# Graph Reinforcement Learning for Network Control via Bi-Level Optimization

**Daniele Gammelli** [1] **James Harrison** [2] **Kaidi Yang** [3] **Marco Pavone** [1] **Filipe Rodrigues** [4] **Francisco C. Pereira** [4]

## Abstract

Optimization problems over dynamic networks have been extensively studied and widely used in the past decades to formulate numerous real-world problems. However, (1) traditional optimization-based approaches do not scale to large networks, and (2) the design of good heuristics or approximation algorithms often requires significant manual trial-and-error. In this work, we argue that data-driven strategies can automate this process and learn efficient algorithms without compromising optimality. To do so, we present network control problems through the lens of reinforcement learning and propose a graph network-based framework to handle a broad class of problems. Instead of naively computing actions over high-dimensional graph elements, e.g., edges, we propose a bi-level formulation where we (1) specify a *desired next state* via RL, and (2) solve a convex program to best achieve it, leading to drastically improved scalability and performance. We further highlight a collection of desirable features to system designers, investigate design decisions, and present experiments on real-world control problems showing the utility, scalability, and flexibility of our framework.

## 1. Introduction

Many economically-critical real-world systems are well framed through the lens of control on graphs. For instance, the system-level coordination of power generation systems (Dommel & Tinney, 1968; Huneault & Galiana, 1991; Bienstock et al., 2014); road, rail, and air transportation systems (Wang et al., 2018; Gammelli et al., 2021); complex manufacturing systems, supply chain, and distribution networks (Sarimveis et al., 2008; Bellamy & Basole, 2013); telecommunication networks (Jakobson & Weissman, 1995; Flood, 1997; Popovskij et al., 2011); and many other systems can be cast as controlling flows of products, vehicles, or other quantities on graph-structured environments.

A collection of highly effective solution strategies exist for versions of these problems. Some of the earliest applications of linear programming were network optimization problems (Dantzig, 1982), including examples such as maximum flow (Hillier & Lieberman, 1995; Sarimveis et al., 2008; Ford & Fulkerson, 1956). Within this context, handling multi-stage decision-making is typically addressed via time expansion techniques (Ford & Fulkerson, 1958; 1962). However, despite their broad applicability, these approaches are limited in their ability to handle several classes of problems efficiently. Large-scale time-expanded networks may be prohibitively expensive, as are stochastic systems that require sampling realizations of random variables (Birge & Louveaux, 2011; Shapiro et al., 2014). Moreover, nonlinearities may result in intractable optimization problems.

In this paper, we propose a strategy for simultaneously exploiting the tried-and-true optimization toolkit associated with network control problems while also handling the difficulties associated with stochastic, nonlinear, multi-stage decision-making. To do so, we present dynamic network problems through the lens of reinforcement learning and formalize a problem that is largely scattered across the control, management science, and optimization literature. Specifically, we propose a learning-based framework to handle a broad class of network problems by exploiting the main strengths of graph representation learning, reinforcement learning, and classical operations research tools (Figure 1).

The contributions of this paper are threefold[1]:

- We present a graph network-based bi-level, RL approach that leverages the specific strengths of direct optimization and reinforcement learning.
- We investigate architectural components and design decisions within our framework, such as the choice of graph aggregation function, action parameterization, how exploration should be achieved, and their impact on system performance.

[1]Stanford University [2]Google Research, Brain Team [3]National University of Singapore [4]Technical University of Denmark. Correspondence to: Daniele Gammelli <gammelli@stanford.edu>.

*Proceedings of the $40^{th}$ International Conference on Machine Learning*, Honolulu, Hawaii, USA. PMLR 202, 2023. 

[1]Code available at: `https://github.com/DanieleGammelli/graph-rl-for-network-optimization`

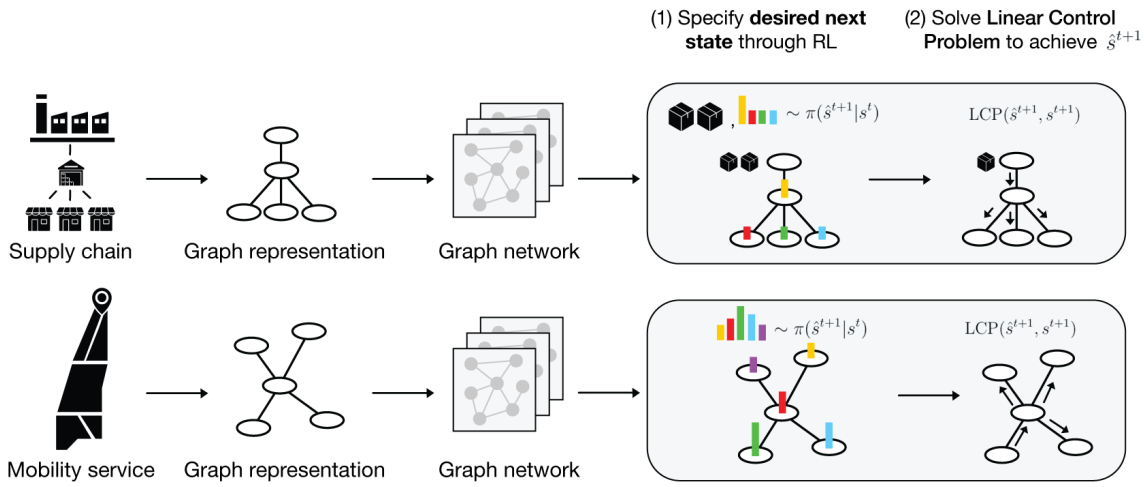



Figure 1: Many real-world systems (left) such as supply chain networks and mobility systems can be cast as controlling quantities within graph-structured environments (center-left). We present a framework that leverages graph networks (center) within a bi-level formulation. Instead of naively computing actions over graph elements, we first specify a *desired next state* through RL (center-right), and then solve a convex program to compute the graph actions that can best achieve it (right).

- We show that our approach is highly performant, scalable, and robust to changes in operating conditions and network topologies, both on artificial test problems, as well as real-world problems, such as supply chain inventory control and dynamic vehicle routing. Crucially, we show that our approach outperforms classical optimization-based approaches, domain-specific heuristics, and pure end-to-end reinforcement learning.

## 2. Related Work

Many real-world network control problems rely heavily on convex optimization (Boyd & Vandenberghe, 2004; Hillier & Lieberman, 1995). This is often due to the relative simplicity of constraints and cost functions; for example, capacity constraints on edges may be written as simple linear combinations of flow values, and costs are linear in quantities due to the linearity of prices. In particular, linear programming (as well as specialized versions thereof) is fundamental in problems such as flow optimization, matching, cost minimization and optimal production, and many more. While algorithmic improvements have made many convex problem formulations tractable and efficient to solve, these methods are still not able to handle (i) nonlinear dynamics, (ii) stochasticity, or (iii) the curse of dimensionality in time-expanded networks. In this work, we aim to address these challenges by combining the strengths of direct optimization and reinforcement learning.

Nonlinear dynamics typically requires linearization to yield a tractable optimization problem: either around a nominal trajectory, or iteratively during solution. While sequential convex optimization often yields an effective approximate solution, it is expensive and practically guaranteeing convergence while preserving efficiency may be difficult (Dinh & Diehl, 2010). Stochasticity may be handled in many ways: common strategies are distributional assumptions to achieve analytic tractability (Astrom, 2012), building in sufficient buffer to correct via re-planning in the future (Powell, 2022), or sampling-based methods, often with fixed recourse (Shapiro et al., 2014). Addressing the curse of dimensionality relies on limiting the amount of online optimization; typical approaches include limited-lookahead methods (Bertsekas, 2019) or computing a parameterized policy via approximate dynamic programming or reinforcement learning (Bertsekas, 1995; Bertsekas & Tsitsiklis, 1996; Sutton & Barto, 1998). However, these policies may be strongly sub-optimal depending on representation capacity and state/action-space coverage. In contrast to these methods, we leverage the strong performance of optimization over short horizons (in which the impact of nonlinearity and stochasticity is typically limited) and exploit an RL-based heuristic for future returns which avoids the curse of dimensionality and the need to solve non-convex or sampled optimization problems.

Our proposed approach results in a bi-level optimization problem. Bi-level optimization—in which one optimization problem depends on the solution to another optimization problem, and is thus nested—has recently attracted substantial attention in machine learning, reinforcement learning, and control (Finn et al., 2017; Harrison et al., 2018; Agrawal et al., 2019a;b; Amos & Kolter, 2017; Landry et al., 2019; Metz et al., 2019). Of particular relevance to our framework are methods that combine principled control strategies with learned components in a hierarchical way. Examples include using LQR control in the inner problem with learnable cost and dynamics (Tamar et al., 2017; Amos et al., 2018;

Agrawal et al., 2019b), learning sampling distributions in planning and control (Ichter et al., 2018; Power & Berenson, 2022; Amos & Yarats, 2020), or learning optimization strategies or goals for optimization-based control (Sacks & Boots, 2022; Xiao et al., 2022; Metz et al., 2019; 2022; Lew et al., 2022).

Numerous strategies for learning control with bi-level formulations have been proposed. A simple approach is to insert intermediate goals to train lower-level components, such as imitation (Ichter et al., 2018). This approach is inherently limited by the choice of the intermediate objective; if this objective does not strongly correlate with the downstream task, learning could emphasize unnecessary elements or miss critical ones. An alternate strategy, which we take in this work, is directly optimizing through an inner controller, thus avoiding the problem of goal misspecification. A large body of work has focused on exploiting exact solutions to the gradient of (convex) optimization problems at fixed points (Amos et al., 2018; Agrawal et al., 2019b; Donti et al., 2017). This allows direct backpropagation through optimization problems, allowing them to be used as a generic component in a differentiable computation graph (or neural network). Our approach leverages likelihood ratio gradients (equivalently, policy gradient), an alternate zeroth-order gradient estimator (Glynn, 1990). This enables easy differentiation through lower-level optimization problems without the technical details required by fixed-point differentiation.

## 3. Problem Setting: Dynamic Network Control

To outline our problem formulation, we first define the linear problem, which yields a classic convex problem formulation. We will then define a nonlinear, dynamic, non-convex problem setting that better corresponds to real-world instances. Much of the classical flow control literature and practice substitute the former linear problem for the latter nonlinear problem to yield tractable optimization problems (Li & Bo, 2007; Zhang et al., 2016; Key & Cope, 1990). Let us consider the control of $N_c$ *commodities* on graphs - for example, vehicles in a transportation problem. A graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ is defined as a set $\mathcal{V}$ of $N_v$ nodes, and a set $\mathcal{E}$ of $N_\epsilon$ ordered pairs of nodes $(i, j)$ called edges, each described by a travel time $t_{ij}$. We use $\mathcal{N}^+(i), \mathcal{N}^-(i) \subseteq \mathcal{V}$ for the set of nodes having edges pointing away from or toward node $i$, respectively. We use $x_i^t(k) \in \mathbb{R}$ to denote the quantity of commodity $k$ at node $i$ and time $t$[2].

[2]We consider several reduced views over these quantities: we write $x_i^t \in \mathbb{R}^{N_c}$ to denote the vector of all commodities, $x^t(k) \in \mathbb{R}^{N_v}$ to denote the vector of commodity $k$ at all nodes, and $x_i(k) \in \mathbb{R}^T$ to denote commodity $k$ at node $i$ for all times $t$.

### 3.1. The Linear Network Control Problem

Within the linear model, our commodity quantities evolve in time as

$$x_i^{t+1} = x_i^t + f_i^t + e_i^t, \quad \forall i \in \mathcal{V} \tag{1}$$

where $f_i^t$ denotes the change due to flow of commodities along edges and $e_i^t$ denotes the change due to exchange between commodities at the same graph node. We refer to this expression as the *conservation of flow*. We also accrue money as

$$m^{t+1} = m^t + m_f^t + m_e^t, \tag{2}$$

where $m_f^t, m_e^t \in \mathbb{R}$ denote the money gained due to flows and exchanges respectively. Our overall problem formulation will typically be to control **flows** and **exchanges** so as to maximize money over one or more steps subject to additional **constraints** such as, e.g., flow limitations through a particular edge.

**Flows.** We will denote flows along edge $(i, j)$ with $f_{ij}^t(k)$. From these flows, we have

$$f_i^t = \sum_{j \in \mathcal{N}^-(i)} f_{ji}^t - \sum_{j \in \mathcal{N}^+(i)} f_{ij}^t, \quad \forall i \in \mathcal{V} \tag{3}$$

which is the net flow (inflows minus outflows). As discussed, associated with each flow is a cost $m_{ij}^t(k)$. Note that given this formulation, the total flow cost for all commodities can be written as $m_{ij}^t \cdot f_{ij}^t = (m_{ij}^t)^\top f_{ij}^t$. Thus, we can write the total flow cost at time $t$ as

$$m_f^t = -\sum_{i \in \mathcal{V}} \left( \sum_{j \in \mathcal{N}^-(i)} m_{ji}^t \cdot f_{ji}^t + \sum_{j \in \mathcal{N}^+(i)} m_{ij}^t \cdot f_{ij}^t \right). \tag{4}$$

**Exchanges.** To define our exchange relations and their effect on commodity quantities and costs, we will write the effect that exchanges have on money for each node; we write this as $m_i^t$. Thus, we have $m_e^t = \sum_{i \in \mathcal{V}} m_i^t$. We assume there are $N_e(i)$ exchange options at each node $i$. The exchange relation takes the form

$$\begin{bmatrix} e_i^t \\ m_i^t \end{bmatrix} = E_i^t w_i^t \tag{5}$$

where $E_i^t \in \mathbb{R}^{(N_c+1) \times N_e(i)}$ is an exchange matrix and $w \in \mathbb{R}^{N_e(i)}$ are the weights for each exchange. Each column in this exchange matrix denotes an (exogenous) exchange rate between commodities; for example, for $i$'th column $[-1, 1, 0.1]^\top$, one unit of commodity one is exchanged for one unit of commodity two plus $0.1$ units of money. Thus, the choice of exchange weights $w_i^t$ uniquely determines exchanges $e_i^t$ and money change due to exchanges, $m_e^t$.

**Convex constraints.** We may impose additional convex constraints on the problem beyond the conservation of flow

we have discussed so far. There are a few common examples that one may use in several applications. A common constraint is the non-negativity of commodity values, which we may express as

$$x_i^t \geq 0, \quad \forall i, t. \tag{6}$$

Note that this inequality is defined element-wise. We may also limit the flow of all commodities through a particular edge via

$$\sum_{k=1}^{N_c} f_{ij}^t(k) \leq \overline{f}_{ij}^t, \tag{7}$$

where this sum could also be weighted per commodity. These linear constraints are only a limited selection of some common examples and the particular choice of constraints is problem-specific.

### 3.2. The Nonlinear Dynamic Network Control Problem

The previous subsection presented a linear, deterministic problem formulation that yields a convex optimization problem for the decision variables—the chosen flows and exchange weights. However, the formulation is limited by the assumption of linear, deterministic state transitions (among others), and is thus limited in its ability to represent typical real-world systems (please refer to Appendix A for a more complete treatment). In this paper, we focus on solving the nonlinear problem (reflecting real, highly-general problem statements) via a bi-level optimization approach, wherein the linear problem (which has been shown to be extremely useful in practice) is used as an inner control primitive.

## 4. Methodology

In this section, we first introduce a Markov decision process (MDP) for our problem setting in Section 4.1. We further describe the bi-level formulation that is the primary contribution of this paper and provide insights on architectural considerations in Sections 4.2 and 4.3, respectively.

### 4.1. The Dynamic Network MDP

We consider a discounted MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$. Here, $s^t \in \mathcal{S}$ is the state and $a^t \in \mathcal{A}$ is the action, both at time $t$. The dynamics, $P : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ are probabilistic, with $P(s^{t+1} \mid s^t, a^t)$ denoting a conditional distribution over $s^{t+1}$. Finally, we use $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ to denote the reward function and $\gamma \in (0, 1]$ the discount factor.

**State and state space.** Real-world network control problems are typically partially-observed and many features of the world impact the state evolution. However, a small number of features are typically of primary importance, and the impact of the other partially-observed elements can be modeled as stochastic disturbances. Our formulation requires, at each timestep, the commodity values $x^t$. Furthermore, the constraint values are required, such as costs, exchange rates, flow capacities, etc. If the graph topology is time-varying, the connectivity at time $t$ is also critical. More precisely, the state elements that we have discussed so far are either properties of the graph nodes (commodity values) or of the edges (such as flow constraints). This difference is of critical importance in our graph neural network architecture.

Generally, the choice of state elements will depend on the information available to a system designer (what can be measured) and on the particular problem setting. Possible examples of further state elements include forecasts of prices, demand and supply, or constraints at future times.

**Action and action space.** As discussed in Section 3, an action is defined as all flows and exchanges, $a^t = (f^t, w^t)$. In the following subsections, we accurately describe the action parametrization under the bi-level formulation.

**Dynamics.** The dynamics of the MDP, $P$, describe the evolution of state elements. We split our discussion into two parts: the dynamics associated with commodity and non-commodity elements.

The commodity dynamics are assumed to be reasonably well-modeled by the conservation of flow (1), subject to the constraints; this forms the basis of the bi-level approach that we describe in the next subsection.

The non-commodity dynamics are assumed to be substantially more complex. For example, buying and selling prices may have a complex dependency on past sales, current demand, current supply (commodity values), as well as random exogenous factors. Thus, we place few assumptions on the evolution of non-commodity dynamics and assume that current values are measurable.

**Reward.** We assume that our reward is the total discounted money earned over the problem duration. This results in a stage-wise reward function that corresponds to the money earned in that time period, or $R(s^t, a^t) = m_e^t + m_f^t$.

### 4.2. The Bi-Level Formulation

The previous subsection presented a general MDP formulation that represents a broad class of relevant network optimization problems. The goal is to find a policy $\tilde{\pi}^* \in \tilde{\Pi}$ (where $\tilde{\Pi}$ is the space of realizable Markovian policies) such that $\tilde{\pi}^* \in \arg\max_{\tilde{\pi} \in \tilde{\Pi}} \mathbb{E}_\tau \left[\sum_{t=0}^{\infty} \gamma^t R(s^t, a^t)\right]$, where $\tau = (s^0, a^0, s^1, a^1, \ldots)$ denotes the trajectory of states and actions. This formulation requires specifying a distribution over all flow/exchange actions, which may be an extremely large space. We instead consider a bi-level formulation

$$\pi^* \in \arg\max_{\pi \in \Pi} \mathbb{E}_\tau \left[\sum_{t=0}^{\infty} \gamma^t R(s^t, a^t)\right] \tag{8}$$

$$\text{s.t. } a^t = \text{LCP}(\hat{s}^{t+1}, s^t), \tag{9}$$

where we compute $a^t$ by replacing a single policy that maps from states to actions (i.e., $s^t \to a^t$) with two nested policies, mapping from states to desired next states to actions (i.e., $s^t \to \hat{s}^{t+1} \to a^t$). As a consequence of this formulation, the desired next state $\hat{s}^{t+1}$ acts as an intermediate variable, thus avoiding the direct parametrization of an extremely large action space, e.g., flows over edges in a graph. This desired next state is then used in a linear control problem (LCP$(\cdot,\cdot)$), which leverages a (slightly modified) one-step version of the linear problem formulation of Section 3 to map from desired next state to action. Thus, the resulting formulation is a bi-level optimization problem, whereby the policy $\tilde{\pi}$ is the composition of the policy $\pi(\hat{s}^{t+1} \mid s^t)$ and the solution to the linear control problem. Specifically, given a sample of $\hat{s}^{t+1}$ from the stochastic policy, we select flow and exchange actions by solving

$$\begin{aligned}
\arg\min_{a^t} \quad & d(\hat{s}^{t+1}, s^{t+1}) - R(s^t, a^t) && \text{(10a)}\\
\text{s.t.} \quad & \text{Conservation of flow (1); Net flow (3);} && \text{(10b)}\\
& \text{Reward (4); Exchange conditions (5);} && \text{(10c)}\\
& \text{Other constraints, e.g. (6) or (7)} && \text{(10d)}
\end{aligned}$$

where $d(\cdot,\cdot)$ is a convex metric which penalizes deviation from the desired next state. The resultant problem is convex and thus may be easily and inexpensively solved to choose actions $a^t$, even for very large problems. Please see Appendix B.2, C for a broader discussion.

As is standard in reinforcement learning, we will aim to solve this problem via learning the policy from data. This may be in the form of online learning (Sutton & Barto, 1998) or via learning from offline data (Levine et al., 2020). There are large bodies of work on both problems, and our presentation will generally aim to be as-agnostic-as-possible to the underlying reinforcement learning algorithm used. Of critical importance is the fact that the majority of reinforcement learning algorithms use likelihood ratio gradient estimation (Williams, 1992), which does not require path-wise back-propagation through the inner problem.

We also note that our formulation assumes access to a model (the linear problem) that is a reasonable approximation of the true dynamics over short horizons. This short-term correspondence is central to our formulation: we exploit exact optimization when it is useful, and otherwise push the impacts of the nonlinearity over time to the learned policy. We assume this model is known in our experiments—which we feel is a reasonable assumption across the problem settings we investigate—but it could be learned from state transitions or as learnable parameters in policy learning.

### 4.3. Architectural Considerations

After having introduced the problem formulation and a general framework to control graph-structured systems from experience, here and in Appendix B.1, we broaden the discussion on specific algorithmic components.

**Network architectures.** We argue that graph networks represent a natural choice for network optimization problems because of three main properties. First, permutation invariance. Crucially, non-permutation invariant computations would consider each node ordering as fundamentally different and thus require an exponential number of input/output training examples before being able to generalize. Second, locality of the operator. GNNs typically express a local parametric filter (e.g., convolution operator) which enables the same neural network to be applied to graphs of varying size and connectivity and achieve non-parametric expansibility. This is a property of fundamental importance for many real-world graph control problems, which will be dynamic or frequently re-configured, and it is desirable to be able to use the same policy without re-training. Lastly, alignment with the computations used for network optimization problems. As shown in (Xu et al., 2020), GNNs can better match the structure of many network optimization algorithms and are thus likely to achieve better performance.

**Action parametrization.** Let us consider the problem of controlling flows in a network. We are interested in defining a desired next state $\hat{s}^{t+1}$ that is ideally (i) lower dimensional, (ii) able to capture relevant aspects for control, and (iii) as-robust-as-possible to domain shifts. At a high level, we achieve this by avoiding the direct parametrization of per-edge desired flow values and compute per-node desired inflow quantities. Concretely, given the total availability $M$ of commodity units in the graph, we define $\hat{s}^{t+1} = \{\hat{q}_i^{t+1}\}_{i\in\mathcal{V}}, \sum_i \hat{q}_i^{t+1} = M$ as a desired per-node number of commodity units. We do so by first determining $\tilde{q}_i^{t+1} = \{\tilde{q}_i^{t+1}\}_{i\in\mathcal{V}}$, where $\tilde{q}_i^{t+1} \in [0,1]$ defines the percentage of currently available commodity units to be moved to node $i$ in time step $t$, and $\sum_{i\in\mathcal{V}} \tilde{q}_i^{t+1} = 1$. We then use this to compute $\hat{q}_i^{t+1} = \lfloor \tilde{q}_i^{t+1} \cdot M \rfloor$ as the actual number of commodity units. In practice, we achieve this by defining the intermediate policy as a Dirichlet distribution over nodes, i.e., $\pi(\hat{s}^{t+1}|s^t) = \tilde{q}^{t+1} \cdot M, \tilde{q}^{t+1} \sim \text{Dir}(\tilde{q}^{t+1}|s^t)$. Crucially, the representation of the desired next state via $\hat{q}_i$ (i) is lower-dimensional as it only acts over nodes in the graph, (ii) uses a meaningful aggregated quantity to control flows, and (iii) is scale-invariant by construction as it acts on *ratios* opposed to raw commodity quantities. Additionally, for problems that require a generation of commodities (e.g., products in a supply chain), we define the desired next state via the exchange weights introduced in Eq (5), $\hat{s}^{t+1} = \{w_i^{t+1}\}_{i\in\mathcal{V}}, w_i^{t+1} \in \mathbb{N}^+$, with $w_i^{t+1}$ representing the number of commodity units to generate. In practice, this can be achieved by defining the intermediate policy as a Gaussian distribution over nodes (followed by rounding), i.e., $\pi(\hat{s}^{t+1}|s^t) = \text{round}(w^{t+1}), w^{t+1} \sim \mathcal{N}(w^{t+1}|s^t)$.

Table 1: Percentage of oracle performance on different minimum cost flow scenarios.

| | Random | MLP-RL | GCN-RL | GAT-RL | MPNN-RL | Oracle |
|---|---|---|---|---|---|---|
| 2-hops | 9.9% ±4.8% | 60.2% ±2.1% | 31.3% ±1.3% | 22.9% ±1.1% | **89.7% ±0.9%** | - |
| 3-hops | 50.3% ±8.4% | 53.8% ±1.6% | 68.7% ±2.0% | 62.4% ±1.9% | **89.5% ±1.1%** | - |
| 4-hops | 63.1% ±3.9% | 67.8% ±2.5% | 71.4% ±1.7% | 68.2% ±2.3% | **87.1% ±1.2%** | - |
| Dynamic travel time | -23.4% ±4.3% | -0.7% ±1.7% | 18.7% ±2.0% | 17.1% ±1.6% | **99.1% ±1.3%** | - |
| Dynamic topology | 42.5% ±6.8% | N/A | 53.4% ±2.8% | 43.4% ±3.1% | **83.9% ±1.0%** | - |
| Multi-commodity | 22.5% ±8.2% | 41.7% ±3.2% | 33.8% ±2.1% | 33.0% ±1.7% | **72.0% ±1.6%** | - |
| Capacity (Success Rate) | 62.6% (82%) | 62.7% (82%) | 65.2% (87%) | 62.9% (80%) | **89.8% (87%)** | - (88%) |

## 5. Experiments

In this section, we first consider an artificial *minimum cost flow problem* as a simple graph control problem that illustrates the basic principles of our formulation and investigates architectural components (Section 5.1). We further assess the versatility of our framework by applying it to two distinct real-world network problems: the *supply chain inventory management* problem (Section 5.2) and the *dynamic vehicle routing* problem (Section 5.2). Specifically, these problems represent two instantiations of economically-critical graph control problems where the task is to control flows of quantities (i.e., packages and vehicles, respectively), generate commodities (i.e., products within a supply chain), or both.

**Experimental design.** While the specific benchmarks will necessarily depend on the individual problem, in all real-world experiments, we will always compare against the following *classes* of methods: (i) an *Oracle* benchmark characterized as an MPC controller which has access to perfect information of all future states of the system and can thus plan for the perfect action sequence, (ii) a *Domain-driven Heuristic*, i.e., algorithms which are generally accepted as go-to approaches for the types of problems we consider, and (iii) a *Randomized* heuristic to quantify a reasonable lower-bound of performance within the environment.

### 5.1. Minimum Cost Flow

Let us consider an artificial minimum cost flow problem where the goal is to control commodities from one or more source nodes to one or more sink nodes, in the minimum time possible. We assess the capability of our formulation to handle several practically-relevant situations. Specifically, we do so by comparing different versions of our method against an oracle benchmark to investigate the effect of different neural network architectures. Results in Table 1 and in Appendix D.1.3, show how graph-RL approaches are able to achieve close-to-optimal performance in all proposed scenarios while greatly reducing the computation cost compared to traditional solutions (Figure 2 and Appendix C.2)[3]. Among all formulations (please refer to Appendix D.1 for additional details), MPNN-RL is clearly the best performing architecture, achieving $86.7\%$ of oracle performance, on average. As discussed in Section 4.3, this highlights the importance of the algorithmic alignment (Xu et al., 2020) between the neural network architecture and the nature of the computations needed to solve the task. Crucially, results show how our formulation is able to operate reliably within a broad set of situations, ranging from scenarios characterized by dynamic travel times (*Dyn. travel time*), dynamic topologies, i.e., with nodes and edges that can be removed or added during an episode (*Dyn. topology*), capacitated-networks (*Capacity*) with different depth (*2-hop*, *3-hop*, *4-hop*), and multi-commodity problems (*Multi-commodity*).

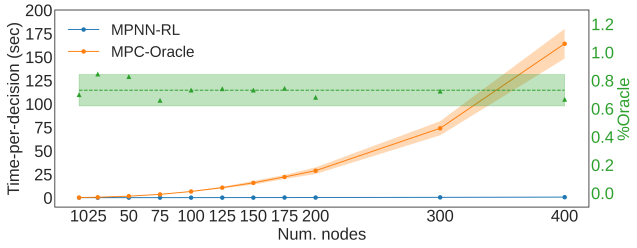



Figure 2: Comparison of computation times between learning-based (blue) and control-based (orange) approaches. Green triangles represent the percentage performance of our RL framework compared to the oracle model.

### 5.2. Supply Chain Inventory Management (SCIM)

In our first real-world experiment, we aim to optimize the performance of a supply chain inventory system. Specifically, this describes the problem of ordering and shipping product inventory within a network of interconnected warehouses and stores in order to meet customer demand while simultaneously minimizing storage and transportation costs. A supply chain system is naturally expressed via a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V} = \mathcal{V}_S \cup \mathcal{V}_W$ is the set of both store $\mathcal{V}_S$ and warehouse $\mathcal{V}_W$ nodes, and $\mathcal{E}$ the set of edges connecting stores to warehouses. Demand $d_i^t$ materializes in stores $i \in \mathcal{V}_S$ at each period $t$. If inventory is available at the store, it is used to meet customer demand and sold at a price $p$. Unsatisfied orders are maintained over time and are represented as a negative stock (i.e., backorder). At each time step, the warehouse orders additional units of inventory $w_i$ from the manufacturers and stores available ones. As commodities travel across the network, they are delayed by transportation times $t_{ij}$. Both warehouses and storage facil-

[3]All methods used the same computational CPU resources, namely a AMD Ryzen Threadripper 2950X (16-Core, 32 Thread, 40M Cache, 3.4 GHz base).

Table 2: System performance on real-world SCIM experiments.

| | Avg. Prod. | S-type Policy | End-to-End RL (MLP/GNN) | Graph-RL (ours) | Oracle |
|---|---|---|---|---|---|
| 1F 2S | -20,334 (± 4,723) | -4,327 (± 251) | -1,832 (± 352) / -17 (± 89) | **192 (± 119)** | 852 (± 152) |
| % Oracle | 0.0% | 75.5% | 87.3% / 95.8% | **96.8%** | 100.0% |
| 1F 3S | -53,113 (± 7,231) | -5,650 (± 298) | -4,672 (± 258) / -810 (± 258) | **997 (± 109)** | 3,249 (± 102) |
| % Oracle | 0.0% | 84.2% | 85.9% / 92.7% | **96.0%** | 100.0% |
| 1F 10S | -114,151 (± 4,611) | -14,327 (± 365) | -587,887 (± 5,255) / -568,374 (± 5,255) | **890 (± 288)** | 1,358 (± 460) |
| % Oracle | 0.0% | 86.4% | N.A. / N.A. | **99.5%** | 100.0% |

ities have limited storage capacities $c_i$, such that the current inventory $q_i$ cannot exceed it. The system incurs a number of operations-related costs: storage costs $m_i^S$, production costs $m_i^O$, backorder costs $m_i^B$, transportation costs $m_{ij}^T$.

**SCIM Markov decision process.** To apply the methodologies introduced in Section 4, we formulate the SCIM problem as an MDP characterized by the following elements (please refer to Appendix D.2.2 for a formal definition):

*Action space ($\mathcal{A}$):* we consider the problem of determining (1) the amount of additional inventory $w_i$ to order from manufacturers in all warehouse nodes $i \in \mathcal{V}_W$, and (2) the flow $f_{ij}$ of commodities to be shipped from warehouses to stores, such that $\mathbf{a}^t = \{w_i^t\}_{i\in\mathcal{V}_W} \cup \{f_{ij}^t\}_{(i,j)\in\mathcal{E}}$.

*Reward ($R(s^t, a^t)$):* we select the reward function in the MDP as the profit of the inventory manager, computed as the difference between sales revenues and costs.

*State space ($\mathcal{S}$):* the state space describes the current status of the supply network, via node and edge features. Node features contain information on (i) current inventory, (ii) current and estimated demand, (iii) incoming flow, and (iv) incoming orders. Edge features are characterized by (i) travel time $t_{ij}$, and (ii) transportation cost $m_{ij}^T$.

**Bi-Level formulation.** In what follows and in Appendix D.2.4, we illustrate a specific instantiation of our framework for the SCIM problem. We define the desired outcome $\hat{s}^{t+1}$ as being characterized by two elements: (i) the desired production in warehouse nodes $\hat{w}_i^{t+1}, \forall i \in \mathcal{V}_W$, and (ii) a desired inventory in store nodes $\hat{q}_i^{t+1}, \forall i \in \mathcal{V}_S$.

The LCP selects flow and production actions to best achieve $\hat{s}^{t+1}$ via distance minimization between desired and actual inventory levels. The LCP is further defined by domain-related constraints, such as ensuring that the inventory in store and warehouse nodes does not exceed storage capacity and that shipped products are non-negative and upper bounded by inventory.

**Inventory management via graph control.** For the SCIM problem, we define the domain-driven heuristic as a prototypical S-type (or "order-up-to") policy, which is generally accepted as an effective heuristic (Van Roy et al., 1997). Appendix D.2 provides further experimental details.

Concretely, we measure overall system performance on three different supply chain networks characterized by increasing complexity. Results in Table 2 show that our framework achieves close-to-optimal performance in all tasks. Specifically, Graph-RL achieves 96.8% (1F2S), 96% (1F3S), and 99.5% (1F10S) of oracle performance. Qualitatively, Figure 3 highlights how Graph-RL learns to control the production and shipping policies to match consumer demand while maintaining low inventory storage. More subtly, Figure 3 shows how policies learned through Graph-RL manage to anticipate demand so that products are promptly available in stores by taking production and shipping time under consideration. Results in Table 2 also show how S-type policies, despite being explicitly fine-tuned for all tasks, are largely inefficient and thus incur unnecessary costs and revenue losses, resulting in a profit gap of approximately 15% compared to Graph-RL, on average.

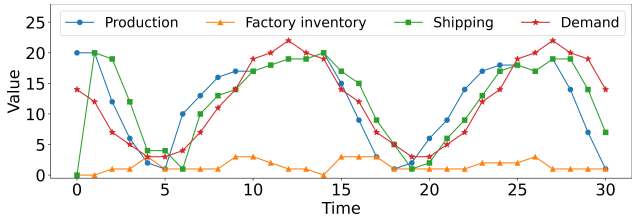
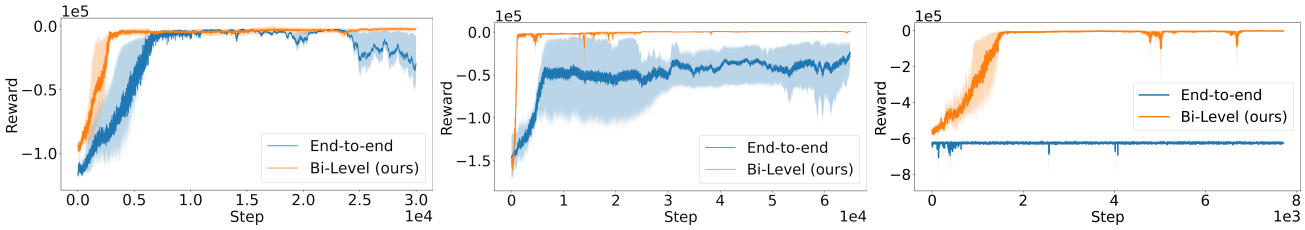



Figure 3: Aggregate behavior of the Graph-RL policies on a test episode for the 1F2S SCIM environment.

**LCP as inductive bias for network computations.** As a further analysis, we compare with an ablation of our framework, which, as in the majority of literature, is defined as a purely end-to-end RL agent that avoids the LCP and directly maps from environment states to production and shipping actions through either MLPs (Peng et al., 2019; Oroojlooyjadid et al., 2022) or GNNs. Results in Figure 4 clearly highlight how the bi-level formulation exhibits significantly improved sample efficiency and performance compared to its end-to-end counterpart, which is either substantially slower at converging to good-quality solutions or does not converge at all, as in Figure 4 (c). We argue that this behavior is due to two main factors: (1) the bi-level agent operates on a lower-dimensional and well-structured representation via $\hat{s}^{t+1}$, and (2) the bi-level formulation provides an implicit inductive bias towards feasible, high-quality solutions via the definition of the LCP. Together, these two properties define an RL agent that exhibits improved efficiency and performance.

Table 3: System performance on real-world DVR experiments.

| | Random | Evenly-balanced System | End-to-end RL | Graph-RL (ours) | Oracle |
|---|---|---|---|---|---|
| New York | -10,778 (± 659) | 9,037 (± 797) | -6,043 (± 2,584) | **15,481 (± 397)** | 16,867 (± 547) |
| % Oracle | 0.0% | 71.6% | 17.2% | **94.9%** | 100.0% |
| Shenzhen | 19,406 (± 1,894) | 29,826 (± 706) | 18,889 (± 1,207) | **36,918 (± 616)** | 40,332 (± 724) |
| % Oracle | 0.0% | 50.1% | -0.02% | **83.8%** | 100.0% |
| Zero Shot NY→SHE | - | - | 18,568 (± 1,358) | 36,100 (± 657) | - |
| Zero Shot SHE→NY | - | - | -4,083 (± 1,278) | 14,495 (± 426) | - |




(a) (b) (c)

Figure 4: Learning curve comparison between an RL agent trained end-to-end (blue) and via our bi-level formulation (orange) on the SCIM task (a) 1F2S (b) 1F3S, and (c) 1F10S

### 5.3. Dynamic Vehicle Routing

In the second real-world experiment, we apply our framework to the field of mobility. Specifically, we focus on the dynamic vehicle routing (DVR) problem, which describes the task of finding the least-cost routes for a fleet of vehicles such that it can satisfy the demand of a set of customers geographically dispersed in a dynamic, stochastic network. Towards this aim, we consider a transportation network $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ with $M$ single-occupancy vehicles, where $\mathcal{V}$ represents the set of stations (e.g., pickup or drop-off locations) and $\mathcal{E}$ represents the set of links in the transportation network (e.g., roads), each characterized by a travel time $t_{ij}$ and cost $m_{ij}$. At each time step, customers arrive at their origin stations and wait for idle vehicles $q_i$ to transport them to their destinations. The trip from station $i \in \mathcal{V}$ to station $j \in \mathcal{V}$ at time $t$ is characterized by a demand $d^t_{ij}$ and a price $p_{ij}$, passengers not served by any vehicle will leave the system and revenue from their trips will be lost. The system operator coordinates a fleet of vehicles to best serve the demand for transportation while minimizing the cost of operations. Concretely, the operator achieves this by controlling the passenger flow $f^t_{ij,P}$ (i.e., vehicles delivering passengers to their destination) and the rebalancing flow $f^t_{ij,R}$ (i.e., vehicles not assigned to passengers and used, for example, to anticipate future demand) at each time step $t$. Please refer to Appendix D.3 for further details.

**DVR Markov decision process.** We formulate the DVR MDP through the following elements:

*Action space ($\mathcal{A}$):* we compute the rebalancing flow $f^t_{ij,R}$, such that $\mathbf{a}^t = \{f^t_{ij,R}\}_{(i,j)\in\mathcal{E}}$. Without loss of generality, we assume the passenger flow is assigned through some independent routine, although the ideas described in this section can be extended to include also passenger flows.

*Reward ($R(s^t, a^t)$):* we select the reward function in the MDP as the operator profit, computed as the difference between trip revenues and operation-related costs.

*State space ($\mathcal{S}$):* the transportation network is described via node features such as the current and projected availability of idle vehicles in each station, current and estimated demand, and provider-level information, e.g., trip price.

**Bi-Level formulation.** We further describe an additional instantiation of our bi-level framework for the DVR problem. First, we define the desired next state $\hat{s}^{t+1}$ to represent the desired number of idle vehicles in all stations $\hat{q}^t_i, \forall i \in \mathcal{V}$. The second step further entails the solution of the LCP to transform the desired number of idle vehicles into feasible environment actions (i.e., rebalancing flows). At a high level, the LCP aims to minimize rebalancing costs while satisfying domain-related constraints such as ensuring that the total rebalancing flow from a region is upper-bounded by the number of idle vehicles in that region and non-negative. Please refer to Appendix D.3.4 for further details.

**Vehicle routing via network flow.** We evaluate the algorithms on two real-world urban mobility scenarios based on the cities of New York, USA, and Shenzhen, China. Results in Table 3 show how Graph-RL is able to achieve close-to-optimal performance in both environments. Specifically, the vehicle routing policies learned through Graph-RL achieve 94.9% (New York) and 83.8% (Shenzhen) of oracle performance, while showing a 23.3% (New York) and 33.7% (Shenzhen) increase in operator profit compared to the domain-driven heuristic, which attempts to preserve equal access to vehicles across stations in the transportation network. As observed for SCIM problems, the results

Table 4: Impact of implicit planning via desired next states.

| | | | Greedy (i.e., $\arg\min_{a^t} -R(s^t, a^t)$) | Graph-RL (i.e., $\arg\min_{a^t} d(\hat{s}^{t+1}, s^{t+1}) - R(s^t, a^t)$) |
|---|---|---|---|---|
| SCIM | 1F2S | Reward | -102,919 ($\pm$ 2,767) | **192** ($\pm$ **119**) |
| | | %Oracle | N.A. | **96.8%** |
| | 1F3S | Reward | -169,433 ($\pm$ 2,880) | **997** ($\pm$ **109**) |
| | | %Oracle | N.A. | **96.0%** |
| | 1F10S | Reward | -587,661 ($\pm$ 3,862) | **890** ($\pm$ **288**) |
| | | %Oracle | N.A. | **99.5%** |
| DVR | New York | Reward | 13,978 ($\pm$ 391) | **15,481** ($\pm$ 397) |
| | | Served Demand | 1,357 ($\pm$ 92) | **1,824** ($\pm$ 87) |
| | | %Oracle | 90.13% | **94.9%** |
| | Shenzhen | Reward | 35,996 ($\pm$ 499) | **36,918** ($\pm$ 616) |
| | | Served Demand | 2,881 ($\pm$ 98) | **3,310** ($\pm$ 92) |
| | | %Oracle | 79.27% | **83.9%** |

confirm that end-to-end RL approaches struggle with high-dimensional action spaces (75 and 90 edges in New York and Shenzhen environments, respectively) and fail to learn effective routing strategies. Lastly, to assess the transferability and generalization capabilities of Graph-RL, we study the extent to which policies can be trained on one city and later applied to the other without further training (i.e., zero-shot). Table 3 shows that routing policies learned in one city exhibit a promising degree of portability to novel environments, with only minimal performance decay. As introduced in Section 4.3, this experiment further highlights the importance of the locality of graph network-based policies: by learning a shared, local operator, policies learned through graph-RL can potentially be applied to arbitrary graph topologies. Crucially, policies with structural transfer capabilities could enable system operators to re-use previous experience, thus avoiding expensive re-training when exposed to new problem instances.

### 5.4. Comparison to Greedy Planning

The role of the distance metric (and the generated desired next state) in Eq. (10a) is to capture the value of future reward in the greedy one-step inner optimization problem, ultimately allowing for implicit long-term planning (please refer to Appendix C.1 for a broader discussion). To quantify this intuition, in Table 4 we compare the proposed bi-level approach to a *greedy* policy that acts optimally with respect to the one-step optimization problem. Concretely, if on one hand the proposed bi-level approach attempts to achieve as best as possible the desired next state (i.e., $\arg\min_{a^t} d(\hat{s}^{t+1}, s^{t+1}) - R(s^t, a^t)$), the greedy policy ignores the distance term and optimizes solely short-term reward (i.e., $\arg\min_{a^t} -R(s^t, a^t)$)). Results in Table 4 highlight how the presence of the desired next state, and ultimately, of the bi-level approach, is instrumental in achieving effective long-term performance. Crucially, since both producing a commodity (SCIM) and rebalancing a vehicle (DVR) are only defined by negative rewards, these only indirectly participate to long-term positive reward via a better (i) product availability or (ii) positioning of vehicles, and thus cannot be measured by the one-step optimization problem. This results in the greedy policy (i) being unable to fulfill any demand in the SCIM problem and (ii) achieving lower profit in the DVR problem. It is important to highlight how, in the DVR problem, the greedy policy achieves reasonably good reward (i.e., profit) because the system can partially self-sustain itself only through passenger trips. However, greediness causes the number of served customers to be considerably smaller, with Graph-RL achieving $\approx$ +35% in New York and $\approx$ +15% in Shenzhen, thus clearly showing the benefit of optimizing for long-term reward via the minimization of the distance metric.

## 6. Conclusion

Research in network optimization problems, in both theory and practice, is largely scattered across the control, management science, and optimization literature, potentially hindering scientific progress. In this work, we propose a general framework that could enable learning-based approaches to help address the open challenges in this space: handling nonlinear dynamics and scalability, among others. Specifically, instead of approaching the problem through pure end-to-end reinforcement learning, we introduced a general bi-level formulation that leverages the specific strengths of direct optimization, reinforcement learning, and graph representation learning. Our approach shows strong performance on all problem settings we evaluate, substantially outperforming both optimization-based and RL-based approaches. In future work, we plan to investigate ways to exploit the non-parametric nature of our approach and take a step in the direction of learning generalist graph optimizers. More generally, we believe this research opens several promising directions for the extension of these concepts to a broader class of large-scale, real-world applications.

## References


Agrawal, A., Barratt, S., Boyd, S., Busseti, E., and Moursi, W. M. Differentiating through a conic program. *Journal of Applied and Numerical Optimization*, 1(2):107–115, 2019a.

Agrawal, A., Barratt, S., Boyd, S., and Stellato, B. Learning convex optimization control policies. In *Learning for Dynamics & Control*, 2019b.

Amos, B. and Kolter, J. Z. OptNet: Differentiable optimization as a layer in neural networks. In *Int. Conf. on Machine Learning*, 2017.

Amos, B. and Yarats, D. The differentiable cross-entropy method. In *Int. Conf. on Machine Learning*, pp. 291–302, 2020.

Amos, B., Jimenez, I., Sacks, J., Boots, B., and Kolter, J. Z. Differentiable mpc for end-to-end planning and control. *Conf. on Neural Information Processing Systems*, 31, 2018.

Astrom, K. J. *Introduction to stochastic control theory*. Courier Corporation, 2012.

Bellamy, M. A. and Basole, R. C. Network analysis of supply chain systems: A systematic review and future research. *Systems Engineering*, 16(2):235–249, 2013.

Bertsekas, D. *Dynamic programming and optimal control*. Athena Scientific, first edition, 1995.

Bertsekas, D. *Reinforcement learning and optimal control*. Athena Scientific, 2019.

Bertsekas, D. and Tsitsiklis, J. N. *Neuro-dynamic programming*. Athena Scientific, 1996.

Bienstock, D., Chertkov, M., and Harnett, S. Chance-constrained optimal power flow: Risk-aware network control under uncertainty. *SIAM Review*, 56(3):461–495, 2014.

Birge, J. R. and Louveaux, F. *Introduction to stochastic programming*. Springer Science & Business Media, 2011.

Boyd, S. and Vandenberghe, L. *Convex Optimization*. Cambridge Univ. Press, 2004.

Daganzo, C.-F. An approximate analytic model of many-to-many demand responsive transportation systems. *Transportation Research*, 12(5):325–333, 1978.

Dantzig, G. B. Reminiscences about the origins of linear programming. *Operations Research Letters*, 1(2):43–48, 1982.

Dinh, Q. T. and Diehl, M. Local convergence of sequential convex programming for nonconvex optimization. In *Recent Advances in Optimization and its Applications in Engineering*. Springer, 2010.

Dommel, H. W. and Tinney, W. F. Optimal power flow solutions. *IEEE Transactions on power apparatus and systems*, (10):1866–1876, 1968.

Donti, P., Amos, B., and Kolter, J. Z. Task-based end-to-end model learning in stochastic optimization. *Conf. on Neural Information Processing Systems*, 30, 2017.

Dumouchelle, J., Patel, R., Khalil, E. B., and Bodur, M. Neur2sp: Neural two-stage stochastic programming. *arXiv:2205.12006*, 2022.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *Int. Conf. on Machine Learning*, 2017.

Flood, J. E. *Telecommunication networks*. IET, 1997.

Ford, L. R. and Fulkerson, D. R. Maximal flow through a network. *Canadian journal of Mathematics*, 8:399–404, 1956.

Ford, L. R. and Fulkerson, D. R. Constructing maximal dynamic flows from static flows. *Operations Research*, 6 (3):419–433, 1958.

Ford, L. R. and Fulkerson, D. R. *Flows in Networks*. Princeton Univ. Press, 1962.

Fujimoto, S., Meger, D., Precup, D., Nachum, O., and Gu, S. S. Why should i trust you, bellman? the bellman error is a poor replacement for value error. *arXiv:2201.12417*, 2022.

Gammelli, D., Yang, K., Harrison, J., Rodrigues, F., Pereira, F. C., and Pavone, M. Graph neural network reinforcement learning for autonomous mobility-on-demand systems. In *Proc. IEEE Conf. on Decision and Control*, 2021.

Gammelli, D., Yang, K., Harrison, J., Rodrigues, F., Pereira, F., and Pavone, M. Graph meta-reinforcement learning for transferable autonomous mobility-on-demand. In *ACM Int. Conf. on Knowledge Discovery and Data Mining*, 2022.

Gilmer, J., Schoenholz, S., Riley, P., Vinyals, O., and Dahl, G. Neural message passing for quantum chemistry. In *Int. Conf. on Machine Learning*, 2017.

Glynn, P. W. Likelihood ratio gradient estimation for stochastic systems. *Communications of the ACM*, 33 (10):75–84, 1990.

Harrison, J., Sharma, A., and Pavone, M. Meta-learning priors for efficient online bayesian regression. In *Workshop on Algorithmic Foundations of Robotics*, pp. 318–337, 2018.

Hillier, F. and Lieberman, G. *Introduction to operations research*. 1995.

Huneault, M. and Galiana, F. D. A survey of the optimal power flow literature. *IEEE transactions on Power Systems*, 6(2):762–770, 1991.

IBM. *ILOG CPLEX User's guide*. IBM ILOG, 1987.

Ichter, B., Harrison, J., and Pavone, M. Learning sampling distributions for robot motion planning. In *Proc. IEEE Conf. on Robotics and Automation*, pp. 7087–7094, 2018.

Jakobson, G. and Weissman, M. Real-time telecommunication network management: Extending event correlation with temporal constraints. In *International Symposium on Integrated Network Management*, pp. 290–301, 1995.

Key, P. B. and Cope, G. A. Distributed dynamic routing schemes. *IEEE Communications Magazine*, 28(10):54–58, 1990.

Kipf, T.-N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *Int. Conf. on Learning Representations*, 2017.

Konda, V. and Tsitsiklis, J. Actor-critic algorithms. In *Conf. on Neural Information Processing Systems*, 1999.

Landry, B., Lorenzetti, J., Manchester, Z., and Pavone, M. Bilevel optimization for planning through contact: A semidirect method. In *The International Symposium of Robotics Research*, pp. 789–804, 2019.

Levine, S., Kumar, A., Tucker, G., and Fu, J. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv:2005.01643*, 2020.

Lew, T., Singh, S., Prats, M., Bingham, J., Weisz, J., Holson, B., Zhang, X., Sindhwani, V., Lu, Y., Xia, F., et al. Robotic table wiping via reinforcement learning and whole-body trajectory optimization. *arXiv preprint arXiv:2210.10865*, 2022.

Li, F. and Bo, R. Dcopf-based lmp simulation: algorithm, comparison with acopf, and sensitivity. *IEEE Transactions on Power Systems*, 22(4):1475–1485, 2007.

Metz, L., Maheswaranathan, N., Nixon, J., Freeman, D., and Sohl-Dickstein, J. Understanding and correcting pathologies in the training of learned optimizers. In *Int. Conf. on Machine Learning*, pp. 4556–4565, 2019.

Metz, L., Harrison, J., Freeman, C. D., Merchant, A., Beyer, L., Bradbury, J., Agrawal, N., Poole, B., Mordatch, I., Roberts, A., et al. Velo: Training versatile learned optimizers by scaling up. *arXiv preprint arXiv:2211.09760*, 2022.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Mnih, V., Puigdomenech, A., Mirza, M., Graves, A., Lillicrap, T.-P., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *Int. Conf. on Learning Representations*, 2016.

Murota, K. *Matrices and Matroids for Systems Analysis*. Springer Science & Business Media, 1 edition, 2009.

Oroojlooyjadid, A., Nazari, M., Snyder, L. V., and Takáč, M. A deep q-network for the beer game: Deep reinforcement learning for inventory optimization. *Manufacturing and Service Operations Management*, 24(1):285–304, 2022.

Paszke, A., Gross, S., Massa, F., Lerer, A., et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

Peng, Z., Zhang, Y., Feng, Y., Zhang, T., Wu, Z., and Su, H. Deep reinforcement learning approach for capacitated supply chain optimization under demand uncertainty. In *2019 Chinese Automation Congress (CAC)*, 2019.

Pereira, M. V. and Pinto, L. M. Multi-stage stochastic optimization applied to energy planning. *Mathematical Programming*, 52(1):359–375, 1991.

Popovskij, V., Barkalov, A., and Titarenko, L. *Control and adaptation in telecommunication systems: Mathematical Foundations*, volume 94. Springer Science & Business Media, 2011.

Powell, W. B. *Reinforcement Learning and Stochastic Optimization: A unified framework for sequential decisions*. Wiley, 2022.

Power, T. and Berenson, D. Variational inference mpc using normalizing flows and out-of-distribution projection. *arXiv:2205.04667*, 2022.

Rawlings, J. and Mayne, D. *Model predictive control: Theory and design*. Nob Hill Publishing, 2013.

Sacks, J. and Boots, B. Learning to optimize in model predictive control. In *Proc. IEEE Conf. on Robotics and Automation*, pp. 10549–10556, 2022.

Sarimveis, H., Patrinos, P., Tarantilis, C. D., and Kiranoudis, C. T. Dynamic modeling and control of supply chain systems: A review. *Computers & operations research*, 35 (11):3530–3561, 2008.

Shapiro, A., Dentcheva, D., and Ruszczyński, A. *Lectures on stochastic programming: Modeling and theory*. SIAM, second edition, 2014.

Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. MIT Press, 1 edition, 1998.

Tamar, A., Thomas, G., Zhang, T., Levine, S., and Abbeel, P. Learning from the hindsight plan—episodic mpc improvement. In *Proc. IEEE Conf. on Robotics and Automation*, pp. 336–343, 2017.

Van de Wiele, T., Warde-Farley, D., Mnih, A., and Mnih, V. Q-learning in enormous action spaces via amortized approximate maximization. *arXiv:2001.08116*, 2020.

Van Roy, B., Bertsekas, D. P., Lee, Y., and Tsitsiklis, J. N. A neuro-dynamic programming approach to retailer inventory management. In *Proc. IEEE Conf. on Decision and Control*, 1997.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, O., and Bengio, Y. Graph attention networks. In *Int. Conf. on Learning Representations*, 2018.

Wang, Y., Szeto, W. Y., Han, K., and Friesz, T. Dynamic traffic assignment: A review of the methodological advances for environmentally sustainable road transportation applications. *Transportation Research Part B: Methodological*, 111:370–394, 2018.

Williams, R.-J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 1992.

Xiao, X., Zhang, T., Choromanski, K. M., Lee, T.-W. E., Francis, A., Varley, J., Tu, S., Singh, S., Xu, P., Xia, F., Takayama, L., Frostig, R., Tan, J., Parada, C., and Sindhwani, V. Learning model predictive controllers with real-time attention for real-world navigation. In *Conf. on Robot Learning*, 2022.

Xu, K., Li, J., Zhang, M., Du, S., Kawarabayashi, K., and Jegelka, S. What can neural networks reason about? In *Int. Conf. on Learning Representations*, 2020.

Zhang, R., Rossi, F., and Pavone, M. Model predictive control of Autonomous Mobility-on-Demand systems. In *Proc. IEEE Conf. on Robotics and Automation*, 2016.

# A. Dynamic Network Control

In this section, we make concrete our discussion on nonlinear problem formulations for network control problems.

**Elements violating the linearity assumption** Real-world systems are characterized by many factors that cannot be reliably modeled through the linear problem described in Section 3. In what follows, we discuss a (non-exhaustive) list of factors potentially breaking such linearity assumptions:

- **Stochasticity.** Various stochastic elements can impact the problem. Commodity transitions in Section 3.1 were defined as being deterministic; in practice in many problems, there are elements of stochasticity to these transitions. For example, random demand may reduce supply by an unpredictable amount; vehicles may be randomly added in a transportation problem; and packages may be lost in a supply chain setting. In addition to these state transitions, constraints may be stochastic as well: flow times or edge capacities may be stochastic, as when a road is shared with other users, or costs for flows and exchange may be stochastic.
- **Nonlinearity.** Various elements of the state evolution, constraints, or cost function may be nonlinear. The objective may be chosen to be a risk-sensitive or robust metric applied to the distribution of outcomes, as is common in financial problems. The state evolution may have natural saturating behavior (e.g. automatic load shedding). Indeed, many real constraints will have natural nonlinear behavior.
- **Time-varying costs and constraints.** Similar to the stochastic case, various quantities may be time-varying. However, it is possible that they are time-varying in a structured way, as opposed to randomly. For example, demand for transportation may vary over the time of day, or purchasing costs may vary over the year.
- **Unknown dynamics elements.** While not a major focus of discussion in the paper up to this point, elements of the underlying dynamics may be partially or wholly unknown. Our reinforcement learning formulation is capapble of addressing this by learning policies directly from data, in contrast to standard control techniques.

# B. Methodology

In this section, we discuss network architectures and RL components more in detail.

## B.1. Network Architecture

Specifically, we first introduce the basic building blocks of our graph neural network architecture. Let us define with $\mathbf{x}_i \in \mathbb{R}^{D_\mathbf{x}}$ and $\mathbf{e}_{ji} \in \mathbb{R}^{D_\mathbf{e}}$ the $D_\mathbf{x}$-dimensional vector of node features of node $i$ and the $D_\mathbf{e}$-dimensional vector of edge features from node $j$ to node $i$, respectively.

We define the update function of node features through either:

- Message passing neural network (MPNN) (Gilmer et al., 2017) defined as

$$\mathbf{x}_i^{(k)} = \bigoplus_{j \in \mathcal{N}^-(i)} f_\theta \left( \mathbf{x}_i^{(k-1)}, \mathbf{x}_j^{(k-1)}, \mathbf{e}_{ji} \right), \tag{11}$$

  where $k$ indicates the $k$-th layer of message passing in the GNN with $k = 0$ indicating raw environment features, i.e., $\mathbf{x}_i^{(0)} = \mathbf{x}_i$, and $\bigoplus$ denotes a differentiable, permutation invariant function, e.g., sum, mean or max.
- Graph convolution network (GCN) (Kipf & Welling, 2017) defined as

$$\mathbf{X}' = f(\mathbf{X}, \mathbf{A}) = \sigma \left( \hat{\mathbf{D}}^{-\frac{1}{2}} \hat{\mathbf{A}} \hat{\mathbf{D}}^{-\frac{1}{2}} \mathbf{X} \mathbf{W} \right), \tag{12}$$

  where $\mathbf{X}$ is the $N_v \times D_\mathbf{x}$ feature matrix, $\mathbf{A}$ is the adjacency matrix with $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ and $\mathbf{I}$ is the identity matrix. $\hat{\mathbf{D}}$ is the diagonal node degree matrix of $\hat{\mathbf{A}}$, $\sigma(\cdot)$ is a non-linear activation function (e.g., ReLU) and $\mathbf{W}$ is a matrix of learnable parameters.

We select the specific architecture based on the alignment with the problem characteristics. We note that these network architectures can be used to define both policy and value function estimator, depending on the reinforcement learning algorithm of interest (e.g., actor-critic (Konda & Tsitsiklis, 1999), value-based (Mnih et al., 2015), etc.). As an example, in our implementation, we define two separate decoder architectures for the actor and critic networks of an Advantage Actor Critic (A2C) (Mnih et al., 2016) algorithm. Below is a summary of the specific architectures used in this work:

- Section 5.1. We use an MPNN as in (11), with a $\max$ aggregation function i.e., $\bigoplus = \max$. We define the output of our policy network to represent the concentration parameters $\alpha \in \mathbb{R}_+^{N_v}$ of a Dirichlet distribution, such that $\mathbf{a}_t \sim \text{Dir}(\mathbf{a}_t|\alpha)$, and where the positivity of $\alpha$ is ensured by a Softplus nonlinearity. On the other hand, the critic is characterized by a *global* sum-pooling performed after $K$ layers of MPNN.
- Section 5.2. We use an MPNN as in (11), with a sum aggregation function i.e., $\bigoplus = \text{sum}$. We define the output of our policy network to represent the (1) concentration parameters $\alpha \in \mathbb{R}_+^{|\mathcal{V}_S|}$ of a Dirichlet distribution for computing the flow actions, and (2) mean $\mu \in \mathbb{R}^{|\mathcal{V}_W|}$ and standard deviation $\sigma \in \mathbb{R}_+^{|\mathcal{V}_W|}$ of a Gaussian distribution for the production action. On the other hand, the critic is characterized by a *global* sum-pooling performed after $K$ layers of MPNN.
- Section 5.2. We use a GCN as in (12). Actor and critic outputs are defined as in the minimum cost flow problem.

**Handling dynamic topologies.** A defining property of our framework is its ability to deal with time-dependent graph connectivity (e.g., edges or nodes are added/dropped during the course of an episode). Specifically, our framework achieves this by (i) considering the problem as a one-step decision-making problem, i.e., avoiding the dependency on potentially unknown future topologies, and (ii) exploiting the capacity of GNNs to handle diverse graph topologies. Crucially, no matter the current state of the graph, GNN-based agents are capable of computing a desired next state for the network, which will then be converted into actionable flow decisions by the LCP.

### B.2. RL Details

We further discuss practical aspects within our bi-level reinforcement learning approach.

**Exploration.** In practice, we choose large penalty terms $d(\cdot, \cdot)$ to minimize greediness. However early in training, randomly initialized penalty terms can harm exploration. We found it was sufficient to down-weight the penalty term early in training. As such, the inner action selection is biased toward short-term rewards, resulting in greedy action selection. However, there are many further possibilities for exploiting random penalty functions to induce exploration, which we discuss in the next section.

**Integer-valued flows.** For several problem settings, it is desirable that the chosen flows be *integer-valued*. For example, in a transportation problem, we may wish to allocate some number of vehicles, which can not be infinitely sub-divided (Gammelli et al., 2021; 2022). There are several ways to introduce integer-valued constraints to our framework. First, we note that because the RL agent is trained through policy gradient—and thus we do not require a differentiable inner problem—we can simply introduce integer constraints into the lower-level problem[4]. However, solving integer-constrained problems is typically expensive in practice. An alternate solution is to simply use a heuristic rounding operation on the output of the inner problem. Again, because of the choice of gradient estimator, this does not need to be differentiable. Moreover, the RL policy learns to adapt to this heuristic clipping. Thus, we in general recommend this strategy as opposed to directly imposing constraints in the inner problem.

## C. Discussion and Algorithmic Components

In this section, we discuss various elements of the proposed framework, highlight correspondences and design decisions, and discuss component-level extensions.

### C.1. Distance metric as value function

The role of the distance metric (and the generated desired next state) is to capture the value of future reward in the greedy one-step inner optimization problem. This is closely related to the value function in dynamic programming and reinforcement learning, which in expectation captures the sum of future rewards for a particular policy. Indeed, under moderate technical assumptions, our linear problem formulation with stochasticity yields convex expected cost-to-go (the negative of the value) (Pereira & Pinto, 1991; Dumouchelle et al., 2022).

There are several critical differences between our penalty term and a learned value function. First, a value function in a Markovian setting for a given policy is a function solely of state. For example, in the LCP, a value function would depend only on $s^{t+1}$. In contrast, our value function depends on $\hat{s}^{t+1}$, which is the output of a policy which takes $s^t$ as

[4]Note that several problems exhibit a *total unimodularity* property (Murota, 2009), for which the relaxed integer-valued problem is tight.

an input. Thus, the penalty term is a function of both the current and desired next state. Given this, the penalty term is better understood as a local approximation of the value function, for which convex optimization is tractable, or as a form of state-action value function with a reduced action space (also referred to as a Q function).

The second major distinction between the penalty term and a value function is particular to reinforcement learning. Value functions in modern RL are typically learned via minimizing the Bellman residual (Sutton & Barto, 1998), although there is disagreement on whether this is a desirable objective (Fujimoto et al., 2022). In contrast, our policy is trained directly via gradient descent on the total reward (potentially incorporating value function control variates). Thus, the objective for this penalty method is better aligned with maximizing total reward.

### C.2. Computational efficiency

Consider solving the full nonlinear control problem via direct optimization over a finite horizon ($T$ timesteps), which corresponds to a model predictive control (Rawlings & Mayne, 2013) formulation. How many actions must be selected? The number of possible flows for a fully dense graph (worst case) is $N_v(N_v - 1)$. In addition to this, there are $\sum_{i \in \mathcal{V}} N_e(i)$ possible exchange actions; if we assume $N_e$ is the same for all nodes, this yields $N_v N_e$ possible actions. Finally, we have $N_c$ commodities. Thus, the worst-case number of actions to select is $TN_cN_v(N_v + N_e - 1)$; it is evident that for even moderate choices of each variable, the complexity of action selection in our problem formulation quickly grows beyond tractability.

While moderately-sized problems may be tractable within the direct optimization setting, we aim to incorporate the impacts of stochasticity, nonlinearity, and uncertainty, which typically results in non-convexity. The reinforcement learning approach, in addition to being able to improve directly from data, reduces the number of actions required to those for a single step. If we were to directly parameterize the naive policy that outputs flows and exchanges, this would correspond to $N_cN_v(N_v+N_e-1)$ actions. For even moderate values of $N_c, N_v, N_e$, this can result in millions of actions. It is well-known that reinforcement learning algorithms struggle with high dimensional action spaces (Van de Wiele et al., 2020), and thus this approach is unlikely to be successful. In contrast, our bi-level formulation requires only $N_c$ actions for the learned policy, while additionally leveraging the beneficial inductive biases over short time horizons.

## D. Additional Experiment Details

In this section, we provide additional details of the experimental set-up and hyperparameters. All RL modules were implemented using PyTorch (Paszke et al., 2019) and the IBM CPLEX solver (IBM, 1987) for the optimization problem.

### D.1. Minimum Cost Flow

We start by describing the properties of the environments in Section D.1.1. We further expand the discussion on model implementation (Section D.1.2), and additional results (Section D.1.3).

#### D.1.1. Environment details

We select environment variables in a way to cover a wide enough range of possible scenarios, e.g., different travel times and thus, different optimal actions.

**Generalities.** As discussed in Section 5, the environments describe a dynamic minimum cost flow problem, whereby the goal is to let commodities flow from source to sink nodes in the minimum time possible (i.e., cost is equal to time). Formally, given a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, the reward function across all environments is defined as:

$$R(s^t, a^t) = -\sum_{(i,j)\in\mathcal{E}} f_{ij}^t t_{ij} + \lambda f_{\text{sink}}^t,$$

where $f_{ij}^t$ and $t_{ij}$ represent flow and travel time along edge $(i, j)$ at time $t$, respectively, $f_{\text{sink}}^t$ is the flow arriving at all sink nodes at time $t$, and $\lambda$ is a weighting factor between the two reward terms. In our experiments, the resulting policy proved to be broadly insensitive to values of $\lambda$, with $\lambda \in [15, 30]$ typically being an effective range.

**2-hop, 3-hop, 4-hop.** Given a single-source, single-sink network, we assume travel times to be constant over the episode and requirements (i.e., demand) to be sampled at each time step as $\rho = 10 + \psi_i, \psi_i \sim \text{Uniform}[-2, 2]$. Capacities $c_{ij}$ are

fixed to a very high positive number, thus not representing a constraint in practice. Cost $m_{ij}$ is equal to the travel time $t_{ij}$. An episode is assumed to have a duration of 30 time steps and terminates when there is no more flow traversing the network. To present a variety of scenarios to the agent at training time, we sample random travel times for each new episode as $t_{ij} \sim \text{Uniform}[0, 10]$ and use the topologies shown in Fig. 6. In our experiments, we apply as many layers of message passing as hops from source to sink node in the graph, e.g., $K = 2$ and $K = 3$ in the 2-hops and 3-hops environment, respectively.

**Dynamic travel times** To train our MPNN-RL, we select the 3-hops environment and generate travel times as follows for every episode: (i) sample random travel times as $t_{ij} \sim \text{Uniform}[0, 10]$, (ii) for every time step, gradually change the travel time as $t_{ij} = t_{ij} + \psi, \psi \sim \text{Uniform}[-1, 1]$.

**Capacity constraints.** In this experiment, we focus on the 3-hops environment and assume a constant value $c_{ij} = 20, \forall (i, j) \in \mathcal{E} : j \neq 7$ while we keep a high value for all the edges going into node 7 (i.e., the sink node) which would more easily generate infeasible scenarios. From an RL perspective, we add the following edge-level features:

- Edge-capacity $\{c_{ij}^t\}_{i,j)\in\mathcal{E}}$ at the current time step $t$.
- Accumulated flow $\{f_{ij}^t\}_{(i,j)\in\mathcal{E}}$ on edge $(i, j)$

**Multi-commodity.** Let $N_c$ define the number of commodities to consider, indexed by $k$. From an RL perspective, we extend the proposed policy to represent a $N_c$-dimensionsional Dirichlet distribution. Concretely, we define the output of the policy network to represent the $N_c \times N_v$ concentration parameters $\alpha \in \mathbb{R}_+^{N_c \times N_v}$ of a Dirichlet distribution over nodes for each commodity, such that $\mathbf{a}_t \sim \text{Dir}\{\mathbf{a}_t|\alpha\}$. In other words, to extend our approach to the multi-commodity setting, we define a multi-head policy network characterized by one head per commodity. In our experiments, we train our multi-head agent on the topology shown in Fig. 10 whereby we assume two parallel commodities: commodity A going from node 0 to node 10, and commodity B going from node 0 to node 11. We choose this topology so that the only way to solve the scenario is to discover distinct behaviours between the two network heads (i.e., the policy head controlling flow for commodity A needs to go up or it won't get any reward, and vice-versa for commodity B).

**Computational analysis.** In this experiment, we generate different versions of the 3-hops environment, whereby different environments are characterized by intermediate layers with increasing number of nodes and edges. The results are computed by applying the pre-trained MPNN-RL agent on the original 3-hops environment (i.e., characterized by 8 nodes in the graph). In light of this, Figure 2 showcases a promising degree of transfer and generalization among graphs of different dimensions.

#### D.1.2. MODEL IMPLEMENTATION

In our experiments, we implement the following methods:

**Randomized heuristics.** In this class of methods, we focus on measuring performance of simple heuristics.

1. *Random policy*: at each timestep, we sample the desired next state from a Dirichlet prior with concentration parameter $\alpha = [1, 1, \ldots, 1]$. This benchmark provides a lower bound of performance by choosing desired next states randomly.

**Learning-based.** Within this class of methods, we focus on measuring how different architectures affect the quality of the solutions for the dynamic network control problem. For all methods, the A2C algorithm is kept fixed, thus the difference solely lies in the neural network architecture.

2. *MLP-RL*: both policy and value function estimator are parametrized by feed-forward neural networks. In all our experiments, we use two layers of 32 hidden unites and an output layer mapping to the output's support (e.g., a scalar value for the critic network). Through this comparison, we highlight the performance and flexibility of graph representations for network-structured data.
3. *GCN-RL*: In all our experiments, we use $K$ layers of graph convolution with 32 hidden units, with $K$ equal to the number of sink-to-source hops in the graph, and a linear output layer mapping to the output's support. See below for a broader discussion of graph convolution operators.
4. *GAT-RL*: In all our experiments, we use $K$ layers of graph attention (Veličković et al., 2018) with 32 hidden units, with $K$ equal to the number of sink-to-source hops in the graph, and single attention head. The output is further computed by a linear output layer mapping to the output's support. Together with GCN-RL, this model represents an approach based on graph convolutions rather than explicit message passing along the edges (as in MPNNs). Through this comparison,

we argue in favor of explicit, pair-wise messages along the edges, opposed to sole aggregation of node features among a neighborhood. Specifically, we argue in favor of the alignment between MPNN and the kind of computations required to solve flow optimization tasks, e.g., propagation of travel times and selection of best path among a set of candidates (max aggregation).

5. *MPNN-RL*: ours. We use $K$ layers of message passing neural network (Gilmer et al., 2017) of 32 hidden units as defined in Section B.1, with $K$ equal to the number of sink-to-source hops in the graph, and a linear output layer mapping to the output's support.

**MPC-based.** Within this class of methods, we focus on measuring performance of MPC approaches that serve as state-of-art benchmarks for the dynamic network flow problem.

6. *Oracle*: we directly optimize the flow using a standard formulation of MPC (Zhang et al., 2016). Notice that although the embedded optimization is a linear programming model, it may not meet the computation requirement of real-time applications (e.g., obtaining a solution within several seconds) for large scale networks. In this work, MPC is assumed to have access to future state elements (e.g., future travel times, connectivity, etc.). Crucially, assuming knowledge of future state elements is equivalent to assuming oracle knowledge of the realization of all stochastic elements in the system. In other words, there is no uncertainty for the MPC (this is in contrast with RL-based benchmarks, that assume access only to *current* state elements). In our experiments, the benchmark with the "Oracle" MPC enables us to quantify the optimal solution for all environments, thus giving a sense of the optimality gap between the ground truth optimum and the solution achieved via RL.

#### D.1.3. ADDITIONAL RESULTS

**Minimum cost flow through message passing.** In this first experiment, we consider 3 different environments (Fig. 6), such that different topologies enforce a different number of required hops of message passing between source and sink nodes to select the best path. Results in Table 1 (*2-hop*, *3-hop*, *4-hop*) show how MPNN-RL is able to achieve at least $87\%$ of oracle performance. Table 1 further shows how agents based on graph convolutions (i.e., GCN, GAT) fail to learn an effective flow optimization strategy.

**Dynamic travel times.** In many real-world systems, travel times evolve over time. To approach this, in Fig. 7 and Table 1 (*Dyn travel time*) we measure results on a dynamic network characterized by two *change-points*, i.e., time steps where the optimal path changes because of a change in travel times. Results show how the proposed MPNN-RL is able to achieve above $99\%$ of oracle performance.

**Dynamic topology.** In real-world systems, operations are often characterized by time-dependent topologies, i.e., nodes and edges can be dropped or added during an episode, such as in roadblocks within transportation systems or the opening of a new shipping center in supply chain networks. However, most traditional approaches cannot deal with these conditions easily. On the other hand, the locality of graph network-based agents, together with the one-step implicit planning of RL, enable our framework to deal with multiple time-varying graph configurations during the same episode. Fig. 8 and Table 1 (*Dyn topology*) show how MPNN-RL achieves 83.9% of oracle performance clearly outperforming the other benchmarks. Crucially, these results highlight how agents based on MLPs result in highly inflexible network controllers that are limited to the same topology they were exposed to during training.

**Capacity constraints.** Real-world systems are often represented as capacity-constrained networks. In this experiment, we relax the assumption that capacities $c_{ij}$ are always able to accommodate any flow on the graph. Compared to previous sections, the lower capacities introduce the possibility of infeasible states. To measure this, the *Success Rate* computes the percentage of episodes which have been terminated successfully. Results in Table 1 (*Capacity*) highlight how MPNN-RL is able to achieve 89.8% of oracle performance while being able to successfully terminate 87% of episodes. Qualitatively, Fig. 9 shows a visualization of the policy for a specific test episode. The plots show how MPNN-RL is able to learn the effects of capacity on the optimal strategy by allocating flow to a different node when the corresponding edge is approaching its capacity limit.

**Multi-commodity.** Often, system operators might be interested in controlling multiple commodities over the same network. In this scenario, we extend the current architecture to deal with multiple commodities and source-sink combinations. Results in Table 1 (*Multi-commodity*) and Fig. 10 show how MPNN-RL is able to effectively recover distinct policies for each commodity, thus being able to operate successfully multi-commodity flows within the same network.

## D.2. Supply Chain Inventory Management

We start by describing the properties of the environments in Section D.2.1. We further expand the discussion on MDP definitions (Section D.2.2), model implementation (Section D.2.3), and specifics on the linear control problem (Section D.2.4).

### D.2.1. Environment details

In our experiments, all stores are assumed to have an independent demand-generating process. We simulate a seasonal demand behavior by representing the demand as a co-sinusoidal function with a stochastic component, defined as follows:

$$d_i^t = \left\lfloor \frac{d_i^{\max}}{2} \left(1 + \cos\left(\frac{4\pi(2i+t)}{T}\right)\right) + \mathcal{U}(0, d_i^{var}) \right\rfloor, \tag{13}$$

where $\lfloor \cdot \rfloor$ is the floor function, $d_i^{\max}$ is the maximum demand value, $\mathcal{U}(0, d_i^{var})$ is a uniformly distributed random variable, and $T$ is the episode length.

Environment parameters are defined as follows:

Table 5: Parameters for the 1F2S environment

| Parameter | Explanation | Value | Parameter | Explanation | Value |
|---|---|---|---|---|---|
| $d^{\max}$ | Maximum demand | [2, 16] | $m^S$ | Storage cost | [3, 2, 1] |
| $d^{var}$ | Demand variance | [2, 2] | $m^O$ | Production cost | 5 |
| $T$ | Episode length | 30 | $m^B$ | Backorder cost | 21 |
| $t^P$ | Production time | 1 | $m^T$ | Transportation cost | [0.3, 0.6] |
| $t_{ij}$ | Travel time | [1, 1] | $p$ | Price | 15 |
| $c$ | Storage capacity | [20, 9, 12] | | | |

Table 6: Parameters for the 1F3S environment

| Parameter | Explanation | Value | Parameter | Explanation | Value |
|---|---|---|---|---|---|
| $d^{\max}$ | Maximum demand | [1, 5, 24] | $m^S$ | Storage cost | [2, 1, 1] |
| $d^{var}$ | Demand variance | [2, 2, 2] | $m^O$ | Production cost | 5 |
| $T$ | Episode length | 30 | $m^B$ | Backorder cost | 21 |
| $t^P$ | Production time | 1 | $m^T$ | Transportation cost | [0.3, 0.3, 0.3] |
| $t_{ij}$ | Travel time | [1, 1, 1] | $p$ | Price | 15 |
| $c$ | Storage capacity | [30, 15, 15, 15] | | | |

Table 7: Parameters for the 1F10S environment

| Parameter | Explanation | Value | Parameter | Explanation | Value |
|---|---|---|---|---|---|
| $d^{\max}$ | Maximum demand | [2, 2, 2, 2, 10, 10, 10, 18, 18, 18] | $m^S$ | Storage cost | $[1, 2 \quad \forall i \in \mathcal{V}/0]$ |
| $d^{var}$ | Demand variance | $[2]_{i\in\mathcal{V}}$ | $m^O$ | Production cost | 5 |
| $T$ | Episode length | 30 | $m^B$ | Backorder cost | 21 |
| $t^P$ | Production time | 1 | $m^T$ | Transportation cost | $[0.3]_{i\in\mathcal{V}}$ |
| $t_{ij}$ | Travel time | $[1]_{i\in\mathcal{V}}$ | $p$ | Price | 15 |
| $c$ | Storage capacity | $[100, 15 \quad \forall i \in \mathcal{V}/0]$ | | | |

### D.2.2. MDP details

In what follows, we complement Section 5.2 with a formal definition of the SCIM MDP.

*Reward* $(R(s^t, a^t))$*:* we select the reward function in the MDP as the profit of the inventory manager, computed as the difference between revenues and the sum of storage, production, transportation, and backorder costs:

$$R(s^t, a^t) = \sum_{i\in\mathcal{V}_W} p \cdot \min(d_i^t, q_i^t) - \left( \sum_{i\in\mathcal{V}} m_i^S \cdot q_i^t + \sum_{i\in\mathcal{V}_W} m_i^O \cdot w_i^t + \sum_{(i,j)\in\mathcal{E}} m_{ij}^T \cdot f_{ij}^t - \sum_{i\in\mathcal{V}_S} m_i^B \cdot \min(0, q_i^t) \right). \tag{14}$$

*State space* $(\mathcal{S})$*:* the state space contains information to describe the current status of the supply network, via the definition of node and edge features. Node features contain information on (i) current inventory $q_i$, (ii) current and estimated demand for

the next $T$ timesteps $\hat{d}_i^{t:t+T}$, (iii) incoming flow for the next $T$ timesteps $\sum_{j\in\mathcal{V}} f_{ji}^{t:t+T}$, and (iv) incoming orders for the next $T$ timesteps $w_i^{t:t+T}$, such that $\mathbf{x}_i = [q_i, \hat{d}_i^{t:t+T}, \sum_{j\in\mathcal{V}} f_{ji}^{t:t+T}, w_i^{t:t+T}]$. Edge features are represented by the concatenation of (i) travel time $t_{ij}$, and (ii) transportation cost $m_{ij}^T$, such that $\mathbf{e}_{ij} = [t_{ij}, m_{ij}^T]$.

#### D.2.3. MODEL IMPLEMENTATION

In what follows, we provide additional details for the implemented baselines and models:

**Randomized heuristics.** In this class of methods, we focus on measuring the performance of simple heuristics.

1. *Avg. Prod*: at each timestep, we (1) select production $w_i^t$ to be the average episode demand across all stores, and (2) sample the desired distribution from a Dirichlet prior with concentration parameter $\alpha = [1, 1, \dots, 1]$ to simulate a random shipping behavior.

**Domain-driven heuristics.** Within this class of methods we measure the performance of heuristics generally accepted as effective baselines.

2. *S-type Policy*: also referred to as "order-up-to" policy, this heuristic is parametrized by two values: a warehouse order-up-to level and a store order-up-to level. At a high level, at each time step the inventory manager aims to order inventory such that all inventory at and expected to arrive at the warehouse and at the stores is equal to the warehouse order-up-to level and the store order-up-to level, respectively. Concretely, we fine-tune the S-type policy on each environment individually by running an exhaustive search for the best order-up-to levels, as shown in Figure 11.

**Learning-based approaches.**

3. *End-to-end RL*: with this benchmark, we evaluate the performance of RL architectures that do not approach the problem via the proposed bi-level formulation. Specifically, as traditionally done in RL, we define the policy network to represent a direct mapping from states to environment actions. In our experiments, both policy and value function estimator are parametrized by feed-forward neural networks with two layers of 64 hidden units followed by linear layers mapping to either (i) mean and standard deviation parameters for the policy network, or (ii) a scalar value function estimate for the critic. Among the three scenarios, we adjust the input layer based on the input dimensionality (which is topology-dependent since we unroll all node and edge features into a vector representation of the graph). Through this comparison, we highlight the benefits of the bi-level formulation for graph control problems.
4. *Graph-RL*: ours. We use $K = 2$ layers of message passing neural network (Gilmer et al., 2017) of 32 hidden units with sum aggregation function as defined in Section B.1 followed by a linear layer mapping to the output's support.

#### D.2.4. LCP FORMULATION

Given a desired next state described by (i) the desired production in warehouse nodes $\hat{w}_i^{t+1}, \forall i \in \mathcal{V}_W$, and (ii) a desired inventory in store nodes $\hat{q}_i^{t+1}, \forall i \in \mathcal{V}_S$, we define the following linear control problem as follows:

$$\min_{f_{ij}^t, w^t, \epsilon_i^f, \epsilon_i^w} \sum_{i\in\mathcal{V}_S} |\epsilon_i^f| + \sum_{i\in\mathcal{V}_W} |\epsilon_i^w| \tag{15a}$$

$$\text{s.t.} \sum_{j\in\mathcal{N}^-(i)} f_{ji}^t = \hat{q}_i^{t+1} + \epsilon_i^f, \qquad i \in \mathcal{V}_S \tag{15b}$$

$$q_i^t + \sum_{j\in\mathcal{N}^-(i)} f_{ji}^t - d_i^t \le c_i^t, \qquad i \in \mathcal{V}_S \tag{15c}$$

$$\sum_{j\in\mathcal{N}^+(i)} f_{ij}^t \le q_i^t, \qquad i \in \mathcal{V}_W \tag{15d}$$

$$q_i^t + w_i^t - \sum_{j\in\mathcal{N}^+(i)} f_{ij}^t \le c_i^t, \qquad i \in \mathcal{V}_W \tag{15e}$$

$$w_i^t = \hat{w}_i^t + \epsilon_i^w, \qquad i \in \mathcal{V}_W \tag{15f}$$

$$f_{ij}^t \ge 0, \qquad (i,j) \in \mathcal{E} \tag{15g}$$

where, as introduced in Section 4, the objective function (15a) represents the distance metric $d(\cdot,\cdot)$ that penalizes the deviation from the desired next states, the constraint (15b) ensures that the total incoming flow in store nodes is as close as possible to the desired inventory, the constraint (15c) represents that the inventory in store nodes after shipping and demand satisfaction does not exceed storage capacity, the constraint (15d) ensures that the shipped products are upper bounded by inventory, constraint (15e) represents that the inventory in warehouse nodes after shipping and re-ordering does not exceed storage capacity, constraint (15f) ensures that orders from manufacturers are close to the desired orders specified through RL, and lastly that commodity flows are non-negative via (15g).

### D.3. Dynamic Vehicle Routing

We start by describing the properties of the environments in Section D.3.1. We further expand the discussion on MDP definition (Section D.3.2), model implementation (Section D.3.3), specifics on the linear control problem (Section D.3.4), and additional results (Section D.3.5).

#### D.3.1. Environment details

We use two case studies from the cities of New York, USA, and Shenzhen, China, whereby we study a hypothetical deployment of taxi-like systems to serve the peak-time commute demand in popular areas of Brooklyn and Shenzhen, respectively. The cities are divided into geographical areas, each of which represents a station. The case studies in our experiments are generated using trip record datasets, which we provide together with our codebase. The trip records are converted to demand, travel times, and trip prices between stations. Here, we consider stochastic time-varying demand patterns, whereby customer arrival is assumed to be a time-dependent Poisson process, and the Poisson rates are aggregated from the trip record data every 3 minutes. We assume the stations to be spatially connected, whereby moving vehicles from one station to the other requires non-trivial sequential actions (i.e., vehicles cannot directly be repositioned from one station to any other station, rather they have to adhere to the available paths given by the city's topology).

The following remarks are made in order. First, we assume travel times are given and independent of operator actions. This assumption applies to cities where the number of vehicles in the fleet constitutes a relatively small proportion of the entire vehicle population on the transportation network, and thus the impact on traffic congestion is marginal. This assumption can be relaxed by training the proposed RL model in an environment considering the endogenous congestion caused by controlled vehicles fleet. Second, without loss of generality, we assume that the arrival process of passengers for each origin-destination pair is a time-dependent Poisson process. We further assume that such process is independent of the arrival processes of other origin-destination pairs and the coordination of vehicles. These assumptions are commonly used to model transportation requests (Daganzo, 1978).

#### D.3.2. MDP details

In what follows, we complement Section 5.2 with a formal definition of the SCIM MDP.

*Reward* ($R(s^t, a^t)$)*:* we select the reward function in the MDP as the operator profit, computed as the difference between revenues and operation-related costs:

$$R(s^t, a^t) = \sum_{(i,j)\in\mathcal{E}} f^t_{ij,P} \cdot (p_{ij} - m_{ij}) - \sum_{(i,j)\in\mathcal{E}} f^t_{ij,R} \cdot m_{ij} \tag{16}$$

*State space ($\mathcal{S}$):* the state space contains information to describe the current status transportation network via the definition of node features. Node features contain information on (i) the current availability of idle vehicles in each station $q_i$, (ii) current and estimated demand for the next $T$ timesteps $\hat{d}_i^{t:t+T}$, (iii) projected availability for the next $T$ timesteps $\hat{q}_i^{t:t+T}$, and (iv) provider-level information such as trip price $p_{ij}$ and cost $z_{ij}$.

#### D.3.3. Model implementation

In what follows, we provide additional details for the implemented baselines and models:

**Randomized heuristics.** In this class of methods, we focus on measuring the performance of simple heuristics.

1. *Random*: at each timestep, we sample the desired distribution from a Dirichlet prior with concentration parameter $\alpha = [1, 1, \ldots, 1]$. This benchmark provides a lower bound of performance by choosing desired next states randomly.

**Domain-driven heuristics.** Within this class of methods, we measure the performance of heuristics generally accepted as reasonable baselines.

2. *Equally-balanced System*: at each decision, we take rebalancing actions so to recover an equal distribution of idle vehicles across all areas in the transportation network. Concretely, the heuristic achieves this by solving the DVR LCP with a fixed desired number of idle vehicles among all stations, i.e., given $M$ available vehicles at time $t$, $\hat{\mathbf{q}}^{t+1} = \{\hat{q}_i^{t+1}\}_{i\in\mathcal{V}} = \{\frac{M}{|\mathcal{V}|}\}_{i\in\mathcal{V}}$.

**Learning-based approaches.**

3. *End-to-end RL*: both policy and value function estimator are parametrized by neural networks that mirror the architecture of Graph-RL. While the critic has the exact same architecture, the actor differs in the last layer, which is characterized by an edge convolution (consisting of 2 linear layers of 32 hidden units) that outputs mean and standard deviation parameters of a Gaussian policy for each edge in the graph.
4. *Graph-RL*: ours. For both actor and critic networks, we use one layer of graph convolution (Kipf & Welling, 2017) with 32 hidden units with sum aggregation function as defined in Section B.1 followed by 2 linear layers of 32 hidden units and a final linear layer mapping to the respective output's support.

#### D.3.4. LCP FORMULATION

Given a desired next state described by the desired number of idle vehicles across stations $\hat{q}_i^{t+1}, \forall i \in \mathcal{V}$, we define the following linear control problem as follows:

$$\min_{f_{ij,R}^t} \sum_{(i,j)\in\mathcal{E}} m_{ij}^t f_{ij,R}^t \tag{17a}$$

$$\text{s.t.} \sum_{j\neq i} (f_{ji,R}^t - f_{ij,R}^t) + q_i^t \geq \hat{q}_i^t, \qquad i \in \mathcal{V} \tag{17b}$$

$$\sum_{j\neq i} f_{ij,R}^t \leq q_i^t, \qquad i \in \mathcal{V} \tag{17c}$$

$$f_{ij,R}^t \geq 0, \qquad (i,j) \in \mathcal{E} \tag{17d}$$

where the objective function (17a) represents the rebalancing cost, constraint (17b) ensures that the resulting number of vehicles is close to the desired number of vehicles, and with constraints (17c), (17d) ensuring that the total rebalancing flow from a region is upper-bounded by the number of idle vehicles in that region and non-negative.

#### D.3.5. ADDITIONAL RESULTS

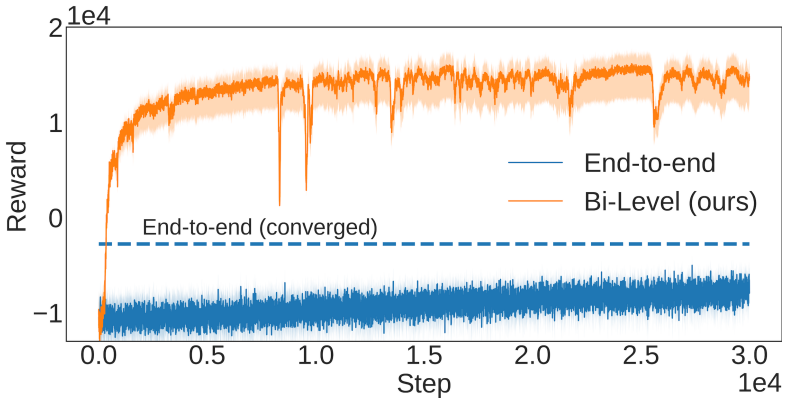



Figure 5: Learning curve comparison between an RL agent trained end-to-end (blue) and via our bi-level formulation (orange) on the DVR New York environment. The dotted line represents the converged performance for the end-to-end agent after 80,000 steps.

Results in Figure 5 highlight the sample efficiency of our bi-level approach compared to its end-to-end counterpart which exhibits (i) much slower convergence and sample inefficiency, and (ii) worse overall performance.

## E. Additional Visualizations

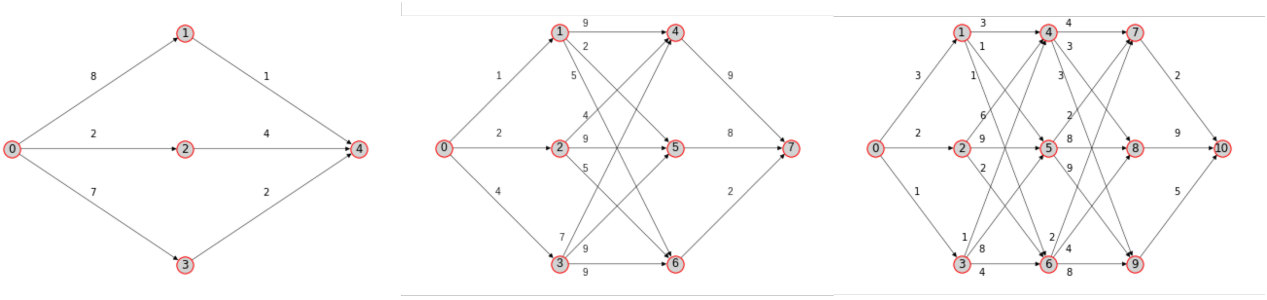


Figure 6: Graph topologies used for the message passing experiments: 2-hops (left), 3-hops (center), 4-hops (right). The source and sink nodes are represented by the left-most and right-most nodes, respectively. Values in the proximity of the edges represent travel times.

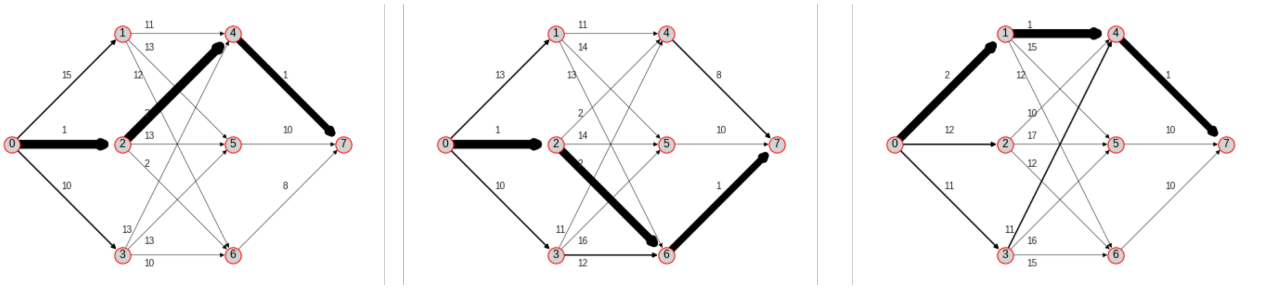


Figure 7: Visualization of a trained instance of MPNN-RL on an environment with dynamic travel times. We simulate a scenario where the optimal path changes three times (left, middle, and right) over the course of an episode. Shaded edges represent actions induced by the MPNN-RL.

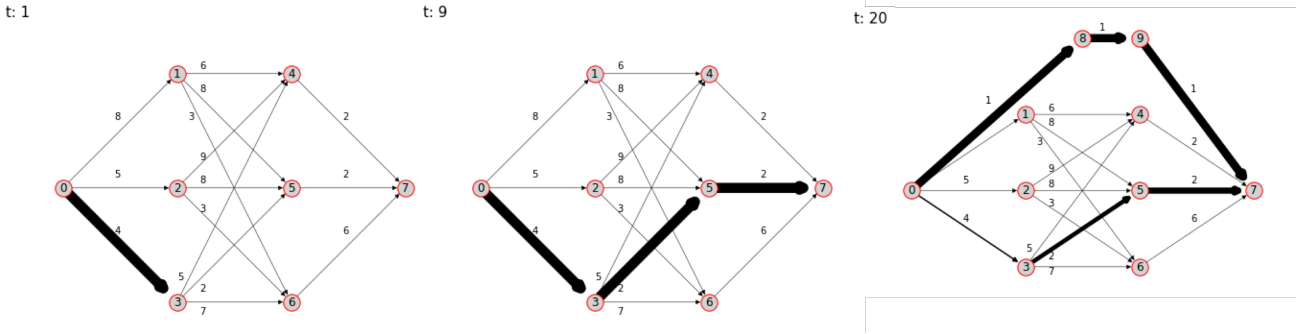



Figure 8: Visualization of a trained instance of MPNN-RL on an environment with dynamic topology. We simulate a scenario where the optimal path changes over the course of an episode because of the addition of a new path. Shaded edges represent actions induced by the MPNN-RL.

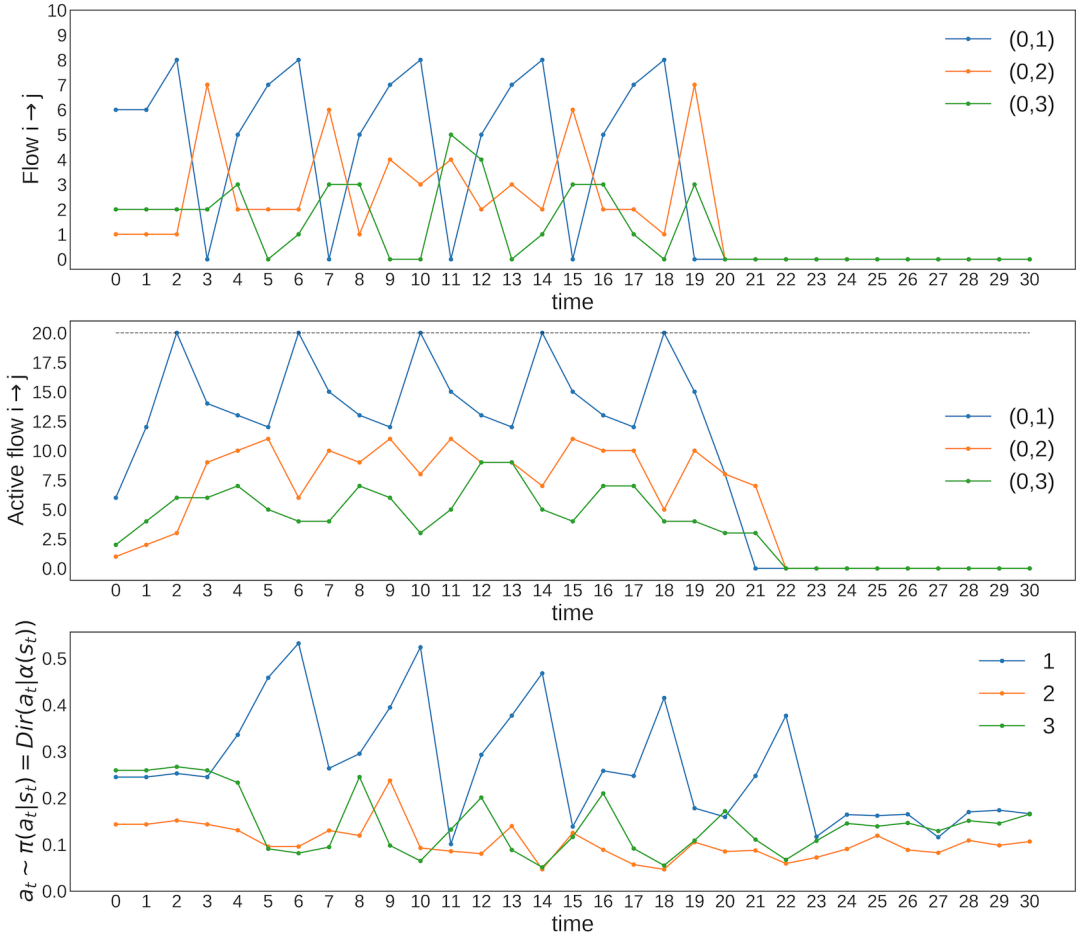



Figure 9: Visualization of the MPNN-RL policy on the capacity-constrained environment. (Top) The resulting flow $f_{ij}$ on the edges $0 \to 1, 0 \to 2, 0 \to 3$. (Center) The accumulated flow on the same edges compared to the fixed capacity $c_{ij} = 20$, represented as a dashed horizontal line. (Bottom) The desired distribution described by the MPNN-RL policy.

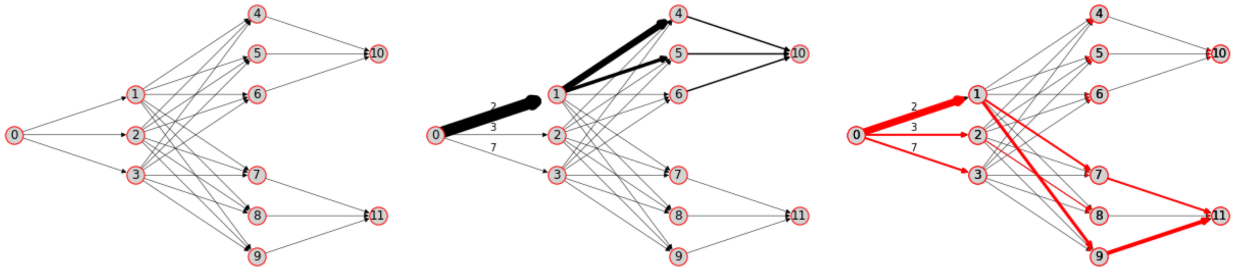
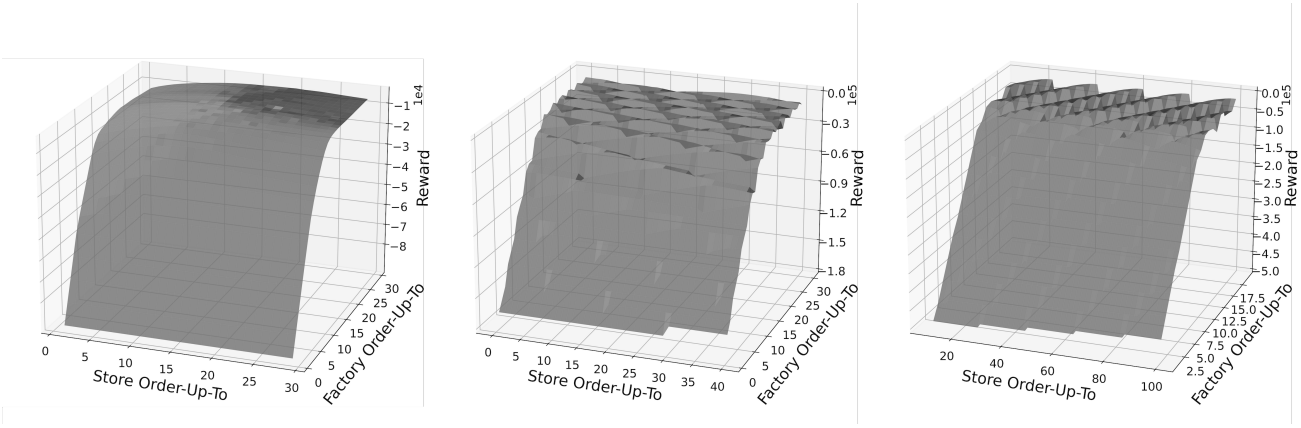


Figure 10: Visualization of the multi-commodity environment. (Left) The topology considered during our experiments. (Center) A visualization of the policy for the first commodity A. (Right) A visualization of the policy for the second commodity B.




(a) (b) (c)

Figure 11: Parameter tuning for the S-type policy on (a) 1F2S, (b) 1F3S, and (c) 1F10S environments.