Solving the shortest path interdiction problem via reinforcement learning

Article *in* International Journal of Production Research · December 2021

DOI: 10.1080/00207543.2021.2002962

4 authors, including:

Dian Huang
Tianjin University
12 PUBLICATIONS **65** CITATIONS
SEE PROFILE

Kan Fang
Tianjin University
25 PUBLICATIONS **1,362** CITATIONS
SEE PROFILE

Solving the shortest path interdiction problem via reinforcement learning

Dian Huang[a,b], Zhaofang Mao[a], Kan Fang[a,*], Lin Chen[c]

[a]*College of Management and Economics, Tianjin University, Tianjin 300072, China*

[b]*School of New Media and Communication, Tianjin University, Tianjin 300072, China*

[c]*Department of Computer Science, Texas Tech University, Lubbock, TX 79409, USA*

Abstract

This paper addresses the shortest path interdiction problem, in which the leader aims to maximize the length of the shortest path that the follower can traverse subject to a limited interdiction budget. To solve this problem, we propose a reinforcement learning framework and use the pointer network to handle the situation of variable output sizes. To evaluate the performance of our proposed reinforcement learning model, we conduct extensive computational experiments on a set of instances that are generated from two different network topologies, i.e., the grid networks and the random graphs. To train the pointer network, we consider three different baselines, i.e., the exponential, critical, and rollout baselines, among which the rollout baseline policy achieves the best computational results, and thus is used as the default baseline during our computational experiments. Moreover, when the size of instances increases, we find that solving the equivalent single-level mixed integer program of the problem could be quite time-consuming, while our proposed reinforcement learning approach can still obtain solutions with good performance effectively for both the grid networks and the random graphs.

Keywords: Network interdiction, Shortest paths, Reinforcement learning, Pointer network, Rollout baseline

*Corresponding author. Email address: kfang@tju.edu.cn (Kan Fang)

Email addresses: huangdian@tju.edu.cn (Dian Huang), maozhaofang@tju.edu.cn (Zhaofang Mao), Lin.Chen@ttu.edu (Lin Chen)

Preprint submitted to International Journal of Production Research

1. Introduction

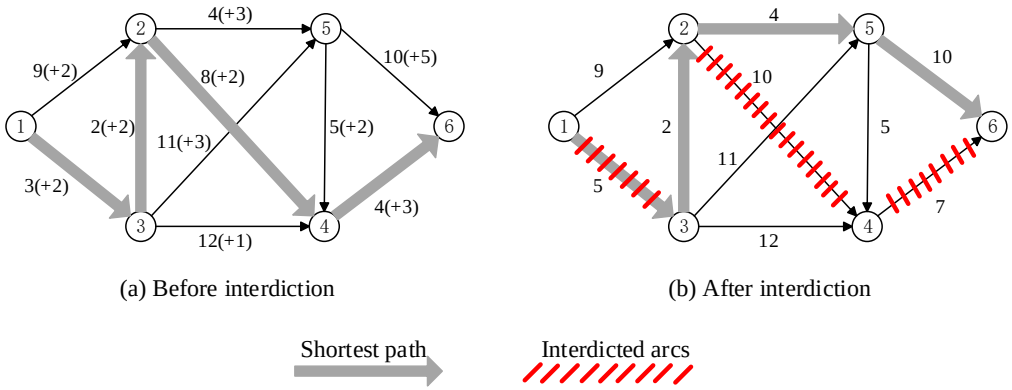
In the past few decades, the network interdiction problem has identified its applications in various situations in which one wishes to determine the weakest components or the worst-case deliberate external disruptions of a system. Until recently, a lot of researchers have used different variations of network interdiction problems to build resilient supply chain and logistic networks such as telecommunication, electric power, transportation, express logistics, water supply, sewage disposal, infection prevention, and natural gas and petroleum distribution systems (Cappanera and Scaparra, 2011). For example, Bier et al. (2007) proposed a greedy algorithm based method to identify promising interdiction strategies on transmission lines in electric power systems, and assess the vulnerability of such systems. Kheirkhah et al. (2016) presented a bilevel programming model to solve the hazardous material distribution problem, in which some arcs of the network can be interdicted by a regulatory agency. Jiang and Liu (2021) addressed the problem of optimizing the defense resource allocation against interdictions on water supply networks, so that the water requirements of public, commercial and industrial activities can be satisfied.

With its numerous applications, the network interdiction problem has played an indispensable role on today's societal, economic and industrial development. In particular, as a typical network interdiction problem, the shortest path interdiction problem has captured many researchers' attention over years in the operations research community. The reason for the rapidly growing studies on the shortest path interdiction problem comes from two-folds: First, many applications of this problem can be found in practical settings, such as nuclear smuggling (Morton et al., 2007), hazardous material transportation (Cappanera and Scaparra, 2011), procurement planning (Prince et al., 2013), border surveillance (Sullivan et al., 2014; Karabulut et al., 2017) and human trafficking (Konrad et al., 2017). Second, by investigating the shortest path interdiction problem as a starting point, the theoretical and algorithmic innovations derived from solving this problem may be used as a basis to solve more generalized network interdiction problems in a variety of scenarios.

To be specific, in a shortest path interdiction problem, two opposing players, called the *leader* (or *attacker, interdictor*) and the *follower* (or *defender*), are engaged in a Stackelberg game as follows: In a directed network, the follower wishes to find a path with minimum length between two specified nodes, i.e., the origin node O and the destination node D. On the other hand, before the follower chooses a path, the leader can take interdiction actions by destroying certain arcs or increasing the effective length of some arcs within limited interdiction

resources, so as to maximize the length of the shortest path that the follower can traverse.

To illustrate, Figure 1 provides a feasible solution to an instance of the shortest path interdiction problem (Smith and Song, 2020), in which the follower aims to find a shortest path from node 1 to node 6, the leader has a budget that can interdict up to three arcs, and the value given in parentheses represents the increased length after interdiction. Obviously, prior to the interdiction, the shortest path from node 1 to node 6 is $1 \to 3 \to 2 \to 4 \to 6$, with a length of 17 (the grey arcs in Figure 1(a)). When the slashed arcs are interdicted by the leader, we can see that the shortest path becomes $1 \to 3 \to 2 \to 5 \to 6$, and the associated length becomes 21 (the grey arcs in Figure 1(b)).



(a) Before interdiction (b) After interdiction

Figure 1: An illustration of the shortest path interdiction problem.

As we know, Fulkerson and Harding (1977) and Golden (1978) are the earliest work that consider the shortest path interdiction model, in both of which the arc lengths can be partially interdicted, and they showed that their problems are polynomial-time solvable, since each of them can be reduced to an equivalent minimum cost network flow problem. However, in many practical settings, the interdiction decisions are usually "all-or-nothing" decisions, i.e., binary variables, and Ball et al. (1989) have shown that a special case of the shortest path interdiction problem, in which the length of an arc becomes arbitrarily large once it is interdicted, and the interdiction resource is constrained by a single cardinality constraint, that is, there are at most k arcs can be interdicted for some pre-fixed value k, i.e., the k-most-vital-arcs problem, remains NP-hard.

Note that the shortest path interdiction problem is a special case of a static Stackelberg game, therefore, it is natural to formulate this problem as a bilevel mixed integer program (MIP). In particular, since the follower only attempts to find the shortest path from O to D, which can be modeled as a convex optimization problem, as a result, the dualize-and-combine

approach can be applied to make both players' problems share the same direction of optimization, and thus any standard optimization solver can be used to solve the corresponding single-level mixed integer program (Smith and Song, 2020). However, given the NP-hardness of this problem, researchers have observed that solving the single-level MIP directly can be extremely difficult even for some modest-sized instances of the problem. In the seminal work of Israeli and Wood (2002), the authors considered the shortest path interdiction problem in its most general form, and proposed two decomposition-based algorithms with the so-called "super-valid inequalities". By testing their proposed algorithms on a set of directed grid networks, the authors showed that such solution approach can achieve substantial speed-ups compared with the traditional linear-programming-based branch and bound method.

Starting with their work, many new variants of the shortest path interdiction problems have been investigated. Depending on the assumptions and objectives that have been considered in these studies, the shortest path interdiction problems can be further categorized with various characteristics, such as: (i) asymmetric information. The leader and follower have different levels of information about the network (e.g. Bayrak and Bailey, 2008; Baycik and Sullivan, 2019). (ii) dynamic interdiction. The leader can interdict arcs sequentially after observing the action of the follower in each period (e.g. Sefair and Smith, 2016). (iii) incomplete information. Both the leader and follower may not have complete information about their opponent or even the underlying network, and they have to learn key parameters from historical data or from their ongoing interactions (e.g. Borrero et al., 2016; Pay et al., 2019; Yang et al., 2021). (iv) fortify strategy. In addition to the above attacker-defender paradigm, researchers have also considered a three-stage defender-attacker-defender game, in which the defender can make a preliminary set of "fortification" decisions to protect infrastructure against potential interdictions (e.g. Cappanera and Scaparra, 2011; Sadeghi et al., 2017; Lozano and Smith, 2017). (v) various risk measures. For stochastic shortest path interdiction problems, in which the traveling cost and interdiction effect may be uncertain, various risk measures have been proposed, such as maximizing the expected cost of the follower (e.g. Holzmann and Smith, 2021), or risk-averse measures (e.g. Song and Shen, 2016). Meanwhile, the maximum reliable path interdiction problem, which can be viewed as a variant of the shortest path interdiction problem, has also been investigated by several researchers in recent years (e.g. Pan and Morton, 2008). To summarize, Table 1 provides the relevant studies on the shortest path interdiction literature.

To solve the shortest path interdiction problems, researchers have mainly used some row-generation techniques to develop various decomposition based algorithms. Among these ap-

Table 1: Relevant studies on the shortest path interdiction literature.

References	Interdiction characteristics	Objective function of leader	Solution methodologies
Bayrak and Bailey (2008)	Deterministic, asymmetric information	Maximize the shortest path	Mixed integer linear program
Khachiyan et al. (2008)	Deterministic, node-wise limited interdiction	Maximize the shortest path	Dijkstra's algorithm
Pan and Morton (2008)	Stochastic, uncertain origin-destination pair	Minimize the detection probability	L-shaped decomposition method
Rocco and Ramirez-Marquez (2010)	Deterministic, multiple objectives	(i)Maximize the shortest path (ii)Minimize the interdiction strategy cost	Evolutionary algorithm
Cappanera and Scaparra (2011)	Deterministic, trilevel optimization, fortification	Maximize the shortest path	Enumeration algorithm
Yates and Lakshmanan (2011)	Deterministic, dynamic interdiction, multiple origins and targets	Minimize the maximum non-detection probability	Knapsack approximation
Borrero et al. (2016)	Deterministic, dynamic interdiction, incomplete information, learning	Maximize the cost of follower	Greedy policies
Sefair and Smith (2016)	Deterministic, dynamic interdiction	Maximize the shortest path	Dynamic programming
Song and Shen (2016)	Stochastic, uncertain arc cost, risk-averse leader	Maximize the shortest path	Branch-and-cut algorithm
Lozano and Smith (2017)	Deterministic, trilevel optimization, fortification	Maximize the shortest path	Backward sampling framework
Sadeghi et al. (2017)	Deterministic, trilevel optimization, partial interdiction, fortification	Maximize the shortest path	Benders decomposition algorithm
Baycik and Sullivan (2019)	Deterministic, asymmetric information, hidden interdictions	Maximize the shortest path	Benders decomposition algorithm
Pay et al. (2019)	Stochastic, incomplete leader's preference	Maximize the shortest path	Branch-and-cut algorithm
Holzmann and Smith (2019)	Deterministic, arc improvement by follower	(i)Maximize the shortest path (ii)Minimize the arc improvement budget	Mixed integer programming
Zhang et al. (2021a)	Deterministic, upgrading edges on trees, ℓ_1 norm	Maximize the shortest path of the tree	Primal dual algorithm
Zhang et al. (2021b)	Deterministic, upgrading edges on trees, hamming distance	Maximize the shortest path of the tree	Dynamic programming
Holzmann and Smith (2021)	Stochastic, randomized interdiction actions	maximize the expected cost of follower	Sample average approximation

proaches, different variations of Benders decomposition have been widely implemented, which can divide the original problem into a master problem and a subproblem, and solve them iteratively to pass information (e.g., the selection of paths by the follower or the resource allocations of the leader) between two players, and generate corresponding valid inequalities to the master problem until an optimal solution is found. In particular, when the decisions of the leader involve integrality restrictions, some additional "super-valid inequalities", which are generated by exploring the combinatorial structures of the problem, could be further used to strengthen the formulation (Israeli and Wood, 2002).

However, for large-scale systems such as the electric power grids or cyber-physical systems, it is still a bottleneck to compute the optimal interdiction strategies efficiently by those developed algorithms, and it is crucial to develop next-generation computationally efficient algorithms to compute near-optimal interdiction strategies. It should also be noticed that all of these state-of-the-art algorithms that have been developed to solve the shortest path interdiction problems are often adapted to particular structures of the problems, and they might not work consistently across all possible instances, that is, once the input of the problem changes slightly, the developed algorithms may need to be revised to guarantee their performance (Bengio et al., 2021).

In recent years, more and more researchers have noticed that machine learning methods have the potential to solve various combinatorial optimization problem in a more principled and optimized way, which requires less handcrafted heuristics for making decisions. For example, Furian et al. (2021) proposed a learning-based branch and price framework to solve a sampled vehicle routing problem with time windows, which is assumed to follow a specific pattern, and showed that their proposed approaches outperform standard algorithms. Chen et al. (2021) took bus scheduling as a case study to show that their proposed framework that integrates analytics and machine learning approaches could be used to address uncertainties and conflicting objectives for challenging real-life optimization problems. Bai et al. (2021) presented a comprehensive review on various hybrid methods that combines analytics techniques with machine learning tools in addressing vehicle routing problems (VRP). Note that it is often difficult to obtain the optimal labels for most combinatorial optimization problems, therefore the supervised learning, which has been widely used in many other machine learning contexts, may be not applicable to most combinatorial optimization problems.

Recently, several researchers have observed that the reinforcement learning, in which an agent interacts with an environment through a markov decision process (MDP) and seeks to

make a series of sequential decisions to maximize the expected sum of future rewards, can be applied to address combinatorial optimization problems, such as the traveling salesman, bin packing, scheduling, and vehicle routing problems (Mazyavkina et al., 2021). For example, Lu et al. (2020) proposed a learning based approach for capacitated vehicle routing problems, which learns to iteratively refine the solution with an improvement operator selected by a reinforcement learning-based controller. Duan et al. (2020) proposed a graph convolutional network based model, which is trained by a hybrid strategy that combines reinforcement learning with supervised learning, to solve practical vehicle routing problems. Lee and Kim (2021) used a reinforcement learning approach to solve the robotic flow shop scheduling problem with processing time variation. Achamrah et al. (2021) considered a dynamic and stochastic inventory routing problem with transshipment and substitution, and proposed a resolution approach based on genetic algorithm and deep reinforcement learning.

In particular, the architecture of pointer networks, which was proposed by Vinyals et al. (2015), enables the reinforcement learning to address problems defined on graphs that arise with variable output dictionaries, that is, the dictionary size of the output is not necessarily equal to the length of the input. For example, Bello et al. (2017) employed the pointer network architecture and presented a framework to solve the traveling salesman and knapsack problems with reinforcement learning and neural networks, and showed that such solution approach can obtain near-optimal solutions. Kool et al. (2019) proposed a model based on attention layers with pointer network and used reinforcement learning to solve the traveling salesman, vehicle routing and orienteering problems.

However, according to our literature review, little work has been done on using machine learning methods to solve the shortest path interdiction problem. The only two exceptions that we found are the following work: Xu et al. (2017) proposed a MDP-based goal recognition approach, and applied it to a dynamic shortest path local network interdiction problem, in which the follower may change its destination node stochastically or even deceptively, and the confrontation between the leader and follower is assumed as a multi-stage process. In a follow-up work, Zeng et al. (2018) further proposed an inverse reinforcement learning-based opponent behavior modeling method in the above goal recognition approach, and applied it to the same problem.

In this work, we aim to integrate reinforcement learning into the shortest path interdiction problem, and hope to initiate more research on the development of machine learning methods for different network interdiction problems. It should be noticed that our approach is quite

different from the above two work in the following aspects: First, the problem considered in this work has a fixed origin and destination nodes on different network topologies, while the above two work focused on a dynamic shortest path interdiction problem on a fixed graph network (i.e., the Chicago Sketch Road Network). Second, they used a human behavior modeling method based on inverse reinforcement learning (IRL), which is suitable to solve the problems where the reward function is hard to calculate, and the experts knowledge is usually needed for agent to learn the reward function during the training process. However, in our work, the reward function can be accurately quantified as the length of the shorted path from the origin node to the destination node after interdiction. Therefore, we prefer to use the reinforcement learning based approach, which requires less input and is more applicable to handle some other more generalized network interdiction problems.

The remainder of this paper is organized as follows. In Section 2, we introduce and formulate the shortest path interdiction problem. In Section 3, we present the framework to train the pointer network and solve the problem with reinforcement learning. Section 4 provides the design of computational experiments and tests the performance of our solution approach. Finally, we conclude our work and provide possible future research directions in Section 5.

2. Problem statement and its formulation

In this work, we consider the following shortest path interdiction problem proposed by Israeli and Wood (2002): Given a directed network $G = (N, A)$, where N is the set of nodes, and A is the set of arcs. Let $c_k \in \mathbb{R}_+$ be the nominal length for each arc $k \in A$, and $d_k \in \mathbb{R}_+$ be the amount by which arc k is lengthened when it is interdicted. In addition, the amount of resource that is required to interdict arc k is denoted as $r_k \in \mathbb{R}_+$, and the total available resource for the leader is denoted as $r_0 \in \mathbb{R}_+$. The objective of the leader is to interdict arcs in the given network to maximize the shortest path length between two specified nodes, i.e., the origin node O and the destination node D, subject to limited interdiction resources. For the purpose of consistency, we use the same name as the one used in Israeli and Wood (2002), and call the above problem the "maximizing the shortest path" problem, or the MXSP problem for short.

Then, by defining the following variables:

- x_k is equal to 1 if arc k is interdicted by the leader, and 0 otherwise,
- y_k is equal to 1 if arc k is traversed by the follower and 0 otherwise,

the MXSP problem can be formulated as the following bilevel mixed integer program:

$$
\begin{aligned}
[\text{MXSP-B}] \quad & \max_{x} \min_{y} \sum_{k \in A} (c_k + x_k \cdot d_k) \cdot y_k && (1.1)\\
\text{s.t.} \quad & \sum_{k \in FS(i)} y_k - \sum_{k \in RS(i)} y_k = 1 \quad \text{for } i = O; && (1.2)\\
& \sum_{k \in FS(i)} y_k - \sum_{k \in RS(i)} y_k = 0 \quad \text{for } i \in N \backslash \{O, D\}; && (1.3)\\
& \sum_{k \in FS(i)} y_k - \sum_{k \in RS(i)} y_k = -1 \quad \text{for } i = D; && (1.4)\\
& \sum_{k \in A} x_k \cdot r_k \leq r_0 \quad \text{for } k \in A; && (1.5)\\
& x_k, y_k \in \{0, 1\} \quad \text{for } k \in A; && (1.6)
\end{aligned}
$$

where $FS(i)$ and $RS(i)$ are the arcs directed out of and into node i, respectively.

As mentioned by Israeli and Wood (2002), the above bilevel optimization model can be transformed to an equivalent single-level optimization model by first taking the dual of the inner minimization in [MXSP-B] and then release variable x. The resulting single-level mixed integer programming model can be written as:

$$
\begin{aligned}
[\text{MXSP-S}] \quad & \max_{\pi, x} \ \pi_D - \pi_O && (2.1)\\
& \pi_j - \pi_i - d_k \cdot x_k \leq c_k \quad \text{for } k = (i, j) \in A; && (2.2)\\
& \sum_{k \in A} x_k \cdot r_k \leq r_0 \quad \text{for } k \in A; && (2.3)\\
& \pi_O = 0; && (2.4)\\
& x_k \in \{0, 1\} \quad \text{for } k \in A; && (2.5)
\end{aligned}
$$

which can be readily solved via a standard optimization solver (e.g. Gurobi or CPLEX).

3. Reinforcement learning model for the MXSP problem

In what follows, we provide the detailed information about how to integrate the reinforcement learning model to solve the MXSP problem. In particular, in this work, we focus on two different network topologies: the directed grid networks and the Erdős-Rényi random graphs. In the directed grid networks, there are $m \times n$ transshipment nodes arranged in a grid of m

rows and n columns. For any of the arcs from O to each transshipment node in the first column or from each transshipment node in the last column to D, we impose that such arc cannot be interdicted and its arc length is equal to 0, while for all the other arcs in the network, we assume that they are interdictable with strictly positive length. Then, it is not difficult to see that the total number of interdictable arcs in a grid network with $m \times n$ transshipment nodes is $(n-2)(5m-4)+3m-2$. Figure 2 shows an example of a 8×12 grid network, in which the dotted arcs cannot be interdicted, and the total number of interdictable arcs (i.e., the solid arcs) is 382. In addition, the grey arcs show the corresponding shortest path when the leader has a budget that can interdict up to nine arcs and the slashed arcs are interdicted.

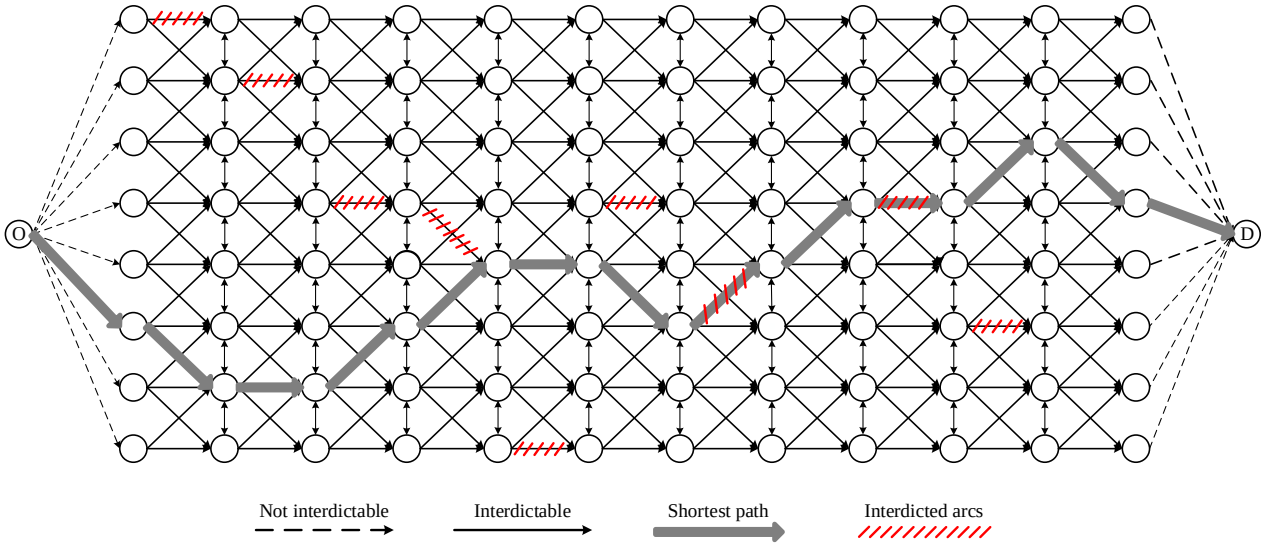
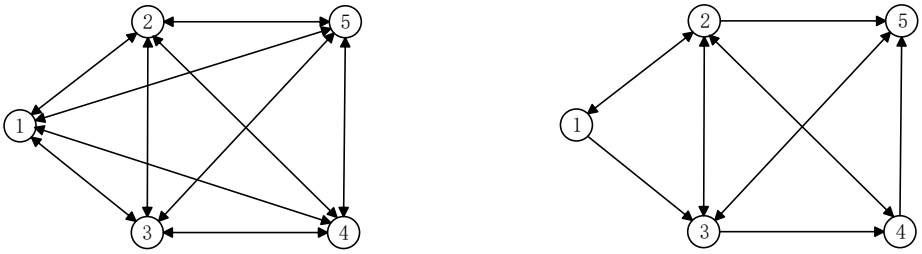


Figure 2: The topology of a 8×12 grid network.

On the other hand, in an Erdős-Rényi random graph, the number n of vertices is given, and each arc between any two vertices is independently connected with probability $\tilde{p}$, that is, the graph is constructed according to one commonly used variant of the Erdős-Rényi model, which is called the $G(n,\tilde{p})$ model (Erdős and Rényi, 1959; Gilbert, 1959). To be specific, given a set of nodes $N=\{1,2,\ldots,n\}$, and a set of arcs $A=\{(i,j): i,j \in N, i \neq j\}$, which forms a directed complete graph (see Figure 3(a) with $n=5$), we choose arcs according to Bernoulli's trials with a fixed probability $\tilde{p} \in [0,1]$, that is, an arc (i,j) is picked with probability $\tilde{p}$ independently of occurrences of other arcs. In particular, from the properties of $G(n,\tilde{p})$, we know that when the number of vertices n tends to infinity, then almost every graph in $G(n, 2\ln(n)/n)$ is connected. In what follows, we always set the probability that each arc is included in the graph to $2\ln(n)/n$, and assume that all the arcs in the generated random graph are interdictable. To illustrate,

Figure 3(b) shows an Erdős-Rényi random graph with 12 arcs that is generated according to the $G(5, 2\ln(5)/5)$ model.



(a) Directed complete graph (b) Erdős-Rényi random graph

Figure 3: An illustration of a directed complete graph and an Erdős-Rényi random graph.

3.1. Applicability of reinforcement learning

As we mentioned in the introduction, the state-of-the-art algorithms that have been proposed for solving different variants of the shortest path interdiction problems are usually problem-tailored, and may not work well for all possible instances. In addition, as shown by Khachiyan et al. (2008), we know that not only the MXSP problem itself is NP-hard, but also the problem of approximating within a factor smaller than 2 for the MXSP problem is still NP-hard. Therefore, when the size of the MXSP problem increases, the computational time for calculating the optimal solutions could become prohibitively time-consuming. As a result, it is natural to investigate if the machine learning methods could be used to solve the MXSP problem. When such solution approach is applicable to the MXSP problem, then we may apply it across many problem instances by exploring and learning the rules from the training data, and fewer manual revision may be required comparing to the heuristics.

As we know, the supervised learning is an important type of the machine learning methods, where the input training data and the corresponding optimal output results are required to be known a *priori*. However, due to the inherent complexity of the MXSP problem, it is difficult to solve the input training data to obtain their optimal labels. Therefore, the supervised learning method may be not suitable for solving the MXSP problem.

Unlike supervised learning methods, reinforcement learning is a machine learning method that does not require optimal labels during the training phase. As introduced by Sutton and Barto (2018), the learner and the thing it interacts with are called *agent* and *environment* respectively. With reinforcement learning methods, the agents learn by constantly interacting with the environment through trial and error to choose the best action to maximize the

expected goals. As a result, such approach can provide a general framework for achieving optimal decisions in dynamic environments, and thus is suitable for solving several combinatorial optimization problems, including some of the network optimization problems. For further pointers to work on reinforcement learning for combinatorial optimization problems, we refer the reader to recent surveys by Mazyavkina et al. (2021) and Wang and Tang (2021).

In particular, the sequence-to-sequence based reinforcement learning has recently been proved to be effective to solve numerous combinatorial optimization problems, and has attracted significant attention from researchers. Bello et al. (2017) introduced a framework of neural combinatorial optimization, which uses reinforcement learning and neural networks to solve traveling salesman problem and achieves close to optimal results on 2D Euclidean graphs with up to 100 nodes. Nazari et al. (2018) proposed a model that consists of a recurrent neural network (RNN) decoder coupled with an attention mechanism, and used a policy gradient algorithm to optimize parameters. It was shown that their trained model can solve VRP with split deliveries and a stochastic variant efficiently. Kool et al. (2019) proposed a model with multi-head attention layers and used REINFORCE with a simple baseline based on a deterministic greedy rollout to train the model. With this model, close to optimal results could be achieved for several classical combinatorial optimization problems, including the TSP, VRP, orienteering problem and the prize collecting TSP. Zhang et al. (2020) proposed a reinforcement learning algorithm called the multi-agent attention model to solve multi-vehicle routing problems with soft time windows, in which an encode-decoder framework with attention layers was proposed to generate tours of multiple vehicles iteratively. Gama and Fernandes (2021) explored the use of pointer network models to solve the orienteering problem with time windows (OPTW), in which the pointer network architecture was customized to better address problems with dynamic time-dependent constraints, and reinforcement learning was used to train the models. In this work, we also choose the sequence-to-sequence based reinforcement learning to solve the MXSP problem.

3.2. The digram of reinforcement learning

During each time step t in reinforcement learning, the agent takes an action A_t based on the policy p mapping states to actions after observing the current environment state S_t. Then the environment responds a new state and reward to the agent. Neural Network (NN) is often used to fit the policy p in reinforcement learning, and the policy is updated during the training step. The objective of the reinforcement learning is to learn a decision policy from the training data that maximizes the given goal.

In the MXSP problem, the input are the arc attributes of the network, and the output are the interdicted arcs that maximizes the shortest path from the origin node O to the destination node D. The policy is represented by a pointer network, a sequence-to-sequence neural network (see Section 3.3 for more details), and we use reinforcement learning algorithm to train the network. The agent learns the rule of the input data (i.e. the network) through trial and error, and selects the appropriate interdictable arcs according to the policy. Therefore, the state can be seen as the current network, the action of the agent can be seen as to choose the arcs to be interdicted, and the reward provided by the environment is the length of the shortest path after interdiction.

To be specific, during each time step t in reinforcement learning, we denote its corresponding state as $S_t = (V, IA \backslash SA, SA)$, where V is the set of vertices, IA is the set of interdictable arcs, and SA is the set of selected arcs to be interdicted. Meanwhile, the action A_t is used to select an arc to be interdicted. Obviously, the maximum size of the eligible action space is $|IA|$. Given a state S_t and an action A_t, the reward can be described as $R(S_t, A_t) = C(S_t)$, where $C(S_t)$ is the length of the shortest path between the original and destination nodes obtained from the state S_t after the action A_t has been taken. In particular, the initial state S_0 is the original network, and the final state S_{final} is a complete solution of the MXSP problem. For example, when we implement the reinforcement learning model for the grid network as shown in Figure 2, the state is the above vector that represents the dynamic status of the grid network, including the set of arcs that have been interdicted, and the corresponding remaining network. The action is to select an arc to be interdicted (i.e., one of the slashed arcs), and the reward is the total length of the corresponding shortest path after interdiction (i.e., the total length of grey arcs).

3.3. Introduction of the pointer network

Similar to many other combinatorial optimization problems, we know that the input in the MXSP problem is a sequence, the output is also a sequence, and these two sequences may not be equal in length. As we know, the sequence-to-sequence neural network (SSNN) proposed by Sutskever et al. (2014) is designed to tackle such sequence-related issues. In a SSNN, a neural network, usually the recurrent neural network (RNN), is used to handle the input sequence (also called encoder), and the output sequence is also handled by another RNN (called decoder). However, the SSNN is hard to solve the problem with unfixed output dictionary, especially with the situations that the output depends on the input. For example, the length of the output may vary from time to time in the orienteering problem. Therefore, the SSNN may be not the best choice for many combinatorial optimization problems including

the MXSP problem.

Recently, the pointer network (PN), a variant of the SSNN introduced by Vinyals et al. (2015), is used to fill this gap, especially for the problems with a discrete and input-dependent output. In particular, due to the good generalization of the PN-based reinforcement learning (PNRL) method, such model can be used to solve new problems (with the same problem structure) and new samples without re-train once the model has been well trained. Meanwhile, when the problem size increases, the PNRL model can still provide relative good results compared to other classical heuristics, and it also has an advantage in terms of computational running time. Therefore, in this paper, we use the pointer network to solve the MXSP problem.

To illustrate, in the MXSP problem with a 8×12 grid network, we know that the number of input arcs is 382 and each arc can be represented by a 3-dimensional vector, which respectively represents the departure node, arrival node, and the corresponding nominal length of the arc. Before we load the input to the encoder of the pointer network, the data of each arc is embedded from 3-dimensional to a ℓ-dimensional vector. In this work, we follow the parameter setting given by Bello et al. (2017) to construct the pointer network, in which both the encoder and decoder use RNN with Long Short-Term Memory (LSTM) cells (Hochreiter and Schmidhuber, 1997). In this work, we let the LSTM contain 128 hidden units, and set $\ell = 128$. Moreover, the adaptive moment estimation (Adam) optimizer is used to train the model (Kingma and Ba, 2015), in which we set the learning rate to 10^{-3} with a decay of 0.96 per epoch.

As shown in Figure 4, the encoder reads the input data step by step, one arc at a time. The input data of each step is a ℓ-dimensional embedded vector. Similarly, the decoder of PN outputs the data step by step, and a pointer mechanism is used to select the best output which points to one of the input. Then, the output of the current step will be taken as the input to the next decoder step, until reaching the final decoder step. Note that the input of the first decoder step is a trainable vector.

3.4. The reinforcement learning model for the MXSP problem

In this section, we provide the details of the reinforcement learning model with pointer network to solve the MXSP problem.

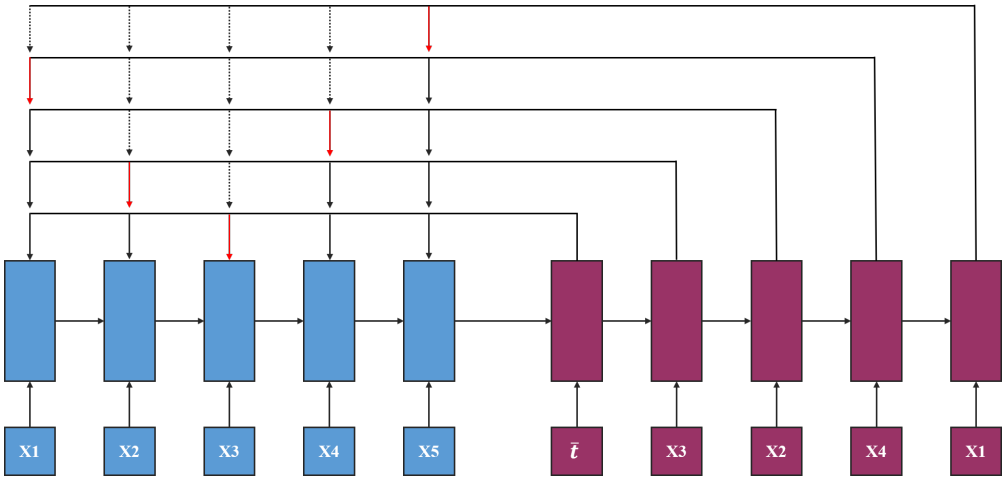


Figure 4: The architecture of the pointer network.

3.4.1. Data input

After we input the information of interdictable arcs to the reinforcement learning model, the interdicted arcs are the output. As we mentioned above, each input arc vector contains three elements, i.e., the departure node, arrival node and its nominal length. The departure and arrival information of each arc are extracted from the generated network and its nominal length is randomly generated. For example, in the grid network as shown in Figure 2, the original input data is a 382×3 vector. Then, after the embedding process, a 382×128 vector is inputted to the encoder.

3.4.2. Cost function

During the training process, the decoder of the pointer network outputs the information of the interdicted arcs π for the encoder input data s. In this work, when an arc k is selected to be interdicted, we assume that the length will be increased by a same amount of $d_k = d$. Then, we reconstruct the remaining network after interdiction, and use Dijkstra's Algorithm to calculate the length of the shortest path from the origin node O to the destination node D in the reconstructed network. In addition, we define the cost function $L(\pi|s)$ as follows:

$$L(\pi|s) = \sum_{k \in SP} c_k,$$

where SP is the set of arcs in the corresponding shortest path. For example, in the grid network as shown in Figure 2, $L(\pi|s) = R(S_{\text{final}}, A_{\text{final}})$ is the total length of the grey arcs.

3.4.3. Policy-based reinforcement learning method

Here, we define the policy of the pointer network as $p(\pi|s)$, and the goal of the policy is to assign high probability to the solution that the corresponding shortest path has large total length and low probability to the solution that with small total length. The policy can be provided by neural network using chain rule. As explained before, the reinforcement learning method is more suitable for the MXSP problem than other supervised learning methods. Therefore, a policy-based reinforcement learning method is used in this paper to optimize the parameters of the pointer network, denoted as θ. In addition, we use the technique of gradient descent to minimize the loss, which can be viewed as the opposite of our goal that maximizes the length of the shortest path of the follower. Therefore, we let $\tilde{L}(\pi|s) = -L(\pi|s)$, and define the loss as the expectation of function $\tilde{L}(\pi|s)$, which can be formulated as: $g(\theta|s) = \mathbb{E}_{p_\theta(\pi|s)}\tilde{L}(\pi|s)$. Then, the loss is optimized by the technique of gradient descent, using REINFORCE Algorithm (Williams, 1992) with baseline $b(s)$:

$$\nabla_\theta g(\theta|s) = \mathbb{E}_{p_\theta(\pi|s)}[(\tilde{L}(\pi|s) - b(s))\nabla_\theta \log p_\theta(\pi|s)] \tag{3}$$

By using Monte Carlo sampling, the gradient in (3) can be reformulated as follows:

$$\nabla_\theta g(\theta) = \frac{1}{B}\sum_{i=1}^{B}(\tilde{L}(\pi_i|s_i) - b(s_i))\nabla_\theta \log p_\theta(\pi_i|s_i), \tag{4}$$

where B is the batch size.

After the policy gradient is defined, an optimizer, such as the Adam optimizer (see Section 3.4.4 for more details), is used to update the neural network.

3.4.4. Baselines and the training procedures

In this work, three different baselines are considered during our training procedures. The most common way may be the exponential moving average method with decay (i.e., the Exponential model), and its corresponding training procedure is presented in Algorithm 3.1. For each epoch, there are T steps (T times B equals to the size of the training data). In each step, we first sample the input data (i.e., the embedded data). For example, in a 8×12 grid network, s_i is a 382×128 vector, and the solution π_i is the output. Then, the baseline b_i is calculated by exponential moving average method. In addition, as we mentioned above, we use REINFORCE Algorithm to calculate the loss function, and use the Adam optimizer to train the model.

Algorithm 3.1 The reinforcement learning training procedure with `Exponential` baseline

Given: number of epochs E, number of steps per epoch T, batch size B, decay factor β
Initialize pointer network parameters θ
for $epoch = 1, ..., E$ **do**
  for $step = 1, ..., T$ **do**
    $s_i \leftarrow SampleInput()$ for $i \in \{1, ..., B\}$
    $\pi_i \leftarrow SampleSolution(s_i, p_\theta)$ for $i \in \{1, ..., B\}$
    if $step == 1$ **then**
      $b_i = \tilde{L}(\pi_i | s_i)$
    else
      $b_i \leftarrow \beta b(i) + (1-\beta)\tilde{L}(\pi_i | s_i)$
    $g_\theta \leftarrow \frac{1}{B} \sum_{i=1}^{B} (\tilde{L}(\pi_i | s_i) - b_i) \nabla_\theta \log p_\theta(\pi_i | s_i)$
    $\theta \leftarrow Adam(\pi, g_\theta)$

The second choice is to use a learnable function (e.g. the Critic model) to estimate the baseline (Bello et al., 2017). Algorithm 3.2 gives the reinforcement learning procedure with the Critic baseline. Similar to the exponential baseline, an expected cost b_i is predicted by the Critic network $b_{\theta_v}(s)$, and the goal is to minimize the mean squared error between b_i and the actual sample cost by the current pointer network policy. Similarly, the Adam optimizer is used to train the Critic network.

Algorithm 3.2 The reinforcement learning training procedure with `Critic` baseline

Given: number of epochs E, number of steps per epoch T, batch size B
Initialize pointer network parameters θ, and Critic network parameters θ_v
for $epoch = 1, ..., E$ **do**
  for $step = 1, ..., T$ **do**
    $s_i \leftarrow SampleInput()$ for $i \in \{1, ..., B\}$
    $\pi_i \leftarrow SampleSolution(s_i, p_\theta)$ for $i \in \{1, ..., B\}$
    $b_i \leftarrow b_{\theta_v}(s_i)$ for $i \in \{1, ..., B\}$
    $g_\theta \leftarrow \frac{1}{B} \sum_{i=1}^{B} (\tilde{L}(\pi_i | s_i) - b_i) \nabla_\theta \log p_\theta(\pi_i | s_i)$
    $\mathcal{L}_v \leftarrow \frac{1}{B} \sum_{i=1}^{B} \left\| b_i - \tilde{L}(\pi_i) \right\|_2^2$
    $\theta \leftarrow Adam(\pi, g_\theta)$
    $\theta_v \leftarrow Adam(\theta_v, \nabla_{\theta_v} \mathcal{L}_v)$

The last baseline is the rollout policy introduced by Kool et al. (2019). The rollout policy keeps freezing for a number of steps (at least one epoch) until the current point network policy

is significantly better than the rollout policy. Then the rollout policy is updated by the latest pointer network policy. As Algorithm 3.3 shows, the parameters in a new rollout model θ^{BL} are initialized to exactly the same as the ones used in the pointer network. The solution π_R^{BL} is found by the rollout model on the rollout dataset R. In the training step, the rollout model validates the training dataset s and the solution is used as a baseline to calculate the loss function. When all the training steps in one epoch are finished, the update checking process begins. A solution π_R is provided by the current pointer network policy p_θ on the rollout dataset R. If π_R improves π_R^{BL} significantly by passing the one side t-test, then the rollout policy θ^{BL}, rollout dataset R and rollout solution π_R^{BL} are updated.

Algorithm 3.3 The reinforcement learning training procedure with `Rollout` baseline

Given: number of epochs E, number of steps per epoch T, batch size B, test significance α
Initialize pointer network parameters θ, and rollout model parameters $\theta^{BL} \leftarrow \theta$
Randomly generate the rollout dataset R
$\pi_R^{BL} \leftarrow RolloutSolution(R, p_{\theta^{BL}})$
for $epoch = 1, ..., E$ **do**
  for $step = 1, ..., T$ **do**
    $s_i \leftarrow SampleInput()$ for $i \in \{1, ..., B\}$
    $\pi_i \leftarrow SampleSolution(s_i, p_\theta)$ for $i \in \{1, ..., B\}$
    $\pi_i^{BL} \leftarrow RolloutSolution(s_i, p_{\theta^{BL}})$ for $i \in \{1, ..., B\}$
    $g_\theta \leftarrow \frac{1}{B} \sum_{i=1}^{B} (\tilde{L}(\pi_i|s_i) - \tilde{L}(\pi_i^{BL}|s_i)) \nabla_\theta \log p_\theta(\pi_i|s_i)$
    $\theta \leftarrow Adam(\pi, g_\theta)$
  $\pi_R \leftarrow RolloutSolution(R, p_\theta)$
  if $OneSideTTest(\pi_R, \pi_R^{BL}) < \alpha$ **then**
    $\theta^{BL} \leftarrow \theta$
    Regenerate the rollout dataset R
    $\pi_R^{BL} \leftarrow RolloutSolution(R, p_{\theta^{BL}})$

4. Experimental study

To evaluate the performance of our proposed reinforcement learning model with pointer network on solving the MXSP problem, we conducted several experiments on a set of instances that are generated from two different network topologies: the directed grid networks and the Erdős-Rényi random graphs. To be specific, we first considered five small-sized instances that are generated from the following grid networks: 5×5, 12×8, 8×12, 10×10, and 15×15, whose numbers of interdictable arcs in each instance are 76, 370, 382, 396, and 966, respectively.

Then, we considered three large-sized instances that are generated from the following grid networks: 20×20, 25×25, and 30×30, whose numbers of interdictable arcs in each instance are 1786, 2856, and 4176, respectively.

Finally, we considered four Erdős-Rényi random graphs with different number of vertices, i.e., $n = 20, 50, 80$, and 100, in which we set the first and last vertices as the origin and destination nodes, respectively. In order to make the generated instances solvable, we restricted that there must exist at least one path from the origin node to the destination node in the generated random graph. Meanwhile, as we know, even for the 5×5 grid network, the minimum number of arcs with strictly positive length in any shortest path is at least 4, therefore, we also restricted that the number of arcs in the random graph that traverse the shortest path between the origin and destination nodes should be at least 4. Otherwise, we repeated the generation process until the above restrictions are satisfied. This way, we obtained four different random graph instances, whose numbers of interdictable arcs in each instance are 103, 374, 651, and 866, respectively. For simplicity, for any of the instances that is generated from grid network $n \times m$, we call the corresponding instance as $Gn \times m$, and call the random graph instance with n vertices as Rn.

In addition, for each arc $k \in A$ of the above instances, we randomly generated its nominal length c_k from the uniform distribution on $\{1, 2, \ldots, 10\}$ when it is interdictable, and set it to 0 when it cannot be interdicted. In addition, once an arc is interdicted, we assumed that the amount of resources it consumes is $r_k = 5$, the amount of length it increases is $d_k = 10$, and the total amount of interdiction resources is $r_0 = 50$. Meanwhile, we set both the validation and rollout sizes to 10000. To restrict our training time within a reasonable range, we set the training size of $G5 \times 5$ to 1,280,000, and set the training sizes of all the other instances to 512,000, respectively.

4.1. Hyperparameters

As we know, the selection of hyperparameters could dramatically affect the performance of our reinforcement learning model. Through some examination, we initialized the parameters of our reinforcement learning model uniformly within [-0.08, 0.08], clipped the $L2$ norm of gradients to 1.0, and set the learning rate to 10^{-3} with a decay of 0.96 per epoch. In addition, the decay factor β used in the Exponential baseline is set to 0.8, the test significance α used in the rollout baseline is set to 0.05, and the training and validation batch sizes are set to 512 and 2048, respectively.

4.2. Computational environment

As we mentioned in Section 2, the MXSP can be formulated as a single-level mixed integer program, i.e., the [MXSP-S] model. In this work, we solved this model by using Gurobi Optimizer 9.0.2 to obtain the exact optimal solutions, which is also an upper bound of our reinforcement learning method. The time limit for running Gurobi Optimizer is set to 7200 seconds. To compare our reinforcement learning model and the MIP model with similar computational environments, we ran the [MXSP-S] model on a server with a 48 virtual CPU system (2 × Intel(R) Xeon(R) Silver 4214), 128GB of RAM and Ubuntu 18.04 operating system, and implemented our reinforcement learning model using PyTorch (Ketkar, 2017) and executed it through the GPU (NVDIA GeForce RTX 2080Ti) on the same server.

4.3. Comparison of the performance of the RL models on $G5 \times 5$ *with different baselines*

As we mentioned above, three baselines, i.e., the Exponential, Critic and Rollout baselines, are used in our reinforcement model. For simplicity, we denote our reinforcement learning models with the above baselines as RL-Exponential, RL-Critic, and RL-Rollout, respectively. To evaluate their performance, we kept these three models all the same except that the corresponding baselines that used to train the pointer network are different, and trained these models for 100 epochs to solve the instance of $G5 \times 5$.

Figure 5 shows the corresponding computational results, in which the x-axis denotes the number of epochs (can be viewed as the training period) and the y-axis denotes the gap, which is defined as the difference between the length of the shortest path after interdiction obtained by our reinforcement models and the one obtained by solving the [MXSP-S] model. From this figure, we can see that the performance of the RL-Rollout model is significantly better than the other two models, as it can always find better results during the training process. In addition, the gap of the RL-Rollout model can converge to a value of around 12%, while the ones of the other two models are around 18%. The reason for the outperformance of the RL-Rollout model may be as follows: As we mentioned before, the baseline policy in the RL-Rollout model keeps freezing for a number of steps until the current neural network policy is significantly better than the baseline policy, and then the rollout baseline policy is updated by the latest neural network policy. This way, the rollout policy guarantees that the baseline could keep relatively stable and is updated only when needed during the training process, and therefore is more suitable for training the reinforcement learning model. In what follows, we used the rollout baseline as the default baseline and implemented the RL-Rollout model to solve all the other instances.

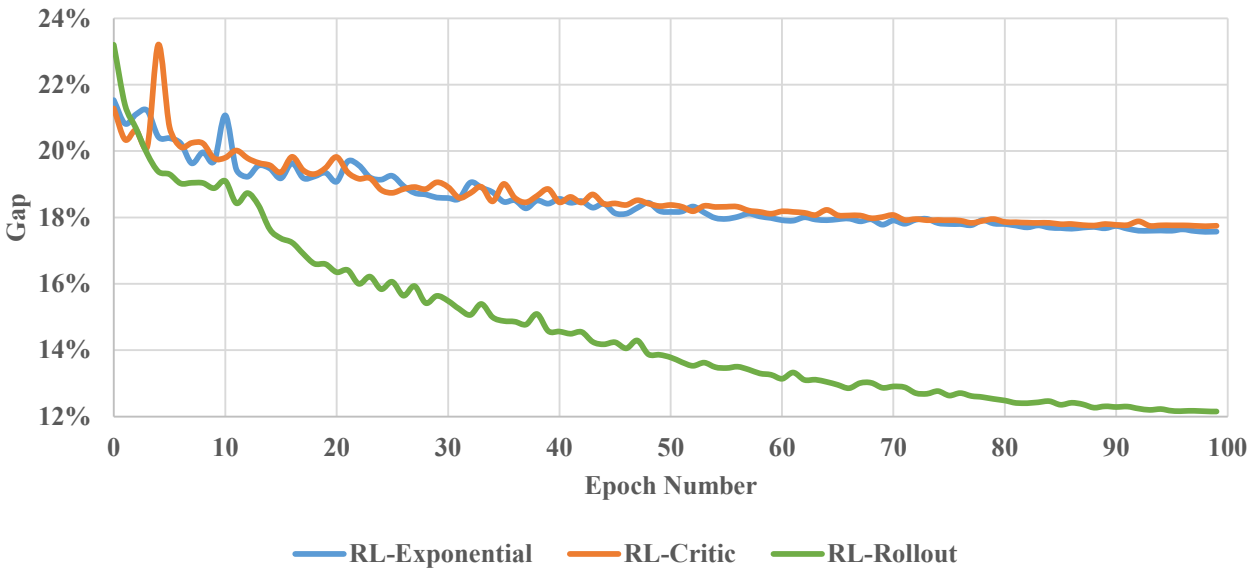


Figure 5: Comparison of the performance of the RL models on $G5 \times 5$ with different baselines in 100 epochs.

4.4. Continuous optimization of the RL-Rollout model on $G5 \times 5$

In this section, we discuss the continuous optimization of the RL-Rollout model. To be specific, we loaded the RL-Rollout model after 100 training epochs in previous section as an initial model for further training, and remained all the other training parameters the same. As shown in Figure 6, the gap of this model can be further decreased to 9.4% after implementing additional 100 training epochs, that is, the performance of the RL-Rollout model can be improved by around 3% compared to our previous result. Such result shows the continuous trainability of the pointer network, and we can always try to increase the training epochs so as to obtain better results for more complex instances.

4.5. Comparison of the performance of the RL-Rollout model and other machine learning methods on $G5 \times 5$

To further test the applicability of other machine learning methods for solving the MXSP problem, we selected two commonly used methods, i.e., the support vector machine (SVM) and the artificial neural network (ANN) methods, as competitors. Unlike the reinforcement learning model, these two methods are supervised model in which the optimal values of the input training data are needed during the training process. Therefore, for all of the 1,280,000 training samples on $G5 \times 5$, we solved them and calculated the labelled training data for these two supervised methods by using Gurobi Optimizer.

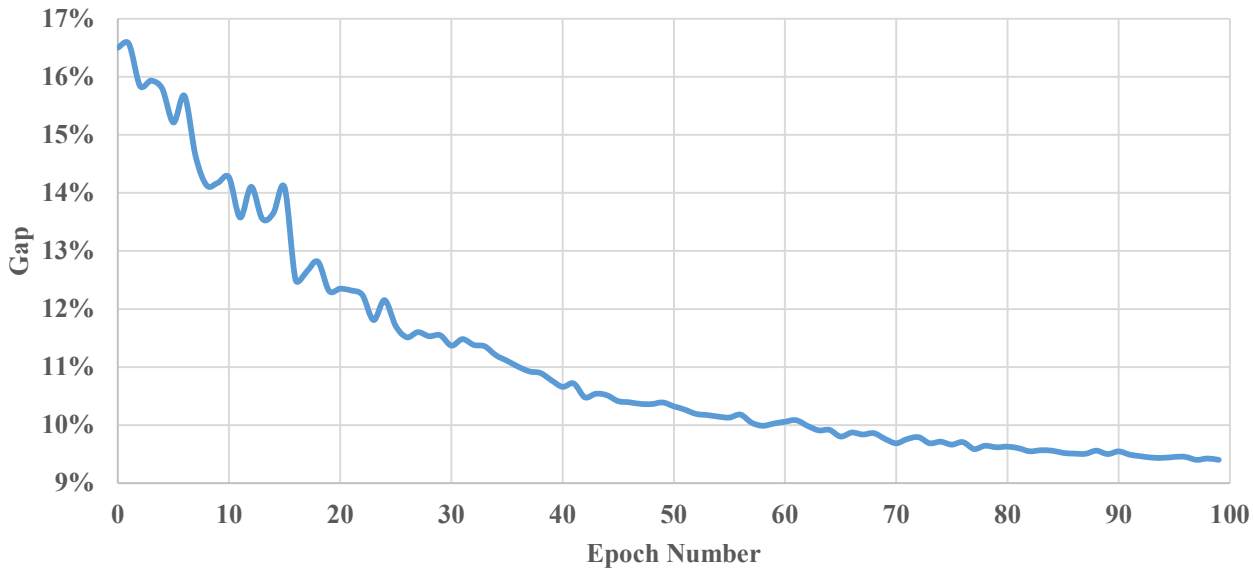


Figure 6: Computational result of the pre-trained RL-Rollout model on $G5 \times 5$.

In the SVM model, each input sample is a $76 \times 3 = 228$ dimensional vector and all the input training samples can be viewed as a $1,280,000 \times 228$ dimensional vector while the output is the length of the shortest path after interdiction, which can be viewed as a 1,280,000 dimensional vector. The SVM model is implemented by calling the SVR function in Scikit-Learn tools (Pedregosa et al., 2011), among which the key parameters are the kernel function, the penalty factor and the tolerance factor. Inspired by the work of Lin et al. (2007), we set radial basis function (RBF) as the kernel function, set penalty factor to 20, and set the tolerance factor for stopping criterion to 0.01.

On the other hand, the ANN model is implemented by Keras tool (Chollet et al., 2015). Adapted from the work of Kitapcı et al. (2014), the ANN model used in this study is constructed as a network with five layers, i.e., the input layer, three hidden layers and the output layer, and the dimensions of the input, hidden, and output layers are 228, 10, and 1, respectively. In addition, the mean squared error (MSE) is set as the loss function, the rectified linear unit (ReLU) activation function is preferred, and the Adam optimizer is used for training. The training epoch number and batch size of the ANN model are set the same as the ones used in the reinforcement learning model, that is, the training epoch number is set to 100 and the batch size is set to 512. The training will be stopped early if the network's performance fails to improve.

Table 2 shows the performance of the SVM, ANN and RL-Rollout models on $G5 \times 5$, in which

the first column presents the disturbance of the labelled data. For example, -5% (5%) means that the labelled values of the training data are reduced (increased) by 5%. From this table, we can see that the SVM and ANN models seem to outperform the RL-Rollout model when the disturbance is less than 10%. However, when the disturbance becomes larger than 10%, these two models are much worse than the RL-Rollout model, which verifies the effectiveness of our proposed reinforcement learning model. As we mentioned before, the MXSP problem is NP-hard, therefore it is usually difficult to obtain the optimal labels of the input training data, which is an essential requirement for supervised machine learning methods. Such features could substantially affect the applicability and accuracy of the supervised methods as the deviation of the labelled values that we obtained is usually greater than 10% or even 50% if the labelled values of the training data are generated randomly. In addition, it seems that the SVM model outperforms the ANN model in terms of gap values.

Table 2: Comparison between RL-Rollout model and other machine learning methods on $G5 \times 5$.

Disturbance	SVM model		ANN model		RL-Rollout Model	
	Gap	Time(s)	Gap	Time(s)	Gap	Time(s)
0	3.41%	0.004	7.22%	0.0003	9.40%	0.003
-1%	3.54%	0.003	7.79%	0.0003		
-5%	4.74%	0.003	8.83%	0.0004		
-10%	9.09%	0.004	7.94%	0.0003		
-20%	19.16%	0.004	16.69%	0.0002		
-30%	29.27%	0.003	26.90%	0.0003		
-40%	39.37%	0.003	37.34%	0.0002		
-50%	49.48%	0.003	47.78%	0.0003		
1%	3.85%	0.004	8.57%	0.0003		
5%	6.48%	0.003	11.10%	0.0004		
10%	11.19%	0.003	15.38%	0.0003		
20%	21.25%	0.003	25.34%	0.0003		
30%	31.36%	0.003	35.76%	0.0003		
40%	41.46%	0.003	46.20%	0.0004		
50%	51.57%	0.004	56.65%	0.0003		

Gap: The average gap between the computational values and the optimal results.

4.6. Comparison of the performance of the RL-Rollout model on small-sized grid network instances

To further test the performance of the RL-Rollout model, we implemented it on the other four small-sized grid network instances, i.e., $G12 \times 8$, $G8 \times 12$, $G10 \times 10$ and $G15 \times 15$. Figure 7 shows the corresponding computational results. From this figure, we can see that this model

can achieve acceptable results for all of these four instances, since the gap for each instance can converge to a value within [14%,17%], and the training processes for these instances are relatively stable.

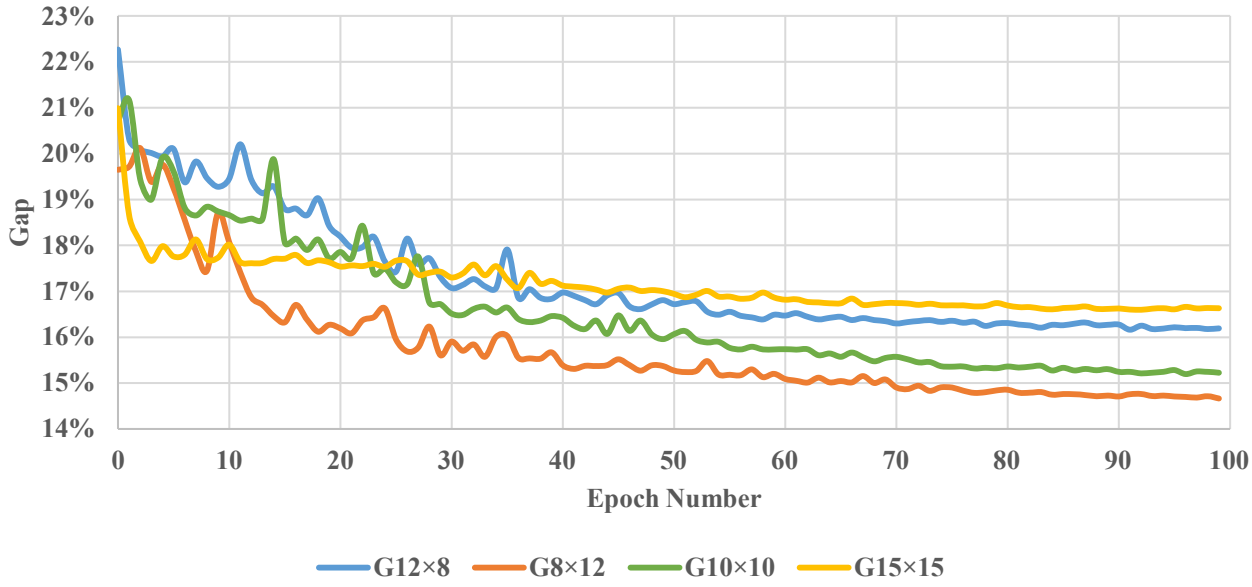


Figure 7: Comparison of the performance of the RL-Rollout model on different instances in 100 epochs.

On the other hand, we also noticed that the learning speeds and the values of gap for these instances after 100 epochs are not as good as the one for $G5 \times 5$. The reason for such phenomenon may be as follows: First, the training size used for $G5 \times 5$ (i.e., 1,280,000) is larger than the ones used for these four instances (i.e., 512,000). As a result, the used training dataset may not fully meet the learning needs of the reinforcement learning models to find better results for these four instances. Second, the structures of the corresponding directed grid networks are more complex for these four instances than the one used in $G5 \times 5$. As a result, the reinforcement learning model may need more time and resource to learn the knowledge behind the data of these instances.

Table 3 shows the corresponding computational results of both the reinforcement learning models based on an average of 10,000 validation samples, and the MIP model solved by Gurobi Optimizer. From this table, we can see that the average gap of the RL-Rollout model is around 14% for all the instances, while the ones of the other two models (i.e., RL-Critic and RL-Exponential) are around 17.5% even only for the instance of $G5 \times 5$. Meanwhile, the computational time of solving the [MXSP-S] model by Gurobi Optimizer is much greater than the ones of the reinforcement learning models, especially when the size of instances increases,

which indicates that our reinforcement learning method could be an effective way to solve the shortest path interdiction problem, since we can always expand the training scale of our reinforcement learning model to achieve better results, which can be addressed offline in the learning phase, and the run-time phase (i.e., the inference phase) can be extremely quick. Note that among the validation dataset of $G15 \times 15$, there are 60 instances which cannot be solved optimally with 7200s time limit by Gurobi Optimizer, and the average gap for these instances is 2.90%.

Table 3: The computational results of the RL models on small-sized grid network instances.

Instance	RL Model				[MXSP-S] Model	
	Baseline	LSP	Gap	Time (s)	OBJ	Time (s)
$G5 \times 5$	Rollout	14.88	9.40%	0.003	16.42	0.12
	Critic	13.51	17.74%	0.003		
	Exponential	13.54	17.57%	0.003		
$G12 \times 8$	Rollout	16.49	15.66%	0.014	19.55	2.50
$G8 \times 12$	Rollout	27.55	14.16%	0.012	32.10	66.09
$G10 \times 10$	Rollout	21.75	14.70%	0.013	25.50	10.53
$G15 \times 15$	Rollout	29.79	16.10%	0.054	35.51	517.03
Average Time	/	/	/	0.015	/	119.25

LSP: The length of the shortest path obtained by the RL models.
OBJ: The objective value obtained by solving the [MXSP-S] model.
Gap: The average gap between LSP and OBJ.

Meanwhile, the $G12 \times 8$ and $G8 \times 12$ instances have similar network structures and input sizes, as the former instance has a 12×8 grid network while the latter one has a 8×12 grid network. From the computational results of these two instances, we can see that the instance of $G8 \times 12$ is more difficult to solve than the instance of $G12 \times 8$ by Gurobi Optimizer, while our proposed reinforcement learning model is much less sensitive on the performance of solving these two instances, which indicates that the reinforcement learning method could have quite good scalability and applicability across different instances with similar structures and sizes.

4.7. The generalization analysis of the RL-Rollout model on small-sized grid network instances

Generalization is another important feature for the reinforcement learning model. In this section, we investigated its generalization for different small-sized grid network instances. To be specific, we first determined the associated trained reinforcement learning model for each instance. For simplicity, we call the RL-Rollout model that is trained based on the instances of $G5 \times 5$ as RL-1 model. Similarly, we call the other RL-Rollout models that are trained based

on the instances of $G12 \times 8$, $G8 \times 12$, $G10 \times 10$, and $G15 \times 15$ as RL-2, RL-3, RL-4, and RL-5 models, respectively. Then, we solved the validation samples of the other instances based on each of these models. Table 4 shows the corresponding computational results. From this table, we can see that the instances that are solved by those trained reinforcement learning models with similar network structure could provide relative good results. For example, for the instances of $G12 \times 8$, when we solved them with the RL-2, RL-3 and RL-4 models, which has similar network structures and sizes, similar values of gap that are around 16% can be obtained. It is also interesting to see that when we used the RL-4 model to solve the validation samples of $G15 \times 15$, it could achieve an average gap that is even better than the one solved by the RL-5 model, which illustrates that our proposed reinforcement learning model could have quite good generalization.

Table 4: The generalization analysis of the RL-Rollout model on small-sized grid network instances.

Instance	RL-1 model		RL-2 model		RL-3 model		RL-4 model		RL-5 model	
	LSP	Gap	LSP	Gap	LSP	Gap	LSP	Gap	LSP	Gap
$G5 \times 5$	**14.88**	**9.40%**	12.3	25.09%	11.35	30.88%	11.96	27.16%	10.3	37.27%
$G12 \times 8$	14.06	28.08%	**16.49**	**15.66%**	16.36	16.32%	16.41	16.06%	14.71	24.76%
$G8 \times 12$	23.91	25.51%	27.01	15.86%	**27.55**	**14.16%**	27.37	14.74%	25.33	21.09%
$G10 \times 10$	18.81	26.24%	21.62	15.22%	21.73	14.78%	**21.75**	**14.70%**	19.81	22.31%
$G15 \times 15$	28.02	21.09%	30.01	15.49%	29.7	16.36%	**30.04**	**15.40%**	29.79	16.10%

LSP: The length of the shortest path obtained by the RL models.
Gap: The average gap between the computational values and the optimal results.
Bolded value: The best average gap for the current instances.

4.8. The performance of the RL-Rollout model on large-sized grid network instances

To further investigate the performance of our proposed reinforcement learning model, we also tested the RL-Rollout model on some large-sized grid network instances. To be specific, we considered three different instances that are generated from the following grid networks: $G20 \times 20$, $G25 \times 25$, and $G30 \times 30$. As we know, when the size of the instances increases, it becomes much more difficult to calculate the optimal solution of the MXSP problem by using Gurobi Optimizer. As a result, many samples cannot be solved optimally within the given time limit. Therefore, we only generated 100 samples to validation for the above three instances. In addition, we used the trained RL-Rollout model based on the instances of $G15 \times 15$, i.e., the RL-5 model, to solve the validation samples of these instances. From Table 5, we can see that the RL-5 model can still provide quite good results, as the average gap for these instances is 15.06%, and the average running time is only 0.34 second, which is significantly smaller than the ones obtained by solving the MXSP problem directly with Gurobi Optimizer. Moreover,

the average percentage of unsolved instances (i.e., PUS) and the average gap for those unsolved instances (i.e., AVG) are 60% and 2.07%, respectively, which indicates that solving the MXSP problem directly by Gurobi Optimizer is not an efficient approach for these large-sized instances, while our proposed reinforcement learning model could still be a quite effective tool to solve them.

Table 5: The computational results of the RL-5 model on large-sized grid network instances.

Instance	RL-5 Model			MXSP Model			
	LSP	Gap	Time (s)	OBJ	PUS	AVG	Time (s)
$G20 \times 20$	38.87	14.72%	0.16	45.58	25%	2.24%	2961.63
$G25 \times 25$	47.32	15.20%	0.31	55.80	68%	1.93%	5563.73
$G30 \times 30$	55.47	15.25%	0.55	65.45	87%	2.03%	6611.62
Average Value	/	15.06%	0.34	/	60%	2.07%	5045.66

LSP: The length of the shortest path obtained by the RL-5 model.
OBJ: The objective value obtained by solving the [MXSP-S] model directly with Gurobi Optimizer.
Gap: The average gap between LSP and OBJ.
PUS: The percentage of unsolved instances by Gurobi Optimizer within 7200s.
AVG: The average gap provided by Gurobi Optimizer for the unsolved instances.

4.9. The performance of the RL-Rollout model on Erdős-Rényi random graphs

Although we have verified that our proposed reinforcement learning model has quite good performance on solving the MXSP problem with grid network topology, we are still unclear whether such reinforcement learning model is also effective for other instances of the MXSP problem with more general network structures. In this section, we further investigated the performance of our proposed RL-Rollout model on Erdős-Rényi random graph instances that we have generated, i.e., the instances of $R20, R50, R80$, and $R100$.

Tabel 6 shows the corresponding computational results, from which we can see that the RL-Rollout model can still solve the random graph instances quite effectively with an average gap of 3.17%. In addition, the smallest gap is 2.39% obtained by solving the instance of $R20$, and the gap increases when the size of instances increases. In fact, by comparing the computational results from Tables 3 and 6, we can see that the RL-Rollout model even performs better on solving the random graph instances than the grid network instances. The reason for such phenomenon may be as follows: On one hand, the grid networks have two different types of arcs, i.e., the interdictable and not interdictable arcs, while all the arcs in the random graphs are interdictable, which may make the RL-Rollout model more difficult to learn the structure of the grid network. On the other hand, in an $m \times n$ grid network, the number of

transshipment nodes between the origin and destination nodes is at least n, as a result, the number of arcs in the obtained shortest path is at least $n+1$, which is generally larger than the one for random graphs, and thus the performance of the RL-Rollout model for the grid network is comparatively worse than the one for random graphs. Moreover, the average running time for the random graphs is also quite small, which shows the efficiency of the RL-Rollout model on both grid networks and random graphs.

Table 6: The computational results of the RL-Rollout model on random graphs.

Instance	RL-Rollout Model			MXSP Model	
	LSP	Gap	Time (s)	OBJ	Time (s)
$R20$	39.59	2.39%	0.005	40.56	0.04
$R50$	31.64	3.16%	0.011	32.64	0.12
$R80$	31.55	3.40%	0.024	32.73	0.26
$R100$	31.91	3.71%	0.039	33.14	0.44
Average Value	/	3.17%	0.020	/	0.22

LSP: The length of the shortest path obtained by the RL-Rollout models.
OBJ: The objective value obtained by solving the [MXSP-S] model with Gurobi Optimizer.
Gap: The average gap between LSP and OBJ.

5. Conclusions

In this work, we proposed a framework to solve the shortest path interdiction problem (i.e., the MXSP problem) with reinforcement learning, in which the pointer network was introduced to handle the situation of variable output sizes for the MXSP problem. In particular, we implemented our reinforcement learning model with three different baselines to train the pointer network, and the results show that the rollout baseline policy could achieve the best performance. To further evaluate the performance of our proposed solution approach, we conducted several computational experiments on various instances that are generated from two different network topologies, i.e., the grid networks and the Erdős-Rényi random graphs. The computational results illustrate the effectiveness of our proposed reinforcement learning methods for randomly generated instances of both the grid network (up to 30×30 transshipment nodes) and the random graphs (up to 100 vertices). To the best of our knowledge, we are one of the first to use RL-based pointer network model to solve the MXSP problem. With those prominent and fruitful results by pioneering studies on various combinatorial optimization problems including the TSP (Bello et al., 2017), VRP (e.g. Nazari et al., 2018; Zhang et al., 2020), and

orienteering problems (e.g. Kool et al., 2019; Gama and Fernandes, 2021), the reinforcement learning has shown its great power in solving various complex time-dependent operations research problems, and we hope that our work could be a starting point to attract further studies on solving more generalized network interdiction problems with powerful machine learning methods.

There are many possible directions for future work stemming from this work: (1) More complex problem settings can be considered, such as variable values of d_k and r_k in the MXSP problem and a knapsack constraint on the total interdiction resources. (2) Calculating the cost function in more efficient ways. In this work, we used Dijkstra's algorithm to calculate the cost for each solution. However, such algorithm is not efficient enough to deal with the batch solutions. According to our experimental observation, the time for calculating the cost functions is the main bottleneck affecting the training speed. Therefore, more efficient algorithms may be needed to calculate the cost function in the future work. (3) More generalized network interdiction problems can be investigated to see if any machine learning method could be used to solve such problems.

Acknowledgements

This research was supported by the National Natural Science Foundation Council of China under Projects 71701144, 71872125 and 91646118, the Ministry of Science and Technology of China under Project 2020IM030300, and the Fundamental Research Funds for the Central Universities (2021XSC-0106, 2021XSC-0125). This work was also supported by the Science & Technology Pillar Key Program of Tianjin Key Research and Development Plan (20YFZCGX00640).

Data Availability Statement

The data that support the findings of this study are available from the corresponding author, [KF], upon reasonable request.

References

Achamrah, F. E., Riane, F., and Limbourg, S. (2021). Solving inventory routing with transshipment and substitution under dynamic and stochastic demands using genetic algorithm and deep reinforcement learning. *International Journal of Production Research*. Forthcoming.

Bai, R., Chen, X., Chen, Z.-L., Cui, T., Gong, S., He, W., Jiang, X., Jin, H., Jin, J., Kendall, G., Li, J., Lu, Z., Ren, J., Weng, P., Xue, N., and Zhang, H. (2021). Analytics and machine learning in vehicle routing research. *arXiv preprint arXiv:2102.10012*.

Ball, M. O., Golden, B. L., and Vohra, R. V. (1989). Finding the most vital arcs in a network. *Operations Research Letters*, 8(2):73–76.

Baycik, N. O. and Sullivan, K. M. (2019). Robust location of hidden interdictions on a shortest path network. *IISE Transactions*, 51(12):1332–1347.

Bayrak, H. and Bailey, M. D. (2008). Shortest path network interdiction with asymmetric information. *Networks*, 52(3):133–140.

Bello, I., Pham, H., Le, Q. V., Norouzi, M., and Bengio, S. (2017). Neural combinatorial optimization with reinforcement learning. In *5th International Conference on Learning Representations (ICLR2017), Toulon, France, April 24-26, 2017, Workshop Track Proceedings*.

Bengio, Y., Lodi, A., and Prouvost, A. (2021). Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421.

Bier, V. M., Gratz, E. R., Haphuriwat, N. J., Magua, W., and Wierzbicki, K. R. (2007). Methodology for identifying near-optimal interdiction strategies for a power transmission system. *Reliability Engineering & System Safety*, 92(9):1155–1161.

Borrero, J. S., Prokopyev, O. A., and Sauré, D. (2016). Sequential shortest path interdiction with incomplete information. *Decision Analysis*, 13(1):68–98.

Cappanera, P. and Scaparra, M. P. (2011). Optimal allocation of protective resources in shortest-path networks. *Transportation Science*, 45(1):64–80.

Chen, B., Bai, R., Li, J., Liu, Y., Xue, N., and Ren, J. (2021). A multiobjective single bus corridor scheduling using machine learning-based predictive models. *International Journal of Production Research*. Forthcoming.

Chollet, F. et al. (2015). Keras. `https://keras.io`.

Duan, L., Zhan, Y., Hu, H., Gong, Y., Wei, J., Zhang, X., and Xu, Y. (2020). Efficiently solving the practical vehicle routing problem: A novel joint learning approach. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3054–3063.

Erdős, P. and Rényi, A. (1959). On random graphs I. *Publicationes Mathematicae*, 6:290–291.

Fulkerson, D. R. and Harding, G. C. (1977). Maximizing the minimum source-sink path subject to a budget constraint. *Mathematical Programming*, 13(1):116–118.

Furian, N., O'Sullivan, M., Walker, C., and Çela, E. (2021). A machine learning-based branch and price algorithm for a sampled vehicle routing problem. *OR Spectrum*, 43:693–732.

Gama, R. and Fernandes, H. L. (2021). A reinforcement learning approach to the orienteering problem with time windows. *Computers & Operations Research*, 133:105357.

Gilbert, E. N. (1959). Random graphs. *The Annals of Mathematical Statistics*, 30(4):1141–1144.

Golden, B. (1978). A problem in network interdiction. *Naval Research Logistics Quarterly*, 25(4):711–713.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Holzmann, T. and Smith, J. C. (2019). Shortest path interdiction problem with arc improvement recourse: A multiobjective approach. *Naval Research Logistics*, 66(3):230–252.

Holzmann, T. and Smith, J. C. (2021). The shortest path interdiction problem with randomized interdiction strategies: Complexity and algorithms. *Operations Research*, 69(1):82–99.

Israeli, E. and Wood, R. K. (2002). Shortest-path network interdiction. *Networks*, 40(2):97–111.

Jiang, J. and Liu, X. (2021). Bayesian stackelberg game model for water supply networks against interdictions with mixed strategies. *International Journal of Production Research*, 59(8):2537–2557.

Karabulut, E., Aras, N., and Altınel, İ. K. (2017). Optimal sensor deployment to increase the security of the maximal breach path in border surveillance. *European Journal of Operational Research*, 259(1):19–36.

Ketkar, N. (2017). Introduction to PyTorch. In *Deep Learning with Python*, pages 195–208. Springer.

Khachiyan, L., Boros, E., Borys, K., Elbassioni, K., Gurvich, V., Rudolf, G., and Zhao, J. (2008). On short paths interdiction problems: Total and node-wise limited interdiction. *Theory of Computing Systems*, 43(2):204–233.

Kheirkhah, A., Navidi, H., and Messi Bidgoli, M. (2016). A bi-level network interdiction model for solving the hazmat routing problem. *International Journal of Production Research*, 54(2):459–471.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations (ICLR2015), San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Kitapcı, O., Özekicioğlu, H., Kaynar, O., and Taştan, S. (2014). The effect of economic policies applied in Turkey to the sale of automobiles: Multiple regression and neural network analysis. *Procedia-Social and Behavioral Sciences*, 148:653–661.

Konrad, R. A., Trapp, A. C., Palmbach, T. M., and Blom, J. S. (2017). Overcoming human trafficking via operations research and analytics: Opportunities for methods, models, and applications. *European Journal of Operational Research*, 259(2):733–745.

Kool, W., van Hoof, H., and Welling, M. (2019). Attention, learn to solve routing problems! In *7th International Conference on Learning Representations (ICLR2019), New Orleans, LA, USA, May 6-9, 2019*.

Lee, J.-H. and Kim, H.-J. (2021). Reinforcement learning for robotic flow shop scheduling with processing time variations. *International Journal of Production Research*. Forthcoming.

Lin, K., Lin, Q., Zhou, C., and Yao, J. (2007). Time series prediction based on linear regression and SVR. In *3rd International Conference on Natural Computation (ICNC 2007)*, volume 1, pages 688–691.

Lozano, L. and Smith, J. C. (2017). A backward sampling framework for interdiction problems with fortification. *INFORMS Journal on Computing*, 29(1):123–139.

Lu, H., Zhang, X., and Yang, S. (2020). A learning-based iterative method for solving vehicle routing problems. In *8th International Conference on Learning Representations (ICLR2020)*.

Mazyavkina, N., Sviridov, S., Ivanov, S., and Burnaev, E. (2021). Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134:105400.

Morton, D. P., Pan, F., and Saeger, K. J. (2007). Models for nuclear smuggling interdiction. *IIE Transactions*, 39(1):3–14.

Nazari, M., Oroojlooy, A., Takáč, M., and Snyder, L. V. (2018). Reinforcement learning for solving the vehicle routing problem. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS2018)*, pages 9861–9871.

Pan, F. and Morton, D. P. (2008). Minimizing a stochastic maximum-reliability path. *Networks*, 52(3):111–119.

Pay, B. S., Merrick, J. R., and Song, Y. (2019). Stochastic network interdiction with incomplete preference. *Networks*, 73(1):3–22.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Prince, M., Geunes, J., and Smith, J. C. (2013). Procurement allocation planning with multiple suppliers under competition. *International Journal of Production Research*, 51(23-24):6900–6922.

Rocco, C. M. and Ramirez-Marquez, J. E. (2010). A bi-objective approach for shortest-path network interdiction. *Computers & Industrial Engineering*, 59(2):232–240.

Sadeghi, S., Seifi, A., and Azizi, E. (2017). Trilevel shortest path network interdiction with partial fortification. *Computers & Industrial Engineering*, 106:400–411.

Sefair, J. A. and Smith, J. C. (2016). Dynamic shortest-path interdiction. *Networks*, 68(4):315–330.

Smith, J. C. and Song, Y. (2020). A survey of network interdiction models and algorithms. *European Journal of Operational Research*, 283(3):797–811.

Song, Y. and Shen, S. (2016). Risk-averse shortest path interdiction. *INFORMS Journal on Computing*, 28(3):527–539.

Sullivan, K. M., Morton, D. P., Pan, F., and Cole Smith, J. (2014). Securing a border under asymmetric information. *Naval Research Logistics*, 61(2):91–100.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014 (NIPS2014), December 8-13 2014, Montreal, Quebec, Canada*, pages 3104–3112.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Vinyals, O., Fortunato, M., and Jaitly, N. (2015). Pointer networks. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015 (NIPS2015), December 7-12, 2015, Montreal, Quebec, Canada*, pages 2692–2700.

Wang, Q. and Tang, C. (2021). Deep reinforcement learning for transportation network combinatorial optimization: A survey. *Knowledge-Based Systems*, 233:107526.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

Xu, K., Xiao, K., Yin, Q., Zha, Y., and Zhu, C. (2017). Bridging the gap between observation and decision making: goal recognition and flexible resource allocation in dynamic network interdiction. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI2017)*, pages 4477–4483.

Yang, J., Borrero, J. S., Prokopyev, O. A., and Sauré, D. (2021). Sequential shortest path interdiction with incomplete information and limited feedback. *Decision Analysis*, 18(3):218–244.

Yates, J. and Lakshmanan, K. (2011). A constrained binary knapsack approximation for shortest path network interdiction. *Computers & Industrial Engineering*, 61(4):981–992.

Zeng, Y., Xu, K., Yin, Q., Qin, L., Zha, Y., and Yeoh, W. (2018). Inverse reinforcement learning based human behavior modeling for goal recognition in dynamic local network interdiction. In *Workshops at the 32nd AAAI Conference on Artificial Intelligence (AAAI2018)*, pages 646–653.

Zhang, K., He, F., Zhang, Z., Lin, X., and Li, M. (2020). Multi-vehicle routing problems with soft time windows: A multi-agent reinforcement learning approach. *Transportation Research Part C: Emerging Technologies*, 121:102861.

Zhang, Q., Guan, X., and Pardalos, P. M. (2021a). Maximum shortest path interdiction problem by upgrading edges on trees under weighted l_1 norm. *Journal of Global Optimization*, 79(4):959–987.

Zhang, Q., Guan, X., Wang, H., and Pardalos, P. M. (2021b). Maximum shortest path interdiction problem by upgrading edges on trees under hamming distance. *Optimization Letters*, 15:2661–2680.