# Reinforcement Learning for Flexibility Design Problems

**Anonymous authors**
Paper under double-blind review

## Abstract

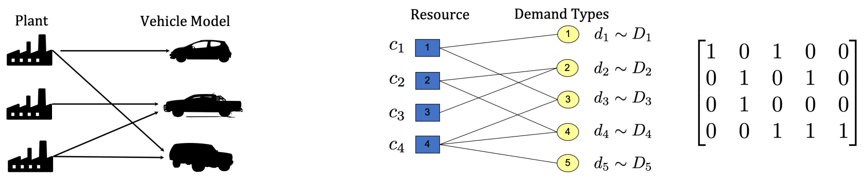
Flexibility design problems are a class of problems that appear in strategic decision-making across industries, where the objective is to design a ($e.g.$, manufacturing) network that affords flexibility and adaptivity. The underlying combinatorial nature and stochastic objectives make flexibility design problems challenging for standard optimization methods. In this paper, we develop a reinforcement learning (RL) framework for flexibility design problems. Specifically, we carefully design mechanisms with noisy exploration and variance reduction to ensure empirical success and show the unique advantage of RL in terms of fast-adaptation. Empirical results show that the RL-based method consistently finds better solutions compared to classical heuristics.

## 1 Introduction

Designing flexibility networks (networks that afford the users flexibility in their use) is an active and important problem in Operations Research and Operations Management (Chou et al., 2008; Wang et al., 2019). The concept of designing flexibility networks originated in automotive manufacturing, where firms considered the strategical decision of adding flexibilities to enable manufacturing plants to quickly shift production portfolios with little penalty in terms of time and cost (Jordan & Graves, 1995). Jordan & Graves (1995) observed that with a well designed flexible manufacturing network, firms can better handle uncertainties and significantly reduce idling resources and unsatisfied demands.

With the increasingly globalized social-media-driven economy, firms are facing an unprecedented level of volatility and their ability to adapt to changing demands has become more critical than ever. As a result, the problem of designing flexibility networks has become a key strategic decision across many different manufacturing and service industries, such as automotive, textile, electronics, semiconductor, call centers, transportation, health care, and online retailing (Chou et al., 2008; Simchi-Levi, 2010; Wang et al., 2019). The universal theme in all of the application domains is that a firm has multiple resources, is required to serve multiple types of uncertain demands, and needs to design a flexibility network that specifies the set of resources that can serve each of its demand types. For example, an auto-manufacturer owns multiple production plants (or production lines), produces a portfolio of vehicle models, and faces stochastic demand for each of its vehicle models. The firm's manufacturing flexibility, coined as process flexibility in Jordan & Graves (1995), is the capability of quickly switching the production at each of its plants between multiple types of vehicle models. As a result, the firm needs to determine a flexible network, where an arc $(i, j)$ in the network represent that plant $i$ is capable of producing a vehicle model $j$ (see Figure 1a for an example). In another example, a logistics firm owns multiple distribution centers and serves multiple regions with stochastic demand. To improve service quality and better utilize its distribution centers, the firm desires an effective flexible network with arcs connecting the distribution centers and regions, where each arc represents an investment by the firm to enable delivery from a distribution center to a region.

In general, the flexibility design problem (FDP) is difficult to solve. It is a combinatorial optimization problem with stochastic demand, that is already NP-hard even when the demand is deterministic. The stochastic nature of the FDP eliminates the possibility of finding the optimal solution for medium-sized instances with $m \cdot n \geq 50$, where $m$ and $n$ denote the number of resources and demand types, even when stochastic programming techniques such as Benders Decomposition are



$$\begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

(a) network for an auto-manufacturer (b) generic network and its matrix representation

Figure 1: *Examples of Flexibility Networks*

applied (Feng et al., 2017). Therefore, researchers have mostly focused on finding general guidelines and heuristics (Jordan & Graves, 1995; Chou et al., 2010; Simchi-Levi & Wei, 2012; Feng et al., 2017). While the heuristics in literature are generally effective, in many applications, even a one percent gain in a FDP solution is highly significant (Jordan & Graves, 1995; Chou et al., 2010; Feng et al., 2017) and this motivates us to investigate reinforcement learning (RL) as a new approach for solving FDPs. In addition, FDPs fall into the class of two-stage mixed-integer stochastic programs, which often arise in strategic-level business planning under uncertainty (see, $e.g.$, Santoso et al. (2005)). Therefore, RL's success in FDPs can lead to future interest in applying RL to more strategic planning problems in business. We believe that RL can be effectively applied in FDPs and more strategic decision problems for two reasons. First, RL is designed to solve problems with stochastic rewards, and the stochasticity of strategic planning problems may even help RL to explore more solutions (Sutton et al., 2000). Secondly, the heuristics mostly relied on, and therefore, are also constrained by human intuitions, whereas RL algorithms with deep neural networks may uncover blind spots and discover new designs, or better, new intuition altogether similar to the successful applications of RL in Atari games and AlphaGo Zero (Silver et al., 2017).

In this paper, we develop an RL framework to solve a given FDP. We first formulate an FDP as a Markov Decision Process (MDP), and then optimize it via policy gradient. In this framework, we design a specific MDP to take advantage of the structure possessed by FDPs. Further, a problem-dependent method for estimating a stochastic objective, $i.e.$, terminal rewards, is proposed. Empirical results show that our proposed method consistently outperforms classical heuristics from the literature. Furthermore, our framework incorporates fast-adaptation for similar problems under different settings, and hence, avoids repeated computations as in classical heuristics.

## 2 Flexibility Design Problem

We present the formal formulation for the FDP. In an FDP, there are $m$ resources (referred to as supply nodes) and $n$ demand types (referred to as demand nodes). The set of feasible solutions $\mathcal{F}$ is all networks with no more than $K$ arcs. Formally, each network $\mathbf{F}$ is represented by an $m \times n$ matrix (see Figure 1b), implying that $\mathcal{F} = \{\mathbf{F} \,|\, \mathbf{F} \in \{0,1\}^{mn}, \sum_{i \in [m], j \in [n]} F_{ij} \leq K\}$, where $[m]$ and $[n]$ denote the set of integers from 1 to $m$ and 1 to $n$, respectively.

The objective of FDP is to find the flexibility network $\mathbf{F} \in \mathcal{F}$ that maximizes the expected profit under stochastic demand. Denoting the random demand vector as $\boldsymbol{d} \in \mathbb{R}^n$ with distribution $\mathbf{D}$, the FDP can be defined as

$$\max_{\mathbf{F} \in \mathcal{F}} \mathbb{E}_{\boldsymbol{d} \sim \mathbf{D}}[P(\boldsymbol{d}, \mathbf{F})] - \sum_{i,j} I_{ij} F_{ij}, \tag{1}$$

where $I_{ij} \geq 0$ is the cost of including arc $(i,j)$ to the flexibility network, while $P(\boldsymbol{d}, \mathbf{F})$ is the profit achieved by network $\mathbf{F}$ after the demand is revealed to be $\boldsymbol{d}$. It is important to note here that the decision on $\mathbf{F}$ is made *prior* to the observation of the demand, and thus, the decision is to create $\mathbf{F}$ such that the expected profit (over $\boldsymbol{d} \sim \mathbf{D}$) is optimized. We next define $P(\boldsymbol{d}, \mathbf{F})$.

Given parameter $\mathbf{p} \in \mathbb{R}^{mn}$ where $p_{ij}$ represents the unit profit of using one unit of resource $i$ to satisfy one demand type $j$ and $\boldsymbol{c} \in \mathbb{R}^n$ where $c_i$ represents the capacity of resource $i$, the function $P(\boldsymbol{d}, \mathbf{F})$ is defined as the objective value of the following linear programming (LP) problem:

$$\max_{\mathbf{f} \in \mathbb{R}^{mn}} \sum_{i,j} p_{ij} f_{ij}, \text{s.t.} \sum_{i \in [m]} f_{ij} \leq d_j, \forall j \in [n], \sum_{j \in [n]} f_{ij} \leq c_i, \forall i \in [m], 0 \leq f_{ij} \leq M_{ij} F_{ij}, \forall i, j,$$

where $M_{ij}$ is a large constant ensuring that the constraint $f_{ij} \leq M_{ij} F_{ij}$ is nonbinding when $F_{ij} = 1$. It is easy to check that it is sufficient to set $M_{ij} = \min\{c_i, d_j\}$ (see Feng et al. (2017) for a discussion). Intuitively, the LP defining $P(\boldsymbol{d}, \mathbf{F})$ solves the optimal resource allocation subject to capacity for resource $i$ being $c_i$ and demand for type $j$ being $d_j$, while only using arcs in $\mathbf{F}$. In general, the LP can be solved efficiently (in a fraction of a millisecond) by LP solvers such as Gurobi (Gurobi Optimization, 2020), and we exploit this in our RL framework.

While $P(\boldsymbol{d}, \mathbf{F})$ is easy to solve, solving FDP defined in equation 1 is difficult. Even when $\boldsymbol{d}$ is deterministic, FDP reduces to a NP-hard problem known as the fixed charge transportation problem (see, $e.g.$, Agarwal & Aneja (2012)). For the FDP with stochastic $\boldsymbol{d}$, FDP is significantly more difficult than the typical NP-hard combinatorial optimization problems, such as travelling salesman or vertex cover problem, because there is no known method for even computing the objective function exactly except for some special cases (Chou et al., 2010). As a result, researchers mostly resort to estimating the objective by repeatedly drawing samples of $\boldsymbol{d}$. While the estimation is accurate when the number of samples of $\boldsymbol{d}$ is large, it reformulates FDP as a mixed-integer linear program (MILP) with a large number of variables and constraints, rendering most of the MILP algorithms impractical in this context.

## 3 RELATED WORK

The research on designing flexibility networks can be divided into three streams that often overlap with each other: $(i)$ extending its model and applications, $(ii)$ conducting theoretical analysis, and $(iii)$ proposing methods to solve FDPs (see Chou et al. (2008); Wang et al. (2019) for more details). The third stream of the literature is most related to our work. Due to the inherent complexity of the FDP, researchers so far have focused on heuristics instead of exact algorithms. Even for the classical flexibility design problem (assuming $p_{ij} = 1$ and $I_{ij} = 0$ for all $i, j$), the optimal solution of FDP is not known except for very restrictive cases (Désir et al., 2016). As a result, various heuristics have been proposed for the classical FDP (Chou et al., 2010; 2011; Simchi-Levi & Wei, 2015). However, while the classical FDP is important, it has been observed that arc costs ($I_{ij}$) and different unit profits ($p_{ij}$) can be key determinants to the value of a flexibility network (Van Mieghem, 1998; Wang et al., 2019). For general FDPs, the most intuitive heuristic is perhaps the greedy heuristic, which at each iteration, adds an arc with the highest performance improvement. Recently, DeValve et al. (2020) derived a performance guarantee on the greedy heuristic when there is no fixed cost on the arcs. Alternatively, the heuristic proposed by Feng et al. (2017) solves a stochastic programming (SP) that relaxes the integer constraint of FDP at each iteration, and uses the fractional solution to make a decision on which arc to add.

Recently, there have been various activities in applying the recent advances in ML tools to solve combinatorial optimization problems, such as the traveling salesman problem (TSP) (Bello et al., 2016; Vinyals et al., 2015), vehicle routing problem (VRP) (Kool et al., 2018; Nazari et al., 2018), min vertex cover problem, and max cut problem (Abe et al., 2019). Interested readers are referred to Bengio et al. (2018) for a framework on how combinatorial problems can be solved using an ML approach and for a survey of various combinatorial optimization problems that have been attempted using ML. To the best of our knowledge, FDPs do not possess the property that solutions are naturally represented as a sequence of nodes, nor the property that the objective can be decomposed into linear objectives of a feasible integer solution. Hence, it is not straightforward to apply the existing RL works in combinatorial problems to FDPs. In addition, Neural Architecture Search (NAS) (Baker et al., 2017a; Zoph & Le, 2017) shares some similarity to FDPs, where it aims at automatically searching for a good neural architecture via reinforcement learning to yield good performance on datasets such as CIFAR-10 and CIFAR-100. Liu et al. (2018) imposed a specific structure as the prior knowledge to reduce the search space and Baker et al. (2017b) proposed performance prediction for each individual architecture.

## 4 PROPOSED METHOD

We present a reinforcement learning framework for flexibility designs. It is typically intractable to perform supervised training since FDPs are NP-hard and the objective is stochastic. However, reinforcement learning (RL) can provide a scalable and fast-adaptable solution based on the partial observation from the performance of a specific FDP solution. We first formulate a given FDP as a finite-horizon RL problem, then introduce policy optimization, evaluation and fast adaptation.

## 4.1 FDPs as Reinforcement Learning

The FDP can be considered as an RL problem, with discrete actions, deterministic state transitions, finite horizon and stochastic terminal rewards. For a given FDP defined in equation 1, an agent will sequentially select at most one arc at a step based on the current state for $K$ discrete steps. Accordingly, we can model the FDP as an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R \rangle$, where $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ is the environment dynamic of the given FDP and $R : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ is the stochastic reward function used to evaluate different arcs. We denote $S$ as the state space, which in our setting is the set of all flexibility networks connecting supply and demand nodes, *i.e.*, $\mathcal{S} = \{\mathbf{F} \,|\, \mathbf{F} \in \{0,1\}^{mn}\}$. We denote $\mathcal{A}$ as the action space, and $\mathcal{A} = \{(i,j) \,|\, i \in [m], j \in [n]\}$. Given state $\boldsymbol{s}_t = \mathbf{F}$, an action $\boldsymbol{a} = (i,j) \in \mathcal{A}$, the deterministic transition is given by $\boldsymbol{s}_{t+1} = \mathbf{F} \cup (i,j)$, where $\mathbf{F} \cup (i,j)$ represents the network with all arcs of $\mathbf{F}$ and arc $(i,j)$.

At each time step $t$, the agent observes the current flexibility network $\boldsymbol{s}_t \in \mathcal{S}$, chooses an action $\boldsymbol{a}_t$, then receives an immediate reward. The initial state, $\boldsymbol{s}_0$, is defined as the matrix with all zeros, *i.e.*, the network with no arcs. The objective of RL is to maximize expected rewards as defined in equation 2:

$$J(\pi) = \mathbb{E}_{\tau \sim p_\pi(\tau)} \left[ \sum_{t=0}^{K} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t) \right], \text{where } r(\boldsymbol{s}_t, \boldsymbol{a}_t) = \begin{cases} I(\boldsymbol{s}_t, \boldsymbol{a}_t), & \text{if } t < K, \\ \mathbb{E}_{\mathbf{d} \sim \mathbf{D}}[P(\boldsymbol{d}, \boldsymbol{s}_t)], & \text{if } t = K, \end{cases} \tag{2}$$

where the trajectory $\tau = (\boldsymbol{s}_0, \boldsymbol{a}_0, \ldots, \boldsymbol{a}_{K-1}, \boldsymbol{s}_K)$ is a sequence of states and actions, the trajectory distribution $p_\pi(\tau) := \prod_{t=0}^{K} \pi(\boldsymbol{a}_t|\boldsymbol{s}_t) T(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$, $I(\boldsymbol{s}_t, \boldsymbol{a}_t) = -I_{ij}$ if $\boldsymbol{a}_t$ is adding arc $(i,j)$ to $\boldsymbol{s}_t$ and $(i,j) \notin \boldsymbol{s}_t$, and $I(\boldsymbol{s}_t, \boldsymbol{a}_t) = 0$ otherwise. The goal of an agent is to learn an optimal policy that maximizes $J(\pi)$.

**Optimal Solution** Considering the designed rewards in equation 2, we have the Bellman equation as follows:

$$V_\pi(\boldsymbol{s}_t) = \begin{cases} \mathbb{E}_\pi\big[I(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma V_\pi(\boldsymbol{s}_{t+1})\big], & \text{if } t < K, \\ \mathbb{E}_{\mathbf{d} \sim \mathbf{D}}[P(\boldsymbol{d}, \boldsymbol{s}_t)], & \text{if } t = K. \end{cases} \tag{3}$$

Note that equation 3 has an optimal deterministic policy, and the optimal policy returns an optimal solution for the FDP defined in equation 1.

We next discuss sample-based estimation for $\mathbb{E}_{\mathbf{d} \sim \mathbf{D}}[P(\boldsymbol{d}, \boldsymbol{s}_K)]$, a quantity that is intractable to compute exactly. As defined in equation 3, all rewards are deterministic except the rewards obtained for the last action. A straightforward way to estimate $\mathbb{E}_{\mathbf{d} \sim \mathbf{D}}[P(\boldsymbol{d}, \boldsymbol{s}_K)]$ is using $\Omega$ i.i.d. samples as

$$\mathbb{E}_{\mathbf{d} \sim \mathbf{D}}[P(\boldsymbol{d}, \boldsymbol{s}_K)] \approx \frac{1}{\Omega} \sum_{1 \le \omega \le \Omega} P(\boldsymbol{d}^\omega, \boldsymbol{s}_K). \tag{4}$$

Intuitively, a higher value of $\Omega$ implies less variance and gives better guidance for RL training, *i.e.*, better sample efficiency. In Section 5.3, we show that the right choice of $\Omega$ is important to the performance of the RL algorithm. When $\Omega = 1$, the rewards are very noisy and the RL converges slowly or even fails to converge. If $\Omega$ is too large, estimating $\mathbb{E}_{\mathbf{d} \sim \mathbf{D}}[P(\boldsymbol{d}, \boldsymbol{s}_K)]$ needs substantially more time, considering $P(\boldsymbol{d}^\omega, \boldsymbol{s}_K)$ is solved by the linear programming solver from Gurobi Optimizer (Gurobi Optimization, 2020). It is interesting to see in numerical experiments that a noisy rewards using a relative small value of $\Omega$ (*e.g.*, setting $\Omega = 50$) can provide solutions that are similar or better compared to solutions from setting $\Omega = 1000$ in similar number of training steps. The observation that a small value of $\Omega$ is enough can be explained by the effectiveness of (noisy) exploration from the literature (Fortunato et al., 2018; Plappert et al., 2018).

**Variance Reduction for Reward Estimation** In addition to increasing $\Omega$, we propose another method for reducing the variance in the reward of the MDP. In this method, instead of using the average profit of $\boldsymbol{s}_K$ with $\Omega$ samples of demand as the estimation of the reward at the last step, we replace it by the difference of the average profits of $\mathbf{F}$ and the flexibility network with all the possible arcs. Formally, we define $\mathbf{1}$ as a $m \times n$ matrix with all ones. Then, we change the estimation to

$$\frac{1}{\Omega} \sum_{1 \le \omega \le \Omega} \big(P(\boldsymbol{d}^\omega, \boldsymbol{s}_K) - P(\boldsymbol{d}^\omega, \mathbf{1})\big). \tag{5}$$

This method reduces the variance of the reward for a fixed value of $\Omega$ because the feasible sets of the LP that defines $P(\boldsymbol{d}, \boldsymbol{s}_K)$ and $P(\boldsymbol{d}, \mathbf{1})$ both increase with $\boldsymbol{d}$, implying that $P(\boldsymbol{d}^{\omega}, \boldsymbol{s}_K)$ and $P(\boldsymbol{d}^{\omega}, \mathbf{1})$ are positively correlated for each $\omega$. The ablation study in Section 5.3 shows the proposed variance reduction method helps for sample efficiency in training.

## 4.2 Learning to Design via Policy Gradient

We considered various RL algorithms for the MDP defined in 4.1, including value-based methods (Mnih et al., 2013; Wang et al., 2015) and policy based method (Sutton et al., 2000; Schulman et al., 2015; 2017). Based on empirical results, we combine the advantage of policy and value based methods to learn a policy $\pi_\theta$. Because the immediate reward released by the environment is the penalty of adding one arc except the terminal state $\boldsymbol{s}_K$, to encourage the agent to focus on achieving better long-term rewards, we train a value network $V_\phi$ via minimizing the squared residual error:

$$J_V(\phi) = \mathbb{E}_{\tau \sim p_\pi(\tau)} \left( V_\phi(\boldsymbol{s}_t) - \hat{V}_\pi(\boldsymbol{s}_t) \right)^2 , \quad (6)$$

where $\hat{V}_\pi(\boldsymbol{s}_t) := \sum_{t'=t}^{K} \gamma^{t'-t} r(\boldsymbol{s}_{t'}, \boldsymbol{a}_{t'})$ is the discounted rewards-to-go. We further define the advantage function $A^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t)$ at step $t$ as

$$A^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t) = Q(\boldsymbol{s}_t, \boldsymbol{a}_t) - V_\phi(\boldsymbol{s}_t) , \quad (7)$$

where $Q(\boldsymbol{s}_t, \boldsymbol{a}_t) := r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \mathbb{E}_{\tau \sim p_\pi(\tau)} V_\phi(\boldsymbol{s}_{t+1})$. The advantage function of a policy $\pi$ describes the gain in the expected rewards when choosing action $\boldsymbol{a}_t$ over randomly selecting an action according to $\pi(\cdot|\boldsymbol{s}_t)$. We would like to maximize the expected reward in equation 2 via policy gradient:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim p_\pi(\tau)} \left[ A^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t) \nabla \log \pi_\theta(\boldsymbol{s}_t, \boldsymbol{a}_t) \right] . \quad (8)$$

As the vanilla policy gradient is unstable, we use proximal policy optimization (PPO) with a clipped advantage function. Details for our implementation are provided in appendix B.

**Policy Evaluation and FDP solutions** Following the literature (Jordan & Graves, 1995), we design our evaluation to leverage the fact that $P(\boldsymbol{d}, \mathbf{F})$ can be computed quickly for a given $\boldsymbol{d}$ and $\mathbf{F}$. The performance of a network is estimated using the objective function of FDP through 5000 random demand samples. Please note that using a very large number of samples in training does not provide better solution and renders much longer training time. To reduce the training time, we adopt an early stopping strategy when there is no performance increase for 48000 steps.

**Meta-Learning with Fast Adaptation** The ability to obtain solutions efficiently for multiple FDP instances is important in the real-world. For example, a firm may need to solve a series of FDP instances with different $K$, before making strategic decisions on how many arcs to have in its flexibility network. While we can apply the RL algorithm to train a different policy for each value of $K$ from scratch, this is usually inefficient. Hence, we consider a model-agnostic meta-learning (MAML) for fast adaption (Finn et al., 2017) to take advantage of the fact that FDP instances with different $K$ can be considered as similar tasks.

**Algorithm 1** Meta-Learning for FDPs

```
1:  Input: initialize parameters $\theta$ and $\phi$
2:  while not done do
3:      Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
4:      for All $\mathcal{T}_i$ do
5:          Sample trajectories $\tau_{\mathcal{T}_i} = \{(\boldsymbol{s}_0, \boldsymbol{a}_0, \ldots, \boldsymbol{s}_{K_{\mathcal{T}_i}})\}$ with $\pi_\theta$.
6:          Update adapted parameters $\theta_{\mathcal{T}_i}$ and $\phi_{\mathcal{T}_i}$ with $\tau_{\mathcal{T}_i}$.
7:          Sample trajectories $\tau = \tau \cup \{(\boldsymbol{s}_0, \boldsymbol{a}_0, \ldots, \boldsymbol{s}_{K_{\mathcal{T}_i}})\}$ with $\pi_{\theta_{\mathcal{T}_i}}$.
8:      end for
9:      Update the meta-policy $\pi_\theta$ and $V_\phi$ with $\tau$.
10: end while
```

Solving FDP with different instances can be regarded as a set of tasks $\{\mathcal{T}_i\}_{i=1}^m$. The tasks $\{\mathcal{T}_i\}_{i=1}^m$ share the same deterministic transitions and similar solution spaces, and the difference between them is at most how many arcs can be added. Specifically, we first train a meta model $\pi_\theta$ using the observations from different tasks ($K$ can be randomly selected). Then, we perform task-adaptation to adapt the meta policy to a specific task, *i.e.*, an FDP with a specific number of $K$ arcs. Intuitively, through pre-training across tasks of different $K$ values, the meta-model learns arc-adding policies that are generically effective for all the tasks, which eliminates the need for each individual task to train from scratch. The ablation study in Section 5.3 shows that a meta-model can quickly adapt to a specific FDP, significantly reducing the training time. In contrast, fast adaptation is not available for many classical heuristics, and the heuristics need to be performed for each task. When we have many different FDPs to solve, our meta learner presents a significant advantage in terms of efficiency and performance as shown in Section 5.3.

## 5 Experimental Results

We report a series of experiments conducted using the RL algorithm described in Section 4, including comparison of the RL algorithm with other heuristics under various scenarios and ablation studies. Our code will be publicly available at https://github.com/anonymous/RL_flex_design for reproduction.

### 5.1 RL Experiment Settings

The RL experiments are carried out on a Linux computer with one GPU of model NVIDIA GeForce RTX 2080 Ti. For all test cases, a three-layer Perceptron is used for both policy and value networks, with hidden layer sizes 1024 and 128 which is selected from ranges [256, 2048] and [64, 1024], respectively. For each epoch, approximately 800 episodes of games are played to collect trajectories for training. For the hyperparameters of the GAE, we select $\gamma = 0.99$ from the range [0.97, 0.999] and $\lambda = 0.999$ from the range of [0.97, 1]. Default values are used for the rest of hyperparameters of PPO, listed as below: clip ratio = 0.2, learning rate of $\pi$ = 3e-4, learning rate of the value function = 1e-3, train iterations for $\pi$ = 80, train iterations for value function =80, and target KL-divergence = 0.01. Finally, for each instance of the FDP, we perform training runs with 12 different seeds, and use the policy from each seed to extract 50 flexibility networks via greedy action selection, producing a total of 600 flexibility networks. Then, we choose the best network among the 600 using a set of 5000 randomly generated demand samples.

We implement two heuristics from the literature to benchmark against the solutions from our RL algorithm. In particular, we implement the greedy heuristic and the stochastic programming based heuristic (referred to as the SP heuristic thereafter) proposed by Feng et al. (2017). The full details for the heuristics are left in appendix A.

### 5.2 RL vs Benchmark Heuristics

We compare our RL algorithm against the two benchmark heuristics in four different scenarios. For each scenario, we create seven FDP instances by setting $K$ equal to $\{n, n+3, \ldots, n+18\}$. We pick these values of $K$ because if $K < n$, then some of the demand nodes would not be connected to any arcs, creating an impractical situation where some demand nodes are being ignored completely. Also, based on the numerical study, the expected profit for all three methods reaches a plateau once $K$ is above $n + 18$. In each FDP instance, the performance for all methods, defined as the expected profit of the returned flexibility networks, are evaluated using 10,000 new randomly generated demand samples. As a benchmark, we compute an upper-bound on the optimal objective by relaxing the integrality constraints and the constraint on $K$. Next, we describe our test scenarios.

**Automotive Manufacturing Scenario** This scenario is motivated by an application for a large auto-manufacturer described in Jordan & Graves (1995). In the scenario, there are 8 manufacturing plants (represented by supply nodes), where each has a fixed capacity, and 16 different vehicle models (represented by demand nodes), where each has a stochastic demand. A unique feature in this scenario is that $I_{ij} = 0$ and $p_{ij} = 1$ for each $i \in [m]$ and $j \in [n]$. The goal in this scenario is to find the best production network (determining which the types of vehicle models that can be produced at each plant) with $K$ arcs that achieves the maximum expected profit.

**Fashion Manufacturing Scenario** This scenario is created based on an application for a fashion manufacturer described in Chou et al. (2014) that extends the setting of Hammond & Raman (1996). In this scenario, a sports fashion manufacturing firm has 10 manufacturing facilities (represented by supply nodes) and produces 10 different styles of parkas (represented by supply nodes) with

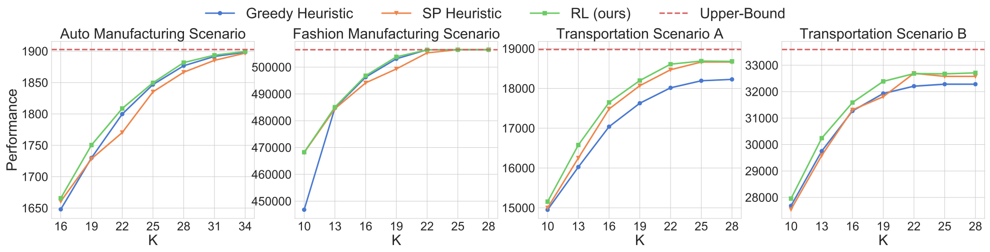


Figure 2: *Comparison of RL with other heuristic methods over four test scenarios.*

stochastic demand. The goal of the firm in this scenario is to find the best production network which determines the styles of parkas that can be produced at each facility with $K$ arcs that achieves the maximum expected profit. A unique feature in this scenario is that $I_{ij} = 0$ but values of $p_{ij}$ are different for different pairs of $i$ and $j$.

**Transportation Scenarios** These two test scenarios are created based on an application for a transportation firm, using publicly available testing instances on the fixed charge transportation problem (FCTP) maintained by Mittelmann (2019). In the transportation scenarios, a firm is required to ship from origins with fixed capacities to destinations with stochastic demand, and the goal is to identify the best set of origin to destination routing network with at most $K$ routes that achieves the maximum expected profit. The two transportation scenarios we create are based on two FCTP instances from Mittelmann (2019), which will be referred to as transportation scenario A and transportation scenario B. In these two scenarios, values of both $I_{ij}$ and $p_{ij}$ are positive and different for different pairs of $i$ and $j$.

**Comparison Summary Across Test Scenarios** In Figure 2, we plot the performances of the greedy heuristic, the SP heuristic, and the RL algorithm. Figure 2 demonstrates that the RL algorithm consistently outperforms both the greedy and SP heuristics. In each of the four scenarios, the RL algorithm is better than both benchmark heuristics in the average performance over different values of $K$. In addition, for all 28 FDP instances across the four testing scenarios, the RL algorithm is better than both benchmark heuristics in 25 of the instances, and only slightly under performs the best benchmark heuristic in the other 3 instances. Therefore, our numerical comparisons demonstrate that the RL algorithm is more effective compared to the existing heuristics. The exact performance values for the greedy heuristic, the SP heuristic, and the RL algorithm, can be found in Tables 2 thru 5 in appendix E. In addition, we plot the training curves by plotting the average MDP return vs the number of training steps for all four testing scenarios (Figure 5 in appendix E). In general, the average return increases (not necessarily at a decreasing rate) with the number of training steps, until it starts to plateau. Also, the number of training steps needed to converge increases with $K$. This is because greater $K$ implies a longer horizon in the MDP and larger solution space, which requires a higher number of trajectories to find a good policy.

It is important to point out that our current RL algorithm has longer computational times compared to both the greedy and SP heuristics. For the test instances with $m = n = 10$, the greedy heuristic takes approximately $6K$ seconds, the SP heuristic takes approximately $10K$ seconds, and the RL algorithm takes about $300K$ seconds. Because FDPs arise in long-term strategic planning problems, the computational time of RL is acceptable. For larger problems, however, the current RL algorithm may be too slow. This motivates us to perform ablation studies investigating different reward estimation and proposing meta-learning with fast adaption to reduce the training steps of RL.

## 5.3 Ablation Studies

We next present several ablation studies for the RL algorithm. All ablation studies are performed under transportation scenario A, with $K = 13$ and $K = 22$. For each instance, we perform 12 training runs with different random seeds.

**Using Different Reward Estimations** In the first ablation study, we compare the RL algorithms with or without the variance reduction (VR) method described in equation 5, for different values of $\Omega$. In Figure 3, we present the training curves for $\Omega \in \{1, 20, 50\}$ with or without the VR method, as well as training curves without the VR method for $\Omega \in \{1, 20, 50, 200, 1000\}$. We find that $\Omega = 1$ is not a good choice, as it takes many steps to converge and converges at a lower average return compared to other choices of $\Omega$. For other values of $\Omega$, the average return after convergence are similar, but they differ in the number of training steps required. In general, either increasing $\Omega$ or adding the VR method reduces the number of training steps. Interestingly, holding everything else equal, the reduction in training steps is smaller when $\Omega$ is large, and thus, there is little benefit in increasing $\Omega$ once in the tested instances when $\Omega$ is above 50. In addition, holding everything else equal, the reduction in train steps also increases with $K$. This is because a larger $K$ implies more steps in the MDP and larger solution space, and the training algorithm in such problems benefits more with smaller return variance.

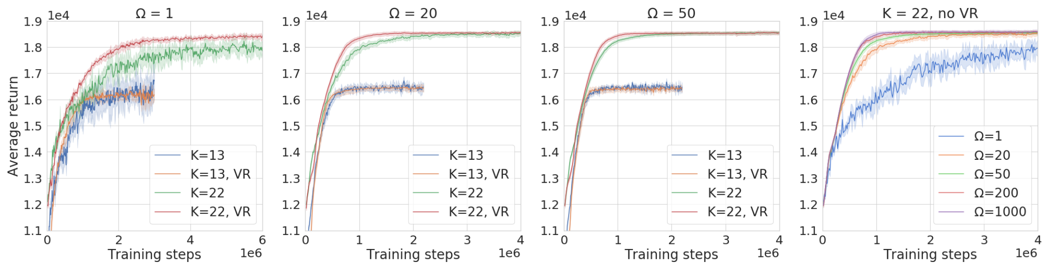


Figure 3: *Training curves with different values $\Omega$ with and without VR.*

**Meta Learning with Fast Adaptation** In the next ablation study, we test the effectiveness of meta-learning with fast adaptation. For this study, we learn a meta-model with different $K$ with first-order MAML, either for 100 epochs (referred to as Meta100), or until convergence (referred to as MetaConvg), and then use the trained parameters from the meta-models to perform adaption for tasks with different values of $K$.

In Figure 4, we present the training curves for meta-learning and fast adaptation with both Meta100 initialization and MetaConvg initialization. We observe that meta-learning can significantly reduce the number of training steps during the fast adaption step, making it very efficient when one needs to solve FDP instances with different values of $K$. Interestingly, we also observe that Meta100 is very much comparable to MetaConvg, implying that it is not necessary to pre-train the meta-model to convergence, thus further reducing the total number of training steps.

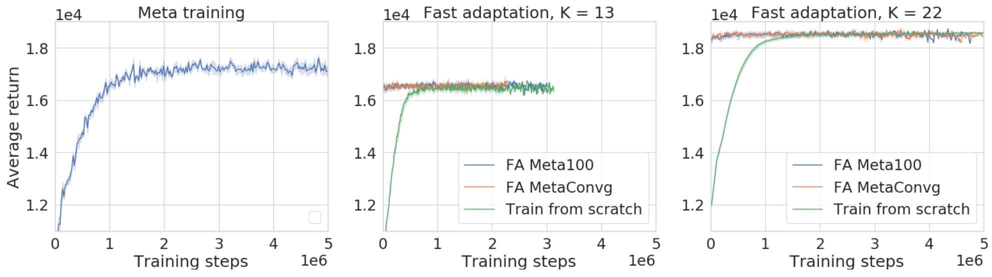


Figure 4: *Training curves for meta-training, and fast adaptions.*

## 6 Conclusion

We have proposed a reinforcement learning framework for solving the flexibility design problem, a class of stochastic combinatorial optimization problems. We find that RL is a promising approach for solving FDP, and our RL algorithm consistently found better solutions compared to the benchmark heuristics. Finally, we believe that the findings of this paper suggest that an important future direction is to apply RL algorithms to other two-stage stochastic combinatorial optimization problems, especially to those problems that arise from strategic planning under uncertainty.

## REFERENCES

Kenshin Abe, Zijian Xu, Issei Sato, and Masashi Sugiyama. Solving np-hard problems on graphs by reinforcement learning without domain knowledge. *arXiv preprint arXiv:1905.11623*, 2019.

Yogesh Agarwal and Yash Aneja. Fixed-charge transportation problem: Facets of the projection polyhedron. *Operations research*, 60(3):638–654, 2012.

Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. In *ICLR*, 2017a.

Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. Accelerating neural architecture search using performance prediction. In *ICLR Workshop*, 2017b.

Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *arXiv preprint arXiv:1811.06128*, 2018.

M. C. Chou, G. A. Chua, C.-P. Teo, and H. Zheng. Design for process flexibility: Efficiency of the long chain and sparse structure. *Operations Research*, 58(1):43–58, 2010.

M. C. Chou, G. A. Chua, C.-P. Teo, and H. Zheng. Process flexibility revisited: the graph expander and its applications. *Operations Research*, 59(5):1090–1105, 2011.

Mabel C Chou, Chung-Piaw Teo, and Huan Zheng. Process flexibility: design, evaluation, and applications. *Flexible Services and Manufacturing Journal*, 20(1-2):59–94, 2008.

Mabel C Chou, Geoffrey A Chua, and Huan Zheng. On the performance of sparse process structures in partial postponement production systems. *Operations research*, 62(2):348–365, 2014.

A. Désir, V. Goyal, Y. Wei, and J. Zhang. Sparse process flexibility designs: Is the long chain really optimal? *Operations Research*, 64(2):416–431, 2016.

Levi DeValve, Saša Pekeč, and Yehua Wei. Matching supply and demand in a resource constrained service network. *working paper*, 2020.

Wancheng Feng, Chen Wang, and Zuo-Jun Max Shen. Process flexibility design in heterogeneous and unbalanced networks: A stochastic programming approach. *IISE Transactions*, 49(8):781–799, 2017.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 2017.

Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, et al. Noisy networks for exploration. 2018.

LLC Gurobi Optimization. Gurobi optimizer reference manual, 2020. URL http://www.gurobi.com.

JH Hammond and A Raman. Sport obermeyer, ltd. harvard business school case 9-695-022. *Harvard Business Review*, 1996.

W. Jordan and S. Graves. Principles on the benefits of manufacturing process flexibility. *Management Science*, 41(4):577–594, 1995.

Wouter Kool, Herke Van Hoof, and Max Welling. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, 2018.

Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. In *ICLR*, 2018.

Hans D Mittelmann. Decision tree for optimization software, 2019.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Mohammadreza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takác. Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems*, pp. 9839–9849, 2018.

Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz. Parameter space noise for exploration. 2018.

Tjendera Santoso, Shabbir Ahmed, Marc Goetschalckx, and Alexander Shapiro. A stochastic programming approach for supply chain network design under uncertainty. *European Journal of Operational Research*, 167(1):96–115, 2005.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

D. Simchi-Levi. *Operations Rules: Delivering Customer Value through Flexible Operations*. MIT Press, Cambridge, MA, 2010.

D. Simchi-Levi and Y. Wei. Understanding the performance of the long chain and sparse designs in process flexibility. *Operations Research*, 60(5):1125–1141, 2012.

D. Simchi-Levi and Y. Wei. Worst-case analysis of process flexibility designs. *Operations Research*, 63(1):166–185, 2015.

Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pp. 1057–1063, 2000.

Jan A Van Mieghem. Investment strategies for flexible resources. *Management Science*, 44(8): 1071–1078, 1998.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in neural information processing systems*, pp. 2692–2700, 2015.

Shixin Wang, Xuan Wang, and Jiawei Zhang. A review of flexible processes and operations. *Production and Operations Management*, 2019.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017.

## A Details for the Benchmark Heuristics.

In the greedy heuristic, we initialize $\mathbf{F}^0$ to be vector correspond to the empty flexibility network. At iteration $k$, we evaluate the expected increase in profit in adding arc $(i,j)$ to $\mathbf{F}^{k-1}$ for every possible arc $(i,j)$ and identify arc $(i^k, j^k)$ with the highest expected profit increase. If the highest expected increase is non-positive, we terminate the algorithm and return $\mathbf{F}^{k-1}$ as the final output. Otherwise, we add $(i^k, j^k)$ to $\mathbf{F}^{k-1}$ to obtain $\mathbf{F}^k$, then return $\mathbf{F}^k$ as the final output if $k = K$ or continue to iteration $k+1$. Because the expected profit for a flexibility network cannot be computed exactly, we will estimate expected profit using a set of $\Omega$ randomly generated demand samples, denoted as $\{\boldsymbol{d}^\omega\}_{1\le\omega\le\Omega}$. More specifically, for any flexibility network $\mathbf{F}$, we estimate its expected profit using the average profit of $\mathbf{F}$ over the demand samples, that is

$$\mathbb{E}_{\mathbf{d}\sim\mathbf{D}}[P(\boldsymbol{d},\mathbf{F})] \approx \frac{1}{\Omega}\sum_{\omega=1}^{\Omega} P(\boldsymbol{d}^\omega,\mathbf{F}). \tag{9}$$

The SP heuristic we implement is suggested by Feng et al. (2017). We note that although there are various heuristics proposed for more specific sub-classes of the FDP ($e.g.$, Chou et al. (2010; 2011); Simchi-Levi & Wei (2015)), with the exception of Feng et al. (2017), most these heuristics are not intended for solving general FDPs. In the SP heuristic, we initialize $\mathbf{F}^0$ to be the empty flexibility network. At iteration $k$, we consider the FDP problem that is approximated using a set of $\Omega$ randomly generated demand samples $\{\boldsymbol{d}^\omega\}_{1\le\omega\le\Omega}$, then relax its integer constraints and enforcing all arcs in $\mathbf{F}^{k-1}$ to be included. In particular, we solve the following linear stochastic programming (SP) problem:

$$\begin{aligned}
\max_{\mathbf{F}\in\mathbb{R}^{mn},\mathbf{f}\in\mathbb{R}^{mn\Omega}} \quad & \frac{1}{\Omega}\sum_{1\le\omega\le\Omega}\ \sum_{i\in[m],j\in[n]} p_{ij}f_{ij}^{\omega} - \sum_{i\in[m],j\in[n]} I_{ij}F_{ij} \\
\text{s.t.} \quad & \sum_{i\in[m]} f_{ij}^{\omega} \le d_j^{\omega},\ \forall j\in[n], \\
& \sum_{j\in[n]} f_{ij}^{\omega} \le c_i,\ \forall i\in[m], \\
& 0 \le f_{ij}^{\omega} \le F_{ij}M_{ij}, \forall i\in[m], j\in[n], \\
& \sum_{i\in[m],j\in[n]} F_{ij} \ge F_{ij}^{k-1}, \\
& \sum_{i\in[m],j\in[n]} F_{ij} \le K.
\end{aligned} \tag{10}$$

After obtaining the solution from equation 10, the heuristic identifies arc $(i^k, j^k)$ with the highest fractional solution value among all arcs that are not in $\mathbf{F}^{k-1}$, and add $(i^k, j^k)$ to $\mathbf{F}^{k-1}$ to $\mathbf{F}^k$. The heuristic terminates after the $K$-th iteration. We refer to this heuristic as the *SP Heuristic*.

For both greedy and SP heuristics, we choose $S = 1000$ and compare the networks returned by the greedy heuristic, the SP heuristic and the network returned by the RL-based algorithm for all test instances.

## B Details for PPO

Our training algorithm is the proximal policy optimization (PPO) algorithm with a clipped advantage function is introduced in Schulman et al. (2017). As a result, we denote our algorithm as PPO-Clip. Under the PPO-Clip algorithm, during the $k+1$-th epoch, we collect a set of trajectories $\mathcal{D}_k$ by running policy $\pi_{\theta_k}$ (a stochastic policy parameterized by $\theta_k$) in the environment $|\mathcal{D}_k|$ times, where each trajectory $\tau\in\mathcal{D}_k$ is represented by a sequence of states and actions $(s_0^\tau, a_1^\tau, s_2^\tau, , ..., s_K^\tau)$. After $\mathcal{D}_k$ is collected, we solve the optimization problem

$$\max_{\theta} \sum_{\tau\in\mathcal{D}_k, t\in[K]} \min\left(\frac{\pi_\theta(a_t^\tau|s_t^\tau)}{\pi_{\theta_k}(a_t^\tau|s_t^\tau)} A^{\pi_{\theta_k}}(s_t^\tau, a_t^\tau), g(\epsilon, A^{\pi_{\theta_k}}(s_t^\tau, a_t^\tau))\right), \tag{11}$$

where $A$ is the advantage function defined in equation 7, and $g$ is the clipping function defined as

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A, \text{ if } A \geq 0, \\ (1-\epsilon)A, \text{ if } A < 0. \end{cases}$$

Once equation 11 is solved, we let $\theta_{k+1}$ to denote the optimal solution obtained in equation 11 and update the stochastic policy by $\pi_{\theta_{k+1}}$. A psuedo-code for how $\pi_{\theta_{k+1}}$ is updated in our learning algorithm is provided in Algorithm 2.

**Algorithm 2** Pseudo Code for PPO-Clip

```
1: Input: initialize parameters $\theta_0$ and $\phi_0$
2: **for** $k = 0, 1, 2, ...$ **do**
3:     Collect set of trajectories $\mathcal{D}$ by running policy $\pi_{\theta_k}$ in the environment.
4:     For each trajectory, compute the rewards-to-go $\{\hat{R}_t^\tau\}_{t=1}^T$.
5:     Compute advantage estimates, $\hat{A}_t$
6:     Update the policy parameter by set $\theta_{k+1}$ to be the solution of the following optimization problem:
           $$\theta_{k+1} = \arg\max_\theta \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k, t \in [T]} \min\left( \frac{\pi_\theta(a_t^\tau | s_t^\tau)}{\pi_{\theta_k}(a_t^\tau | s_t^\tau)} \hat{A}^{\pi_{\theta_k}}(s_t^\tau, a_t^\tau), g(\epsilon, \hat{A}^{\pi_{\theta_k}}(s_t^\tau, a_t^\tau)) \right).$$
7:     Update the value function parameter by set $\phi_{k+1}$ to be the solution of the following optimization problem:
           $$\phi_{k+1} = \arg\min_\phi \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k, t \in [T]} \left( V_\phi(s_t) - \hat{R}_t^\tau \right)^2.$$
8: **end for**
```

## C Parameters for the Test Scenarios

**Automotive Manufacturing Scenario** In the automotive manufacturing scenario, the parameters for the FDP instances are set as follows.

The capacity vector for resource nodes, $\boldsymbol{c}$, and the mean for the demand distribution at the demand nodes ($\mathbb{E}[\mathbf{D}]$), denoted by $\boldsymbol{\mu}$, are given in Jordan & Graves (1995) as

$$\boldsymbol{c} = [380, 230, 250, 230, 240, 230, 230, 240],$$
$$\boldsymbol{\mu} = [320, 150, 270, 110, 220, 110, 120, 80, 140, 160, 60, 35, 40, 35, 30, 180].$$

We set that the distribution of $\mathbf{D}$ is independent and normal with mean $\boldsymbol{\mu}$ and standard deviation $\boldsymbol{\sigma} = 0.8\boldsymbol{\mu}$. Furthermore, for each $j$, we truncate $d_j$ to be within $[0, \mu_j + 2\sigma_j]$ to avoid negative demand instances. Finally, like Jordan & Graves (1995), we set $p_{ij} = 1$, $I_{ij} = 0$ for each $1 \leq i \leq m$ and $1 \leq j \leq n$.

**Fashion Manufacturing Scenario** In the fashion manufacturing scenario, the parameters for the FDP was selected based on the setting described in Chou et al. (2014). Specifically, the capacity vector for resources, $\boldsymbol{c}$, and the mean for demand nodes, $\boldsymbol{\mu}$, are set to be equal to each other with values

$$\boldsymbol{c} = \boldsymbol{\mu} = [1017, 1042, 1358, 2525, 1100, 2150, 1113, 4017, 3296, 2383].$$

We assume that the distribution of $\mathbf{D}$ is independent and normal with mean $\boldsymbol{\mu}$ and standard deviation $\boldsymbol{\sigma}$, where the values of $\boldsymbol{\sigma}$ is obtained from Chou et al. (2014), with $\boldsymbol{\sigma} = [194, 323, 248, 340, 381, 404, 524, 556, 1047, 697]$. Furthermore, for each $j$, we truncate $d_j$ to be within $[0, \mu_j + 2\sigma_j]$ to avoid negative demand instances. In addition, the prices of the parkas styles are given by

$$\boldsymbol{q} = [110, 99, 80, 90, 123, 173, 133, 73, 93, 148],$$

and the profit margin is 24%. Thus, we set the unit profit for using one resource $i$ to satisfy one demand $j$, $p_{ij}$, to $p_{ij} = 0.24q_j$, for each $1 \leq i \leq m$ and $1 \leq j \leq n$. Finally, like Chou et al. (2014), we choose $I_{ij} = 0$ for all $1 \leq i \leq m$ and $1 \leq j \leq n$.

**Transportation Scenarios** We create transportation scenario A and transportation scenario B from two FCTP instances from Mittelmann (2019), which are named as 'ran10x10a' and ''ran10x10b' in Mittelmann (2019). In an FCTP, the firm is required to ship from origins with fixed capacities to destinations with fixed demands, and the goal is to minimize the total transportation cost plus the fixed cost of transporting on a origin-destination transportation network. To convert an FCTP into an FDP, we use the supply nodes to represent the origins and the demand nodes to represent the destinations, and perturb the demand of the destinations with a random noise. More specifically, we set the capacity vector $\boldsymbol{c}$ as the capacity of the origins and the mean for demand nodes as the demand of the destinations from the FCTP problem. We further set that the distribution of $\mathbf{D}$ is independent and normal with mean $\boldsymbol{\mu}$ and standard deviation $\boldsymbol{\sigma} = 0.8\boldsymbol{\mu}$. For each $1 \le i \le m$ and $1 \le j \le n$, the value of $I_{ij}$ is set to the fixed charge of each arc of the FCTP instance. In addition, let the unit transportation cost from the FCTP instance for each original $i$ and destination $j$ to be $t_{ij}$. We set the unit profit from resource $i$ to demand $j$, $p_{ij}$, to be a large constant $P_{const}$ minus $t_{ij}$ where $P_{const}$ is computed as

$$P_{const} = \max_{(i,j)\in\mathbb{R}^{mn}} (I_{ij} + t_{ij}).$$

This choice of $P_{const}$ and $p_{ij}$ allows the FTP problem with an unbounded number of arcs to be equivalent to the original FCTP testing instance when the standard deviation of $\mathbf{D}$ is the zero vector.

## D Additional Ablation Study

**Enlarging the Action Set** We consider a different action set where given state $s = \mathbf{F}$, each action can remove arc $(i, j)$ from $\mathbf{F}$ if $F_{ij} = 1$, add $(i, j)$ if $F_{ij} = 0$, or choose to do nothing and keep the current state ($i.e.$, no-op). We can use this action set to replace the original action set defined in the MDP and obtain a new MDP that is also equivalent to the FDP problem.

This action set is considered because the new MDP has more opportunities to change the state than the original action set, and thus help us to check if the additional opportunities in the new action set may fasten the training process or help our RL algorithm to find policies that identify better FDP solutions compared to the original action set. We denote the default action set as **add/no-op** and the new action set as **add/delete/no-op**.

Table 1: Average performance, best performance, and average training steps for different action sets

| | $K = 13$ | | | $K = 22$ | | |
|---|---|---|---|---|---|---|
| | Avg Perf | Best Perf | Avg Steps | Avg Perf | Best Perf | Avg Steps |
| add/no-op | 16463 | 16597 | 132640 | 18529 | 18605 | 181360 |
| add/delete/no-op | 16469 | 16670 | 154640 | 18528 | 18622 | 218640 |

In Table 1, we present the average performance and best performance of the structures returned by the MDP with the original action set (add/no-op) and the add/delete/no-op action set under transportation scenario A with $K = 13$ and $K = 22$. We find no significant difference in the performances between the two action sets. In addition, we also provide the average number of training steps used in training for both action sets. The RL algorithm takes on average more training steps under the new action set, and therefore require a longer computational time as well. Therefore, we conclude that there seems to be no benefit to use the add/delete/no-op action set.

## E Additional Tables and Graphs for Experimental Results

We provide the specific values for the expected profit achieved by each of the methods for all of the test scenarios, via Tables 2 thru 5. For each instance, the method with the highest expected profit is highlighted in bold. The training curves for RL for all test scenarios are presented as Figure 5

Table 2: Expected Profit for RL vs Greedy and SP Heuristics for the Auto Manufacturing Scenario

| **Methods** | $K$**=16** | **19** | **22** | **25** | **28** | **31** | **34** |
|---|---|---|---|---|---|---|---|
| Greedy | 1648.0 | 1730.0 | 1799.8 | 1846.9 | 1876.8 | 1891.6 | 1898.3 |
| SP | 1660.7 | 1728.5 | 1770.3 | 1835.2 | 1866.5 | 1885.4 | 1896.9 |
| RL | **1665.7** | **1750.5** | **1808.7** | **1849.7** | **1881.8** | **1893.7** | **1899.6** |

Table 3: Expected Profit for RL vs Greedy and SP Heuristics for the Fashion Manufacturing Scenario

| **Methods** | $K$ **= 10** | **13** | **16** | **19** | **22** | **25** | **28** |
|---|---|---|---|---|---|---|---|
| Greedy | 446809.9 | 484788.8 | 496262.8 | 503107.5 | **506480.3** | 506497.2 | 506497.2 |
| SP | 468137.4 | 484474.0 | 494125.4 | 499337.6 | 505321.3 | **506499.9** | 506500.1 |
| RL | **468248.3** | **485085.0** | **496812.6** | **503852.2** | 506449.5 | 506497.0 | **506501.3** |

Table 4: Expected Profit for RL vs Greedy and SP Heuristics for Transportation Scenario 3A

| **Methods** | $K$ **= 10** | **13** | **16** | **19** | **22** | **25** | **28** |
|---|---|---|---|---|---|---|---|
| Greedy | 14950.8 | 16025.0 | 17038.3 | 17627.4 | 18014.4 | 18191.6 | 18226.3 |
| SP | 14999.2 | 16250.4 | 17483.7 | 18067.6 | 18468.4 | 18661.4 | 18661.8 |
| RL | **15155.6** | **16577.9** | **17648.3** | **18197.1** | **18608.0** | **18686.4** | **18678.7** |

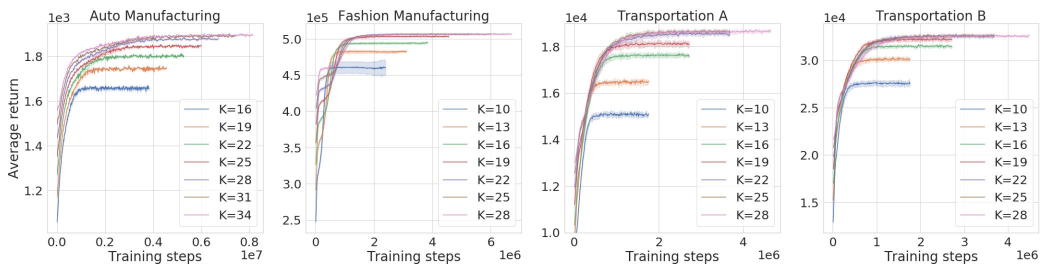


Figure 5: *Training curve for different values of $K$.*

Table 5: Expected Profit for RL vs Greedy and SP Heuristics for Transportation Scenario 3B

| **Methods** | $K$ **= 10** | **13** | **16** | **19** | **22** | **25** | **28** |
|---|---:|---:|---:|---:|---:|---:|---:|
| Greedy | 27682.5 | 29751.5 | 31271.1 | 31929.1 | 32207.2 | 32282.6 | 32282.6 |
| SP | 27546.1 | 29587.2 | 31312.2 | 31808.0 | **32695.0** | 32578.0 | 32578.0 |
| RL | **27956.2** | **30241.9** | **31588.1** | **32388.5** | 32682.5 | **32674.8** | **32710.1** |