MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 29, NO. 3, 2022
journal homepage: www.mais-journal.ru

THEORY OF COMPUTING

Towards Neural Routing with Verified Bounds on Performance

I. P. Buzhinsky[2], A. A. Shalyto[1]

DOI: 10.18255/1818-1015-2022-3-228-245

[1]ITMO University, 49 Kronverksky pr., Saint Petersburg 197101, Russia.
[2]Aalto University, 8 Maarintie, Espoo 02150, Finland.

MSC2020: 68T07
Research article
Full text in English

Received June 16, 2022
After revision August 25, 2022
Accepted August 26, 2022

When data-driven algorithms, especially the ones based on deep neural networks (DNNs), replace classical ones, their superior performance often comes with difficulty in their analysis. On the way to compensate for this drawback, formal verification techniques, which can provide reliable guarantees on program behavior, were developed for DNNs. These techniques, however, usually consider DNNs alone, excluding real-world environments in which they operate, and the applicability of techniques that do account for such environments is often limited. In this work, we consider the problem of formally verifying a neural controller for the routing problem in a conveyor network. Unlike in known problem statements, our DNNs are executed in a distributed context, and the performance of the routing algorithm, which we measure as the mean delivery time, depends on multiple executions of these DNNs. Under several assumptions, we reduce the problem to a number of DNN output reachability problems, which can be solved with existing tools. Our experiments indicate that sound-and-complete formal verification in such cases is feasible, although it is notably slower than the gradient-based search of adversarial examples.

The paper is structured as follows. Section 1 introduces basic concepts. Then, Section 2 introduces the routing problem and DQN-Routing, the DNN-based algorithm that solves it. Section 3 proposes the contribution of this paper: a novel sound and complete approach to formally check an upper bound on the mean delivery time of DNN-based routing. This approach is experimentally evaluated in Section 4. The paper is concluded with some discussion of the results and outline of possible future work.

Keywords: formal verification; trustworthy AI; deep neural networks; routing problem

INFORMATION ABOUT THE AUTHORS

Igor Petrovich Buzhinsky correspondence author	orcid.org/0000-0003-3713-6051. E-mail: igor.buzhinsky@gmail.com Postdoctoral reseacher, Doctor of Science (Technology).
Anatoly Abramovich Shalyto	orcid.org/0000-0002-2723-2077. E-mail: anatoly.shalyto@gmail.com Professor, Doctor of Technical Sciences, Professor.

Funding: The work was financially supported by the Russian Science Foundation (Project 20-19-00700).

For citation: I. P. Buzhinsky and A. A. Shalyto, “Towards Neural Routing with Verified Bounds on Performance”, *Modeling and analysis of information systems*, vol. 29, no. 3, pp. 228-245, 2022.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 29, № 3, 2022
сайт журнала: www.mais-journal.ru

THEORY OF COMPUTING

На пути к нейросетевой маршрутизации с верифицированными границами эффективности

И. П. Бужинский[2], А. А. Шалыто[1]

DOI: 10.18255/1818-1015-2022-3-228-245

[1]Университет ИТМО, Кронверкский пр., д. 49, г. Санкт-Петербург, 197101 Россия.
[2]Университет Аалто, ул. Мааринтие, д. 8, г. Эспоо, 02150 Финляндия.

УДК 004.8
Научная статья
Полный текст на английском языке

Получена 16 июня 2022 г.
После доработки 25 августа 2022 г.
Принята к публикации 26 августа 2022 г.

Когда алгоритмы на основе данных, особенно основанные на глубоких нейронных сетях (ГНС), заменяют классические, их более высокая производительность часто сопряжена с трудностями при анализе. Чтобы компенсировать этот недостаток, для ГНС были разработаны методы формальной верификации, которые могут предоставить надежные гарантии поведения программы. Эти методы, однако, обычно рассматривают только саму ГНС, исключая среду, в которой она работает, и применимость методов, учитывающих такие среды, часто ограничена. В данной работе рассматривается задача формальной верификации нейросетевого контроллера для задачи маршрутизации в конвейерной сети. В отличие от известных постановок задачи, рассматриваемые ГНС выполняются в распределенной среде, и производительность алгоритма маршрутизации, которая измеряется как среднее время доставки, зависит от многократного выполнения этих ГНС. При некоторых предположениях, проблема верификации сводится к ряду проблем достижимости выходов ГНС, которые можно решить с помощью существующих программных средств. Эксперименты показывают, что в таких случаях возможна строгая и полная формальная верификация, хотя она заметно медленнее, чем градиентный поиск состязательных примеров.
Статья построена следующим образом. Раздел 1 вводит основные понятия. Затем в Разделе 2 представлена проблема маршрутизации и алгоритм DQN-маршрутизации на основе ГНС, который ее решает. В Разделе 3 описывается вклад данной статьи: новый надежный и полный подход к формальной проверке верхней границы среднего времени доставки маршрутизации на основе ГНС. Этот подход экспериментально оценивается в Разделе 4. Статья завершается обсуждением результатов и описанием возможной будущей работы.

Ключевые слова: формальная верификация; надежный ИИ; глубокие нейронные сети; задача маршрутизации

ИНФОРМАЦИЯ ОБ АВТОРАХ

Игорь Петрович Бужинский автор для корреспонденции	orcid.org/0000-0003-3713-6051. E-mail: igor.buzhinsky@gmail.com постдок, кандидат технических наук.
Анатолий Абрамович Шалыто	orcid.org/0000-0002-2723-2077. E-mail: anatoly.shalyto@gmail.com профессор, доктор технических наук, профессор.

Финансирование: Работа выполнена при финансовой поддержке Российского Научного Фонда (Проект 20-19-00700).

Для цитирования: I. P. Buzhinsky and A. A. Shalyto, “Towards Neural Routing with Verified Bounds on Performance”, *Modeling and analysis of information systems*, vol. 29, no. 3, pp. 228-245, 2022.

Introduction

The recent success of deep neural networks (DNNs) has motivated researchers and practitioners to apply them in many fields. When DNNs are required to interact with the real world, they are usually trained with reinforcement learning (RL), based on the feedback from the environment. In particular, RL has been applied to the problem of routing [1, 2], with the work [1] utilizing DNNs.

Unlike traditional controllers that are programmed explicitly [3], data-driven models such as DNNs are hard to analyze. Especially when the decisions of a DNN-based controller influence the real world, this leads to difficulties in establishing safety and reliability guarantees of such controllers. These concerns are exacerbated by the possibility of adversarial attacks [4–6] both in the digital and the physical worlds. The problem of acquiring guarantees on program behavior is usually referred to as formal verification [7] and is more computationally expensive compared to testing and simulation. Formal verification approaches were developed not only for traditional programs, but also for DNNs [8–13]. These approaches usually consider DNNs in isolation, i.e. the connection of their inputs and outputs to the rest of the world is not modeled.

In this work, we consider the classical routing problem being solved by a distributed family of DNN-based controllers, focusing on the baggage handling problem [3, 14, 15]. These controllers can be trained as a single DNN for all nodes of the network or as node-specific DNNs. For such a family of controllers, we are interested in proving an upper bound on the mean delivery time from the chosen source node to the chosen destination node. As a concrete routing algorithm, we selected DQN-routing [1] based on deep RL.

The proposed verification approach involves adjusting the routing probabilities generated by DNN agents, representing the conveyor network as a discrete-time Markov chain, and reducing the verification problem to a number of problems of verifying a reachability property for a multilayer perceptron in isolation, which is done using existing software [13]. We assume that the routing DNNs do not change during operation. While this assumption reduces the applicability of the approach in the context of RL, it also enables verification of DNN controllers obtained by means other than RL.

1. Preliminaries

1.1. Deep neural networks

A *deep neural network* (DNN) is a parameterized function trained to predict some outcome from input data. More formally, we will assume that a DNN $\mathcal{N}$ predicts an outcome $y \in \mathbb{R}^{d_{\text{out}}}$ from a vector $x \in \mathbb{R}^{d_{\text{in}}}$, and it is an MLP:

$$\mathcal{N} = \mathcal{A}_1 \circ f \circ \mathcal{A}_2 \circ f \circ \cdots \circ \mathcal{A}_k, \tag{1}$$

where k is the number of layers in the network (here, we do not count the input layer), $\mathcal{A}_1, \ldots, \mathcal{A}_k$ are learnable affine operators $\mathcal{A}_i(\mathbf{x}) = \mathbf{A}_i\mathbf{x} + \mathbf{b}_i$, possibly with different input and output dimensions, and f is a continuous scalar nonlinear *activation function* that is applied to vectors element-wise. A popular choice of f, both in learning and verifying DNNs, is the rectified linear unit (ReLU): $f(x) = \max(0, x)$. In DQN-routing, $k = 3$ and f is ReLU.

We use the following notation. For matrices $\mathbf{A}_1, \ldots, \mathbf{A}_k$, the block diagonal matrix constructed from them is denoted $\text{diag}(\mathbf{A}_1, \ldots, \mathbf{A}_k)$. For column vectors $\mathbf{b}_1, \ldots, \mathbf{b}_k$, the column vector produced as their concatenation is $\text{concat}(\mathbf{b}_1, \ldots, \mathbf{b}_k)$. Block diagonal matrices and vector concatenations produced from m identical matrices $\mathbf{A}$ or vectors $\mathbf{b}$ are denoted $\text{diag}(\mathbf{A}, m)$ and $\text{concat}(\mathbf{b}, m)$.

1.2. Adversarial examples for DNNs

DNNs with large d_{in} may exhibit intuitively unwanted and unexpected properties: a slight modification of the input vector often makes the DNN produce a prediction error [6]. This is especially visible on the problem of image classification, where an eye-imperceptible change of an image can result in a classification error. Many approaches, such as [16–19], perform gradient-based optimization of such *adversarial examples* and measure *adversarial robustness*, the robustness of DNNs to such examples. For discrete output predictions,

a possible choice is to define adversarial robustness as the norm (often ℓ_2 or ℓ_∞) of the minimum change that, when added to the original input, makes the classifier produce an incorrect outcome. One popular method for finding adversarial examples is *projected gradient descent* (PGD) [20], in which the classification error is maximized w.r.t. the input vector of the DNN. The input vector is restricted to a ball centered at a given point: if the vector escapes this ball on some step of gradient descent, it is projected back onto this ball.

However, evaluation of adversarial robustness metrics based on gradient-based optimization provides only upper bounds on these metrics' values. To accurately determine whether adversarial examples exist in a given input space region, algorithms based on other principles must be used.

1.3. Formal verification of neural networks

Certified defenses [21–24] and formal verification techniques [8–13] focus on reliable guarantees on adversarial robustness. Typically, DNN verification approaches check a property on the output of the DNN, such as reachability of a certain output space region, when the input vector is constrained to a specified input space region. In particular, by checking that one component of the output vector cannot exceed the others when the inputs belong to a ball centered at a real data sample, it is possible to check a bound on the minimum norm of the adversarial perturbation for this sample.

Most commonly, DNN verification targets the case of the DNN being executed only once, on fixed inputs, and only the input-output dependency is modeled. However, these assumptions are unacceptable when the DNN implements a controller, e.g. trained with deep RL [25, 26]: the environment (physical or simulated) where the DNN is executed becomes important. First, the property to verify in this case is usually formulated in terms of the environment (e.g., check reachability of a certain unsafe environment state region), so its state needs to be modeled in addition to the DNN execution. Second, the dynamics of the environment may influence future DNN inputs. Finally, in deep RL, the DNN itself may change based on the feedback obtained from the environment.

1.4. Related work: Verification of neural network controllers

Existing approaches that verify DNN controllers [27–33] are model-based: the environment is modeled explicitly and analyzed together with the DNN. A popular choice used in [27, 28, 32] for the model of the environment is a Markov decision process (MDP), which is used to formalize RL. Reasoning about such models is possible with probabilistic [34] and hybrid model checking [29, 30, 33]. The source of distortions to which the controller must be robust could be the initial state of the environment [33], its stochastic behavior [27], possibility of an adversarial change in transition probabilities of the environment [32], variability of the system's inputs [29, 30], or possible failures in the controlled plant [27].

Our approach does not account for environment randomness: decision probabilities depend only on the DNN controller. Thus, our environment model reduces to a discrete Markov chain whose transition probabilities are decided by the DNN. This allows reducing our verification problem to a family of reachability problems for an isolated DNN.

2. Distributed routing with a neural controller

In this work, we focus on verifying a particular deep RL method [1] that learns DNNs to deliver packages in a network. Hereinafter, we limit the routing problem to delivering material objects in a conveyor network, using the airport baggage delivery problem [3, 14, 15] as an example.

2.1. Conveyor networks and graphs

We describe the delivery problem starting from *conveyor networks*. Examples of conveyor networks used in this paper are shown on the left in Fig. 1 and Fig. 2: both are synthetic examples from the literature [1, 15]. A conveyor network is composed of *conveyors* that transport *packages* (e.g., bags), it has a physical layout, which defines conveyor lengths, *checkpoints*, and positions of the checkpoints on the conveyors. There may be several types of checkpoints.

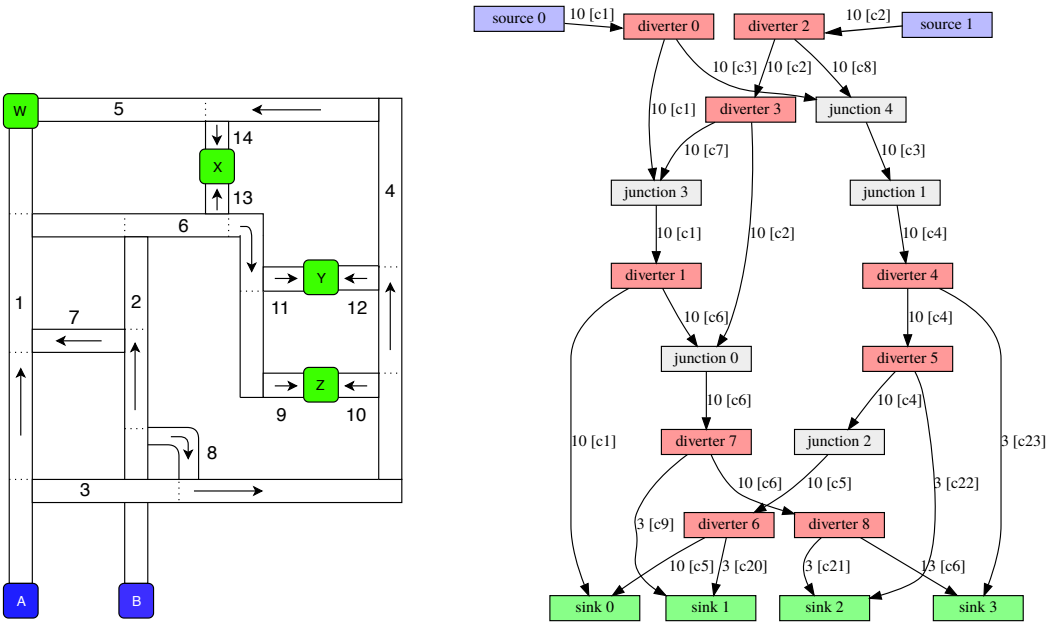

Fig. 1. Left: conveyor network from [1], with the focus on real-world conveyor layout. Sources are shown in blue and sinks are shown in green. Right: corresponding conveyor graph

A *source* is a point of package arrival. If a conveyor is linked to a source, then this source must be located at the beginning of the conveyor. A *sink* is a point of package removal from the network. If a conveyor is linked to a sink, this sink must be located at the end of the conveyor. Each package that enters the conveyor network is assumed to have a fixed destination sink.

A *junction* is a connection point of two conveyors c_1 and c_2, such that c_1 ends in the junction, and packages are redirected to a specified non-terminal location of c_2. Junctions redirecting packages to the same location of c_2 are not allowed.

A *diverter* is a manipulator that connects two conveyors c_1 and c_2: a package reaching the position of the diverter at c_1 may be redirected to the beginning of c_2. For each package, the decision whether to redirect it to c_2 is controlled.

A *conveyor graph* (V, E) is a weighted digraph with nodes corresponding to the checkpoints ($V = I \sqcup O \sqcup J \sqcup D$, where the disjoint components correspond to sources I, sinks S, junctions J, and diverters D), and arcs connecting the checkpoints that are directly linked by conveyor sections. The arcs are weighted by the lengths of corresponding conveyor sections. Due to the assumptions above, the indegree and outdegree of each node is at most two and is always determined by its type, except for junctions, which may only have an indegree of two (normal junction) or one (a new conveyor begins at the end of another one).

Fig. 1 and Fig. 2 show conveyor graphs (on the right) produced from networks shown on the left: they will be used as examples to evaluate the proposed verification approach. Each node is annotated with its type and index, with a separate numbering for the nodes of each type. The arcs are labeled with the lengths of the corresponding conveyor sections in meters and with the index of the conveyor in square brackets. The example from [1] (Fig. 1) was taken as is, the one from [15] (Fig. 2) was adapted to match our assumptions: some nodes were split into two. For example, hybrid junction/diverter nodes were split into separate junction and diverter nodes that are separated with a short (5 m) conveyor section. The example in Fig. 2 is more interesting for verification due to the presence of cycles, which may potentially lead to unbounded delivery times.

Suppose that all conveyor network components are operational and working properly. Then the *delivery problem* is, by controlling the routing decisions of diverters, to ensure that all arriving packages are

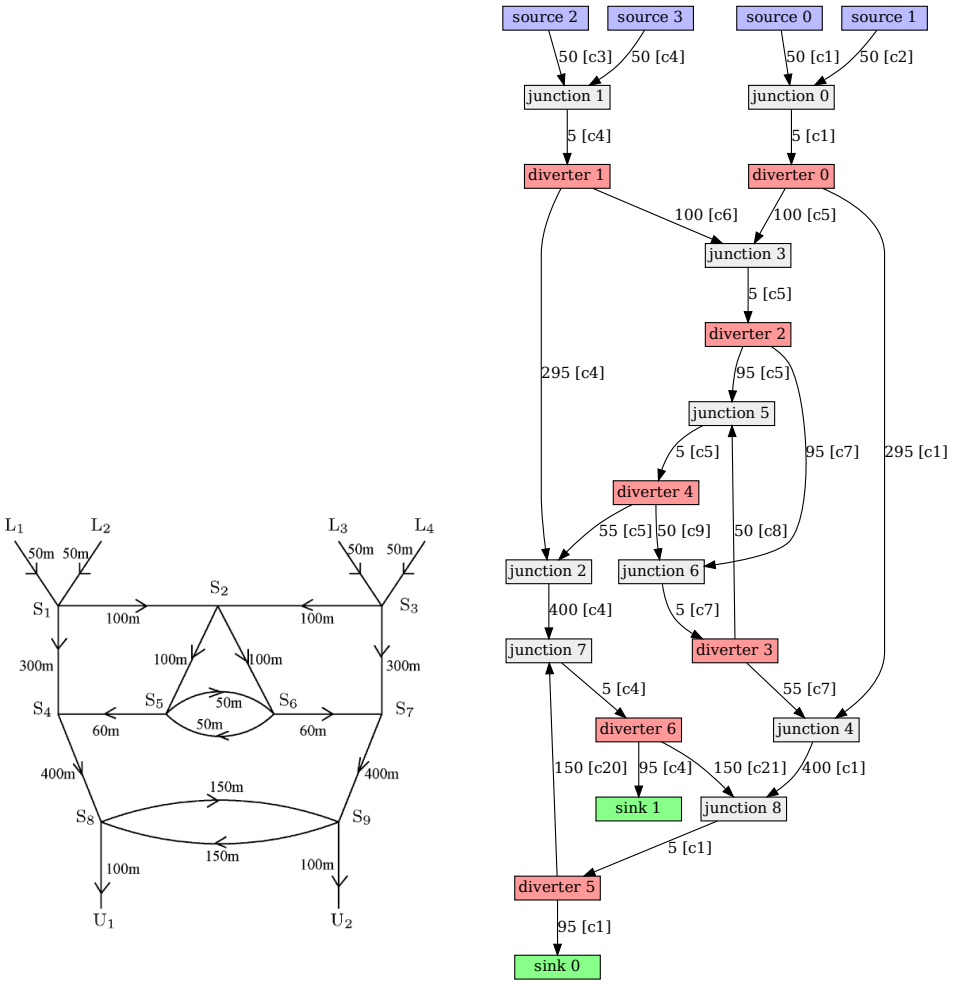

Fig. 2. Left: conveyor network from [15]. Arks correspond to conveyor sections and are labeled by their lengths. Sources are labeled as $L_1, \ldots, L_4$ and sinks are labeled as U_1, U_2. Right: corresponding conveyor graph

delivered to their target sinks so that the delivery time and, optionally, some other metrics (e.g., energy consumption [1]) are minimized. In this work, we will focus solely on delivery time.

2.2. Routing algorithm

Assume that the routing decisions in the conveyor network are made by a DNN $\mathcal{N}$. In DQN-routing, $\mathcal{N}$ is trained with RL. Although it is possible to learn either one DNN or custom DNNs for each node, w.l.o.g. we assume that an instance of the same DNN $\mathcal{N}$ is located at each non-sink node (this choice will not have a notable effect on the verification approach). We use an architecture based on [1], with the following enhancements in node representation to support the possibility of dynamic changes in the conveyor network:

Each node is represented by its *embedding*, a real vector trained based on the conveyor graph. Embeddings are computed using a variant of the Laplacian Eigenmaps [35] algorithm. Routing outcomes are obtained by executing $\mathcal{N}$ on embeddings of various nodes.

The output of $\mathcal{N}$ depends on the embeddings of three nodes: $\mathbf{e}_c$, embedding of the current node c (where a package waits for a routing decision), $\mathbf{e}_d$, embedding of the destination node d (sink), and $\mathbf{e}_n$, embedding of the candidate neighbor n to which the package can be routed. As input, $\mathcal{N}$ receives concat$(\mathbf{e}_d - \mathbf{e}_c, \mathbf{e}_n - \mathbf{e}_c)$. $\mathcal{N}$ is trained as a deep Q network [36] with two hidden layers and predicts the Q value (expected cumulative future reward) of the routing decision. The rewards are based on package delivery time of the conveyor

network, so the predicted Q value is the negation of the expected remaining delivery time of the current package.

At diverters, routing decisions are made according to a *decision distribution*, which converts the predicted Q values $q_i = \mathcal{N}(\text{concat}(\mathbf{e}_d - \mathbf{e}_c, \mathbf{e}_{n_i} - \mathbf{e}_c))\,(1 \le i \le 2)$ to the probabilities p_1 and $p_2 = 1 - p_1$ of routing the package to n_1 and n_2, respectively. Below, we assume that n_1 is always the next checkpoint on the current conveyor and n_2 is a checkpoint on a different conveyor. The decision distribution is a Boltzmann distribution:

$$(p_1, 1 - p_1) = \text{softmax}((q_1, q_2)/T), \tag{2}$$

where $T > 0$ ("temperature") is a hyperparameter. Thus,

$$p_1 = \sigma((q_1 - q_2)/T), \tag{3}$$

where σ is the sigmoid (logistic) function. The probabilistic routing rule is not used when the destination is unreachable from one of the neighbors (reachability is computed before routing): in this case the package is routed to the other neighbor. Assuming that package destinations are always reachable from the corresponding sources, this rule implies that there is always at least one neighbor suitable for routing. Although Q values are also predicted at sources and junctions, routing decisions are not made there as the next checkpoint is unique.

Keeping in mind the rules of making routing decisions, we see that the delivery depends only on the decisions at a subset of diverters. For a source v_1 and sink v_2, these are only diverters v_3 such that (1) v_3 is reachable from v_1 and (2) v_2 is reachable from both successors of v_3. For a fixed source/sink pair, we will call such deverters *nontrivial*.

3. Proposed verification approach

3.1. Verification problem

Hereinafter, we are interested in the delivery time of a package between a chosen source and sink. Due to the stochasticity of DNN routing decisions, it is not meaningful to verify the maximum possible delivery time as it is only determined by the conveyor graph: e.g., it will be infinite if the conveyor graph has cycles reachable from the source and from which the sink is reachable. Verifying bounds on the delivery time statistics (means or medians) is more comprehensible.

We define the *cost* C_{π,v_1,v_2} of delivering a package π from source $v_1 \in I$ to sink $v_2 \in O$ as the total time spent on the delivery. For fixed v_1 and v_2, the embeddings of conveyor graph nodes $\mathbf{e}_1, \dots, \mathbf{e}_{|V|}$, and the maximum allowed ℓ_∞ discrepancy of these embeddings ϵ, we aim to verify that the expectation of C_{π,v_1,v_2} taken over the stochastic routing decisions of DNN agents, does not exceed the given bound c_0:

$$\forall \pi\ \forall \mathbf{e}'_1, \dots, \mathbf{e}'_{|V|} \left(\bigwedge_{i=1}^{|V|} \left\| \mathbf{e}'_i - \mathbf{e}_i \right\|_\infty \le \epsilon \implies \mathrm{E}\, C_{\pi,v_1,v_2} \le c_0 \right). \tag{4}$$

Intuitively, this means that routing decisions cannot deteriorate if the input of $\mathcal{N}$ changes slightly (ϵ). The input change may result from altering the conveyor graph, e.g., due to conveyor break. The choice of the ℓ_∞ norm as the input metric is based on the restrictions of DNN verification tools, and the choice of delivery cost expectation (and not, e.g., its median) is justified by the possibility to compute this value analytically for an arbitrary conveyor graph as explained below.

3.2. Expected delivery cost

If the speeds of all conveyors are known constants, C_{π,v_1,v_2} can be computed based on the lengths of traversed conveyor sections. For simplicity, we assume that the speeds of all conveyors are 1 m/s, and the

lengths of conveyors are measured in meters. This makes the numerical values of conveyor lengths and their traversal times equal, allowing us to substitute the latter with the former.

Let us compute the expectation of C_{π,v_1,v_2}, assuming that $\mathcal{N}$ is fixed during the delivery of π. This assumption may be justified by the possibility to temporarily stop the training of $\mathcal{N}$ when its parameters have converged and the distribution of incoming packages is fixed. For fixed $\mathcal{N}$, package deliveries become independent (under the assumption that packages do not collide), and the delivery of each package can be described as a discrete-time Markov chain $\mathcal{C}$:

- the states of $\mathcal{C}$ correspond to conveyor graph nodes;
- the initial state of $\mathcal{C}$ corresponds to v_1;
- the transition probabilities of $\mathcal{C}$ are determined by the decision distribution (2) of $\mathcal{N}$, and transitions have *weights* equal to conveyor section lengths.

As C_{π,v_1,v_2} only depends on v_1 and v_2, from now on we omit π in C_{π,v_1,v_2}. $\mathrm{E}\, C_{v_1,v_2}$ is the expected time of reaching v_2 starting from v_1. Since the routing algorithm never routes packages to nodes without a path to the destination and this path can always be selected with a non-zero probability, $\mathrm{E}\, C_{v_1,v_2}$ is finite. Since $\mathrm{E}\, C_{v_1,v_2}$ is similar to the expected hitting time in a Markov chain (unlike the hitting time, it accounts for transition weights), it can be found [37] by solving a system of $|V|$ linear equations. This system can be simplified by excluding the nodes that do not belong to any path from v_1 to v_2.

Next, suppose that routing action probabilities are not fixed. Then, for each nontrivial diverter $1 \le i \le t$, suppose that p_i and $1 - p_i$ are the probabilities of routing the package to the checkpoint on the current conveyor and on the adjacent conveyor, respectively, and $\mathbf{p} = (p_1, \dots, p_t)$. Here, the components of $\mathbf{p}$ do not need to sum up to one.

Suppose that each component p_i of $\mathbf{p}$ cannot have extreme values: $\mathbf{p} \in (0,1)^t$. This is true if the decision distribution is adjusted as explained in the next subsection. Suppose also that the following holds: if $\mathbf{p} \in (0,1)^t$, then the aforementioned equation system has a unique solution. This implication was validated for the two example conveyor graphs considered in this paper, but checking it for arbitrary graphs is retained for future work. Now, if both assumptions hold, the solution is unique and can be found symbolically, giving the expected delivery time of a package between chosen source and sink $\tau(\mathbf{p}) := \mathrm{E}\, C_{v_1,v_2} = f_1(\mathbf{p})/f_2(\mathbf{p})$, where f_1 and f_2 are some polynomials. In this representation, we can take $f_2(\mathbf{p})$ to be the (non-zero) determinant of the system.

3.3. Probability smoothing

If $\mathcal{N}$ is very confident in routing decisions, the distribution (2) generates probabilities very close to zero or one, and they may be processed as zeros and ones due to machine rounding, possibly making package delivery impossible. What is more, if we would like to maximize $\tau(\mathbf{p})$ w.r.t. its argument with gradient-based approaches, probabilities with extreme values may cause vanishing gradients.

To resolve these issues, we modify (2) to separate the probabilities from zero and one by introducing *probability smoothing* by analogy with label smoothing [38] in classifier training:

$$\mathrm{smooth}(p) = (1 - \mu)p + \mu/2, \tag{5}$$

where $\mu \in (0,1)$ is a hyperparameter. This rule is applied right after (2) and is used when $\mathcal{N}$ is already trained. If training is done with RL, probability smoothing would encourage the agents to explore more, but a small μ would not alter decisions significantly and at the same time would ensure that the probabilities belong to the interval $[\mu/2, 1 - \mu/2]$, i.e., are separated from the extreme values.

3.4. Simultaneous computation of routing probabilities

We would like to reduce our verification problem to verifying a single DNN. Unfortunately, a DNN that encodes $\tau(\mathbf{p})$ does not fit into the layered structure (1), making existing DNN verification tools inapplicable. As a remedy, we can model the computation of $\mathbf{p}$. To do this, we find all the Q values used in its computation

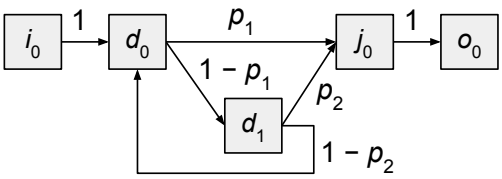

Fig. 3. Conveyor graph example; arcs are labeled with routing probabilities, the length of each section equals one

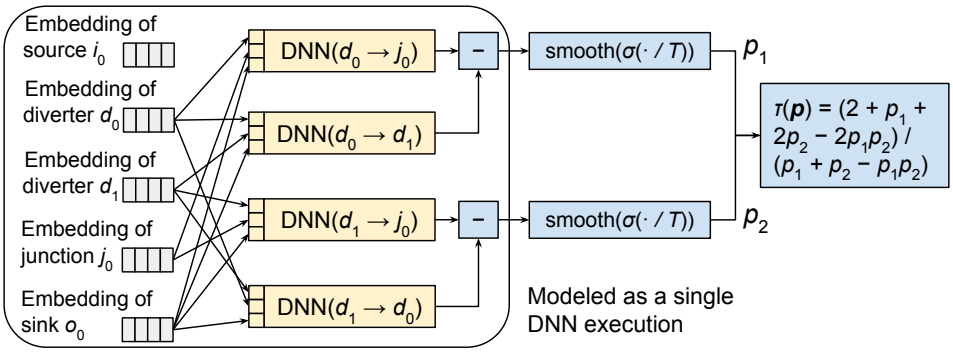

Fig. 4. Computation of $\tau(\mathbf{p})$ for the example in Fig. 3

by expressing them as a result of a single forward pass of a larger DNN $\mathcal{M}$, which we will construct below. The number of such Q values is $2t$, thus all executions of $\mathcal{N}$ can be modeled by repeating $\mathbf{A}_i$ and $\mathbf{b}_i$ for all layers i as $\mathrm{diag}(\mathbf{A}_i, 2t)$ and $\mathrm{concat}(\mathbf{b}_i, 2t)$. The idea of modeling several DNN executions as a single one is illustrated in Fig. 4 for a tiny conveyor graph shown in Fig. 3.

However, if the inputs of $\mathcal{N}$ are also duplicated, solving the problem (4) would be impossible as the same embedding vectors would be allowed to take different values. To resolve this, all different embeddings $\mathbf{e}'_1, \dots, \mathbf{e}'_{|V|}$ must be given to $\mathcal{M}$ only once, as their concatenation. We also need to subtract the embeddings of current nodes from other embeddings. Both duplication and subtraction can be done with a single linear transformation $\mathbf{B} = \{\mathbf{B}_{r,s}\}$ with a block structure, where $\mathbf{B}_{r,s}$ ($1 \le r \le 2t$, $1 \le s \le |V|$) are square matrices with the side equal to the dimension of an embedding, and are either zero, identity, or negated identity matrices:

$$\begin{bmatrix} \mathbf{e}_{d_{1,1}} - \mathbf{e}_{c_1} \\ \mathbf{e}_{d_{1,2}} - \mathbf{e}_{c_1} \\ \dots \\ \mathbf{e}_{d_{t,1}} - \mathbf{e}_{c_t} \\ \mathbf{e}_{d_{t,2}} - \mathbf{e}_{c_t} \end{bmatrix} = \begin{bmatrix} \mathbf{B}_{1,1} & \dots & \mathbf{B}_{1,|V|} \\ \dots & \dots & \dots \\ \mathbf{B}_{2t,1} & \dots & \mathbf{B}_{2t,|V|} \end{bmatrix} \begin{bmatrix} \mathbf{e}'_1 \\ \dots \\ \mathbf{e}'_{|V|} \end{bmatrix}. \tag{6}$$

Thus, the actual first-layer matrix of $\mathcal{M}$ is $\mathrm{diag}(\mathbf{A}_1, 2t) \cdot \mathbf{B}$.

Note that the embeddings of some nodes are not needed to compute $\tau(\mathbf{p})$: it is sufficient to use embeddings for all nontrivial diverters, their successors in the graph, and v_2. Formula (6) implies that other embeddings are always multiplied by zero blocks. To reduce the computational complexity, we filter out unused embeddings from $\mathbf{e}'_1, \dots, \mathbf{e}'_{|V|}$ prior to constructing $\mathcal{M}$.

In addition, several of the decisions above depend on the considered routing algorithm, DQN-routing. Yet, encoding multiple outputs of DNNs that operate differently is also possible. First, $\mathcal{N}$ could accept the embeddings of both neighbors and produce the score (e.g., logit) used to compute the current component of $\mathbf{p}$ right away, without Q values. In this case, it would suffice to repeat $\mathcal{N}$ only t times instead of $2t$. Second, having different DNNs $\mathcal{N}^1, \dots, \mathcal{N}^t$ with weight matrices $\mathbf{A}_i^1, \dots, \mathbf{A}_i^t$ and bias vectors $\mathbf{b}_i^1, \dots, \mathbf{b}_i^t$ of the same shape in different nodes instead of the single DNN $\mathcal{N}$ can be handled by composing their parameters as $\mathrm{diag}(\mathrm{diag}(\mathbf{A}_i^1, 2), \dots, \mathrm{diag}(\mathbf{A}_i^t, 2))$ and $\mathrm{concat}(\mathrm{concat}(\mathbf{b}_i^1, 2), \dots, \mathrm{concat}(\mathbf{b}_i^t, 2))$.

3.5. Proposed verification algorithm

The ability to capture the Q values computed by all nodes allows us to bound their differences and thus, by calling a DNN verification tool, compute the reachability of a certain hyperrectangle R of probability vectors. The concrete bounds needed to check the reachability of R can be found by reverse application of (3) and (5). In turn, the transition from the probability region to the verification outcome can be handled by a satisfiability modulo theories (SMT) solver [39].

The proposed verification algorithm is given in Alg. 1 and is based on a dichotomic procedure parameterized by the probability region R. The regions to be processed are placed in a queue U, i.e., the space of probability vectors is processed in the breadth-first search (BFS) order. Initially, the problem needs to be solved for all possible probability vectors $R = [\mu/2, 1 - \mu/2]^t$, which is the initial region in the queue (line 2). Mapping between probability vectors and outputs of $\mathcal{M}$ is done with a component-wise function g (line 3), which is monotonic and trivial to inverse. Each region R (line 5) is analyzed, resulting in proving the bound, finding a counterexample, or splitting the problem.

1. If $\tau(\mathbf{p}) \leq c_0$ (line 6) for all probability vectors in R, then the cost bound is proven for R and we proceed to the next unprocessed region (line 7). Since $\tau(\mathbf{p})$ is expressed symbolically, this check can be performed by an SMT solver.
2. Alternatively, the algorithm checks whether R is reachable for some embeddings $\mathbf{e}'_1, \ldots, \mathbf{e}'_{|V|}$ within the robustness assumption. This can be done by requesting a DNN verification tool to check the reachability of $g^{-1}(R)$, which, like R, is an intersection of hyperplanes and can be specified with linear constraints. If it is unreachable, we have a proof for R. In this case, the condition on line 8 fails and we proceed to the next unprocessed region.
3. Alternatively, if R is reachable and the bound is violated for the counterexample returned by the verification tool (lines 8–10), the algorithm has found a true counterexample and returns it immediately (line 11).
4. Otherwise, no conclusion can be made, and the algorithm attempts to find a proof or refutation by splitting R into two regions (line 12) and scheduling them to be processed later (line 13). We divide R into two equal rectangular halves by splitting the longest dimension.

Finally, if all regions have been processed without reporting a counterexample, this means that all parts of the initial probability region were verified (line 15). The idea behind the algorithm is illustrated in Fig. 5.

Alg. 1 is sound and complete unless it is run for the real maximum of τ, but its worst-case execution time is unbounded (see Technical Appendix for proofs). In practice, the performance of the algorithm deteriorates when c_0 approaches the approximately computed real maximum of τ.

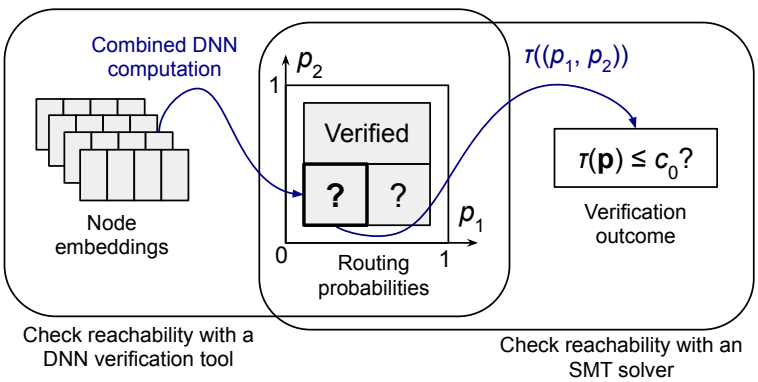

Fig. 5. The idea behind Alg. 1. In this example, $t = 2$. The algorithm works with a region of the probability space and uses third-party tools that can connect it to node embeddings and mean delivery cost whose bound must be checked.

Data: neural network $\mathcal{M}$ as defined above, number of nontrivial diverters t, temperature T, smoothing parameter μ, cost bound c_0, cost function τ, default embeddings $\mathbf{e} = \text{concat}\left(\mathbf{e}_1, \ldots, \mathbf{e}_{|V|}\right)$

Result: `Verified` or `Counterexample`$\left(\mathbf{e}', \mathbf{p}, c\right)$

$U \leftarrow$ `FIFOQueue()`
`enqueue`$\left(U, [\mu/2, 1 - \mu/2]^t\right)$
$g \leftarrow \text{smooth}(\sigma(\cdot/T))$
while U is not empty **do**
  $R \leftarrow$ `dequeue`(U)
  if $\forall \mathbf{p} \in R\ \tau(\mathbf{p}) \le c_0$ **then**
    continue
  if $\exists \mathbf{e}' = \text{concat}\left(\mathbf{e}'_1, \ldots, \mathbf{e}'_{|V|}\right) : \left\|\mathbf{e}' - \mathbf{e}\right\|_\infty \le \epsilon \wedge \mathcal{M}(\mathbf{e}') \in g^{-1}(R)$ **then**
    $\mathbf{p} \leftarrow g(\mathcal{M}(\mathbf{e}'))$
    if $\tau(\mathbf{p}) > c_0$ **then**
      return `Counterexample`$\left(\mathbf{e}', \mathbf{p}, \tau(\mathbf{p})\right)$
    $(R_1, R_2) \leftarrow$ `split`(R)
    `enqueue`(U, R_1, R_2)
return `Verified`

Algorithm 1. Checking robustness of the expected package delivery time w.r.t. node embeddings.

4. Experiments

4.1. Experimental setup

The proposed formal verification approach was implemented in Python[1]. As software tools to verify DNNs, solve SMT, and perform symbolic computations, we used Marabou [13], Z3 [40], and SymPy [41], respectively. We also implemented gradient-based search of adversarial examples with PGD, using PyTorch[2]. All experiments were run on an Intel Core i7-9750H 2.6 GHz CPU, using one core.

The approach was evaluated on conveyor graphs from Fig. 1–2, later referred to as graph 1 and graph 2. For both graphs, a DNN with two hidden layers of size 64 was trained according to [1]. Hyperparameter values were chosen heuristically to make mean delivery time plots of DQN-routing comparable with other routing algorithms considered in [1]. In particular, we chose 10 and 8 as embedding dimensions, and $T = 1.5$ and $T = 10$ as temperatures. As a smoothing parameter, we used $\mu = 0.01$, which is small enough not to change the behavior of DQN-routing significantly.

4.2. Symbolic computation of expected delivery time

As a preliminary step, our approach computes the expected delivery time $\tau(\mathbf{p})$ by symbolically solving a linear system of up to $|V|$ equations. In our experiments, the number of equations in this system after excluding irrelevant nodes was between 12 and 16, and $\tau(\mathbf{p})$ was computed instantly.

Some examples of the computed expressions for $\tau(\mathbf{p})$ are given below (the components of $\mathbf{p}$ correspond to nontrivial diverters v_c). Below, we denote the k-th source, sink, junction, and diverter as i_k, o_k, j_k, and d_k, respectively. The indices of all the nodes are given in Fig. 1–2. Let $\mathrm{P}(v_n|v_c, v_d)$ be the probability of routing the package to node v_n given that now it is at node v_c and its destination is v_d.

[1] https://github.com/ctlab/dqnroute
[2] https://github.com/pytorch/pytorch

For graph 1, assuming delivery from i_1 to o_3, $\tau(\mathbf{p}) = -20p_1p_2 + 40p_1 + 43$, where $p_1 = \mathrm{P}(d_3|d_2, o_3)$ and $p_2 = \mathrm{P}(j_0|d_3, o_3)$. Note that this expression does not depend on the routing probabilities given the package is at d_0 since this diverter is unreachable from i_1, on similar probabilities assuming that the package is at d_5 or d_6 since the sink o_3 is unreachable from these diverters, and on similar probabilities at d_1, d_4, d_7 and d_8 since only one arc from each of these nodes makes o_3 reachable and thus routing decisions are deterministic at these diverters. Also, even if probability smoothing is disabled, $\tau(\mathbf{p})$ is bounded, which complies with the acyclicity of the graph, and cannot exceed 83.

For graph 2, assuming delivery from i_0 to o_1, $\tau(\mathbf{p}) = (-155p_1p_2p_3p_4p_5 + 55p_1p_2p_3p_5 - 55p_1p_2p_4p_5 + 245p_1p_3p_4p_5 - 145p_1p_3p_5 - 245p_1p_4p_5 + 110p_1p_5 + 155p_2p_3p_4p_5 - 55p_2p_3p_5 + 55p_2p_4p_5 + 450p_3p_4p_5 + 310p_3p_4 - 550p_3p_5 - 310p_3 - 450p_4p_5 - 310p_4 - 110p_5)/(p_5(p_3p_4 - p_3 - p_4))$, where $p_1 = \mathrm{P}(j_4|d_0, o_1)$, $p_2 = \mathrm{P}(j_5|d_2, o_1)$, $p_3 = \mathrm{P}(j_4|d_3, o_1)$, $p_4 = \mathrm{P}(j_2|d_4, o_1)$, $p_5 = \mathrm{P}(o_1|d_6, o_1)$. Note that, if probability smoothing is disabled, this value is unbounded (it approaches infinity when $p_5 \longrightarrow 0$), which complies with the cyclicity of the graph. This value also does not depend on routing probabilities at d_1 since this diverter is unreachable from i_0, and at d_5 since this diverter cannot route the package to o_0, and thus its decision is deterministic.

In the experiments below, we consider two source/sink pairs per graph: the examples above and two more, from i_1 to o_2 in graph 1, and from i_2 to o_0 in graph 2. These pairs correspond to verification problems with $t = 2$ for graph 1 and $t = 5$ for graph 2. These dimensions are further reduced to 2 and 5 respectively since each $\tau(\mathbf{p})$ does not depend on one of the components of $\mathbf{p}$ (in the examples above, these are components p_0 and p_1 respectively). These are the maximum dimensions of the probability space that are possible for these graphs.

4.3. Gradient-based search of adversarial examples

Prior to formal verification, it is useful to study the considered verification problems by running imprecise, but faster methods: e.g., searching for adversarial examples with PGD provides approximate values on the actual maxima of the expected delivery cost τ. Found values are used to determine the thresholds for formal verification. We investigate various ℓ_∞-norm embedding discrepancies ϵ from 0 to 6.4. Note that $\epsilon = 0$ gives unique embeddings that DQN-routing computes with the Laplacian Eigenmaps algorithm: in this case the verification problem is essentially reduced to testing.

When running PGD, we performed ten restarts with 100 optimization steps of magnitude 0.02ϵ in each. Each run used a different starting point. The overall running time of PGD for a fixed value of ϵ was 4–5 s for graph 1 and 7–9 s for graph 2. The found maxima of τ are shown in Table 1: the expected delivery cost is robust to moderate embedding discrepancies, but under large discrepancies it can be made significantly larger.

Table 1. Expected delivery cost maxima found with PGD

ϵ	Graph 1		Graph 2	
	$i_1 \longrightarrow o_3$	$i_1 \longrightarrow o_2$	$i_0 \longrightarrow o_1$	$i_2 \longrightarrow o_0$
0.00	43.10	53.03	818.62	818.43
0.01	43.10	53.04	818.63	818.44
0.10	43.10	53.09	818.76	818.48
0.20	43.11	53.10	818.98	818.53
0.40	52.27	54.11	819.79	818.73
0.80	80.82	72.79	824.99	819.72
1.60	82.70	72.80	919.36	831.37
3.20	82.70	72.80	3727.38	963.18
6.40	82.70	72.80	11786.13	11721.99

4.4. Formal verification

We ran DNN verification for each value of ϵ used in the PGD experiments. The approximate expected delivery cost maxima found with PGD were used to select the upper bounds c_0 to be verified. These bounds were chosen to expect both positive and negative verification outcomes for different ϵ: e.g., to consider hard instances of the verification problem, we selected some c_0 values to be close to the approximate maxima. Each run was limited to 2 hours and 12 GB of RAM.

Verification results are shown in Table 2. With the increase of ϵ, verification becomes more computationally demanding and in most cases eventually violates either the time or the memory limit. As the resource consumption of the verification approach is dominated by Marabou, with SMT solver executions times being negligible, a possible explanation for this is that Marabou might have been optimized for small input discrepancies more typical in adversarial example search. As for verification outcomes, they are consistent with PGD results. However, verification time increases when approaching the approximate maxima, especially from above. In this case, not only the running time of Marabou increases (from a couple of seconds to almost two hours), but also the required number of calls to this tool.

Table 2. Results of formal verification. Verification outcomes are denoted as "+" (verified), "–" (counterexample found) and "?" (unknown). For a completed verification run, its execution time in seconds is shown in parentheses. For an unknown outcome, the parentheses instead show its reason: reaching the time limit (TL) or the memory limit (ML)

c_0	$\epsilon = 0$	$\epsilon = 0.01$	$\epsilon = 0.1$	$\epsilon = 0.2$	$\epsilon = 0.4$	$\epsilon = 0.8$	$\epsilon = 1.6$	$\epsilon = 3.2$	$\epsilon = 6.4$
Graph 1, delivery from i_1 to o_3									
81.00	+ (3)	+ (4)	+ (4)	+ (4)	? (TL)	? (TL)	? (ML)	? (ML)	? (ML)
44.00	+ (7)	+ (12)	+ (20)	+ (716)	? (TL)	? (TL)	– (969)	– (1423)	– (343)
43.50	+ (8)	+ (13)	+ (21)	+ (3298)	? (TL)	? (TL)	– (970)	– (1379)	– (347)
43.12	+ (13)	+ (23)	+ (37)	? (TL)	? (TL)	? (TL)	– (965)	– (1324)	– (346)
43.10	– (3)	– (3)	– (4)	– (4)	– (6)	– (17)	– (263)	– (567)	– (369)
Graph 1, delivery from i_1 to o_2									
72.80	+ (3)	+ (4)	+ (7)	+ (7)	? (TL)	? (TL)	? (ML)	? (ML)	? (ML)
54.12	+ (5)	+ (12)	+ (100)	? (TL)	? (TL)	– (362)	? (ML)	? (ML)	– (412)
53.11	+ (9)	+ (77)	+ (2597)	? (TL)	? (TL)	– (382)	? (ML)	? (ML)	– (411)
53.10	+ (11)	+ (87)	+ (4160)	? (TL)	? (TL)	– (385)	? (ML)	? (ML)	– (412)
53.00	– (4)	– (3)	– (3)	– (5)	– (10)	– (46)	– (397)	? (ML)	– (412)
Graph 2, delivery from i_0 to o_1									
825.0	+ (90)	+ (165)	+ (222)	+ (434)	+ (619)	? (ML)	? (ML)	? (ML)	? (ML)
820.0	+ (123)	+ (233)	+ (492)	+ (1727)	? (TL)	? (ML)	? (ML)	? (ML)	? (ML)
819.0	+ (147)	+ (370)	+ (2006)	? (TL)	? (TL)	? (ML)	? (ML)	? (ML)	? (ML)
818.6	– (4)	– (6)	– (7)	– (32)	– (17)	– (8)	– (85)	? (ML)	? (ML)
818.0	– (4)	– (7)	– (7)	– (10)	– (10)	– (9)	– (18)	? (ML)	? (ML)
Graph 2, delivery from i_2 to o_0									
830.0	+ (81)	+ (161)	+ (215)	+ (561)	+ (477)	? (TL)	? (ML)	? (ML)	? (ML)
820.0	+ (118)	+ (224)	+ (394)	+ (862)	+ (997)	? (TL)	? (ML)	? (ML)	? (ML)
819.0	+ (136)	+ (282)	+ (527)	+ (1275)	+ (3277)	? (TL)	? (ML)	? (ML)	? (ML)
818.4	– (4)	– (7)	– (7)	– (10)	– (14)	– (10)	– (21)	? (ML)	? (ML)
818.0	– (4)	– (6)	– (7)	– (9)	– (15)	– (10)	– (20)	? (ML)	? (ML)

As for the routing algorithm under verification, we conclude that the delivery time performance of DQN-routing is robust w.r.t. the node representations. This can be explained by relatively small input dimensions of constructed DNNs $\mathcal{M}$ (60 and 96 for graphs 1 and 2 respectively).

Discussion and conclusion

In this paper, we have considered the problem of verifying the performance of a DNN-based routing algorithm, where routing is done by a family of DNN agents located in the nodes of the routing network, and the performance indicator is the mean delivery time of a package between two chosen nodes of the network. To our best knowledge, this is the first application of sound and complete formal verification to a distributed system of DNNs (or multiple instances of the same DNN) operating together to achieve a common goal. The importance of this problem is justified by considering a distributed system of DNNs (or multiple instances of the same DNN) that operate together to achieve a common goal, which, to the best of our knowledge, has not yet been approached with sound and complete formal verification.

To solve this problem, we proposed a verification approach that can prove or refute the given upper bound on the mean delivery time. We focused on a particular class of routing problems, baggage handling, and a particular routing algorithm, DQN-routing, as the source of DNNs to be verified. The verification approach does not depend on the way the DNNs are trained, and we believe that it can be adapted to other routing problems. To make the problem solvable, we accepted several simplifying assumptions. We regarded the DNN agents to be fixed during the delivery, and focused on the stability of the expected delivery time w.r.t. network node representation (embeddings). We also assumed that packages cannot collide and the network is static and fault-free. Removal of the these assumptions is part of future work.

The proposed verification approach was evaluated on two conveyor networks with 20–22 nodes. The approach is fast when the allowed discrepancies of node embeddings approach zero, i.e., the verification problem approaches the one of testing. When these discrepancies are large, or, more importantly, we want to check a delivery time bound which is close to an approximation of the actual maximum of the mean delivery time, the approach becomes more computationally expensive. On the other hand, this performance decrease is relevant largely for the case of verifying satisfied bounds only since the violations of bounds can usually be found with PGD if it is run prior to formal verification.

As for the verified routing algorithm, DQN-routing, on the considered examples, we concluded that it learns DNNs that are relatively robust to the changes in node embeddings. While we focused on the support of this particular algorithm based on DNNs, the proposed verification approach may be adapted to others. In particular, the origins of node embeddings as well as the DNNs to be verified are not significant. Yet, our approach requires each node to have at most two successors. While this is a reasonable assumption for conveyor networks, it is not so for computer networks. Extending the proposed approach to handle routing in computer networks will require support of arbitrary node degrees. The RL approach that we verify, DQN-Routing, is applicable not only for conveyor delivery, but also for package routing in computer networks.

In addition to broadening the supported class of routing problems, we envisage several ways of improving the proposed approach. First, though the trade-off between completeness of analysis and computational complexity is unsurprising, there might be room for efficiency improvement.

Second, we assumed that the routing agents are static during delivery. When the DNNs under verification are trained with RL (as in DQN-routing), this limits the applicability of the proposed approach, though it is possible to use it after training, or periodically between RL iterations. Verifying DNNs in an RL framework is subject of future work.

Third, our approach assumes that the conveyor network is fixed. Violation of this assumption is partially covered by allowing node embedding discrepancies. As the most adverse topology change is a malfunction, future work may address simultaneous verification for multiple conveyor graphs obtained from the original one by removing one or more arcs.

A. Appendix: proofs of correctness

In this appendix, we prove several properties of Alg. 1. Below, we assume that the checks on lines 6 and 8 of Alg. 1 are performed with third-party tools that implement sound and complete algorithms, meaning that these checks always terminate and return correct results.

Let c_* be the maximum of τ on the set $S = \{\mathbf{e}' : \|\mathbf{e}' - \mathbf{e}\|_\infty \le \epsilon\}$ of allowed node embeddings. Let $\hat{R}$ be the set of probability vectors reachable from S. Then, $c_* = \tau(\mathbf{p}_*)$ for some $\mathbf{p}_* \in \hat{R}$. Recall that $\mathbf{p}$ is continuous as a function of node embeddings (it is computed by applying continuous functions to the outputs of the DNN $\mathcal{M}$, which is assumed to use a continuous activation function), and τ is continuous as a function of $\mathbf{p}$ (it is an arithmetic expression with non-zero denominator).

Theorem 1. *If Alg. 1 terminates, then its verification result is correct.*

Proof. First, suppose that Alg. 1 is run for $c_0 \ge c_*$, which means that the bound is satisfied. In this case, Alg. 1 will never return a counterexample since it is impossible to satisfy the conditions on lines 8 and 10 simultaneously. The only remaining way for the algorithm to terminate is to return a positive verification outcome on line 15.

Then, suppose that Alg. 1 is run for $c_0 < c_*$, which means that the bound is violated. In this case, $\mathbf{p}_*$ belongs to some hyperrectangle R_* such that $\forall \mathbf{p} \in R_*\ \tau(\mathbf{p}) > c_0$ and $R_* \cap \hat{R} \ne \varnothing$ (both sets contain $\mathbf{p}_*$). For any hyperrectangle containing R_*, the algorithm will be unable to report a positive verification result (condition on line 6 will fail and condition on line 8 will pass). Moreover, the condition on line 10 will be satisfied and a counterexample will be reported. □

Theorem 2. *If Alg. 1 is requested to check the bound $c_0 \ne c_*$, it will always terminate.*

Proof. Suppose that a call of Alg. 1 never terminates. First, suppose that Alg. 1 is run for $c_0 > c_*$. Then there is a sequence of nested hyperrectangles R_i such that R_{i+1} is one of the parts of R_i produced on lines 12–14. The algorithm splits the regions using the longest dimension, and thus there is a point $\mathbf{q}$ such that $\cap_{i=1}^{\infty} R_i = \{\mathbf{q}\}$. There is a sequence of points $\mathbf{p}_i$ that certify that the conditions on line 6 are violated, i.e., $\tau(\mathbf{p}_i) > c_0$. This sequence converges to $\mathbf{q}$, and since τ is continuous, $\tau(\mathbf{q}) \ge c_0$. With $c_0 > c_*$, this is only possible if $\mathbf{q} \notin \hat{R}$. There is also a sequence $\mathbf{p}'_i$ of points assigned on line 9, which also converges to $\mathbf{q}$. However, $\mathbf{p}'_i$ all belong to $\hat{R}$, which is compact as a continuous image of a compact set S. But then $\mathbf{q} \in \hat{R}$ as a limit of points in a closed set. Contradiction.

Then, suppose that Alg. 1 is run for $c_0 < c_*$. Recall the region R_* from the proof of the previous theorem for this case. If the algorithm never terminates, given the BFS order of the traversal of the set of probability vectors, it will necessarily consider a region that belongs to R_*. However, it must have reported a counterexample for this region. Contradiction. □

Theorem 3. *There is a family of verification problems parameterized only by c_0 such that the execution time of Alg. 1 is unbounded on it, and this family does not need to include the case $c_0 = c_*$*

Proof. We will show that for each integer n, it is possible to select an instance of the verification problem such that Alg. 1 processes more than n probability regions. First, we take $t = 1$, $\tau(\mathbf{p}) = 1 + p_1$. This situation corresponds to a tiny conveyor graph with one diverter that decides whether the package follows the path with the length of 1 or the path with the length of 2. Then, we can select the DNN such that $c_* = 3/2$ and request the algorithm to check the bound $c_0 = 3/2 + \gamma$. In this case, the actual verification outcome is positive, but by making γ sufficiently small, we can cause the algorithm to reach line 12 for as many intervals as we wish: these are the intervals that approach $p_1 = 1/2$ from above. □

References

[1] D. Mukhutdinov, A. Filchenkov, A. Shalyto, and V. Vyatkin, "Multi-agent deep learning for simultaneous optimization for time and energy in distributed routing system", *Future Generation Computer Systems*, vol. 94, pp. 587–600, 2019.

[2] J. A. Boyan and M. L. Littman, "Packet routing in dynamically changing networks: A reinforcement learning approach", in *Proceedings of the 6th International Conference on Neural Information Processing Systems*, 1993, pp. 671–678.

[3] G. Black and V. Vyatkin, "Intelligent component-based automation of baggage handling systems with IEC 61499", *IEEE Transactions on Automation Science and Engineering*, vol. 7, no. 2, pp. 337–351, 2009.

[4] A. Athalye, L. Engstrom, A. Ilyas, and K. Kwok, "Synthesizing robust adversarial examples", in *Proceedings of the 35th International Conference on Machine Learning*, 2018, pp. 284–293.

[5] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, "Robust physical-world attacks on deep learning visual classification", in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1625–1634.

[6] C. Szegedy, W. Zaremba, I. Sutskever, J. B. Estrach, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks", in *International Conference on Learning Representations*, 2014.

[7] R. Drechsler, Ed., *Advanced formal verification*. 2004, vol. 122.

[8] G. Anderson, S. Pailoor, I. Dillig, and S. Chaudhuri, "Optimization and abstraction: A synergistic approach for analyzing neural network robustness", in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2019, pp. 731–744.

[9] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari, "Output Range Analysis for Deep Feedforward Neural Networks", in *NASA Formal Methods*, A. Dutle, C. Muñoz, and A. Narkawicz, Eds., Cham: Springer International Publishing, 2018, pp. 121–138.

[10] Y. Y. Elboher, J. Gottschlich, and G. Katz, "An abstraction-based framework for neural network verification", in *Computer Aided Verification*, 2020, pp. 43–65.

[11] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks", in *Computer Aided Verification*, 2017, pp. 3–29.

[12] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient SMT solver for verifying deep neural networks", in *Computer Aided Verification*, 2017, pp. 97–117.

[13] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljić, *et al.*, "The Marabou framework for verification and analysis of deep neural networks", in *Computer Aided Verification*, 2019, pp. 443–452.

[14] M. Johnstone, D. Creighton, and S. Nahavandi, "Status-based routing in baggage handling systems: Searching verses learning", *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, vol. 40, no. 2, pp. 189–200, 2009.

[15] A. N. Tarau, B. De Schutter, and H. Hellendoorn, "Model-based control for route choice in automated baggage handling systems", *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 40, no. 3, pp. 341–351, 2010.

[16] O. Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. Nori, and A. Criminisi, "Measuring neural net robustness with constraints", in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, 2016, pp. 2613–2621.

[17] A. Fawzi, H. Fawzi, and O. Fawzi, "Adversarial vulnerability for any classifier", in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 2018, pp. 1178–1187.

[18] A. Fawzi, O. Fawzi, and P. Frossard, "Analysis of classifiers' robustness to adversarial perturbations", *Machner Learning*, vol. 107, no. 3, pp. 481–508, 2018.

[19] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "DeepFool: a simple and accurate method to fool deep neural networks", in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2574–2582.

[20] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks", in *International Conference on Learning Representations*, 2017.

[21] A. Boopathy, T.-W. Weng, P.-Y. Chen, S. Liu, and L. Daniel, "CNN-Cert: An efficient framework for certifying robustness of convolutional neural networks", in *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 3240–3247.

[22] P.-y. Chiang, R. Ni, A. Abdelkader, C. Zhu, C. Studor, and T. Goldstein, "Certified defenses for adversarial patches", in *International Conference on Learning Representations*, 2020.

[23] M. Lecuyer, V. Atlidakis, R. Geambasu, D. Hsu, and S. Jana, "Certified robustness to adversarial examples with differential privacy", in *IEEE S & P*, 2019, pp. 656–672.

[24] A. Raghunathan, J. Steinhardt, and P. Liang, "Certified Defenses against Adversarial Examples", in *International Conference on Learning Representations*, 2018.

[25] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with deep reinforcement learning", in *NIPS Deep Learning Workshop*, 2013.

[26] R. S. Sutton, A. G. Barto, *et al.*, *Introduction to reinforcement learning*. MIT press, 1998, vol. 135.

[27] E. Bacci and D. Parker, "Probabilistic Guarantees for Safe Deep Reinforcement Learning", in *Formal Modeling and Analysis of Timed Systems*, N. Bertrand and N. Jansen, Eds., Cham: Springer International Publishing, 2020, pp. 231–248.

[28] O. Bastani, Y. Pu, and A. Solar-Lezama, "Verifiable reinforcement learning via policy extraction", in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 2018, pp. 2494–2504.

[29] R. Ivanov, T. J. Carpenter, J. Weimer, R. Alur, G. J. Pappas, and I. Lee, "Case study: verifying the safety of an autonomous racing car with a neural network controller", in *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*, 2020, pp. 1–7.

[30] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, and I. Lee, "Verisig: verifying safety properties of hybrid systems with neural network controllers", in *Proceedings of the 22nd International Conference on Hybrid Systems: Computation and Control*, 2019, pp. 169–178.

[31] Y. Kazak, C. Barrett, G. Katz, and M. Schapira, "Verifying deep-RL-driven systems", in *Proceedings of the 2019 Workshop on Network Meets AI & ML*, 2019, pp. 83–89.

[32] L. Oakley, A. Oprea, and S. Tripakis, "Adversarial Robustness of AI Agents Acting in Probabilistic Environments", in *Workshop on Foundations of Computer Security*, 2020.

[33] H.-D. Tran, F. Cai, M. L. Diego, P. Musau, T. T. Johnson, and X. Koutsoukos, "Safety verification of cyber-physical systems with reinforcement learning control", *ACM Transactions on Embedded Computer Systems*, vol. 18, no. 5s, pp. 1–22, 2019.

[34] A. Bianco and L. De Alfaro, "Model checking of probabilistic and nondeterministic systems", in *Foundations of Software Technology and Theoretical Computer Science*, 1995, pp. 499–513.

[35] M. Belkin and P. Niyogi, "Laplacian eigenmaps and spectral techniques for embedding and clustering", in *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, 2002, pp. 585–591.

[36] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning", *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[37] J. R. Norris, *Markov chains*. Cambridge University Press, 1998.

[38] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception architecture for computer vision", in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2818–2826.

[39] C. Barrett and C. Tinelli, "Satisfiability Modulo Theories", in *Handbook of Model Checking*, E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, Eds. Cham: Springer International Publishing, 2018, pp. 305–343.

[40] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver", in *Tools and Algorithms for the Construction and Analysis of Systems*, 2008, pp. 337–340.

[41] A. Meurer, C. P. Smith, M. Paprocki, O. Čertık, S. B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J. K. Moore, S. Singh, *et al.*, "SymPy: symbolic computing in Python", *PeerJ Computer Science*, vol. 3, e103, 2017.