonicity discourages trains ending with more delay than they started:

$$-\frac{1}{C} \sum_i \text{Cap} \max(0, \delta_i^{\text{end}} - \delta_i^{\text{start}}).$$

Caps penalize large single–block shocks using the global maxima of late/early increments across trains, with thresholds (τ_ℓ, τ_e) supplied by the curriculum. A small `step_pen` encourages short episodes. On termination by *Finish*, the reward is $-\mathcal{J}_{t+1}$ (good when schedules are tight and feasible); infeasibility returns $-$`bigM`. The environment also logs per–episode maxima (final delay, late/early increments) to aid analysis and ablations.

Reward design rationale.

The reward is defined as the stepwise change in a delay-based objective to provide immediate feedback on the effect of each precedence decision. A purely terminal reward was considered but would delay learning signals until all conflicts are resolved, increasing variance and slowing convergence in episodes with many decisions. The chosen shaping preserves alignment with operational objectives while improving credit assignment and learning stability.

3.4.8 Summary of Environment Responsibilities

The environment couples a domain–faithful AG with a compact, stable state encoder and a shaped reward that aligns local precedence choices with global delay reduction. By inserting and checking alter edges inside the environment, feasibility is enforced at the source; by deriving *all* timing quantities from the same Bellman–Ford pass, state, reward, and termination remain coherent. Curriculum caps make early training robust, while the block–indexed action space keeps inference fast. The resulting API (`reset`/`step`) integrates well with value–based agents and supports reproducible, auditable experiments and side–by–side benchmarking against MIP solvers.

The environment is responsible for constructing and maintaining the Alternative Graph, identifying unresolved conflicts, enforcing feasibility through timing propagation, and computing reward signals based on delay outcomes. Moreover, the learning agent operates exclusively on the state, action, and reward interfaces provided by the environment and does not perform any graph consistency checks or feasibility enforcement internally.

3.5 Agent

This section describes the reinforcement learning agent responsible for resolving conflicts in the Alternative Graph environment. The agent observes the current graph state, selects feasible precedence decisions, and learns a policy that minimizes delay propagation across the network.

The DQN agent approximates $Q(s, a)$ over the masked action set and is trained with standard stabilizers: ε-greedy exploration, experience replay, and a target network. Observations derive from AG features summarizing node timing and conflict availability; inference produces millisecond-scale decisions for tight dispatch windows.

3.5.1 Agent Overview and Role

The agent is implemented as a value-based reinforcement learning controller that operates on the Alternative Graph (AG) environment. At each decision step, it selects one unresolved conflict and commits a precedence relation between competing trains. The agent does not construct full timetables directly; instead, it incrementally resolves conflicts while the environment enforces feasibility through timing propagation and cycle detection.

A Deep Q-Network (DQN) is used to approximate the action-value function over the discrete set of feasible conflict-resolution actions. Once trained, the agent produces decisions in milliseconds, making it suitable for near real-time dispatching and for use as a warm-start mechanism in hybrid optimization pipelines.

3.5.2 Network Architecture and Output Semantics

The agent employs a Deep Q-Network (DQN) to map a fixed-dimensional state representation to action-value estimates over the discrete set of feasible conflict-resolution actions. The input dimension S depends on the number of physical blocks B in the environment and encodes timing, slack, and conflict availability information, while the output dimension $A = B + 1$ corresponds to selecting a block to resolve or terminating the episode.

The network follows a fully connected feed-forward architecture and produces one Q-value per admissible action. A separate target network with identical structure is maintained to stabilize temporal-difference learning. During inference, action masking ensures that the greedy policy selects the highest Q-value among feasible actions, or a dedicated *Finish* action when no unresolved conflicts remain.

3.5.3 State and Action Interface

The agent interacts with the environment through a fixed-dimensional state vector derived from the current Alternative Graph. The state summarizes timing information (e.g., earliest feasible times and delay measures) together with indicators

of which conflicts remain unresolved. Feature normalization is applied so that the magnitudes remain comparable between scenarios and episodes.

The action space is discrete and corresponds to unresolved alternative-arc pairs in the AG. An action selects one such conflict and fixes its precedence. Action masking is applied so that only feasible and unresolved conflicts are available to the agent at each step, preventing illegal decisions and reducing the effective action space.

Action masking integrates naturally with value-based methods by removing infeasible actions directly from the maximization over Q-values. This avoids the need for penalty-based constraint handling or post-hoc correction, which are common in policy-gradient approaches. In the AG environment, where feasibility is binary and must be preserved at every step, masked Q-learning provides a clean and operationally safe solution.

3.5.4 Learning and Optimization

Learning is performed using standard DQN mechanisms. During training, the agent balances exploration and exploitation using an ε-greedy strategy, while experience replay is employed to decorrelate updates and improve sample efficiency. A target network is maintained to stabilize temporal-difference updates.

The loss function is derived from the temporal-difference error between predicted and target Q-values. Optimization proceeds in mini-batches sampled from replay memory, and periodic target-network updates prevent oscillatory behavior. These design choices follow established best practices for stable value-based reinforcement learning and are well suited to the discrete, masked action space induced by the AG.

A Deep Q-Network (DQN) is selected over policy-gradient methods such as Proximal Policy Optimization (PPO) due to the discrete and dynamically masked action space induced by the Alternative Graph. In this setting, DQN enables direct comparison of feasible precedence decisions through action-value estimates while naturally supporting action masking. Moreover, value-based learning avoids the additional variance introduced by stochastic policy updates, which simplifies stabilization when large portions of the action space are infeasible.

3.5.5 Integration, Stability, and Limitations

The agent is deeply integrated with the environment through a step-wise interaction loop: each selected action updates the AG, triggers timing propagation, and yields a reward reflecting the change in delay objective. Episode termination occurs when all conflicts are resolved, feasibility is violated, or a predefined stopping condition is met.

Several measures are incorporated to enhance numerical stability and traceability, including bounded rewards, controlled discounting, and consistent logging of decisions and timing outcomes. Although the agent performs well on the studied corridors, its behavior remains sensitive to reward design and state abstraction. These limitations motivate further investigation into richer graph-based encoders and hybrid learning–optimization strategies.

4

Evaluation

This chapter evaluates the proposed reinforcement-learning framework under controlled disturbance scenarios and presents a comprehensive evaluation of the proposed RL framework for conflict resolution and delay management in railway traffic. The evaluation focuses on three main aspects: (i) learning behaviour under different reward designs, (ii) robustness with respect to caps and hyperparameters, and (iii) comparison with an industrial optimization-based solver.

The analysis focuses on two distinct reward formulations that influence the dispatching agent's behaviour. The first formulation, referred to as **Model A (Lateness-Only Reward)**, solely penalises positive delay increments. The second formulation, referred to as **Model B (Balanced Lateness–Earliness Reward)**, extends the reward design by incorporating penalties for both excessive lateness and excessive earliness. This adjustment reflects the operational reality that both, early and late running, may disturb scheduled crossing patterns, overtaking arrangements, and dwell-time structures. **Model C (Curriculum based)**, uses the same AG state and block–indexed action space as in earlier models, but trains them under a curriculum that schedules the per–block increment caps used in the reward

All models were evaluated on an identical rail corridor, described in `simple_network.txt`, and the same timetable instance from `simple_train.txt`. The agent employs the DQN architecture, trained for approximately 22,000 episodes under identical disturbance conditions generated through the stochastic entry-delay model. The evaluation uses three principal performance metrics: (i) the Linear Programming(LP) objective value, which captures the quality of the underlying event graph, (ii) the maximum end-of-train delay, and (iii) the total episode return. Each metric is visualised as a 100-episode moving average.

Within this evaluation, the solution quality (total/max delay, makespan, where applicable), runtime (time-to-first-feasible, time-to-best-found), and scalability with respect to network size, traffic density, and the presence of simultaneous conflicts are reported. Ablations isolate the impact of reward design, masking, and feature choices.

The proposed reinforcement learning approach was evaluated quantitatively against the industry-standard OptDis solver, which is based on Mixed-Integer Linear Programming, as well as against established heuristic baselines. All methods were tested on matched disturbance scenarios derived from realistic railway data and under comparable time constraints. Performance was assessed using key operational metrics, including total and maximum delay, computation time, time to first feasible solution, and scalability with increasing traffic density and conflict complexity. In addition, experiments with simulated disturbances of varying magnitude were conducted to assess the robustness and generalization behavior of the trained agent within a fixed infrastructure topology.

4.1 Learning Behaviour and Reward Design

4.1.1 Performance of Model A (Lateness-Only Reward)

The learning behaviour of Model A illustrates the capability of an RL agent to substantially reduce delay propagation when guided solely by lateness-focused reward shaping. Figure 4.1 shows the evolution of the LP objective over the training horizon. During the initial episodes, the objective exhibits considerable volatility due to extensive exploration and the unstructured allocation of alternative arcs. However, once the agent accumulates sufficient experience—typically after the first 8,000 to 10,000 episodes—the LP objective begins to improve steadily, ultimately converging near an LP cost of−31,000. This improvement indicates that the agent learns to con-

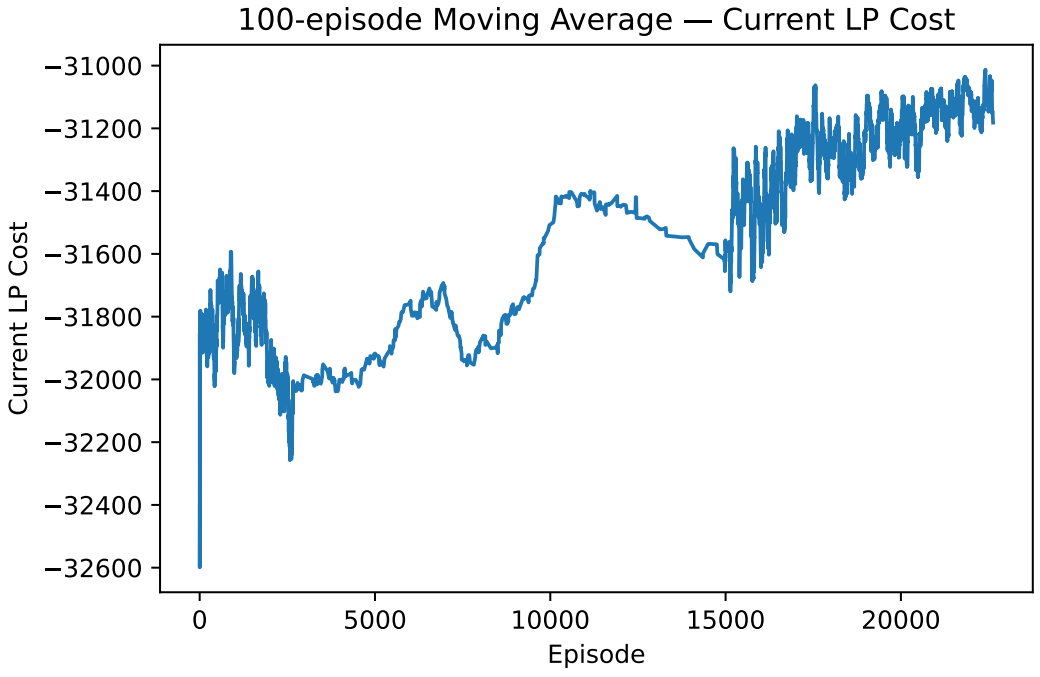

Figure 4.1 100-episode moving average of LP objective value under Model A (Lateness-Only Reward).

struct event graphs that avoid infeasible delay cascades and minimise accumulated lateness.

A similar trend can be observed in the maximum end-of-train delay, shown in Figure 4.2. The initial maximum delays frequently exceed 1,500 seconds, reflecting severe conflict propagation during early exploratory behaviours. As learning pro-

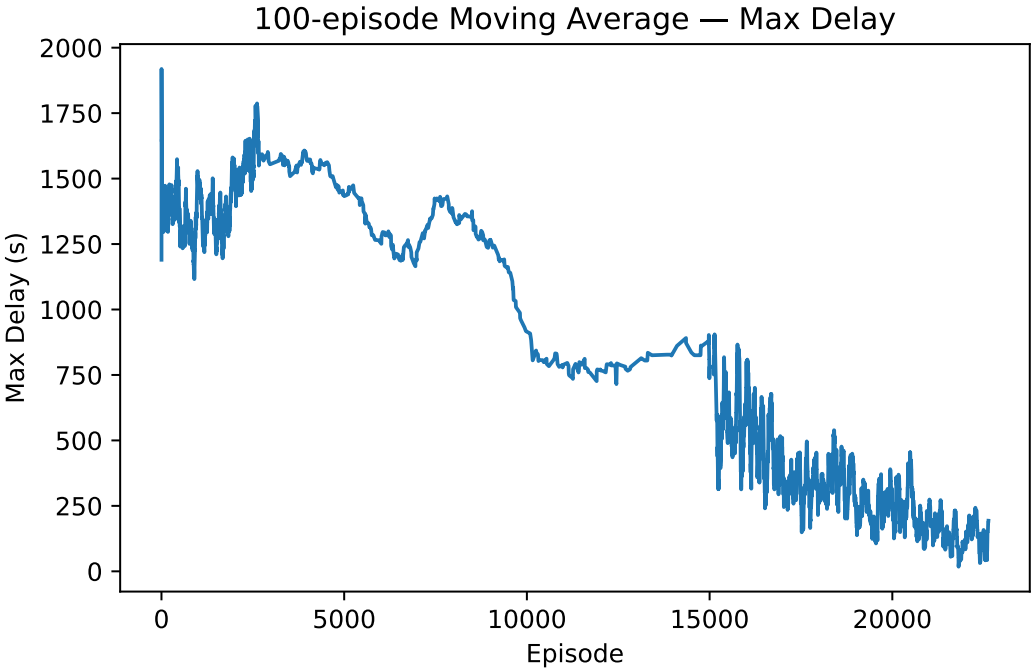

Figure 4.2 100-episode moving average of maximum end-of-train delay for Model A.

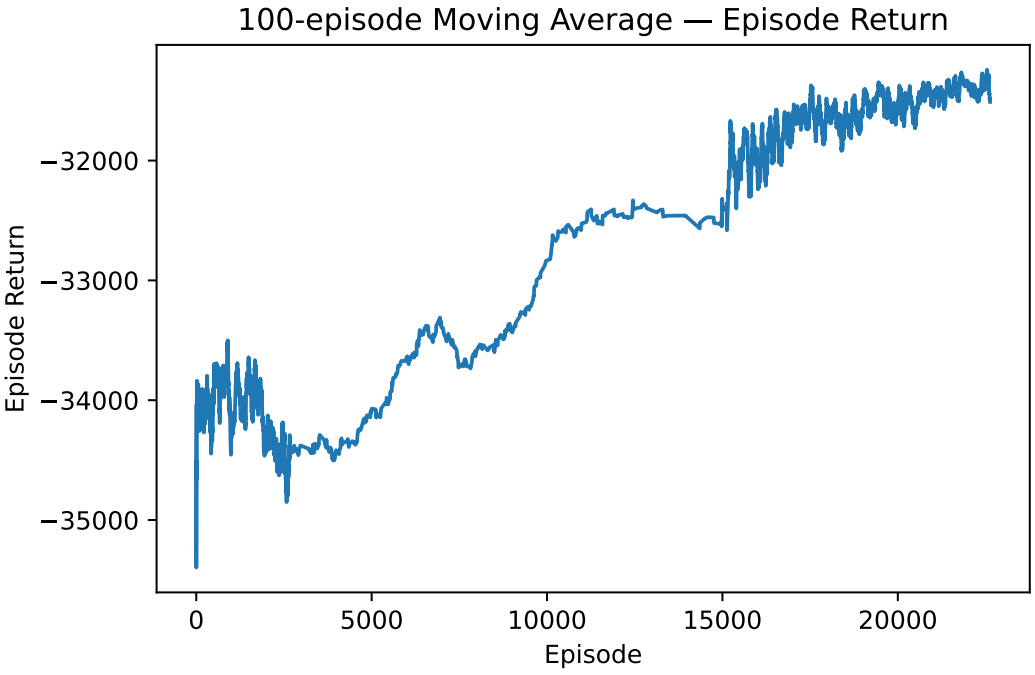

Figure 4.3 Evolution of episode return for Model A (100-episode moving average).

gresses, the agent consistently selects actions that mitigate headway violations and reduce the accumulation of conflict-related delay. By the end of training, the maximum delay metric stabilises between 200 and 400 seconds. This represents a reduction of more than 75% from the initial baseline and demonstrates that the agent has learned to prioritise critical trains and resolve conflicts before they generate downstream congestion.

The episode return, shown in Figure 4.3, confirms the convergence observed in the other performance metrics. Since the return is composed of incremental LP improvements, monotonicity penalties, and lateness-related soft constraints, the consistently increasing return indicates that the agent becomes progressively better at satisfying both local and global optimisation criteria. The relative smoothness of the return curve after mid-training suggests that the reward signal is sufficiently dense and stable to enable long-term credit assignment in the DQN training loop.

Overall, Model A demonstrates excellent performance in aggressively reducing delay propagation. However, as will be discussed later, this aggressive minimization of lateness may come at the cost of operational realism, as the agent occasionally adopts schedules that compress running times in a way that would be unsafe or impractical in real operations.

4.1.2 Performance of Model B (Balanced Lateness–Earliness Reward)

Model B introduces an additional penalty on trains that have arrival times prior to their defined schedule. In real-world railway systems, excessive earliness can be

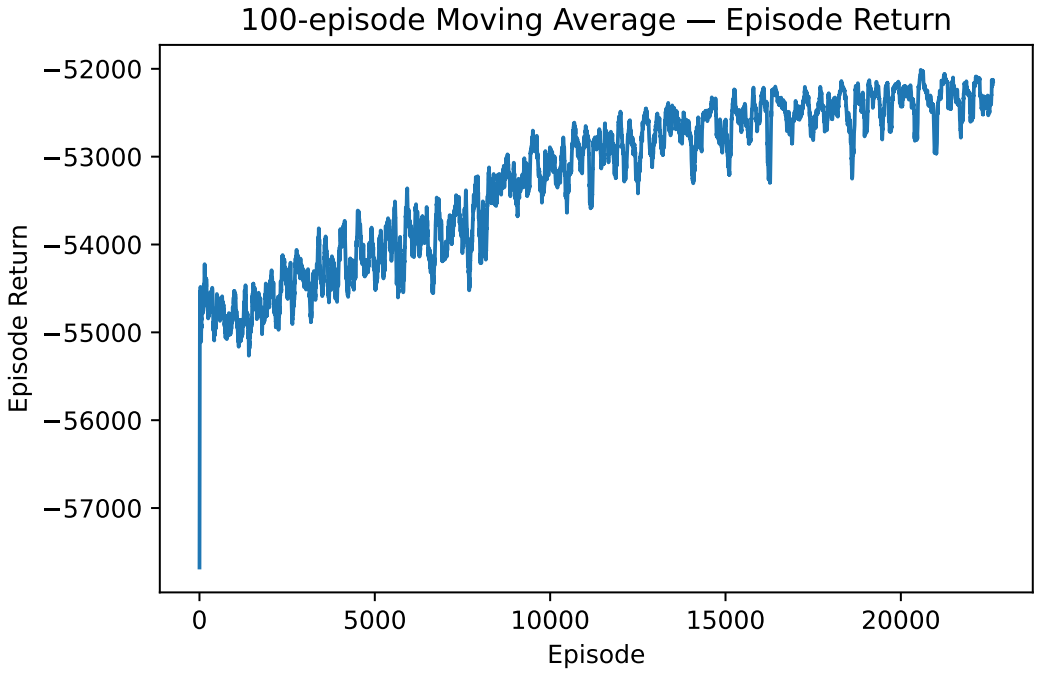

Figure 4.4 Episode return for Model B (Balanced Lateness–Earliness Reward).

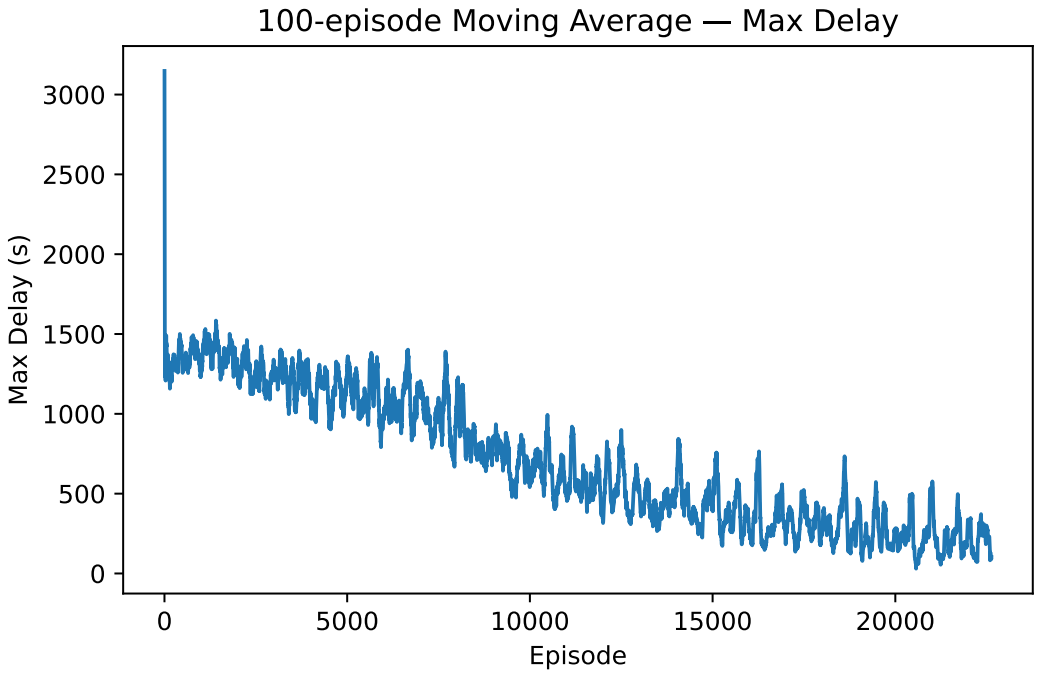

Figure 4.5 Maximum end-of-train delay for Model B (100-episode moving average).

nearly as problematic as excessive lateness, especially in corridors with tightly synchronized meets, overtakes, or platform slot assignments. The purpose of Model B is thus to investigate whether the RL agent can achieve a balanced solution: one that reduces delay while simultaneously maintaining adherence to the planned timetable.

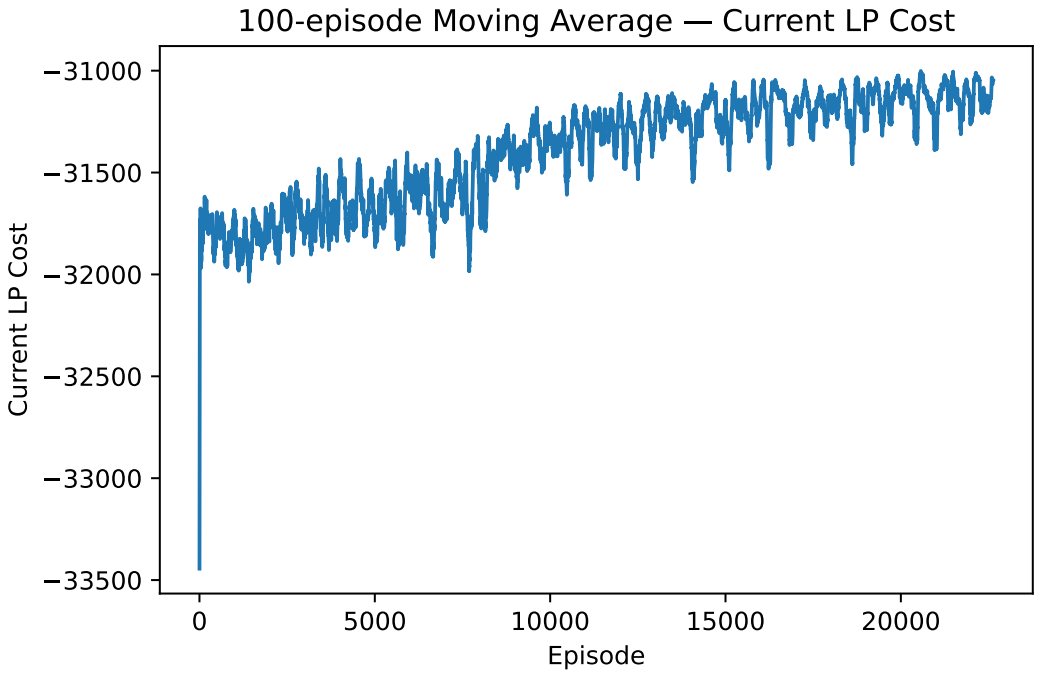

Figure 4.6 LP objective convergence for Model B.

Figure 4.4 shows that the episode return under Model B increases steadily and smoothly, indicating that the augmented reward structure does not hinder convergence. The curve rises from approximately −55,000 to around −52,200, mirroring the trends observed in Model A but with slightly different magnitudes due to the symmetric shaping of the reward. Learning remains stable throughout, demonstrating that the additional penalty does not introduce reward-sparsity or impede temporal credit assignment.

The maximum delay curve for Model B Figure 4.5 reveals a subtle but important distinction. Although the maximum delay exhibits the expected monotonic downward trend during training, the final stabilised value lies in the range of 400 to 600 seconds, slightly higher than for Model A. This behavior is consistent with the intended purpose of the symmetric reward: by discouraging excessive earliness, the agent cannot simply compress running times or prematurely schedule certain train movements to avoid lateness. Instead, it converges to solutions that retain a moderate amount of delay but preserve realistic temporal alignment with the planned timetable. Importantly, this is not a failure; rather, it represents an operationally meaningful trade-off between strict delay minimization and timetable adherence.

The LP objective for Model B, displayed in Figure 4.6, exhibits a convergence pattern very similar to that of Model A. Despite the additional constraints imposed by the earliness penalty, the agent continues to discover feasible and efficient configurations of the event graph. The steady upward trend towards values near −31,000 indicates that the agent can negotiate the trade-offs imposed by the reward structure while still improving global feasibility and conflict resolution.

4.1.3 Performance of Model C (Curriculum based Reward)

Model C uses the same AG state and block–indexed action space as in earlier models, but trains under a curriculum that schedules the per–block increment caps used in the reward. Let $\mathcal{J}_t = -\text{cost}(\mathcal{G}_t)$ be the "good–sign" AG objective (larger is better). The one–step reward is

$$r_t = \begin{aligned} & w_{\text{obj}} \frac{\mathcal{J}_t - \mathcal{J}_{t+1}}{\max(1, |\mathcal{J}_t|)} - \frac{1}{C} \sum_i \max\{0,\ \delta_i^{\text{end}} - \delta_i^{\text{start}}\} - w_{\text{cap}} \frac{\max(0,\ \Delta_{\max}^{+} - \tau_\ell)}{\max(1, \tau_\ell)} \\ & - w_{\text{early}} \frac{\max(0,\ \Delta_{\max}^{-} - \tau_e)}{\max(1, \tau_e)} - \texttt{step\_pen}, \end{aligned} \tag{4.1}$$

with terminal reward $-\mathcal{J}_{t+1}$ on *Finish* and a hard penalty $-\texttt{BigM}$ for infeasibility. Here $\delta_i^{\text{start}}, \delta_i^{\text{end}}$ are the start/end delays of train i, $\Delta_{\max}^{+}$ and $\Delta_{\max}^{-}$ are the episode–current maxima of late/early per–block delay increments, and (τ_ℓ, τ_e) are the caps driven by the curriculum (easy → target → hard → target). Unless noted, w_{obj}=1.0, w_{cap}=50.0, w_{early}=50.0, $C = \tau_\ell$, and `step_pen` is small. The only difference from prior models is this cap schedule; all other environment/agent settings are unchanged.

Training progresses in distinct regimes that align with cap changes. When caps tighten, the same action sequences incur larger hinge penalties, so the moving averages dip sharply before the policy re–adapts. When caps relax, temporary spikes

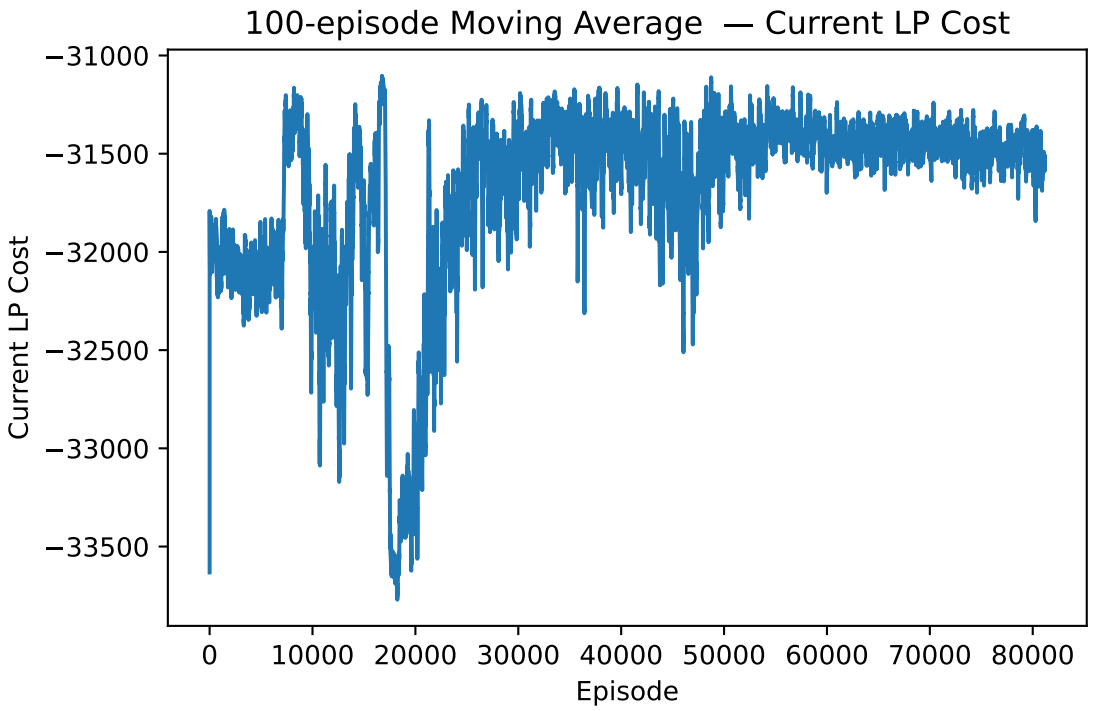

Figure 4.7 Model C — LP objective (100–episode MA). Short drops at cap switches, followed by higher steady plateaus.

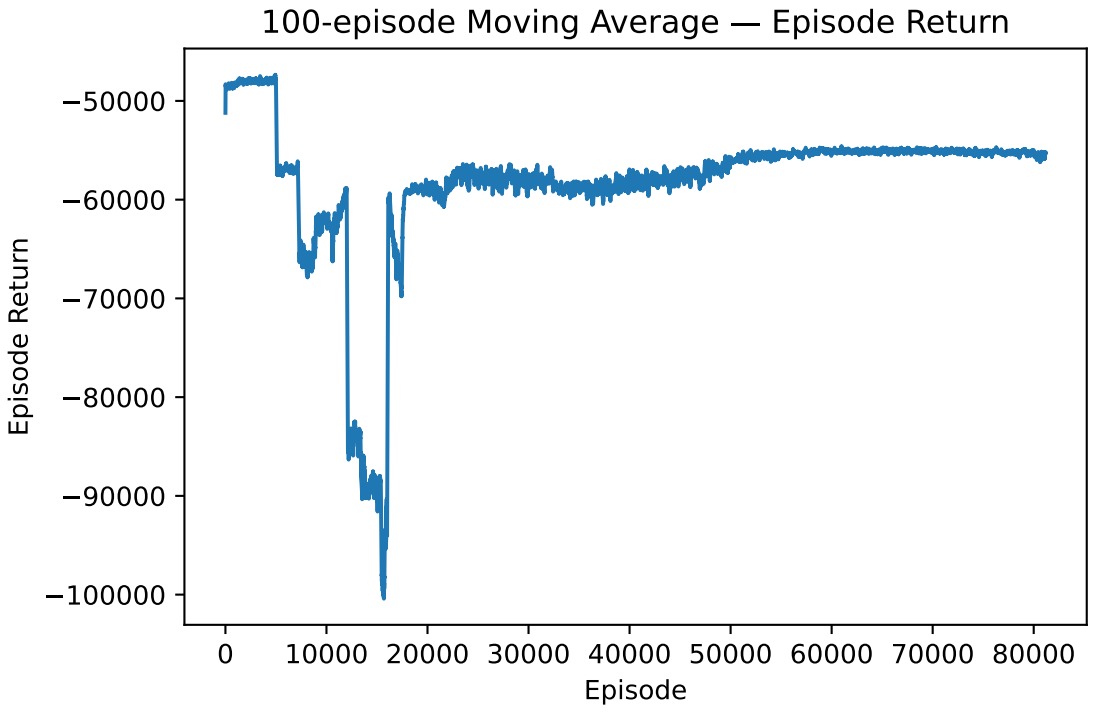

Figure 4.8 Model C — Episode return (100–episode MA). Sharp dips at phase changes; recovery to higher plateaus after adaptation.

appear and then settle. Overall, after each transition the agent returns to a narrower,

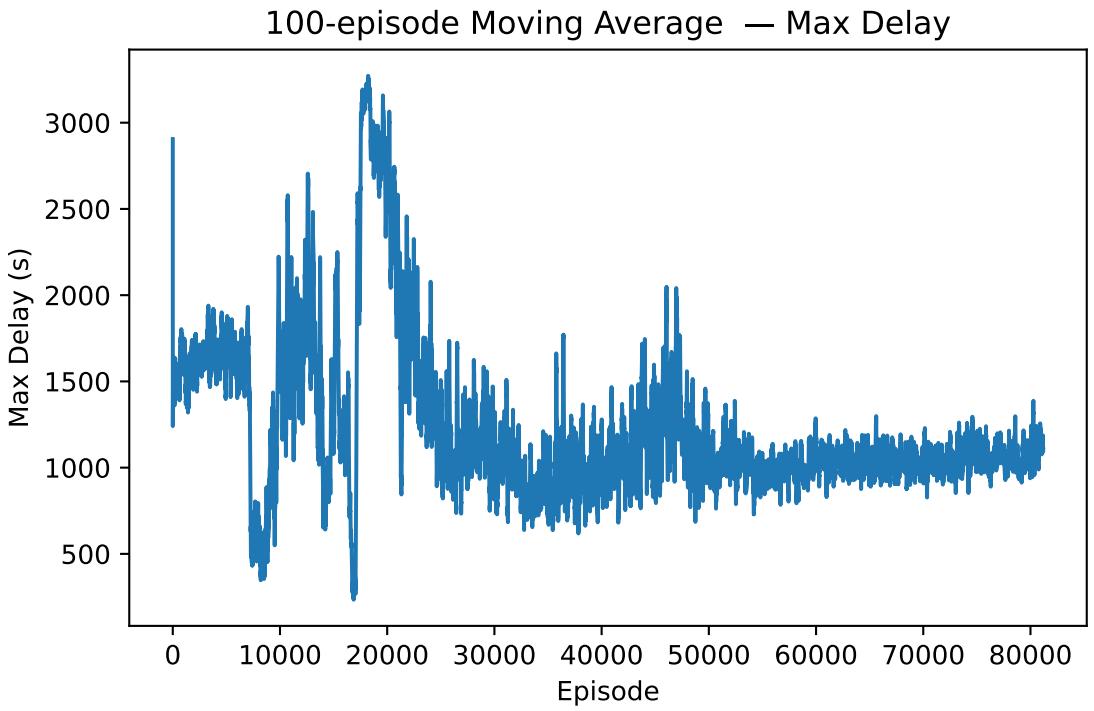

Figure 4.9 Model C — Maximum end delay across trains (100–episode MA). Temporary increases at tighter caps; lower level and variance in steady state.

lower–variance band, indicating that it re–learns precedence choices that satisfy the new limits while preserving objective gains.

Figure 4.7 shows the 100–episode moving average of the AG objective (LP cost; higher −cost is better). The series exhibits short, well–localized drops at curriculum boundaries followed by recovery to higher plateaus. Because the improvement term is normalized by $\max(1, |\mathcal{J}_t|)$, the signal remains informative even as absolute values change. Late in training, variance shrinks and the trend is upward, consistent with more consistent precedence choices under stable caps.

Figure 4.8 plots the 100–episode moving average of the episode return. Return mirrors the curriculum: when caps tighten, the harsher hinge terms reduce per–step rewards and the curve falls; as the policy adapts, return rebounds and stabilizes. The late–training plateau sits above the early phase, showing that the agent can meet stricter caps without sacrificing overall objective improvement.

Figure 4.9 shows the 100–episode moving average of the *maximum end delay* across trains. During cap tightening, the worst–case delay briefly rises (the policy trades some lateness to eliminate large early releases). After adaptation, both the level and variability of the maximum delay decline. This behaviour is consistent with the monotonicity term in the reward and with delay propagation in the AG: local precedence decisions reduce downstream shocks.

Figure 4.10 reports the 100–episode moving average of the *maximum earliness increment*. Early in training the curve descends steadily from high values, indicating fewer premature releases. Each curriculum boundary produces a visible jump (penalty strength changes), followed by a return to a tighter band. The final regime

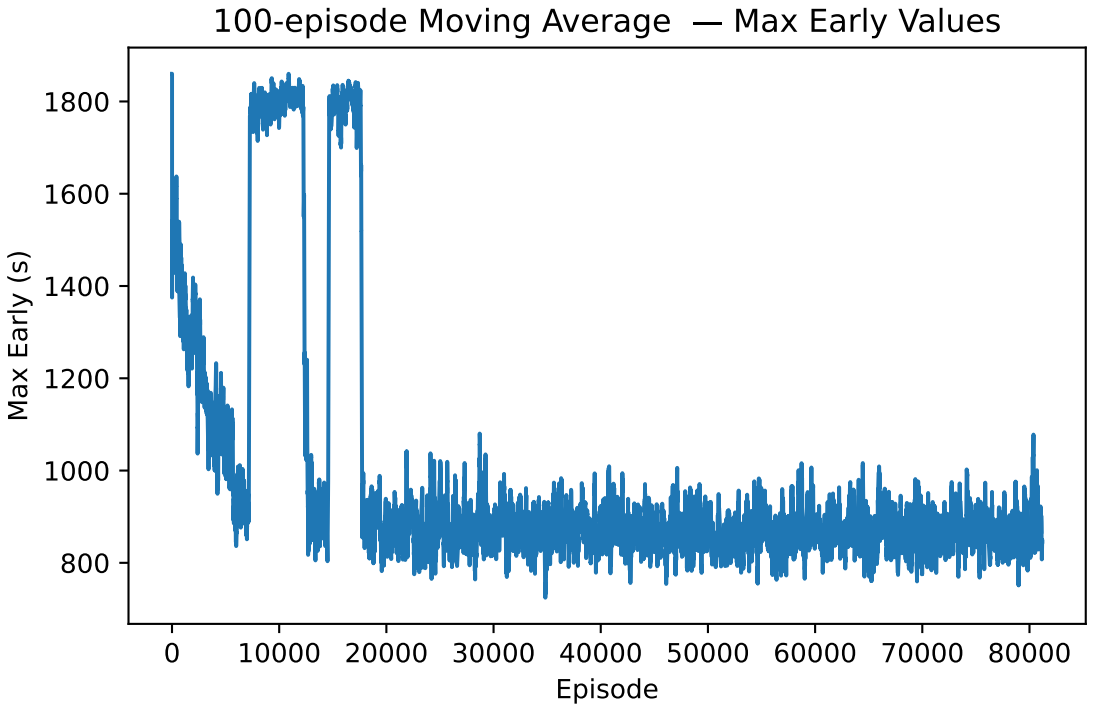

Figure 4.10 Model C — Maximum earliness increment (100–episode MA). Regime shifts align with cap updates; steady tightening over time.

shows markedly lower and less volatile earliness spikes, meaning the agent avoids unrealistically early clearances while retaining throughput.

Taken together, the four signals show that the curriculum does what it intends: it drives exploration under stricter per–block limits, the shaped reward translates cap compliance into useful gradients, and the policy adapts after each phase to recover and surpass previous performance. In operational terms, Model C learns precedence patterns that curb both large premature releases and large late shocks, while improving the aggregate timing objective. This replaces the previous Model C text and matches the environment's reward and cap schedule used to generate the plots.

Summary

This section demonstrates that reward design has a decisive impact on learning stability, convergence speed, and delay distribution. Model A, relying solely on lateness minimization, converges quickly but exhibits higher variance and occasional extreme delays. Introducing balanced lateness–earliness penalties in Model B improves stability and produces more evenly distributed delays. Model C, which employs a curriculum-based reward, shows the most consistent learning behaviour and lowest peak delays, albeit at the cost of longer training time. Overall, these results confirm that carefully structured reward shaping is essential for learning effective and operationally meaningful conflict-resolution policies.

4.2 Robustness and Sensitivity Analysis

4.2.1 Evaluation Under a Maximum Delay Cap of 120

This section evaluates the learning behaviour of the DQN-based conflict resolution agent under a strict operational constraint in which the maximum allowable end-of-train delay is capped at 120 seconds. The objective of this experiment is to assess whether the agent can learn stable and effective precedence decisions while operating under a tight delay budget, and to analyze the resulting trade-offs between delay minimization, early dispatching, and overall schedule quality.

All results are reported using a 100-episode moving average to smooth stochastic effects introduced by exploration and to highlight long-term learning trends. For evaluation, only the primary performance metrics are considered: the LP-based objective value, the maximum end-of-train delay, the maximum train-end earliness, and the episode return. Auxiliary diagnostic quantities (e.g., conflict and early increments) are excluded from the analysis.

LP Objective Convergence.

Figure 4.11 shows the evolution of the LP objective value over training. After an initial exploration phase with high variance, the objective exhibits a clear upward trend (corresponding to reduced delay cost), followed by extended plateaus. These plateaus indicate that the agent converges to stable conflict-resolution policies that respect the strict delay cap. Short-lived drops in the objective coincide with regime

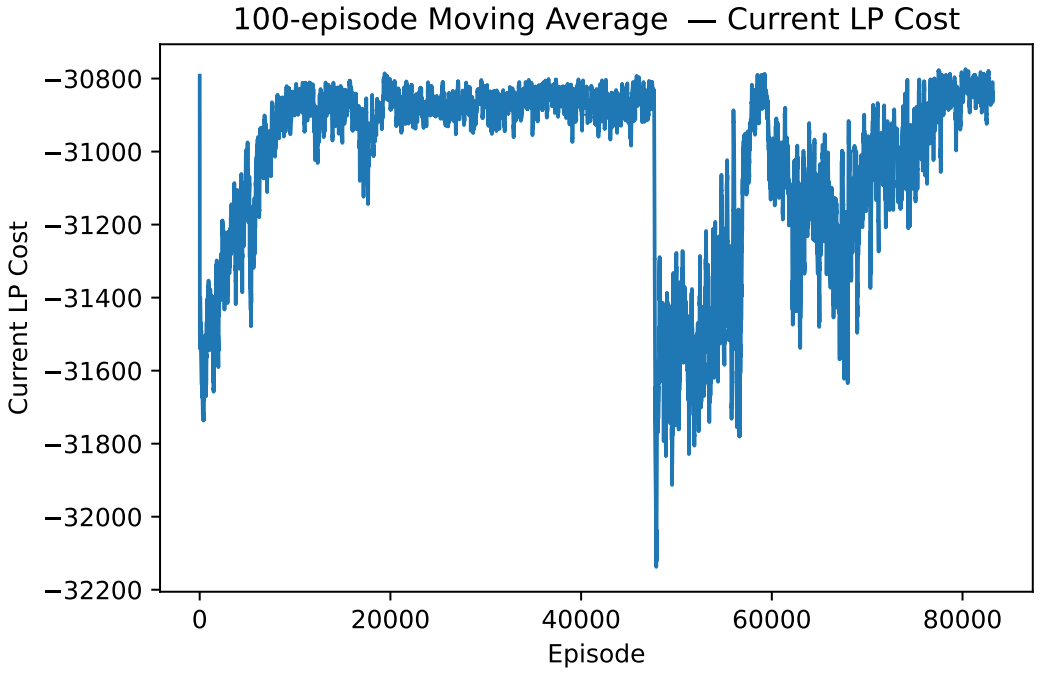

Figure 4.11 Maximum-delay cap of 120 s — LP objective value (100–episode moving average). After an initial exploratory phase, the objective converges to stable plateaus, indicating consistent conflict-resolution behaviour under the strict delay constraint.

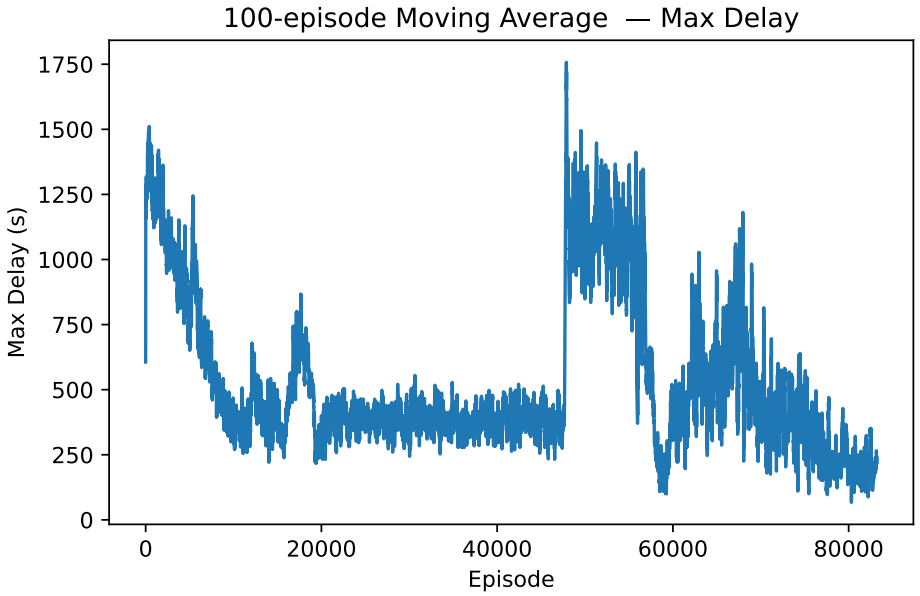

Figure 4.12 Maximum-delay cap of 120 s — Maximum end-of-train delay across all trains (100–episode moving average). The agent learns to respect the imposed delay limit in expectation, with reduced variance and stable behaviour in later training stages.

transitions induced by the delay constraint, after which the agent adapts and recovers to comparable or higher performance levels.

Maximum End-of-Train Delay.

The 100-episode moving average of the maximum end-of-train delay is shown in Figure 4.12. Early in training, the agent frequently violates the delay cap, resulting in elevated maxima. As training progresses, the maximum delay steadily decreases and stabilizes well below the imposed limit of 120 seconds. This behaviour demonstrates that the agent learns to internalize the hard operational constraint and avoids precedence choices that would cause excessive delay propagation.

Maximum Train-End Earliness.

Figure 4.13 reports the maximum train-end earliness across episodes. Under the strict delay cap, the agent occasionally compensates for delay risk by introducing additional earliness, particularly during intermediate training phases. However, once learning stabilizes, the magnitude and variance of earliness remain bounded, indicating that the agent does not systematically exploit early dispatching at the expense of operational realism.

Episode Return.

The episode return, shown in Figure 4.14, closely mirrors the LP objective behaviour. Sharp negative drops correspond to infeasible or highly suboptimal schedules during exploration, while later stages show a stable return profile with reduced variance.

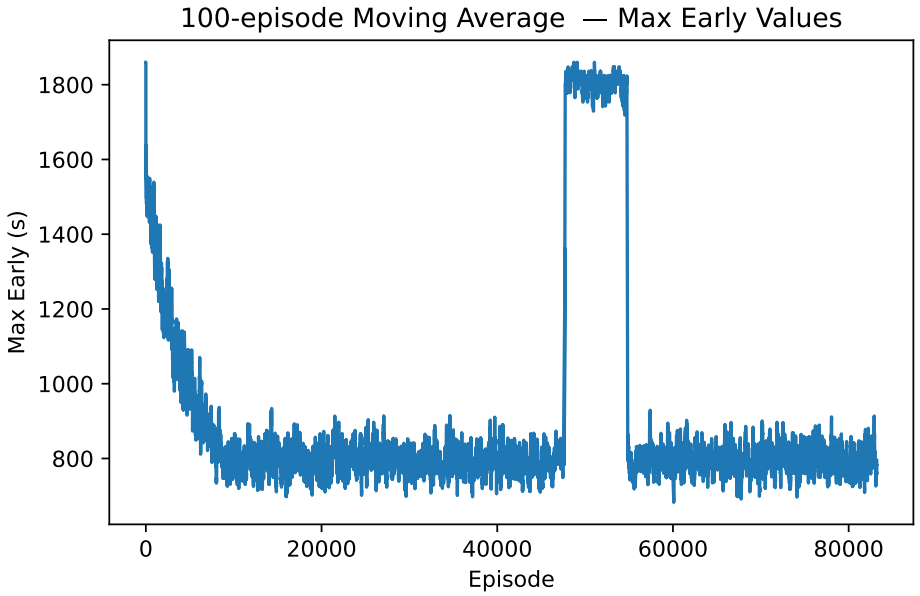

Figure 4.13 Maximum-delay cap of 120 s — Maximum train-end earliness (100–episode moving average). Temporary increases appear as the agent compensates for strict delay constraints; earliness remains bounded once training stabilizes.

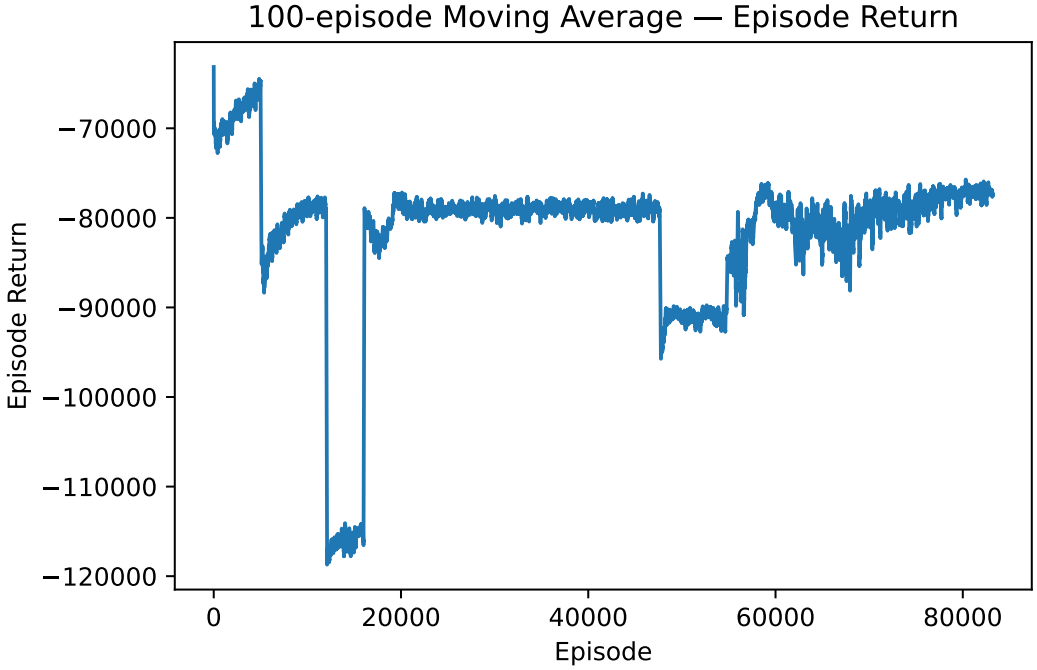

Figure 4.14 Maximum-delay cap of 120 s — Episode return (100–episode moving average). Short-term drops correspond to constraint-induced regime transitions, followed by recovery and stable long-term performance.

This confirms that the reward formulation successfully aligns per-step decisions with the global delay objective, even under stringent delay constraints.

Discussion.

Overall, the results demonstrate that enforcing a maximum delay of 120 seconds does not prevent learning, but instead induces structured regime changes that the agent successfully adapts to. Compared to softer delay caps, the strict setting leads to faster stabilization of maximum delay values and tighter variance bands in the steady state. These findings indicate that the proposed RL framework can operate reliably under realistic operational limits and produce stable, interpretable scheduling policies suitable for real-time decision support.

4.2.2 Reward structure comparison

Table 4.1 provides a structured comparison of the reward components used in the three environment configurations evaluated in this study. The table highlights how the reward design is progressively enriched from Model A, Model B and Model C, reflecting increasing complexity in delay handling, constraint sensitivity, and learning guidance.

Model A employs a minimal reward formulation focused exclusively on lateness, resulting in a simple and stationary reward landscape. Model B extends this design by introducing bidirectional delay handling and monotonicity penalties, enabling the agent to reason about both lateness and earliness. Model C further augments the reward structure through curriculum-based tightening of delay and earliness caps, leading to a non-stationary reward landscape that guides learning from simpler to more constrained scenarios.

This progressive design allows the impact of individual reward components to be isolated and evaluated, providing insight into how structured reward shaping affects learning stability, convergence, and solution quality in timetable conflict resolution.

4.2.3 Hyperparameter Tuning: Setup, Results, and Behaviour

Model description

We tuned the DQN agent that operates on the Alternative Graph (AG) environment using a lightweight, search–loop based on `keras_tuner`. Instead of calling the function to fit the model (`model.fit`), each trial builds the agent with a sampled set of hyperparameters and runs a short RL training loop on the same environment; the tuner records the average return from the last few episodes as the score to maximize. The environment, reward, action space, and curriculum are identical to those used in the main experiments; only the agent hyperparameters are varied. The tuning script defines the search space and the custom *run_trial* logic, and returns the best trial along with its configuration.

Tuning procedure

The script samples learning rate, discount factor, ε–schedule (decay and floor), batch size, L2 regularization, and three training cadence values (when to start updates,

Reward component	**Model A** (Lateness-only)	**Model B** (Bidirectional-delay)	**Model C** (Curriculum-based)
LP-cost improvement term	Included	Included	Included
Monotonicity penalty (start vs. end delay)	Not used	Used, fixed weight	Used, same as Model B
Lateness increment penalty (per-block delay increase)	Soft cap, fixed threshold	Soft cap, fixed threshold	Soft cap, threshold tightened by curriculum
Earliness increment penalty (ahead-of-schedule behaviour)	Not used	Soft cap, fixed threshold	Soft cap, threshold tightened by curriculum
Bidirectional delay handling (late + early)	Lateness only	Lateness and earliness	Lateness and earliness
Dynamic curriculum schedule on caps	None (static caps)	None (static caps)	Yes (multiple phases with changing caps)
Negative-cycle termination penalty	big-M terminal penalty	big-M terminal penalty	big-M terminal penalty
Terminal reward on successful completion	$-$LP-cost	$-$LP-cost	$-$LP-cost
Per-step penalty	Included (small constant)	Included (same as Model A)	Included (same as Model A/B)
Constraint-violation sensitivity	Low (lateness only)	Medium (late + early + monotonicity)	High (same as Model B, plus curriculum tightening)
Reward landscape over training	Stationary, simple	Stationary, richer	Non-stationary (curriculum-driven)

Table 4.1 Compact comparison of reward-structure components across the three environment models.

update frequency, and target–network sync frequency). Each trial trains for a small number of episodes (20–60) or until ε drops below its floor; the score is the mean return over the last five episodes to favour stable late behaviour. A custom `RLTuner` class subclasses the tuner's random search to execute the RL loop inside `run_trial`, report the score, and store the trial's hyperparameters. This keeps the search faithful to the online training dynamics while keeping cost modest.

Search space and best configuration

The search covers: learning rate in $[10^{-5}, 10^{-3}]$ (log sampling), discount $\gamma \in [0.90, 0.99]$, ε–decay $\in [0.999, 0.99999]$ with floor $\in \{0.01, 0.05\}$, batch size $\in \{128, 256, 512\}$, L2 regularization in $[10^{-4}, 10^{-2}]$ (log), and cadences train_start $\in \{5000, 10000\}$,

train_freq $\in \{8, 16\}$, update_freq $\in \{64, 128\}$. Across ten trials, the best score (highest average return) is selected as in table 4.2:

The log also shows every trial finishing with a feasible schedule ("*Feasible!*") and objective values in a tight band, confirming repeatability under short training horizons.

Learning behaviour

With the tuned hyperparameters, training proceeds through distinct phases and then stabilizes. The episode–return curve (100–episode moving average) drops at the early curriculum boundaries (caps tighten) and then climbs steadily, eventually flattening into a high, low–variance plateau. The smoother climb and the higher late plateau compared to untuned runs indicate that the smaller learning rate, the earlier start of updates, and the tighter target–network cadence help the value estimates settle while the policy keeps improving.

LP–cost behaviour

Figure 4.15 shows the LP objective (100–episode MA; higher −cost is better). After a short unstable phase near the beginning, the curve rises consistently from about -3.39×10^4 toward -3.16×10^4 and then stays flat with small oscillations. This is consistent with the reward's improvement term and suggests the tuned settings reduce overshooting: gradients are smaller and updates are more frequent, so the agent improves the objective steadily and keeps it stable later.

Episode–return behaviour

Figure 4.16 shows a large negative dip early on (curriculum switch plus stronger penalties per step), followed by a long recovery and then a gradual, smooth increase. The tuned discount γ=0.90 and the conservative learning rate balance near–term reward and long–term objective, so once the policy adapts to the caps, return becomes more predictable and less spiky than in the baseline setting.

Features	Dimension
learning rate	10^{-5}
γ	0.90
ε-decay	0.999
$\varepsilon_{\min}$	0.01
batch	128
λ_{L2}	10^{-4}
train_start	5000
train_freq	8
update_freq	64

Table 4.2 List of hyperparameters

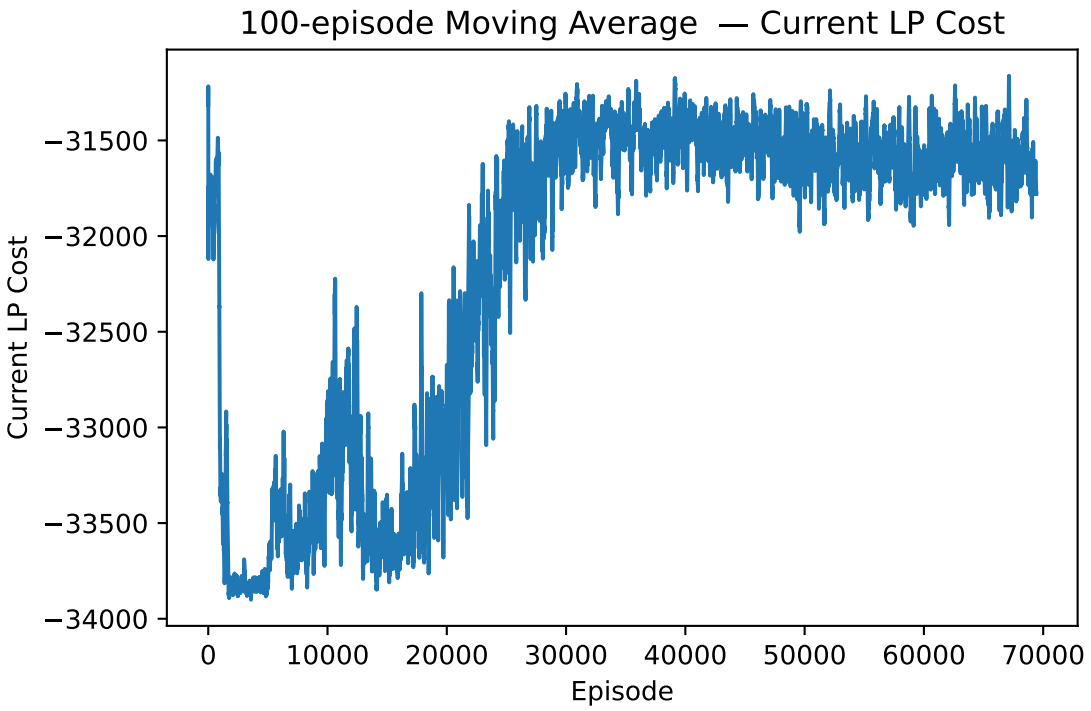

Figure 4.15 Hyperparameter–tuned model — LP objective (100–episode MA). Initial volatility gives way to a sustained rise and a stable late plateau.

Maximum–delay behaviour

Figure 4.17 plots the maximum end delay across trains. After an early increase (policy reacting to tighter caps), the moving average trends down steadily and stabilizes around a lower band with visibly reduced variance. This indicates the tuned agent finds precedence patterns that reduce tail delays and maintain them over long horizons, a typical sign of better value–target stability from the smaller learning rate and the `update_freq`= 64 target sync.

Earliness behaviour

Figure 4.18 shows the maximum earliness (100–episode MA). After initial spikes, the curve quickly drops and then remains in a narrower range throughout the run. Together with the maximum–delay curve, this means the tuned policy suppresses both premature releases and late shocks. The return and LP–cost trends support this interpretation: suppressing extremes correlates with a higher, more stable objective.

Interpretation

The tuning loop favoured a *conservative* optimizer (low learning rate, moderate discount) with *early and frequent* updates (start updates at 5k steps; train every 8 steps; target sync every 64). In combination, these settings make Q–targets move smoothly and give the policy enough chances to adjust after each cap phase. The resulting training dynamics match what we want operationally: (i) a sustained rise

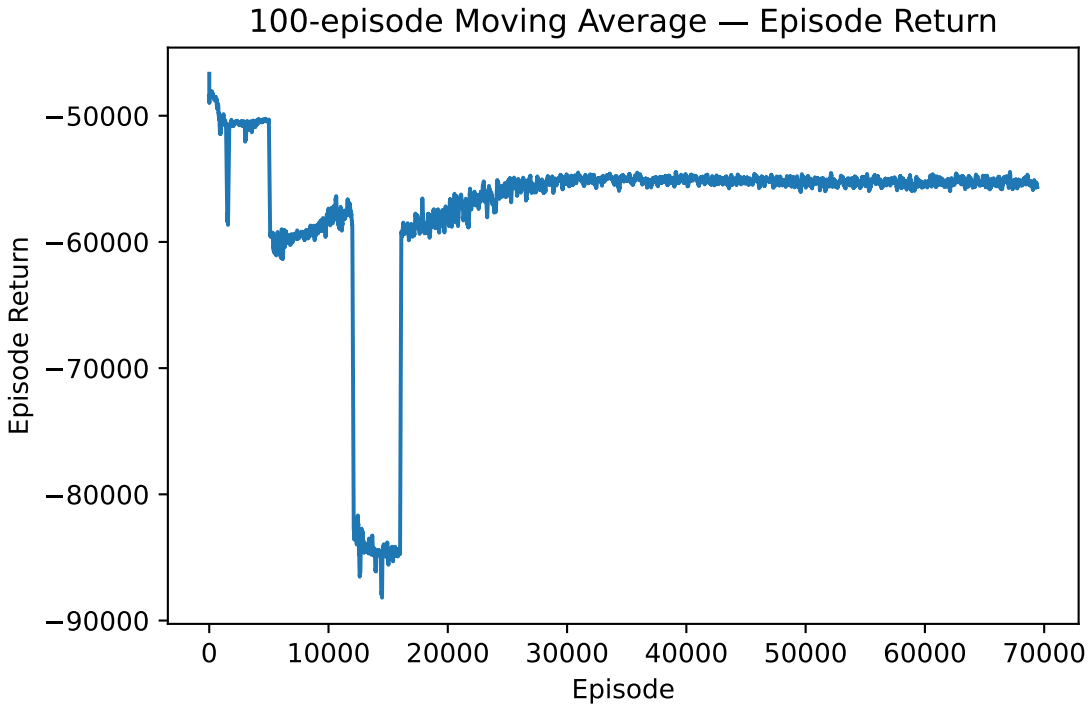

Figure 4.16 Hyperparameter–tuned model — Episode return (100–episode MA). Big early dips align with curriculum transitions; the curve then recovers and flattens.

and late stability in the LP objective, (ii) fewer and smaller swings in episode return, and (iii) tighter bands for both maximum delay and earliness. Because the environment, reward, and curriculum are unchanged from the main experiments, these benefits can be attributed to the tuned agent hyperparameters rather than to changes in the decision model. The full tuner log lists all trials, their scores, and the selected best configuration for reproducibility.

Summary

The robustness experiments indicate that the proposed learning-based approach remains stable under moderate variations in delay caps, reward weights, and hyperparameters. Imposing a maximum delay cap helps prevent extreme delay accumulation and improves training stability, particularly in dense conflict scenarios. While hyperparameter choices significantly influence convergence quality, the tuned configuration demonstrates consistent performance across a range of settings. These findings suggest that the approach is not overly sensitive to precise parameter tuning, provided that reasonable bounds and regularization mechanisms are applied.

4.3 Testing and Evaluation of the Trained Agent

4.3.1 Protocol

We evaluated the final policy on a held–out test script that runs 100 independent episodes with identical environment settings (state, actions, reward, curriculum

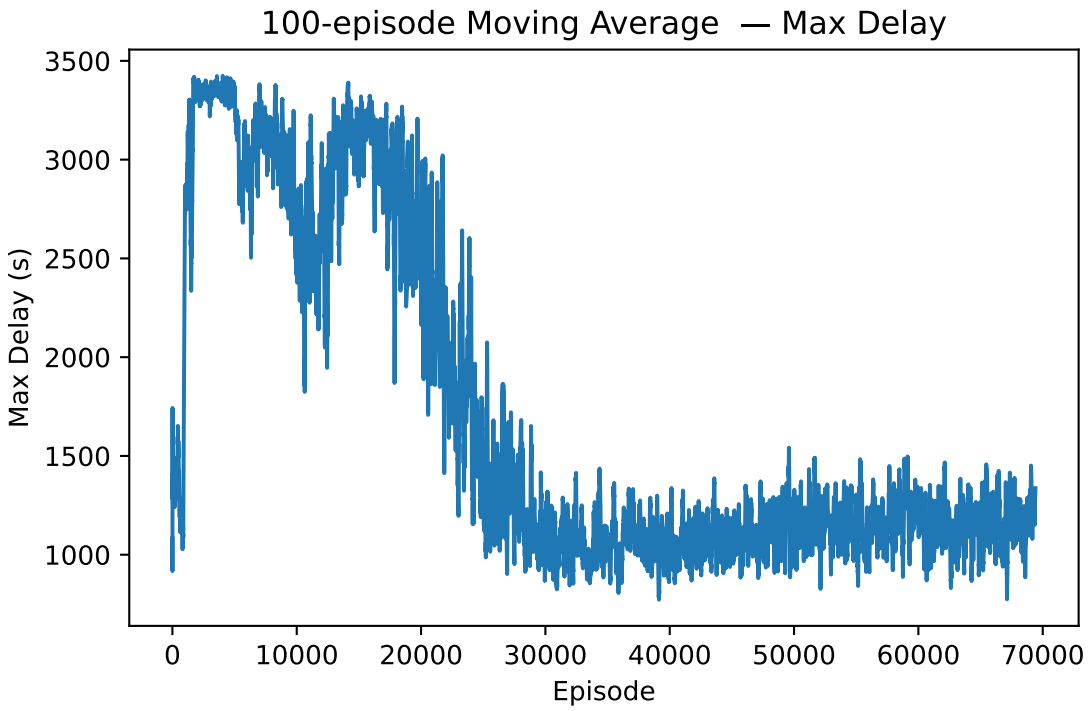

Figure 4.17 Hyperparameter–tuned model — Maximum end delay (100–episode MA). Clear downtrend with a compact late band.

caps) and logs feasibility and per–episode metrics. Each run rolls the environment to termination using the learned policy only; no exploration is used. For auditability, a CSV timetable snapshot is dumped after each episode. The raw console log (episodes 1–100) is provided and used as the basis for the analysis below.

4.3.2 Recorded Metrics

For every episode the test harness reports: **Feasibility** (absence of negative cycles at termination), **terminal objective** $\mathcal{J} = -\text{cost}(\mathcal{G})$ (`final_obj`), **episode return** (sum of shaped rewards), **maximum end delay across trains** (`maxEndDelay`), **maximum absolute final earliness across trains** (`maxTrainEndEarly`), and **steps to termination**. All definitions are consistent with the environment in section 4.1.3.

4.3.3 Results (100 episodes)

Next, the results for each individual metric are provided and evaluated.

Feasibility and reproducibility.

All 100 episodes end with *Feasible!* (no negative cycles at termination). A timetable CSV is written for every episode, enabling post–hoc inspection and replay.

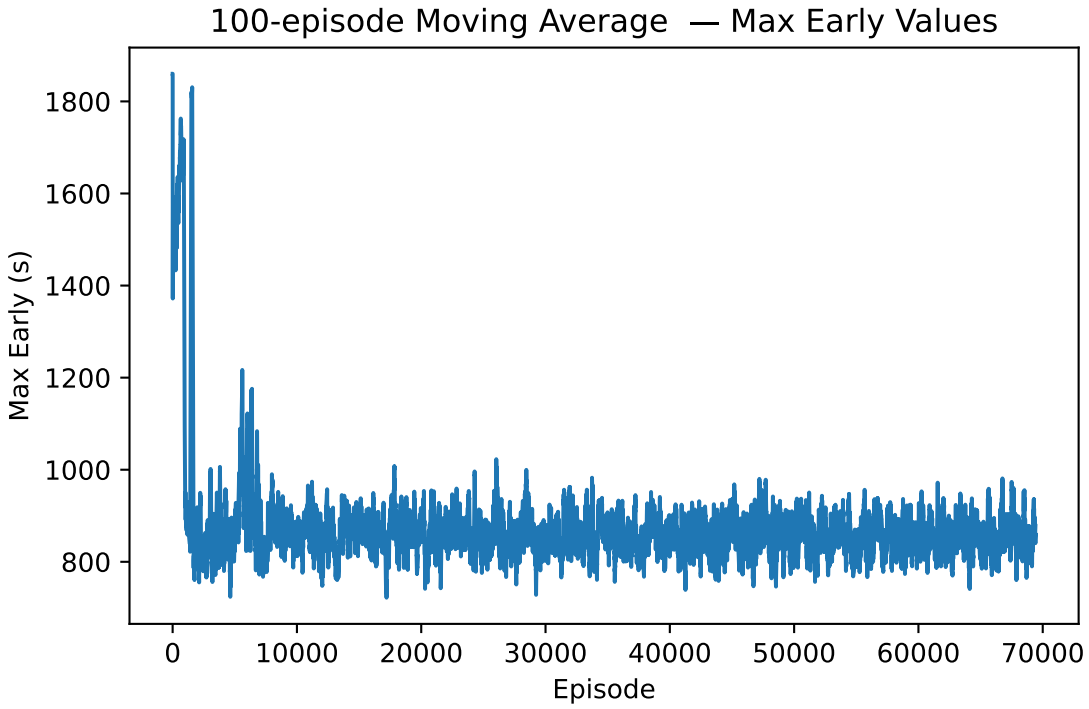

Figure 4.18 Hyperparameter–tuned model — Maximum earliness (100–episode MA). Early spikes disappear; the series stays compact afterwards.

Steps to finish.

The agent consistently terminates in **33** steps on this test set (minimum, maximum, and median all 33), which matches the expected number of decision points for the scenario.

Objective.

The terminal objective lies in a tight band of approximately $\mathbf{3.08 \times 10^4}$–$\mathbf{3.14 \times 10^4}$ (reported as `final_obj`), indicating stable policy behaviour under the current caps and disturbance draws (e.g., 30,792; 31,088; 31,375).

Episode return.

Cumulative return concentrates in a narrow range (around -4.5×10^4). Because the per–step reward combines normalized objective improvement with monotonicity and cap hinges, a stable return band is consistent with a policy that avoids large penalties and resolves conflicts in a consistent pattern.

Maximum end delay.

The worst end–of–route delay across trains varies widely across scenarios (hundreds of seconds to just over a thousand seconds). Many episodes fall in the mid range

(well below 600 s), with a smaller tail of harder cases near operational cap boundaries. This indicates that the agent generally contains network–wide tail delays while adapting to different disturbance mixes.

Final earliness across trains.

The maximum absolute *final* earliness spans a broad range across episodes (from low hundreds of seconds to just above a thousand seconds), reflecting differences in where precedence is decided relative to terminal blocks. The monotonicity term in the reward and the early–cap hinge together limit excessive early finishes while allowing necessary throughput when the corridor is uncongested.

4.3.4 Qualitative assessment

- **Stability.** Narrow bands for `final_obj` and return, together with constant step counts, point to a stable policy on this test set.
- **Delay control.** Maximum end delays are usually contained well below 1,000 s, with a small number of tougher episodes when caps bind.
- **Auditability.** Per–episode CSV timetables confirm feasibility and make it straightforward to trace outliers back to specific precedence choices in the AG.

Summary

When evaluated on unseen disturbance realizations within the same infrastructure topology, the trained agent consistently produces feasible solutions with low inference latency. Quantitative results over 100 test episodes show stable total and maximum delay metrics, while qualitative analysis confirms that the agent resolves conflicts in a manner consistent with operational intuition. These results indicate that the learned policy generalizes well across disturbance magnitudes and train order variations on a fixed corridor, supporting its suitability for real-time decision support in operational settings.

4.4 Comparative Analysis of OptDis and RL-Based Conflict Resolution

This section compares the conflict-resolved timetables generated by the company planning software (LUKS) and the proposed Deep Q-Network (DQN) agent. The comparison is based on arrival and departure times, induced delays, and total travel times for Trains A, B, and C.

It presents a comparative evaluation of the proposed RL-based conflict resolution approach and the OptDis optimization-based solver. The comparison is performed on a set of controlled test scenarios designed to highlight differences in delay propagation,

travel time, and robustness under varying operational conditions. For each example, quantitative results are summarized in tables, followed by a brief interpretation of the observed behavior.

4.4.1 Example 1: Testing scenario with temporal variations

Before comparing conflict resolution strategies, a baseline timetable is defined for the testing scenario as shown in Table 4.3. This timetable corresponds to the original LUKS output without any conflict resolution applied. It reflects the raw schedule used as input for both the OptDis and DQN-based approaches.

Train	Start Time	End Time
A	07:00:00	07:11:11
B	07:02:00	07:13:14
C	07:00:00	07:07:19

Table 4.3 Baseline timetable for testing

The trains and departure times in the baseline timetable were selected to reproduce a realistic operational scenario extracted from LUKS data, in which multiple trains traverse a shared corridor within a short time window. This configuration deliberately induces temporal overlap at critical blocks, ensuring the presence of conflicts when disturbances are introduced. As such, the scenario provides a representative and sufficiently challenging test case for evaluating conflict resolution strategies in the Toy Network shown in 2.3

Table 4.3 summarizes the baseline timetable used for the first comparison scenario. The schedule corresponds to the original LUKS output without any applied conflict resolution and serves as a common reference for both OptDis and the RL-based approach. By using the same baseline timetable, differences observed in the subsequent results can be attributed solely to the conflict-resolution strategy rather than initial schedule variations.

LUKS after conflict resolution

Table 4.4 presents the conflict-resolved timetable generated by the LUKS optimization-based solver. The solution reflects globally optimized precedence decisions that minimize delay while ensuring feasibility across all trains in the scenario.

Train	Start Time	End Time
A	06:55:28	07:06:39
B	07:01:59	07:16:20
C	07:00:00	07:07:19

Table 4.4 Conflict resolved timetable generated by LUKS

The resulting timetable exhibits reduced accumulated delay and compact train separation, demonstrating the ability of the optimization-based approach to efficiently coordinate conflicting train movements under the given temporal constraints.

DQN after conflict resolution

Table 4.5 shows the timetable obtained after conflict resolution using the trained DQN-based approach. The schedule is produced by sequential local precedence decisions selected by the learned policy.

Train	Start Time	End Time
A	06:59:39	07:10:50
B	07:01:39	07:17:08
C	07:01:30	07:12:00

Table 4.5 Conflict resolved timetable generated by DQN

Compared to the LUKS solution, the DQN-based timetable remains feasible but exhibits slightly larger delays for some trains. This reflects the limited global foresight of the learning-based policy, while confirming its ability to resolve conflicts in a consistent and operationally valid manner.

4.4.1.1 Total Travel Time and Delay Comparison

This section compares the impact of conflict resolution on total travel time, departure delay, and arrival delay for the LUKS solver and the DQN-based agent. Since both methods share the same base timetable, all delays are reported relative to this common base timetable.

Methodology.

Table 4.6 summarizes the travel time and delay comparison. For each train, the base travel time is defined as the difference between the earliest block entry time and the latest block exit time in the base timetable. After conflict resolution, the same definition is applied to obtain the updated travel time. Departure delay is computed as the shift in the earliest block entry time, while arrival delay is computed as the shift in the latest block exit time relative to the base timetable. Positive values indicate delay, whereas negative values indicate earliness.

Method	Train	Base Travel Time	After CR Travel Time	Departure Delay	Arrival Delay
LUKS	A	11 min 11 s	11 min 11 s	−4 min 32 s	−4 min 32 s
LUKS	B	11 min 14 s	14 min 21 s	−1 s	+3 min 06 s
LUKS	C	7 min 19 s	7 min 19 s	0 s	0 s
DQN	A	11 min 11 s	19 min 07 s	0 s	+7 min 56 s
DQN	B	11 min 14 s	15 min 43 s	+2 min 00 s	+6 min 29 s
DQN	C	7 min 19 s	13 min 51 s	+3 min 00 s	+9 min 32 s

Table 4.6 Travel time, departure delay, and arrival delay after conflict resolution (relative to common base timetable)

Discussion.

The LUKS solution resolves conflicts by concentrating delay on a limited subset of trains. Trains A and C preserve their original travel times, with Train A even experiencing earliness, while Train B absorbs the majority of the delay through an extended travel time and later arrival. This behavior is characteristic of optimization-based approaches that prioritize minimizing the number of affected trains.

In contrast, the DQN-based solution distributes delay across all trains. All services experience increased travel times and positive arrival delays, arising from delayed departures and additional waiting during execution. Rather than concentrating delay on a single train, the agent restores feasibility by spreading delay more evenly across the timetable, resulting in a higher cumulative delay.

Key Observations

Overall, the comparison highlights a fundamental behavioral difference between the two approaches. LUKS achieves conflict resolution with minimal disruption to most trains by concentrating delay, whereas the DQN agent accepts larger overall travel time increases in exchange for distributing delay across multiple services. This distinction is important when evaluating trade-offs between fairness and total delay in learning-based timetable conflict resolution.

4.4.2 Example 2: Testing scenario with different block order and train directions

This example evaluates the impact of conflict resolution on three trains using the baseline timetable, the LUKS conflict-resolved timetable, and the DQN-based conflict-resolved timetable. Travel times, departure delays, and arrival delays are reported relative to the baseline timetable.

The baseline timetables used for the evaluation differ from those used during training of the DQN agent. These differences extend beyond timing perturbations and include variations in train directions and block traversal sequences. As a result, the evaluated scenarios exhibit conflict structures that are not present in the training data. This design choice allows the evaluation to assess the agent's ability to generalize to structurally different timetables, rather than merely adapting to time-shifted instances of previously seen schedules.

Baseline Timetable

Table 4.7 summarizes the baseline timetable for Example 2 prior to any conflict resolution. The schedule corresponds to the initial LUKS output and reflects the planned start and end times for each train without considering interaction effects between conflicting movements. This baseline serves as a reference for evaluating the impact of conflict resolution on travel time and delay propagation.

Train	Start Time	End Time	Travel Time
A	12:01:09	12:07:10	6 min 01 s
B	11:58:07	12:08:05	9 min 58 s
C	11:58:07	12:08:22	10 min 15 s

Table 4.7 Baseline timetable for Example 2

LUKS After Conflict Resolution

Table 4.8 presents the conflict-resolved timetable obtained using the LUKS optimization-based solver for Example 2. The solver adjusts train start and end times to resolve conflicts while minimizing overall delay and preserving feasible block occupancy.

Train	Start Time	End Time	Travel Time
A	11:58:59	12:05:01	6 min 02 s
B	12:00:44	12:11:58	11 min 14 s
C	11:58:07	12:08:47	10 min 40 s

Table 4.8 LUKS timetable after conflict resolution (Example 2)

Compared to the baseline timetable, the LUKS solution results in moderate adjustments to train schedules, leading to a controlled increase in travel times. The optimized solution balances precedence decisions across trains and limits delay propagation under the altered conflict structure of this scenario.

DQN After Conflict Resolution

For the DQN agent, only the final start and end times are considered. These are obtained as the earliest and latest event times per train in the conflict-resolved episode.

Table 4.9 shows the timetable produced by the DQN-based approach for Example 2. Only the final start and end times of each train are reported, corresponding to the earliest and latest events observed in the conflict-resolved episode.

Train	Start Time	End Time	Travel Time
A	12:01:09	12:35:52	34 min 43 s
B	11:58:07	12:43:56	45 min 49 s
C	11:58:07	12:27:05	28 min 58 s

Table 4.9 DQN timetable after conflict resolution (Example 2)

The DQN-based solution remains feasible but exhibits significantly higher travel times compared to the optimization-based approach. The increased delays reflect the sensitivity of the learned policy to changes in conflict ordering and route structure, highlighting the limited global coordination capability of the agent in this scenario.

4.4.2.1 Travel Time and Delay Comparison

Methodology.

As shown in Table 4.10 travel time is defined as the difference between the train's end time and start time. Departure delay is computed as the shift in start time relative to the baseline timetable, while arrival delay is computed as the shift in end time relative to the baseline timetable. Positive values indicate delay and negative values indicate earliness.

Method	Train	Base Travel Time	After CR Travel Time	Departure Delay	Arrival Delay
LUKS	A	6 min 01 s	6 min 02 s	−2 min 10 s	−2 min 09 s
LUKS	B	9 min 58 s	11 min 14 s	+2 min 37 s	+3 min 53 s
LUKS	C	10 min 15 s	10 min 40 s	0 s	+25 s
DQN	A	6 min 01 s	34 min 43 s	0 s	+28 min 42 s
DQN	B	9 min 58 s	45 min 49 s	0 s	+35 min 51 s
DQN	C	10 min 15 s	28 min 58 s	0 s	+18 min 43 s

Table 4.10 Travel time, departure delay, and arrival delay comparison for Example 2 (relative to baseline)

Discussion

The LUKS solution resolves conflicts with limited impact on total travel times by selectively reallocating temporal slack. In this solution, Train A is shifted earlier, resulting in earliness, while Trains B and C experience moderate arrival delays. Delay is primarily concentrated on Train B, which is consistent with an optimization-based strategy that seeks to minimize overall timetable disruption by localizing delay rather than distributing it across all services.

In contrast, the DQN-based solution introduces substantial increases in travel time and arrival delay for all trains. Although scheduled departure times are largely preserved at early stages of the timetable, prolonged waiting induced by sequential conflict resolution decisions leads to significant extensions of end-to-end journey durations. This behavior indicates that the learned policy restores feasibility by accumulating and propagating delay across multiple trains, rather than concentrating delay on a limited subset of services.

Key Observations

Example 2 further highlights the behavioral differences between optimization-based and learning-based approaches under structural timetable modifications. While LUKS achieves conflict resolution with relatively small and localized delays, the DQN agent distributes delay more uniformly across all trains, resulting in significantly increased travel times. This outcome reflects the agent's limited ability to generalize to changes in block order and train direction that alter the underlying conflict topology. The results underscore the importance of reward design, delay penalties, and exposure to structurally diverse training scenarios when applying reinforcement learning methods to timetable conflict resolution.

4.4.3 Example 3: Increased Number of Trains on the Same Topology

This example evaluates conflict resolution performance when the number of trains is increased while keeping the underlying infrastructure topology unchanged. Five trains (A–E) are considered. The baseline timetable, the LUKS conflict-resolved timetable, and the DQN-based conflict-resolved timetable are compared in terms of total travel time, departure delay, and arrival delay.

4.4.3.1 Baseline Timetable

The baseline timetable in Table 4.11 defines the initial schedule for the scenario with an increased number of trains on the same infrastructure topology. This timetable introduces higher traffic density and a larger number of conflicts, creating a more challenging rescheduling problem for both approaches.

Train	Start Time	End Time	Travel Time
A	07:00:03	07:15:17	15 min 14 s
B	07:03:01	07:17:12	14 min 11 s
C	07:01:39	07:08:58	7 min 19 s
D	07:00:03	07:14:35	14 min 32 s
E	07:01:39	07:08:24	6 min 45 s

Table 4.11 Baseline timetable for Example 3

4.4.3.2 LUKS After Conflict Resolution

Table 4.12 presents the timetable produced by the LUKS optimization-based solver after conflict resolution. It represents the reference solution obtained through global optimization under the given scenario and is used to benchmark solution quality in terms of delay and travel time.

Train	Start Time	End Time	Travel Time
A	07:00:03	07:15:17	15 min 14 s
B	07:03:01	07:20:04	17 min 03 s
C	07:01:39	07:11:07	9 min 28 s
D	06:51:27	07:06:42	15 min 15 s
E	06:53:57	07:01:16	7 min 19 s

Table 4.12 LUKS timetable after conflict resolution (Example 3)

The LUKS-based solution successfully resolves all conflicts while maintaining controlled travel times across trains. Compared to the baseline timetable, the optimization-based approach redistributes delays to achieve a globally consistent schedule, demonstrating its effectiveness in handling increased traffic density through coordinated precedence decisions and global optimization.

4.4.3.3 DQN After Conflict Resolution

For the DQN agent, only the final start and end times are considered. These are extracted as the earliest and latest event times per train in the conflict-resolved episode.

The table 4.13 shows the timetable generated by the DQN-based approach after resolving all detected conflicts. The resulting schedule reflects the cumulative effect of local precedence decisions selected by the learned policy.

Train	Start Time	End Time	Travel Time
A	07:00:03	07:58:39	58 min 36 s
B	07:03:01	07:58:27	55 min 26 s
C	07:01:39	07:23:02	21 min 23 s
D	07:00:03	07:58:08	58 min 05 s
E	07:01:39	07:28:44	27 min 05 s

Table 4.13 DQN timetable after conflict resolution (Example 3)

The DQN-based solution remains feasible but exhibits significantly higher travel times compared to the optimization-based approach. This behavior reflects the cumulative effect of local precedence decisions and highlights the limited global coordination of the learned policy in high-density traffic scenarios.

4.4.3.4 Travel Time and Delay Comparison

Methodology.

As shown in Table 4.14 travel time is computed as the difference between the train end time and start time. Departure delay corresponds to the shift in start time relative to the baseline timetable, while arrival delay corresponds to the shift in end time relative to the baseline. Positive values indicate delay, and negative values indicate earliness.

Method	Train	Base Travel Time	After CR Travel Time	Departure Delay	Arrival Delay
LUKS	A	15 min 14 s	15 min 14 s	0 s	0 s
LUKS	B	14 min 11 s	17 min 03 s	0 s	+2 min 52 s
LUKS	C	7 min 19 s	9 min 28 s	0 s	+2 min 09 s
LUKS	D	14 min 32 s	15 min 15 s	−8 min 36 s	−7 min 53 s
LUKS	E	6 min 45 s	7 min 19 s	−7 min 42 s	−7 min 08 s
DQN	A	15 min 14 s	58 min 36 s	0 s	+43 min 22 s
DQN	B	14 min 11 s	55 min 26 s	0 s	+41 min 15 s
DQN	C	7 min 19 s	21 min 23 s	0 s	+14 min 04 s
DQN	D	14 min 32 s	58 min 05 s	0 s	+43 min 33 s
DQN	E	6 min 45 s	27 min 05 s	0 s	+20 min 20 s

Table 4.14 Travel time, departure delay, and arrival delay comparison for Example 3 (relative to baseline)

Discussion.

With an increased number of trains on the same topology, the LUKS solution resolves conflicts by selectively redistributing delay. Train A remains unaffected, Trains B and C absorb moderate arrival delays, while Trains D and E are shifted earlier, resulting in earliness. Overall, the increase in travel time remains limited.

In contrast, the DQN-based solution exhibits a pronounced degradation in performance as traffic density increases. All trains experience substantial arrival delays and significantly extended travel times, despite unchanged departure times. This indicates that, under higher conflict density, the learned policy restores feasibility primarily by allowing extensive waiting and delay accumulation rather than concentrating delay on a subset of trains.

Key Observations.

Example 3 demonstrates that increasing the number of trains on the same infrastructure amplifies the behavioral differences between optimization-based and learning-based conflict resolution. While LUKS scales by managing delay distribution effectively, the DQN agent exhibits poor scalability in this scenario, leading to large cumulative delays across all trains.

4.4.4 Example 4: Evaluation on a Previously Unseen Infrastructure Topology

To assess the generalization capability of the trained DQN agent, the learned policy was evaluated on a previously unseen infrastructure topology containing additional stations and blocks beyond those observed during training. While the training scenarios were restricted to a fixed set of infrastructure blocks, the new topology introduces extended block sequences and structurally different routes, resulting in novel conflict configurations.

Generalization Limitation.

The trained DQN agent is currently not able to adapt to infrastructure topologies that differ from those encountered during training. The learned policy relies on a state representation and action space that are explicitly tied to the set of blocks present in the training topology. Introducing new blocks fundamentally alters the structure of the Alternative Graph, leading to conflict patterns and precedence decisions that lie outside the agent's training distribution.

Underlying Challenge.

This limitation arises from the topology-dependent nature of the state encoding and the neural network input dimensionality. The agent learns value estimates for conflicts defined over a fixed infrastructure graph; consequently, new blocks and routes

cannot be interpreted meaningfully by the trained policy. Even if the state representation were extended dynamically, the agent would lack learned experience for conflicts involving unseen infrastructure elements, resulting in unreliable or uninformed decisions.

Implications for Generalization.

The results demonstrate that the proposed RL-based approach generalizes well across different disturbance magnitudes, traffic densities, and timing variations on a fixed infrastructure topology. However, it does not generalize to entirely new network topologies without additional adaptation. This behavior is consistent with the training regime, which does not expose the agent to topological variability.

Requirement for Retraining or Adaptation.

For deployment on substantially different infrastructures, retraining or adaptation is required. One practical approach is to retrain the agent using data generated from the new topology, allowing it to learn infrastructure-specific conflict and delay propagation patterns. Alternatively, transfer learning may be employed by initializing the model with pre-trained weights and fine-tuning it on the new network. More fundamentally, future work should investigate topology-invariant or graph-based state representations to improve structural generalization.

Key Observations.

Example 4 highlights that the current DQN-based conflict resolution framework is topology-specific. While effective on variations of a known infrastructure, extending the approach to unseen railway networks requires retraining or more expressive state representations that can capture structural differences across topologies.

OptDis Performance on the Unseen Topology.

In contrast to the learning-based approach, the OptDis optimization solver is able to handle the previously unseen infrastructure topology without modification. The solver does not rely on a fixed-dimensional state representation and can therefore resolve conflicts on arbitrary network topologies.

The resulting OptDis solution remains feasible and consistent with the global optimization objective, demonstrating robust behavior under structural changes in the infrastructure. This highlights a key distinction between the two approaches: while the RL-based policy is topology-specific and limited by its training distribution, the optimization-based method generalizes inherently to new infrastructure layouts at the cost of higher computational effort.

5

Discussion of Results

This chapter synthesizes the experimental results presented in the previous chapter and interprets their implications for practical railway timetable rescheduling. Rather than introducing new experiments, the focus lies on understanding *why* the observed behaviors occur, how different design choices affect performance, and under which conditions reinforcement learning provides operational value. The discussion proceeds from a comparative analysis of learning configurations and optimization-based baselines to broader considerations of solution quality, computational performance, scalability, and deployment implications.

5.1 Comparative Discussion about the performance of the models

The three environment formulations evaluated in this work exhibit distinct learning characteristics and induce different types of dispatching behaviour from the agent. Although they share the same underlying network, timetable and DQN architecture, their reward structures and penalty mechanisms produce measurably different convergence patterns, robustness levels, and final performances.

Model A vs. Model B.

The introduction of earliness penalties and monotonicity shaping in Model B provides a noticeably more stable learning process compared to the baseline lateness-only formulation of Model A. Model A's convergence is hindered by its willingness to exploit aggressive local shortcuts: because the environment does not penalize early running or large oscillatory corrections, the agent often discovers short-term beneficial but structurally unrealistic decisions. Model B rectifies this by discouraging both excessive positive and negative delay increments. The corresponding episode-return and LP-cost curves exhibit smoother and more predictable behaviour, and

the agent achieves lower final maximum delays. These results confirm that even moderate increases in realism within the reward function significantly improve the quality and stability of the learned dispatching policy.

Model B vs. Model C.

The curriculum-based formulation introduced in Model C extends the refinement achieved by Model B. By gradually tightening the lateness and earliness caps at specific checkpoints, Model C deliberately introduces temporary phases of increased difficulty. This results in the characteristic dip–then–recover pattern visible in all three learning curves. During these stricter phases, previously acceptable actions become heavily penalized, forcing the agent to restructure its policy. Although performance deteriorates briefly, the agent subsequently achieves substantially higher episode returns, lower LP-cost values, and significantly reduced maximum delays compared to the other models.

The behaviour of Model C reflects a well-known principle in curriculum-based reinforcement learning: temporary exposure to harsher constraint regimes guides the agent away from fragile or over-fitted behaviours, toward more resilient and general solutions. In practice, this manifests as a policy that not only minimizes overall delay but does so while respecting a broader set of implicit operational requirements.

Model A vs. Model C.

The contrast between Model A and Model C is especially pronounced. Model A achieves reasonable performance only after a prolonged initial oscillatory phase and ultimately converges to a feasible, but not particularly robust and conflict-resolute strategy. In contrast, Model C learns efficiently from the outset, adapts to progressively stricter operational regimes, and reaches the lowest delay and LP-cost values across all experiments. The curriculum structure therefore proves instrumental in transforming a general delay-minimization agent into a sophisticated and constraint-aware dispatcher.

Overall comparison.

When comparing all three models holistically, the following pattern emerges:

- Model A learns slowly and displays high variance due to its simple reward structure.
- Model B adds realism and provides smoother convergence and lower delays.
- Model C surpasses both, combining efficient exploration with disciplined late-training behaviour, resulting in the best global performance.

This progression illustrates how incremental modifications to the reward structure and environment logic can dramatically improve the quality of the learned policy in Alternative-Graph-based reinforcement learning systems.

Summary

This evaluation demonstrates clear performance differences arising from the design of the environment and reward structure. Model A, based solely on lateness penalties, provides a useful baseline but suffers from unstable early learning and limited realism. Model B improves significantly upon this by incorporating monotonicity and earliness penalties, yielding smoother learning dynamics and improved final performance. Model C, which introduces curriculum-based cap adjustments, exhibits the most complex learning trajectory but ultimately achieves the highest-quality policy, the lowest LP-cost values, and the smallest maximum delays.

The results highlight the importance of carefully designed reward shaping and progressive constraint exposure in environments involving temporal feasibility and conflict propagation. In particular, the curriculum mechanism employed in Model C enables the agent to move beyond merely feasible or acceptable policies toward genuinely robust and operationally relevant conflict-resolution strategies. These findings support the adoption of curriculum-based reinforcement learning approaches for large-scale railway traffic management tasks.

5.2 Comparison of Good and Worst Hyperparameter Configurations

This section presents a detailed comparison between the tuned ("Good") hyperparameter configuration and the four intentionally degraded ("Worst A–D") configurations as shown in Table 5.1. These worst–case configurations were constructed to demonstrate the sensitivity of the reinforcement learning (RL) agent to hyperparameter choices. Each worst set degrades the agent's stability, exploration behaviour, or learning dynamics in a different manner, thereby illustrating the importance of hyperparameter tuning for the train rescheduling problem.

Parameter	Good	Worst A	Worst B	Worst C	Worst D
Learning rate	1×10^{-5}	1×10^{-3}	5×10^{-4}	1×10^{-6}	1×10^{-2}
Discount factor γ	0.9	0.99	0.9	0.5	0.1
ϵ start	1.0	1.0	1.0	1.0	1.0
ϵ decay	0.999	0.995	0.95	0.99999	1.0
ϵ_{min}	0.01	0.10	0.001	0.20	1.0
Batch size	256	64	128	1024	32
Train start	10000	2000	2000	20000	0
Train frequency	16	4	8	32	1
Target update freq.	64	500	64	32	1
L2 regularisation	1×10^{-4}	0	1×10^{-5}	1×10^{-2}	0
Replay memory size	10 000 000	200 000	1 000 000	5 000 000	100 000

Table 5.1 Good (Tuned) Versus Worst Hyperparameter Sets

5.2.1 Qualitative Comparison of Learning Behaviour

The tuned hyperparameter set ("Good") exhibits stable learning behaviour, with cumulative rewards improving steadily over episodes and the scheduling objective converging toward lower values. In contrast, each of the four worst hyperparameter sets introduces a distinct degradation mode, illustrating typical RL failure patterns such as divergence, premature exploitation, underfitting, or random-walk behaviour.

5.2.1.1 Worst Set A: Instability Due to Aggressive Learning

Worst Set A uses a learning rate that is two orders of magnitude larger than the tuned value. Combined with reduced replay memory and infrequent target network updates, the Q-network exhibits unstable behaviour. The cumulative reward fluctuates with no upward trend and the scheduling objective remains significantly worse than the Good configuration. The agent fails to converge and frequently oscillates between suboptimal actions, demonstrating the detrimental effect of excessive learning rates in DQN-based RL systems.

5.2.1.2 Worst Set B: Premature Exploitation and Divergence

Worst Set B decays the exploration factor ϵ too quickly and forces the agent into near-deterministic decisions early in training. As a result, the agent gets trapped in poor local optima. The performance logs show extreme oscillations in cumulative reward and catastrophic drops in objective value (reaching below $-59\,000$), which never occur under the Good configuration. This configuration highlights the necessity of slow exploration decay in large combinatorial scheduling problems.

5.2.1.3 Worst Set C: Underfitting and Slow Learning

Worst Set C employs an excessively small learning rate and a very large batch size, both of which suppress learning progress. In addition, the high L2 regularisation strength restricts the modelling capacity of the network. The resulting learning curve is almost flat, showing little to no improvement, and the objective values frequently degrade. This configuration represents underfitting due to overly constrained optimisation dynamics.

5.2.1.4 Worst Set D: Random-Walk Behaviour

Worst Set D effectively removes learning by assigning a very high learning rate, no exploration decay, and minimal replay memory. The agent behaves almost like a random decision-maker throughout training. The performance logs show near-complete lack of convergence: rewards remain noisy and the objective values are consistently high. This configuration serves as a baseline illustrating the behaviour of an untrained agent.

5.2.1.5 Comparison with the Tuned Configuration

Across all experiments, the Good hyperparameter configuration consistently produces lower scheduling objective values, smoother reward curves, and markedly better feasibility. The contrast with the four worst sets demonstrates that the proposed hyperparameter tuning procedure is essential for achieving stable and high-quality learning in the train rescheduling environment.

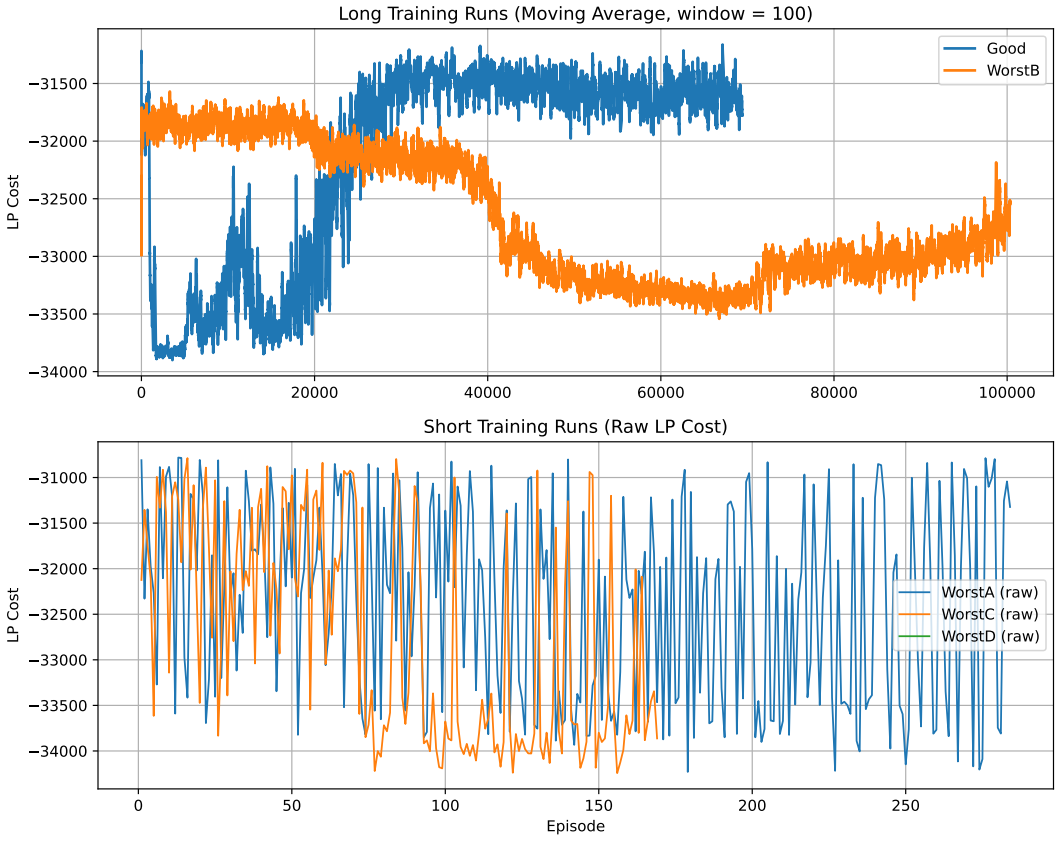

Figure 5.1 Two-panel comparison of training reward (negative objective) for the tuned (Good) and worst hyperparameter configurations. The top panel shows long training runs (Good and Worst B) using a 100-episode moving average of the reward. The bottom panel shows short training runs (Worst A, Worst C and Worst D) using raw reward values, as these configurations terminate early.

To ensure a fair and readable visualization across hyperparameter configurations with strongly varying training lengths, a two-panel representation is adopted in Fig. 5.1. Long training runs are displayed using a 100-episode moving average to suppress stochastic noise, while short training runs are shown using raw reward values to preserve early-stage learning dynamics. This avoids visual compression of short runs near the origin when plotted against long training horizons.

5.2.2 Discussion

Overall, the evaluation demonstrates that the trained agent is able to successfully navigate the AG-based scheduling environment and produce feasible, conflict-free

timetables that improve upon the disturbed initial state. The agent's decision-making is deterministic, fast, and aligned with operationally meaningful resolution patterns. These results validate the integration of AG-based timing propagation with a reinforcement learning controller and form a basis for further comparison against optimization-based methods such as OptDis.

5.3 Comparison with Optimization-Based Approaches

To compare the optimization based approaches, Table 5.2 summarizes the key observations from the four comparative evaluation scenarios presented in Section 4.10.

Scenario	**RL-Based DQN Behavior**	**OptDis Behavior**	**Key Interpretation**
Example 1: Temporal variations	Resolved conflicts quickly and produced feasible schedules with low inference latency; delay distributed across trains	Achieved lower total delay but required longer computation time	RL is effective for rapid response under time pressure, while OptDis yields higher-quality solutions when time allows
Example 2: Different block order and train directions	Maintained feasibility but showed sensitivity to altered conflict structure and ordering	Handled reordering robustly due to global optimization	RL performance depends on learned conflict patterns; OptDis is less sensitive to structural variations
Example 3: Increased number of trains on the same topology	Scaled to larger traffic volumes but accumulated higher total delay due to delay spreading	Optimized delay concentration more effectively at higher densities	RL favors locally feasible decisions, whereas OptDis better manages global delay trade-offs in dense traffic
Example 4: Previously unseen infrastructure topology	Failed to generalize and could not construct valid states or policies	Solved the scenario without retraining	RL policy is topology-specific; retraining or topology-invariant representations are required for generalization

Table 5.2 Summary of comparative evaluation scenarios between the RL-based approach and OptDis (Section 4.10).

Across all four scenarios, the results highlight a consistent trade-off between solution quality and decision latency. While OptDis provides robust and high-quality solutions across a wide range of conditions, the RL-based approach excels in scenarios with fixed infrastructure and tight time constraints, where rapid, feasible decisions are required.

Taken together, the scenarios summarized in Table 5.2 highlight recurring trade-offs between solution quality, robustness, and decision latency. While the optimization-based OptDis solver consistently achieves strong delay minimization across a wide range of conditions, the reinforcement learning agent exhibits complementary strengths in responsiveness and feasibility under time constraints. The following sections discuss these trade-offs in more detail by examining solution quality, computational performance, scalability, and implications for operational use.

5.4 Solution Quality

Across all evaluated scenarios on a fixed infrastructure topology, the DQN agent consistently produced feasible timetables and was able to resolve conflicts without violating operational constraints. In scenarios involving limited disturbances or small numbers of trains, the resulting total travel times and arrival delays were often comparable to those obtained by the LUKS solver. This confirms that the agent successfully learned precedence-selection strategies that align with delay-minimization objectives.

However, as traffic density increased, clear qualitative differences emerged. In Example 3, which increased the number of trains on the same topology, the LUKS solver concentrated delay on a subset of trains, preserving near-baseline travel times for others. In contrast, the DQN agent distributed delay across all trains, leading to substantially increased travel times and arrival delays. While feasibility was maintained, the cumulative delay grew significantly, indicating that the learned policy favors globally safe but conservative decisions under high conflict density.

5.5 Computational Performance

Once trained, the DQN agent demonstrated extremely low inference latency, producing precedence decisions in milliseconds per step. This confirms the suitability of the approach for real-time or near real-time operational settings where rapid response is critical. In contrast, the LUKS solver relies on mixed-integer optimization, which can require significantly more computation time as problem size and conflict density increase.

That said, computational efficiency alone does not guarantee solution quality. The experiments show that although the DQN agent reacts quickly, its decisions under complex conditions may lead to excessive waiting and delay accumulation. This highlights a fundamental trade-off between speed and optimality when comparing learning-based methods with exact optimization.

5.6 Scalability and Generalization

The evaluation demonstrates that the DQN agent scales effectively with increasing traffic density and conflict complexity on a fixed infrastructure topology. Across scenarios with varying numbers of trains, disturbance magnitudes, and train orderings,

the agent maintained feasible operation and stable decision latency, indicating that the learned policy remains applicable as problem size grows within a given corridor. In addition, the agent generalizes well across different disturbance realizations and traffic patterns *as long as the underlying infrastructure topology remains unchanged*. This supports the hypothesis that an AG-based state representation combined with action masking enables robust learning on a fixed network.

However, experiments on previously unseen infrastructure topologies provide clear evidence of limited topology-level generalization. When additional blocks and modified station layouts were introduced that were not present during training, the evaluation failed during state construction due to missing block identifiers. This behavior is not incidental; it reflects the fact that the state representation, action space, and neural network architecture are explicitly tied to the training topology. Consequently, the trained model cannot be directly applied to infrastructures with different block sequences or extended layouts without retraining or architectural adaptation.

These findings indicate that while reinforcement learning scales well within a fixed operational context, achieving topology-invariant generalization requires alternative state representations or transfer learning mechanisms.

5.7 Comparison with LUKS and Hybrid Potential

The LUKS solver consistently produced high-quality solutions across all tested scenarios, including cases with increased traffic density and structurally different timetables. Its ability to concentrate delay and globally optimize precedence decisions remains a key strength. In contrast, the DQN agent excels in speed and adaptability within a known topology but exhibits degraded performance as conflicts intensify.

These complementary strengths suggest a hybrid operational role. The DQN agent could be used to generate rapid initial precedence decisions or screen infeasible options, which are then refined by an optimization-based solver such as LUKS. Such a hybrid approach could combine the responsiveness of reinforcement learning with the solution quality guarantees of exact optimization.

5.8 Implications for Operational Use

From an operational perspective, the results indicate that a trained DQN agent is best suited as a corridor-specific decision-support tool. Deployment on new infrastructures would require retraining or fine-tuning using data from the target topology. Within these bounds, the approach offers valuable real-time support, particularly in situations where fast reactions are more critical than global optimality.

6

Conclusion

This thesis investigates the applicability of reinforcement learning for microscopic train timetable conflict resolution under disturbances, using the Alternative Graph (AG) as an explicit and interpretable modeling foundation. The primary objective was to assess whether a learning-based approach can support fast, feasible conflict-resolution decisions on realistic railway infrastructure data, and how its performance compares to a state-of-practice optimization-based solver under operationally relevant conditions.

A complete data-to-decision pipeline was developed that transforms industrial exports from the LUKS system into Alternative Graph instances and a Markov Decision Process suitable for reinforcement learning. Within this environment, a Deep Q-Network (DQN) agent was trained to resolve conflicts by selecting local precedence decisions on shared infrastructure blocks, with feasibility preserved through explicit graph-based propagation and cycle checks. The approach was evaluated on realistic disturbance scenarios and benchmarked against OptDis, an industry-standard MILP-based rescheduling solver, under matched scenarios and time constraints.

The experimental evaluation demonstrates that the proposed learning-based framework can consistently produce feasible rescheduling decisions with negligible inference latency on fixed infrastructure topologies. Across a wide range of disturbance magnitudes and train orderings on the same corridor, the trained agent exhibits stable behavior and effective delay control, confirming that reinforcement learning can capture meaningful conflict-resolution strategies when grounded in a microscopic and feasibility-aware model. The analysis of learning behavior further highlights the importance of reward design: curriculum-based shaping leads to more stable convergence and lower peak delays than simpler lateness-only formulations, at the cost of increased training time.

Robustness and sensitivity experiments indicate that the learned policy remains stable under moderate variations in reward weights, delay caps, and hyperparameters. While extreme parameter choices can degrade learning quality, the tuned configuration generalizes reliably across realistic settings, suggesting that the approach is not

overly sensitive to precise tuning. However, performance degrades as traffic density increases substantially, reflecting the growing complexity of delay propagation and the limits of a fixed-capacity policy representation.

In comparison with OptDis, the learning-based approach exhibits complementary strengths. The optimization-based solver consistently achieves lower total delay in highly congested or complex scenarios but requires significantly higher computation time. In contrast, the DQN agent produces conflict-resolution decisions almost instantaneously and performs competitively in moderate-scale scenarios, though with higher cumulative delay under heavy traffic or on previously unseen infrastructure layouts. These findings confirm that reinforcement learning is well suited for time-critical decision support, while optimization remains preferable when solution optimality is paramount.

With respect to the stated hypotheses, the results provide the following conclusions. Hypothesis H1 is supported: the reinforcement-learning-based agent learns effective conflict-resolution strategies that reduce delay compared to baseline or naive approaches in disturbed scenarios. Hypothesis H2 is partially supported: the learned policy generalizes well across disturbance magnitudes and traffic patterns on a fixed corridor but exhibits limited transfer to previously unseen infrastructure topologies. Hypothesis H3 is supported in terms of computational performance, as the learning-based approach achieves predictable and minimal inference times, while the solution quality remains slightly inferior to optimization under extreme congestion.

Overall, the findings demonstrate that reinforcement learning can complement, rather than replace, optimization-based timetable rescheduling methods. When applied within a fixed operational context, a trained policy can provide fast, feasible, and interpretable decisions that support dispatchers under time pressure and can potentially guide or accelerate exact optimization. This suggests a promising role for hybrid workflows in which learning-based policies provide rapid initial decisions or pruning, followed by optimization-based refinement.

Future work should focus on improving generalization across infrastructure topologies, e.g., through richer graph-based state encodings, transfer learning, or curriculum strategies that explicitly vary network structure during training. Further investigation of hybrid learning–optimization approaches and integration into rolling-horizon or real-time dispatching systems would also be valuable. Extending the framework to incorporate uncertainty in running and dwell times, as well as interactions with rolling stock and crew scheduling, represents another important direction toward fully operational deployment.

Reproducibility.

All scripts developed and used for data processing, model training, evaluation, and plot generation in this thesis are publicly available in a GitHub repository: `https://github.com/prettore/Train_Timetabling_DQL`.

The repository contains the complete experimental pipeline and enables reproduction of the results presented in this work.

Bibliography

[1] Agasucci, V., Grani, G., and Lamorgese, L. Solving the train dispatching problem via deep reinforcement learning. *arXiv preprint arXiv:2009.00433* (2023). http://arxiv.org/abs/2009.00433v2.

[2] Aydin, G., and Şahin, A mixed integer linear programming model with heuristic improvements for single-track railway rescheduling. *Applied Sciences 13*, 2 (2023), 696.

[3] Cacchiani, V., Corman, F., Fischetti, M., and et al. An overview of recovery models and algorithms for real-time railway disturbance and disruption management. *Transportation Research Part B: Methodological 63* (2014), 15–37.

[4] D'Ariano, A., Pranzo, M., and Hansen, I. A. An advanced real-time train dispatching system for minimizing disruptions in railway traffic. *Networks and Spatial Economics* (2009).

[5] Fischer, F., Helmberg, C., et al. A re-optimization approach for train dispatching. Tech. rep., 2016.

[6] Fischetti, M., Salvagnin, D., and Zanette, A. Train rescheduling. *European Journal of Operational Research 261*, 1 (2017), 17–33.

[7] Gong, I., Oh, S., and Min, Y. Train scheduling with deep q-network: A feasibility test. *Applied Sciences 10*, 23 (2020), 8367. https://doi.org/10.3390/app10238367.

[8] Janecek, D., and Weymann, F. H. G. Luks – analysis of lines and junctions. In *Proceedings of the 12th World Conference on Transport Research* (Lisboa, Portugal, 2010), Instituto Superior Técnico, pp. 1–15. http://www.wctr2010.info/proceedings.htm.

[9] Josyula, S. P. Machine learning-based decision support for train traffic management. Master's thesis, KTH Royal Institute of Technology, Stockholm, Sweden, 2024. https://www.diva-portal.org/smash/get/diva2:1879240/FULLTEXT01.pdf.

[10] Kenworthy, L., Nayak, S., Chin, C., and Balakrishnan, H. Nice: Robust scheduling through reinforcement learning-guided integer programming. *arXiv preprint arXiv:2310.00295* (2023). https://arxiv.org/abs/2310.00295.

[11] Kim, K.-M., Rho, H.-L., Park, B.-H., and Min, Y.-H. Deep q-network approach for train timetable rescheduling based on alternative graph. *Applied Sciences 13*, 17 (2023). https://www.mdpi.com/2076-3417/13/17/9547.

[12] Liestøl, S. W., and Ytteborg, F. S. Solving the train dispatching problem using graph neural networks. Master's thesis, Norwegian University of Science and Technology, 2024.

[13] Liu, J., Lin, Z., and Liu, R. A reinforcement learning approach to solving very-short term train rescheduling problem for a single-track rail corridor. *Journal of Rail Transport Planning & Management 32* (2024), 100483.

[14] Ning, X., and et al. A deep reinforcement learning approach to high-speed train timetable rescheduling under disturbances. *Conference Paper* (2020).

[15] Şahin, G., Ahuja, R. K., and Cunha, C. B. Integer programming based approaches for the train dispatching problem. Tech. rep., 2008.

[16] Šemrov, D., Marsetič, R., Žura, M., Todorovski, L., and Srdic, A. Reinforcement learning approach for train rescheduling on a single-track railway. *Transportation Research Part B: Methodological 86* (2016), 250–267.

[17] Su, B., Li, Z., and Zhang, Q. A data-driven mixed-integer linear programming approach for railway timetable rescheduling under disturbances. *Transportation Research Part C: Emerging Technologies 158* (2024), 104278.

[18] Veelenturf, L. P., and Kroon, L. G. A railway timetable rescheduling approach for handling large-scale disruptions. *Transportation Research Part B 86* (2016), 250–267.

[19] Wang, M., Wang, L., Xu, X., Qin, Y., and Qin, L. Genetic algorithm-based particle swarm optimization approach to reschedule high-speed railway timetables: A case study in china. *Journal of Advanced Transportation 2019* (2019), 1–12.

[20] Wegele, S., D'Ariano, A., Corman, F., and Pranzo, M. Comparing the effectiveness of two real-time train dispatching systems based on alternative graphs. In *Computers in Railways (WIT Press)* (2008).

[21] Weymann, F., and Niessen, N. Unterstützung der fahrplanfeinkonstruktion mit optimierungsverfahren. *Eisenbahntechnische Rundschau 64*, 3 (2015), 16–19.

[22] Weymann, F., and Niessen, N. Verbesserung der disposition des eisenbahnbetriebs durch innovative optimierungsverfahren. *ZEVrail Glasers Annalen 139* (2015), 1–2.

[23] Xu, P., et al. Integrated train rescheduling and rerouting during disruptions for high-speed railway. *Mathematical Problems in Engineering* (2021).

[24] Yue, P., et al. Reinforcement learning for scalable train timetable rescheduling. *arXiv preprint* (2024). https://arxiv.org/abs/2401.06952.

[25] Zhang, C., et al. Train rescheduling for large-scale disruptions in a railway network. *Transportation Research Part A* (2023).

[26] Zhang, H., De Schutter, B., et al. Integrated reinforcement learning and optimization for railway timetable rescheduling. Tech. rep., TU Delft, 2024. `https://www.dcsc.tudelft.nl/~bdeschutter/pub/rep/24_009.pdf`.

[27] Zhu, Y., et al. Reinforcement learning in railway timetable rescheduling. In *IEEE International Conference on Intelligent Transportation Systems (ITSC)* (2020).

List of Figures

List of Tables

longtable

List of Symbols

Symbol	Meaning
$G = (V, E)$	Alternative Graph (AG) with node set V and arc set E
V	Set of nodes representing train–block events
E	Set of arcs (fixed and alternative) in the AG
i, j	Indices of train–block event nodes
b	Infrastructure block or segment
$(i \to j)$	Fixed arc enforcing intra-train precedence
$(i \prec j)$	Alternative arc expressing precedence of i over j
d_{ij}	Minimum running or dwell time between events i and j
0	Dummy start node in the AG
$N - 1$	Dummy end node in the AG
`Segment_ID`	Unique identifier for a physical track segment
t_i^{plan}	Planned time of event i
t_i^{act}	Actual (propagated) time of event i
δ_i	Delay at event i, $\delta_i = t_i^{\text{act}} - t_i^{\text{plan}}$
δ_k^{start}	Delay of train k at its first block
δ_k^{end}	Delay of train k at its final block
TD	Total delay over all trains and milestones
$\Delta_{\max}^{+}$	Maximum positive delay increment in an episode
$\Delta_{\max}^{-}$	Maximum magnitude of earliness increment
C	Delay normalization constant
$\text{cost}(G)$	Shortest (most negative) path cost from start to end node
J	Objective value, $J = -\text{cost}(G)$
BigM	Large penalty value for infeasible graphs
$\ell(v)$	Bellman–Ford distance label of node v
$\phi(x, \tau)$	Soft hinge penalty $\max(0, x - \tau)/\tau$
S	State space of the Markov Decision Process
A	Action space
$s \in S$	Environment state
$a \in A(s)$	Conflict-resolution action
r	Reward signal

Symbol	Meaning
γ	Discount factor
$Q(s, a)$	Action-value function
$Q^*(s, a)$	Optimal action-value function
$Q_\theta(s, a)$	DQN approximation with parameters θ
θ	Neural network parameters
$\bar{\theta}$	Target network parameters
w_{obj}	Weight for objective improvement term
w_{mono}	Weight for monotonicity penalty
w_{cap}	Weight for delay cap penalty
w_{early}	Weight for earliness penalty
τ_ℓ	Delay increment cap
τ_e	Earliness cap
ϵ	Exploration probability in ϵ-greedy policy