Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling (ICAPS 2023)

Fast and Robust Resource-Constrained Scheduling with Graph Neural Networks

Florent Teichteil-Königsbuch[1], Guillaume Povéda[1], Guillermo González de Garibay Barba[2], Tim Luchterhand[3], Sylvie Thiébaux[3,4]

[1]Airbus AI Research
[2]Airbus Spain
[3]LAAS-CNRS, ANITI, Université de Toulouse, INSA
[4]Australian National University
florent.teichteil-koenigsbuch@airbus.com, guillaume.poveda@airbus.com, guillermo.garibay@airbus.com, tim.luchterhand@laas.fr, sylvie.thiebaux@laas.fr

Abstract

Resource-Constrained Project Scheduling Problems (RCPSPs) are NP-complete, which makes it challenging to efficiently solve large instances and robustify solutions in the presence of uncertainty. To remedy this, we learn to efficiently mimic the solutions produced by Constraint Programming (CP) solver, using a Graph Neural Network (GNN) architecture designed to capture the structure of RCPSPs. Since the GNN solution may violate constraints, we ensure schedule feasibility at inference time by extracting the task ordering from the GNN schedule and post-processing it with the well-known Schedule Generation Scheme (SGS). We find that SIREN, the resulting algorithm, produces schedules that are of higher quality than those produced by the CP solver within the same computation time budget. The speed and solution quality of SIREN make it suitable as a component of an on-line scenario-based optimisation procedure for RCPSPs with *stochastic* durations. This leads to the SERENE system, which robustly selects, in real-time, the best next tasks to start in order to minimise the average makespan over the scenarios. Empirically, SERENE achieves better average makespan over different realisations of uncertainty than deterministic algorithms that continuously reschedule on the basis of either the worst, best or average task durations.

Introduction

Scheduling tasks constrained by precedence relations and shared limited resources is an ubiquitous problem found in many areas of human activities, e.g. university class timetabling, public transport optimisation, manufacturing task scheduling, and daily activity planning, to name but a few. Such problems belong to the widely studied class of Resource-Constrained Project Scheduling Problems (RCPSPs) (Özdamar and Ulusoy 1995). Unfortunately, solving RCPSPs optimally is NP-complete in general (Ganian, Hamm, and Mescoff 2020), which makes computing high quality schedules for large problems challenging. Many polynomial-time heuristics have been designed to compute schedules for large RCPSPs in reasonable time, but their quality remains far from that of the best-known solutions obtained with Constraint Programming (CP) (Schutt et al. 2013). One of those heuristics, known as Schedule Generation Scheme (SGS), generates a feasible schedule, i.e. a schedule respecting all the precedence and cumulative resource constraints, from a given ordered list of the tasks (Hartmann and Kolisch 2000). Many different task ordering heuristics have been proposed in the literature to feed SGS, for instance based on latest task finish times (Hartmann and Kolisch 2000) or on genetic programming (Regnier-Coudert and Povéda 2021).

In this paper, we investigate the use of machine learning to provide good and fast solutions to RCPSPs. Specifically, we consider Graph Neural Networks (GNNs) which are an extension of Deep Neural Networks to learn to approximate functions whose values are correlated following a graph structure (Wu et al. 2021). GNNs have proven to provide good approximation metrics in, e.g., path finding, social media networks (Fan et al. 2019), molecule design (Mercado et al. 2021), and classical planning heuristics (Shen, Trevizan, and Thiébaux 2020). We show that an RCPSP can be naturally represented as a graph linking resource nodes and task nodes, where the starting dates of the task nodes depend on the remaining level of resources fed by the predecessor resource nodes and on the ending dates of the predecessor task nodes. A GNN with this graph structure can be trained to learn an approximation of the task starting dates. Moreover GNNs have the property that they are able to work with different graph structures and inputs of varying sizes, and are hence applicable to unseen RCPSPs with different number of tasks and resources, and different constraint structures – at least provided that the distribution of the test instances follows a similar distribution to that of the training instances.[1]

We use our GNNs to learn to mimic the solutions produced by OR-Tools' CP-SAT solver (Perron and Furnon 2022), by training on a set of more than 1600 RCPSPs of different size and structure. Since the GNN infers an approximate schedule which does not need to exactly satisfy the precedence and resource constraints, we make the schedule feasible by extracting the task ordering from the task starting dates inferred by the GNN, and passing them to the SGS procedure. Our experiments on a test set of over 400 RCPSPs of different size and structure show that the learned GNN followed by the SGS procedure is able to infer schedules up to 4 orders of magnitude faster than the vanilla CP solver for the

[1]Note that we do not study the ability of our GNNs to handle instances of larger sizes than those they were trained on.

same schedule quality, and that the quality of the inferred GNN solutions corrected by SGS are on average only 5% worse than the CP solutions. We name our approach SIREN, for Scheduling wIth gRaph nEural Networks.

Building on the ability of SIREN to efficiently find good solutions, we then extend our approach to stochastic RCPSPs with uncertain task durations by adapting Hindsight Optimisation (HOP) (Chong, Givan, and Chang 2000), a well-known framework for on-line scenario-based stochastic optimisation. At each given step of the execution, we sample a number of task duration scenarios for the tasks that have not yet been executed. We then solve the corresponding deterministic RCPSPs using SIREN, and select, amongst the set of tasks which any of the scenarios' solutions prescribe to start next, the subset which results in the smallest expected makespan. This requires adapting HOP to choose sets of tasks to start without exploring an exponential number of subsets. We call this extension SERENE for Stochastic schEduling with gRaph nEural NEtworks. Our experiments show that the quality of the schedules produced by SERENE significantly exceeds that of standard reactive baselines that reschedule from the current executing state on the basis of the scenario with average, longest, or shortest task durations. SERENE also produces solutions of similar quality as those obtained by substituting CPSAT for SIREN in HOP, albeit with significantly reduced computation time, thereby enabling fast and robust schedule adaptation in the face of uncertainty.

To summarize, this paper's contributions are:

1. a Graph Neural Network encoding of RCPSPs;
2. the SIREN framework which combines GNNs and SGS to produce feasible solutions to RCPSPs with different sizes and structures;
3. the SERENE extension to RCPSPs with stochastic task durations which adapts Hindsight Optimisation to robustly select the next tasks to start.

Related Work

In recent years, there has been much interest in leveraging progress in machine learning (ML), especially deep neural networks, to solve combinatorial optimization (CO) problems. This includes work on specific problems such as the traveling salesman problem (TSP) (Joshi et al. 2022; Deudon et al. 2018) and job-shop scheduling (Park et al. 2021; Song et al. 2022), as well as work on problem-agnostic techniques such as satisfiability (SAT) (Kurin et al. 2020) and mixed-integer programming (MIP) (Khalil, Morris, and Lodi 2022; Nair et al. 2020). Graph neural networks (GNNs) enjoy particularly high popularity due to their ability to capture and exploit the inherent graph structure of these problems.

The majority of proposed approaches fall into three major categories as identified in (Bengio, Lodi, and Prouvost 2021). The first directly imitates an optimal solver by learning to produce the same kind of solutions. The second employs ML to replace parts of a classical optimization algorithm. And finally the third uses ML to configure an optimization algorithm. In the following we focus on the first two of these categories.

Integrating ML into an existing solver. The aim is to guide a classical solver to find solutions more quickly and has the advantage of maintaining the guarantees of said classical algorithm, e.g. optimality or the ability to provide upper and lower bounds of the objective. Gasse et al. (2019) represent a MIP as a bipartite graph with nodes for variables and constraints. They then train a GNN to efficiently approximate the well known *strong branching* heuristic (Applegate et al. 1995), which makes high-quality branching decisions, however at a considerable computational cost.

MIP-GNN developed by Khalil, Morris, and Lodi (2022) follows a somewhat similar approach. Here a GNN is trained to estimate the average values of the variables in a binary MIP, the so called *variable bias*. This bias can then be used to either directly produce partial variable assignments for warm-starting the MIP-solver, or to predict a score for each variable that can ultimately be used as a branching heuristic.

In theory, any instance of an RCPSP can be encoded as a MIP by either discretizing time or using more advanced approaches like (Artigues, Michelon, and Reusser 2003). However, both types of encodings might produce prohibitively large MIPs for large RCPSP instances. Thus, while both of the previously mentioned architectures for MIPs could work for RCPSPs in principle we argue that it is more efficient to encode the RCPSP directly as a graph. By relaxing the strict bipartite topology, we are able to encode precedence constraints between task nodes alongside resource constraints between task and resource nodes. Furthermore, state of the art RCPSP solvers are based on CP methods rather than MIP (Schutt et al. 2013; Laborie 2018).

Direct learning of solutions. Given a sufficiently large CO problem or strict runtime constraints, employing an optimal solver might be intractable in general. In these cases, neural approaches belonging to the second category, i.e. that produce high quality solutions on their own (independently of a classical solver) become of interest. Apart from the approach proposed in this paper, there exist further examples in the literature. Khalil et al. make use of the *structure2vec* network architecture (Dai, Dai, and Song 2016) to produce a graph embedding for typical graph problems like TSP, minimum vertex cover and maximum cut. Based on this embedding, a greedy policy is learned using reinforcement learning (RL). While their algorithm produces suboptimal solutions in general, it does find high quality solutions more quickly than the optimal solver CPLEX and of higher quality than established heuristic approaches from the literature. Additionally, it is able to run on GPUs and thus exploit their parallelization capabilities.

More closely related to the RCPSP problem structure is the work on Job-Shop problems in (Song et al. 2022; Park et al. 2021). For instance, Song et al. employ a GNN working on a similar graph structure as SIREN, i.e. operations and machines are represented as two types of nodes and precedences as well as dependencies on machines can be specified simultaneously. Still, the Job-Shop formulation only allows for exclusive resource access, whereas RCPSPs

in general include cumulative resources. Moreover, these works focus on Deep Reinforcement Learning (DRL) methods, whereas we try to exploit the strengths of model-based approaches, by learning the solutions produced by a CP solver with a GNN. The advantage of our supervised method over DRL is a substantial saving in training budget.

Stochastic problems. Finally and perhaps most importantly, all of the aforementioned contributions focus on deterministic problems. In contrast, the proposed SERENE-algorithm is able to solve stochastic RCPSPs, i.e. problem instances where the duration of a task is non-deterministic. It does so by combining SIREN with hindsight optimization (HOP). Since this leads to multiple calls of SIREN, execution speed is top priority. Luckily, SIREN produces high-quality solutions in, on average, significantly less time than optimal solvers. To our best knowledge, there currently exists no approach to stochastic RCPSPs that leverages the reasoning power and generalization ability of GNNs.

Resource-Constrained Project Scheduling

A resource-constrained project scheduling problem (RCPSP) considers limited resources and tasks defined by durations, resource needs and precedence relations. The problem usually consists in finding a schedule of minimal duration (or makespan), by assigning a start time to each task, so as to satisfy the precedence relations and the resource availability constraints.

Problem Definition

Formally a RCPSP is defined by a tuple $(T, R, P, \mathbf{r}, \mathbf{C}, \mathbf{d})$ where: T is a set of tasks; R is a set of resource types; $P \subseteq T \times T$ is a set of precedence constraints between tasks so that each pair $(i, j) \in P$ indicates that task i should be finished before task j can be started; $\mathbf{r} = (r_{i,k})_{i \in T, k \in R}$ where $r_{i,k}$ is the quantity of resource of type $k \in R$ required to perform task $i \in T$; $\mathbf{C} = (c_k)_{k \in R}$ where c_k is the total resource capacity of resource k available at any time; $\mathbf{d} = (d_i)_{i \in T}$ where d_i is the duration of task i. Solving an RCPSP consists in assigning a start time x_i to each task $i \in T$, so as to minimize the makespan defined in (1).

$$f = \max_{i \in T}(x_i + d_i) \tag{1}$$

This is subject to the following constraints: the precedence constraints eq. (2) and the cumulative resource constraints eq. (3) stating that at any time within the scheduling horizon h, the capacity of each resource must not be exceeded by the tasks that are concurrently running.

$$x_i + d_i \leq x_j \ \forall (i, j) \in P \tag{2}$$

$$\sum_{\substack{i \in T \text{ s.t.} \\ x_i \leq t < x_i + d_i}} r_{i,k} \leq c_k \ \forall k \in R \ \forall t \leq h \tag{3}$$

For more information on RCPSPs, we refer the reader to (Artigues, Demassey, and Neron 2008). Solving an RCPSP is known to be NP-hard, but efficient algorithms have been developed in the last decades to get reasonably good solutions. Methods fall mainly into two classes: inexact methods based on local search and meta-heuristics, and exact mathematical programming methods including Mixed Integer Linear Programming (MILP) and Constraint Programming (CP). Overview of methods can be found in (Pellerin, Perrier, and Berthaut 2020). CP has gained popularity in the last decade and outperforms other approaches on classical benchmarks (Schutt et al. 2013; Laborie 2018). In this paper, we will use OR-Tools' CP-SAT solver (Perron and Furnon 2022) as an efficient solver to learn from, and as a baseline in our experimental results. OR-Tools won several CP contests involving scheduling benchmarks, including the latest Minizinc Challenge[2].

Serial Schedule Generation Scheme

A schedule generation scheme is a routine able to compute a feasible schedule solution from another representation of a solution. For RCPSPs, the input space of a generation scheme is a priority list of tasks (equivalent to a permutation of task), from which the SGS algorithm is able to compute a feasible schedule as the output. Different schedule generation schemes have been introduced in (Kolisch 1994). They are widely used to quickly compute solutions, feeding from local search algorithms that optimize the priority list given to the SGS procedure (Ayodele, McCall, and Regnier-Coudert 2017; Van Peteghem and Vanhoucke 2014). In this paper we will use the *serial SGS* routine: it takes the tasks in the order of the priority list and insert them in the schedule at their earliest possible date, considering currently scheduled tasks. It can be shown that for classical RCPSPs, there exists a priority list from which the serial SGS routine will compute an optimal schedule.

Learning to Solve Deterministic RCPSPs

In this section we present our first contribution. It consists of a GNN that we train to imitate and directly output the solutions found by an exact CP solver. Before developing the method presented in this section, we first experimented with various Deep Reinforcement Learning (DRL) approaches, also shaping the reward with well known RCPSP heuristics. Except for the smallest instances, the DRL agents kept exploring the huge space of feasible solutions without ever finding any of the (rare) reasonably good ones. On the contrary, we found that training the GNN in a supervised fashion, using the solutions provided by the CP solver as labels, produced the best results by harnessing the relationship between the feasible solutions and the structure of the CP models.

GNNs are designed to have an inductive bias that is appropriate for data that is best represented as a graph (Bronstein et al. 2021). A distinctive property of GNNs is the invariance of the model regarding the ordering of the nodes. Furthermore, the same model and parameters can be applied to graphs with different number of elements and different topologies. This is what allows us to use a single GNN model across different RCPSP instances. Also, like in CNNs, the models have local connectivity. This is important because it significantly reduces the computational cost (i.e.

[2]https://www.minizinc.org/challenge2022/results2022.html

in contrast to a fully connected model), and it has some regularizing effect.

The use of GNNs to learn to solve RCPSPs raises two major issues which we address below. The first is that of designing the right GNN architecture to capture the most important relations in RCPSPs. The second is how to obtain a feasible schedule at inference time, given that the GNN-predicted schedules are approximate and are not necessarily feasible. We name our overall learning and inference framework `SIREN`, for *Scheduling wIth gRaph nEural Networks*.

Graph Neural Network Representation of RCPSPs

Given an instance of an RCPSP as introduced previously we define the GNN graph as a tuple $G = (T, R, E, \mathbf{V}, \mathbf{E})$ where tasks T and resources R are directly represented as two different types of nodes. Furthermore, there are three types of edges $E = E_P \cup E_R \cup E_{\text{rev}}$, i.e. precedence edges $E_P = P$, resource demand edges $E_R = \{(i,k), i \in T, k \in R \mid r_{i,k} > 0\}$ and parallel reverse links $E_{\text{rev}} = \{(j,i) \mid (i,j) \in E_P \cup E_R\}$ that are added to enable bidirectional information propagation. Each edge $(i,j) \in E$ is associated with an edge feature $\mathbf{e}_{ij} \in \mathbf{E}$, that allows the encoding of resource consumption and the distinction of the different edge types. Features for precedence edges $(i,j) \in E_P$ are defined as follows:

$$\mathbf{e}_{ij} = [1,0,0,0,0];\ \text{reverse link}\ \mathbf{e}_{ji} = [0,0,1,0,0], \quad (4)$$

while features for resource consumption edges $(k,i) \in E_R$ take the following form:

$$\mathbf{e}_{ki} = [0,1,0,0,r_{i,k}];\ \text{reverse link}\ \mathbf{e}_{ik} = [0,0,0,1,r_{i,k}]. \quad (5)$$

Analogously, the task durations and quantities of available resources are represented by task node features $\mathbf{v}_i \in \mathbf{V}_T, i \in T$ and resource node features $\mathbf{v}_k \in \mathbf{V}_R, k \in R$ respectively:

$$\mathbf{v}_i = [0,1,0,d_i],\ i \in T \quad (6)$$
$$\mathbf{v}_k = [1,0,c_k,0],\ k \in R \quad (7)$$

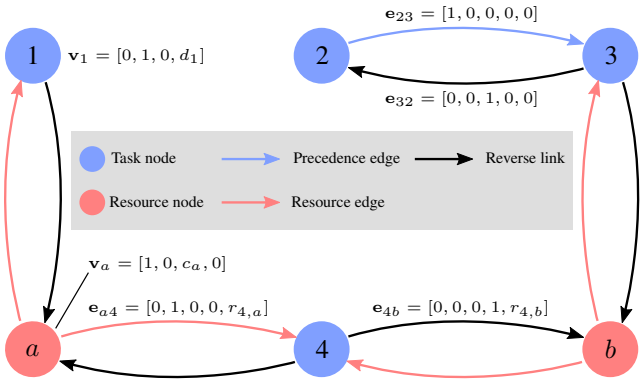

Figure 1: GNN graph representation of an RCPSP. Note that all edges and nodes are annotated with features but only some examples are shown here for the sake of simplicity

As opposed to the edge features $\mathbf{E}$, the node features $\mathbf{V} = \mathbf{V}_T \cup \mathbf{V}_R$ are changed by each Graph Transformer layer as described in the following paragraph.

There are certainly multiple possible ways of representing an RCPSP as a graph. The representation discussed above and depicted in Figure 1 enables an immediate processing with known GNN models.

GNN architecture. The architecture used in this work is a graph transformer as defined in (Shi et al. 2021) and implemented in pytorch-geometric (Fey and Lenssen 2019). This is a message passing model consisting of multiple layers. Each layer transforms the current graph G into a graph G' with identical topology but updated node features. Each node $i \in T \cup R$ aggregates information from its connected neighbors $\mathcal{N}(i) = \{j \in T \cup R \mid (j,i) \in E\}$ according to

$$\begin{aligned}\mathbf{v}'_i &= \phi(\mathbf{v}_i; \mathbf{V}, \mathbf{E}) \\ &= \mathbf{W}_1\mathbf{v}_i + \sum_{j \in \mathcal{N}(i)} \alpha_{j,i} \left(\mathbf{W}_2\mathbf{v}_j + \mathbf{W}_5\mathbf{e}_{ji}\right), \end{aligned} \quad (8)$$

where the weights $\alpha_{i,j}$ are calculated using an attention mechanism similar to (Vaswani et al. 2017):

$$\alpha_{j,i} = \text{softmax}\left(\frac{(\mathbf{W}_3\mathbf{v}_i)^\top(\mathbf{W}_4\mathbf{v}_j + \mathbf{W}_5\mathbf{e}_{ji})}{\sqrt{o}}\right). \quad (9)$$

The transformation in eqs. (8) and (9), is parameterized by the learnable weight matrices $\mathbf{W}_i, i \in \{1,\dots,5\}$. In eq. (9), o denotes the output dimension of the layer which we set to 256 in our experiments. Note that each layer has its own set of weights.

To facilitate the training of the neural network, we use a residual connection (He et al. 2015) for each layer. Additionally, a non-linear transformation $\text{ReLU}(x) := \max(0,x)$ and a normalization ψ are applied to the input of the residual layer. The complete chain of operations of one residual transformer layer is

$$\mathbf{v}'_i = \mathbf{v}_i + \phi(\text{ReLU}(\psi(\mathbf{v}_i)), \mathbf{V}, \mathbf{E}), \quad (10)$$
$$\psi(\mathbf{v}_i) = \mathbf{v}_i / C \ \text{ where } C = \|\mathbf{v}_i\|_2. \quad (11)$$

Note that partial derivatives with respect to C are discarded during gradient descent. We stack a total of 15 such residual transformer layers in order to allow information to propagate through large graph diameters.

Variable magnitude encoding. Our GNN is trained to approximate the task starting dates. Since we have RCPSPs of different sizes, we can have a large variability in the magnitudes of these starting dates.

Instead of letting the GNN directly predict the starting dates for task i as a scalar x_i^{gnn}, we have found that a different encoding of the output by the GNN can lead to faster convergence. We apply a standard multi-layer perceptron with linear activation to the output of the last transformer layer in order to obtain N scalars $x_{i,j}^{gnn}, j \in \{1,\dots,N\}$ for each task i. The final starting date x_i^{gnn} is then calculated by

$$x_i^{gnn} = \sum_{j=1}^{N} 2^{j-1} \cdot x_{i,j}^{gnn}. \quad (12)$$

In our experiments, we use $N = 8$. This encoding is fully differentiable.

Algorithm 1: SIREN: GNN training

Input: Set $\mathcal{P}$ of RCPSP problems with their corresponding known solutions $\mathcal{X} = (\mathbf{x}_P)_{P \in \mathcal{P}}$

Output: GNN weights $\mathbf{W}$ trained to fit the solutions of $\mathcal{P}$'s problems

▷ Build set $\mathcal{B}_{train}$ of training batches containing GNN encodings of problems in $\mathcal{P}$ and their solutions from $\mathcal{X}$

for all $0 \leqslant epoch < nb\_epochs$ **do**

  for all batch $b \in \mathcal{B}_{train}$ **do**

    ▷ Predict starting dates $\hat{\mathbf{x}}_b$ from the GNN batch b

    ▷ Compute the mean-squared error loss $\mathcal{L}_{MSE}$ between $\hat{\mathbf{x}}_b$ and the known solutions $\mathbf{x}_b$ of the RCPSPs encoded in the GNNs in b

    ▷ Backpropagate $\mathcal{L}_{MSE}$ and update $\mathbf{W}$

  end for

end for

return $\mathbf{W}$

The SIREN Training and Inference Framework

The overall training and inference procedures of SIREN[3], which are detailed below, are illustrated in Figure 2.

Training. Algorithm 1 presents our training procedure. It consists first in building a batch of GNNs which represent each a RCPSP problem as described in the previous subsection. Each batch contains a set of GNNs which are processed at once, which is helpful for the back-propagation phase where the GNN weights are updated to improve the prediction fitness. In Algorithm 1 we note $\mathbf{x}_b$ (resp. $\hat{\mathbf{x}}_b$) the solution vector obtained when concatenating the solution labels (resp. GNN predictions) of all the RCPSPs encoded in the GNNs of a given batch b.

We trained our GNN weights on RCPSP instances from psplib (Kolisch and Sprecher 1997) obtained from the github `kobe-scheduling` repository[4]. 1632 random instances of various sizes and structures (80% of the instances) have been chosen for training during 50000 epochs, which took less than a day on a AWS machine image equipped with 32 Intel(R) Xeon(R) CPU running at 2.30GHz and 2 GM204GL [Tesla M60] GPUs. The solutions were provided by CP-SAT which was run with a timeout of 15 minutes. Note that our approach is totally agnostic to the RCPSP solver used to label the RCPSP problem data with their respective solutions. Figure 3 shows the training statistics averaged over all the training instances: even if only the loss is used to train the GNN weights, we also report the evolution of the makespan (relative to the labeling solutions' average makespan) and of the constraint violations in terms of percentage of violated constraints and violation magnitude. The precedence (resp. resource) violation magnitude is the average absolute value of the overlapping duration for precedence tasks (resp. of the over-consumption of resources beyond their capacities). The precedence violation percentage increases during training because the GNN tries to learn schedules of makespans minimized by the CP solver, increasingly pressuring the precedence constraints as learning goes. Resource constraints are not impacted by this phenomenon: the GNN improves its ability to satisfy those constraints from the CP solution examples. Although the constraint violation percentage is roughly in the range 15%-20%, the magnitude of the violations is less than 1 unit, which means that the GNN can find nearly feasible schedules; in terms of precedence constraints for instance, it indicates that the resulting scheduled tasks overlap by less than 1 time unit.

[3]https://github.com/fteicht/gnn4rcpsp

[4]https://github.com/ptal/kobe-scheduling

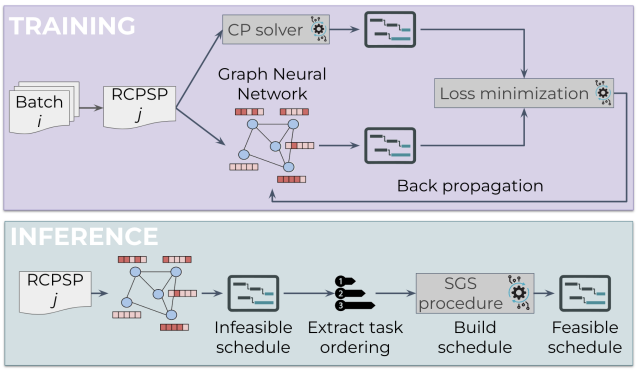

Figure 2: SIREN's learning and inference

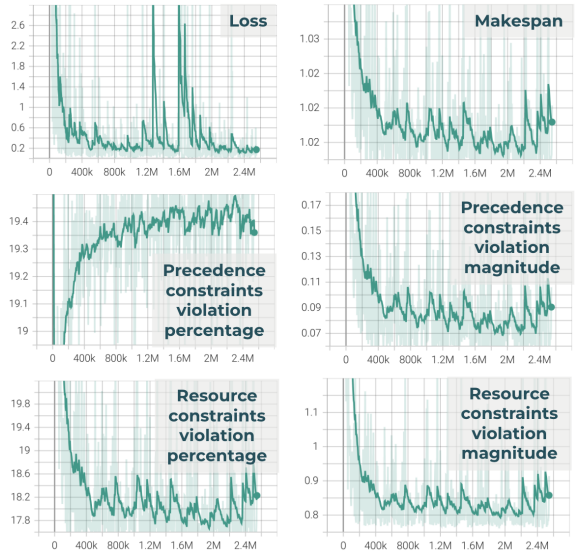

Figure 3: SIREN's training statistics. The horizontal (resp. vertical) axes represent the learning steps (resp. metrics values).

Inference. The slight violation of the constraints by the GNN prediction observed during training tells us that the GNN predictions must be corrected in order to make a feasible schedule. The first approach we tried consisted in calling the CP-SAT solver that we warm-started with the solution inferred by the GNN in order to find the first feasible solu-

Algorithm 2: SIREN: GNN inference with SGS correction

Input: Given RCPSP P with unknown solution; GNN trained weights $\mathbf{W}$

Output: Solution schedule $\mathbf{x}$ of P

▷ Build GNN $G_{\mathbf{W},P}$ from P and using weights $\mathbf{W}$

▷ Predict from $G_{\mathbf{W},P}$ the solution $\hat{\mathbf{x}}$ of P

▷ Extract tasks ordering $O_{\hat{\mathbf{x}}}$ from inferred starting dates $\hat{\mathbf{x}}$

▷ Construct solution schedule $\mathbf{x}^\star$ by running SGS on $O_{\hat{\mathbf{x}}}$

return $\mathbf{x}^\star$

tion of the problem from the GNN prediction. Unfortunately, this proved inefficient since for many RCPSP instances, the total computation time of the GNN prediction followed by the CP solver feasibility request, exceeded that of the vanilla CP solver to get the same solution quality. The CP solver often could not figure how to slightly move the task starting dates in order to satisfy all the constraints, even if the GNN solution was of high quality with very little task overlapping.

We thus went for another strategy described in Algorithm 2 and in the bottom picture of Figure 2. Instead of calling CP to get a feasible schedule, we extract the tasks ordering from their starting dates, and pass it to the Serial SGS procedure described earlier in order to build a feasible schedule. Since the Serial SGS procedure is proven to build optimal schedules from tasks ordering corresponding to optimal schedules, we expect the schedules reconstructed from the nearly optimal GNN schedules to be of high quality.

Comparison of SIREN with Pure Search

Our test dataset consists of 408 random instances (the remaining 20% of instances) of varying complexity and size from the `kobe-scheduling`, all different from the training ones. The distribution of the training (resp. testing) sets per instance size in terms of number of tasks is the following: 23.7% (22.5%) instances of size 30, 23.5% (23.5%) of size 60, 22.8% (26.5%) of size 90, 30% (27.5%) of size 120.

SIREN vs CP

In this section we compare the performance of SIREN to CP in two different ways. First we want to demonstrate that SIREN is able to find good quality solutions in considerably less time than CP.

Figure 4 compares the inference time of SIREN to the runtime of CP when trying to achieve a similar makespan as the former. As is apparent, in more than 82% of problems CP takes more time than SIREN to achieve a solution of comparable quality. In over 40% of the cases, there is a considerable computational overhead ranging from 10 times up to over 20,000 times the computation time of SIREN. In contrast, in the cases where CP is superior, it is at most two times faster than SIREN.

Second, we compare the quality of SIREN's solutions to CP. In this experiment, CP was given a maximum computation time of 15 minutes. The results are shown in Figure 5. As expected, SIREN exhibits a measurable performance decay compared to CP. However, in the large majority of cases, it performs almost as good as CP, even returning optimal solutions (same makespan as the known kobe optimum reference) for 203 out of the 408 test instances. And on average, SIREN produces schedules with less than 5% longer makespans. Table 1 gives an overview over the results of both experiments.

SIREN vs Custom Heuristics

Our previous analysis showed that SIREN was on average faster than CP to get to a specific quality. However it reflects that SIREN is only doing fast inference and post-processing, leading to a feasible schedule before CP is able to produce even 1 feasible solution. It is therefore important to compare SIREN's schedule quality with that of other fast heuristics. We will compare with several heuristics:

- Dummy heuristic (DUM): it consists in running serial SGS with the priority list of task $[1, 2..., |T|]$
- Max Descendant Priority Rule (MDPR) from (Regnier-Coudert and Povéda 2021), a greedy heuristic scheduling first the available tasks that have the most descendants in the precedence tasks graph.
- Custom Critical Path Method (CCPM): it lexicographically orders tasks by the attributes (*latest start date*, *lat-*

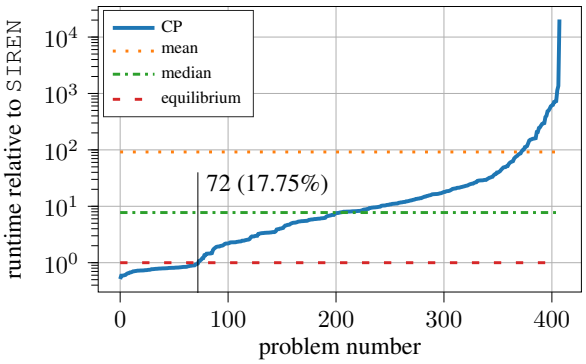

Figure 4: Runtime of vanilla CP relative to SIREN. Note that the problems have been ordered by the time it took CP to find a solution of similar quality to that of SIREN

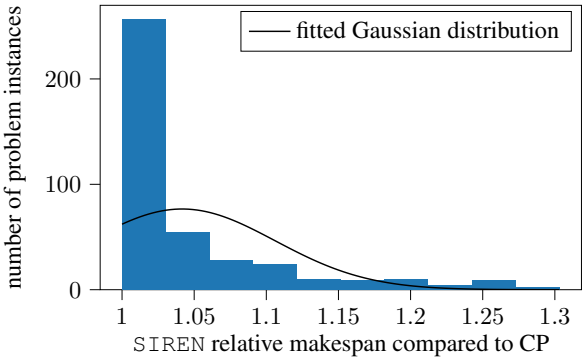

Figure 5: Performance of SIREN relative to the best CP solutions found within a timeout of 15 minutes. On average, SIREN performs 4.2% worse than CP

category	relative performance min	max	median	mean	std
inf. time CP vs SIREN	0.513	2.05e4	7.74	91.8	1.02e3
makespan SIREN vs CP	1.00	1.30	1.01	1.04	6.45e−2

Table 1: Summary of the data from Figures 4 and 5. For the first row, we measured the time it takes CP to find a solution of similar quality to that of SIREN. For the second row, CP was given a timeout of 15 minutes

algorithm	relative % overcost compared to best CP solution mean	std	25%	50%	75%	max
DUM	12.07	10.12	0.0	11.81	20.46	36.79
MDPR	7.72	7.22	0.0	6.59	12.70	36.29
CCPM	6.21	7.34	0.0	2.21	11.87	30.95
SIREN	4.16	6.45	0.0	0.93	5.73	30.32

Table 2: Statistics of relative overcost compared to best CP solutions on the 408 test instances. (% are for percentiles)

est start date − *earliest start date*), found by classical forward and backward recursion, and runs SGS with this prioritized list.

Table 2 shows the relative overcost of each of the methods compared to the best CP solution. The table shows that SIREN produces shorter schedules statistically on our test set, providing evidence that the GNN inference learnt a relatively good ordering of tasks. SIREN is therefore competitive with custom scheduling heuristics.

On-line Stochastic Optimisation with GNN Prediction and SGS Correction

The fast inference time of SIREN compared to the computation time of the vanilla CP solver allows us to integrate SIREN in an online stochastic scenario-based optimisation framework to solve RCPSPs with stochastic task durations, which we name SERENE for *Stochastic schEduling with gRaph nEural NEtworks*. Every time the execution of a task terminates, SERENE samples several deterministic future scenarios reflecting possible durations for the tasks still in play. These scenarios are each separately solved by SIREN, in order to help SERENE decide of the next best tasks to start and of their starting dates so as to minimize the expected makespan of the executed schedule.

SERENE is depicted in Figure 6 and described in Algorithm 3. It relies on two important bricks: the construction of a *"remaining"* stochastic RCPSP, and a double-pass sampling process which aims at approximating the best set of tasks to start next, without having to enumerate all the possible subsets of tasks that could be potentially started, as their number grows exponentially with the number of those tasks.

Stochastic RCPSP. A stochastic RCPSP is a tuple $(T, R, P, \mathbf{r}, \mathbf{C}, \mathbf{D})$ where T, R, P, $\mathbf{r}$ and $\mathbf{C}$ are defined as in deterministic RCPSPs and $\mathbf{D} = (\mathbb{D}_i)_{i \in T}$ where $\mathbb{D}_i$ is a probability distribution over the possible durations of task i. We assume that the actual duration d_i of task i is observed when the task ends, which implies that the probabilistic remaining duration of a running task evaluated at the current execution time, depends on how long this task has already been running. The objective is to minimize the *average* makespan

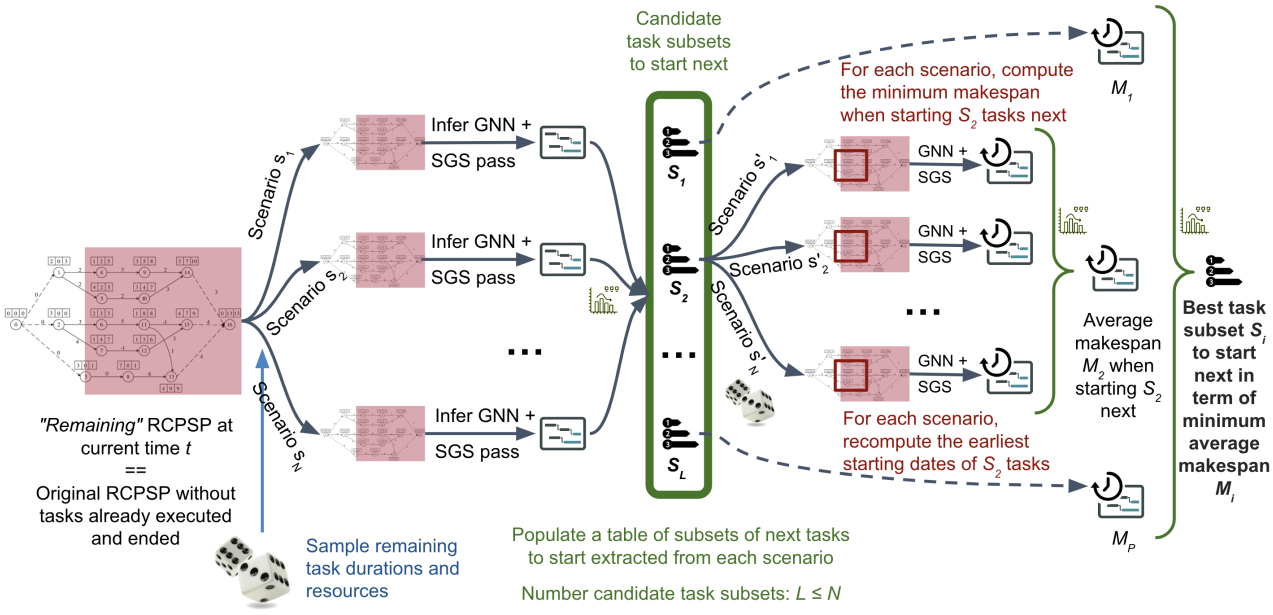

Figure 6: SERENE: double-pass on-line scenario-based optimisation scheme with GNN prediction and SGS correction

defined as $\max_{i \in T} \int_0^{+\infty} (x_i + d_i) d\mathbb{D}_i(d_i)$. Because of the stochastic duration of the tasks, it is impossible to decide of the starting dates x_i of the tasks upfront before the schedule execution: they have to be adapted online to the observed duration of the tasks that have already been executed.

Remaining stochastic RCPSP. Let's consider a current execution time t and a stochastic RCPSP $\mathcal{P} = (T, R, P, \mathbf{r}, \mathbf{C}, \mathbf{D})$. We define $T_L = \{i \in T : x_i \leqslant t\}$ as the set of tasks that have been launched before time t, $T_E = \{i \in T_L : x_i + d_i \leqslant t\}$ as the set of tasks that have already ended, and $T_R = \{i \in T_L : x_i + d_i > t\}$ as the set of tasks running at time t. Since the solver does not need to reason about the tasks that have already ended, we will represent all the tasks in T_E as an aggregated zero-duration task ε. While ε is not theoretically needed, we keep it in practice along with the precedence constraints from the tasks in T_E, that we now make originate from ε, to the tasks in $T \setminus T_L$ which remain to be executed. Indeed, all the RCPSP benchmarks in the `kobe-scheduling` repository, from which we have learned our GNN weights, have such a zero-duration "source" task, which is why we prefer maintaining this structure for inference.

The *remaining* stochastic RCPSP $\overline{\mathcal{P}}$ of $\mathcal{P}$ at the current time t is defined as the tuple $\overline{\mathcal{P}} = (\overline{T}, R, \overline{P}, \overline{\mathbf{r}}, \mathbf{C}, \overline{\mathbf{D}})$ where:

- $\overline{T} = \{\varepsilon\} \cup (T \setminus T_E)$;
- $\overline{P} \subseteq \overline{T} \times \overline{T}$ with $\overline{P} = \{(\varepsilon, j) : j \in T \setminus T_E, \exists i \in T_E \text{ s.t. } (i,j) \in P\} \cup \{(i,j) : i \in T \setminus T_E, j \in T \setminus T_E, (i,j) \in P\}$;
- $\overline{\mathbf{r}} = (\overline{r}_{i,k})_{i \in \overline{T}, k \in R}$ with $\overline{r}_{\varepsilon,k} = 0$ for all $k \in R$ and $\overline{r}_{i,k} = r_{i,k}$ for all $i \in \overline{T} \setminus \varepsilon$ and $k \in R$;
- $\overline{\mathbf{D}} = (\overline{\mathbb{D}}_i)_{i \in \overline{P}}$ with $\overline{\mathbb{D}}_\varepsilon = \delta_0$ and $\overline{\mathbb{D}}_i(d) = \frac{\mathbb{D}_i(d+t-x_i)}{\int_{t-x_i}^{+\infty} \mathbb{D}_i(\mu) d\mu}$ for all $i \in T_R$ and $\overline{\mathbb{D}}_i = \mathbb{D}_i$ for all $i \in T \setminus (T_E \cup T_R)$ where δ is the Dirac probability distribution.

Basically, the remaining RCPSP $\overline{\mathcal{P}}$ is obtained by replacing all launched and ended tasks in the original RCPSP $\mathcal{P}$ by a single zero-duration task ε, linking ε to the successor tasks of the ended tasks in $\mathcal{P}$, and updating the probability distribution on the duration of the running tasks according to the time elapsed since they started.

Double-pass sampling process. `SERENE`, detailed in Algorithm 3, builds the remaining stochastic RCPSP at the current execution time, then it samples a number of deterministic scenarios on which `SIREN` is called to evaluate the makespan and the best next tasks to start for each scenario. We then statistically compute the average makespan over all the scenarios for each set of next tasks to start in order to select the set with the lowest average makespan. Note that `SIREN` can be replaced in this framework with any other deterministic RCPSP solver, especially CP-SAT, which we will exploit in the experiments later in order to compare the benefit of using GNNs over the vanilla CP-SAT solver when solving online stochastic RCPSPs.

However, contrary to standard online scenario-based optimisation algorithms for *sequential* probabilistic planning problems – sometimes named Optimisation in Hindsight or HOP for short (Yoon et al. 2008) – we cannot enumerate all the possible next actions to consider at the current time before statistically evaluating each of them on the set of sampled scenarios. The reason is that, in scheduling, many different tasks can be potentially started simultaneously, which leads to a number of task subsets to consider for starting next which is exponential in the number of tasks that can start given the constraints. In order to overcome this combinatorial explosion issue, we propose a double-pass sampling approach (see Algorithm 3 for details). First, we sample different scenarios from which we extract the subsets of tasks to start next found by `SIREN`, and we insert them in a set $\mathcal{C}$ of candidate task subsets. This first phase allows us to extract the most promising subsets of tasks to start without enumerating all the possible combinations. Second, as in traditional HOP frameworks, we evaluate each subset $\mathcal{S}_C$ of tasks to start one by one by sampling different scenarios on which `SIREN` is run. Contrary to the first pass, we force `SIREN` to start the tasks in $\mathcal{S}_C$ but we let it decide freely of the starting dates of the remaining tasks. We then

Algorithm 3: The on-line stochastic `SERENE` algorithm

Input: Current time t; Set T_L of launched tasks so far; Original stochastic RCPSP $\mathcal{P}$; GNN scheduler $\mathcal{G}$; Number of scenario samples N

Output: Best set of tasks to start next and their starting date

▷ Compute the set T_E of ended tasks at time t from T_L
▷ Compute the set T_R of running tasks at time t from T_L
▷ Compute remaining RCPSP $\overline{\mathcal{P}}$ from t, T_E, T_R and $\mathcal{P}$
▷ Create empty set $\mathcal{C}$ of candidate subsets of tasks to start
for all scenario $0 \leqslant i < N$ **do**
  ▷ Sample deterministic RCPSP scenario $\mathcal{P}_i$ by sampling task durations from $\overline{\mathcal{P}}$
  ▷ Call `SIREN` on $\mathcal{P}_i$ and add best tasks $\mathcal{S}_i$ to start to $\mathcal{C}$
end for
▷ Create a table M that maps candidate task subsets $\mathcal{S}_C \in \mathcal{C}$ to the average makespan value obtained when forcing the tasks in $\mathcal{S}_C$ to start next
▷ Create a table D that maps candidate task subsets $\mathcal{S}_C \in \mathcal{C}$ to the list of earliest possible starting dates of tasks in $\mathcal{S}_C$ when forced to start next
for all Candidate task subset $\mathcal{S}_C \in \mathcal{C}$ **do**
  ▷ $M[\mathcal{S}_C] \leftarrow 0$
  for all Scenario $0 \leqslant j < N$ **do**
    ▷ Sample deterministic RCPSP scenario $\mathcal{P}_j$ by sampling task durations from $\overline{\mathcal{P}}$
    ▷ Compute the earliest starting dates of tasks in $\mathcal{S}_C$ when forced to start next and record them in $D[\mathcal{S}_C]$
    ▷ Call `SIREN` on $\mathcal{P}_j$ by forcing the SGS subprocedure to start tasks in $\mathcal{S}_C$ at the dates stored in $D[\mathcal{S}_C]$ and update $M[\mathcal{S}_C]$ online as the obtained makespan M_j arises: $M[\mathcal{S}_C] \leftarrow \frac{j \cdot M[\mathcal{S}_C] + M_j}{j+1}$
  end for
end for
▷ Best next tasks to start : $\mathcal{S}_{C^\star} \leftarrow \text{argmin}_{\mathcal{S}_C \in \mathcal{C}} M[\mathcal{S}_C]$
return $\mathcal{S}_{C^\star}$ and average starting date $\text{mean}(D[\mathcal{S}_{C^\star}])$

average the makespan obtained by SIREN over all the scenarios, which provides a statistical estimate of the average achievable makespan when we start the tasks in $\mathcal{S}_C$ next.

We should note here that, due to the stochastic remaining duration of the running tasks, the starting date of the next tasks to start depends on the sampled scenario. Moreover, while the starting dates of the tasks of a given subset $\mathcal{S}_C$ were all the same after the first sampling pass, it does not need to be the case when we re-sample the scenarios in the second pass. This is why we need to recompute the earliest starting dates of the tasks in $\mathcal{S}_C$ when we evaluate a given scenario in the second pass. This can be done with a procedure not detailed in this paper but which is similar to the insertion in the schedule of a task from the priority list in SGS. Finally, since the starting dates of the tasks in the best subset $\mathcal{S}_{C^*}$ can be all different for the same reason, we return as best expected starting date the average starting date over all the tasks in $\mathcal{S}_{C^*}$. If this value is 0, we start all the tasks in $\mathcal{S}_{C^*}$ now, otherwise we will call SERENE again when one of the currently running tasks finishes, informing in the meantime the user that we will *probably* start the next tasks in $\mathcal{S}_{C^*}$ at their computed average starting date.

Empirical Evaluation of SERENE

We now compare SERENE to several other baselines:

- SIREN in reactive mode using the average scenario (*SIREN-REACTIVE_AVG*)
- SIREN in reactive mode using the pessimistic scenario (*SIREN-REACTIVE_WORST*)
- SIREN in reactive mode using the optimistic scenario (*SIREN-REACTIVE_BEST*)
- CP-SAT solver in reactive mode using the average scenario (*CPSAT-REACTIVE_AVG*). Each CP call has a timeout of 0.5 sec.
- SERENE using CP-SAT in place of SIREN to solve sampled scenarios. (*CPSAT-HINDSIGHT_DBP*). Each CP call has a timeout of 0.2 sec.
- *Foresight* : CP solver that knows the ground truth scenario in advance. This method will play as a theoretical upper bound.

SERENE and *CPSAT-HINDSIGHT_DBP* are sampling $N = 30$ sampled scenarios at each rescheduling steps. All reactive methods recompute new schedules at each step of execution, by solving one future scenario for the remaining stochastic RCPSP $\overline{\mathcal{P}}$. In the average, pessimistic, and optimistic scenarios, task durations take their expected, maximum, and minimum value, respectively.

The evaluation focuses on 37 base instances of the kobe dataset that are the hardest for CP-SAT. For these problems the CP solver took more than 0.5 seconds to produce solutions as good as SIREN. Then 10 deterministic ground truth scenarios per base stochastic instance are sampled to evaluate our different methods. Therefore we end up having 370 experimental runs. Each stochastic instance corresponds to one of the 37 aforementioned base deterministic instances from the kobe dataset, where we replaced the deterministic task durations of the original instances by uniform probability distributions centered around the original durations. The support of the uniform distributions has a maximum deviation of 8 on both sides of the original durations, which represents 80% of the original duration of the longest tasks in the dataset. We truncate the support of the distributions whenever necessary in order to ensure to sample positive task durations.

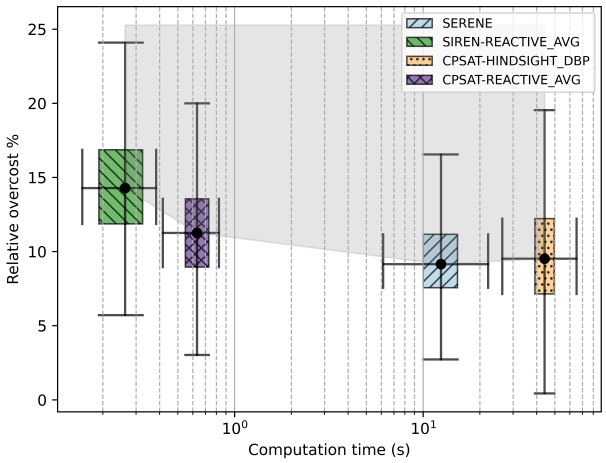

Figure 7: Relative overcost (%) and mean computation time per reschedule (s) joint-boxplot per methods

Figure 7 represents the tradeoff between the computation time of each method (x-axis) and its overcost relative to *Foresight* (y-axis). For visualisation purposes we only plot one *SIREN-REACTIVE* version. The three methods *AVG*, *WORST*, *BEST* indeed have very similar statistics with $\approx 2.5 \cdot 10^{-2}$ sec median computation time for all methods, and respectively 14.3, 15.9, 16.3% as median overcosts. Two algorithms have lower overcost than the others: SERENE and *CPSAT-HINDSIGHT_DBP*, which empirically get results of similar quality. This demonstrates the advantages of hindsight optimisation over (*REACTIVE*) mono-scenario re-optimisation. The variance of the overcost of SERENE is however lower ($\sigma = 3.0$) than that of *CPSAT-HINDSIGHT_DBP* ($\sigma = 4.11$). SERENE therefore has a more "predictable" performance than the CPSAT version, and it also has smaller overcost values: e.g $p_{90\%} = 13.1$ versus $p_{90\%} = 15.1$ for the 90^{th} percentile. Additionally to difference in quality, each method has different computation times at all rescheduling steps. Figure 7 highlights that reactive methods are (as expected) the fastest, and that SERENE is around 4 times faster than *CPSAT-HINDSIGHT_DBP* on average ($\approx$ 12s compared to $\approx$ 45s). SERENE is therefore the preferred method in this experiment, as it offers the best compromise between computation time and quality.

Conclusion

To our knowledge, we propose in this paper one of the first successful attempts to learn to solve any-size Resource-Constrained Project Scheduling Problems, including in pres-

ence of uncertain task duration. Our method relies on a GNN that learns task starting dates, from which task priorities are extracted and passed to the SGS procedure in order to produce a feasible schedule of high quality. Uncertain task durations are handled by a double-pass scenario-based sampling approach which allows us to greedily compute the best subset of tasks to start next in real time, without requiring to enumerate all the possible combinations of such subsets. Our results show that we can infer schedules on unseen instances of higher quality than those produced by CP within the same computation time budget, in both deterministic and stochastic settings.

This work can be extended in several interesting directions. In the short term, we would like to extend the framework to more complex variants of RCPSPs of particular interest in many real applications, e.g. Multi-Skill Preemptive RCPSPs. This will require us to adapt the GNN architecture to the additional constraint relations of those extended problem classes. Secondly, we used SGS as a schedule repair method. To obtain solutions of higher quality, an interesting alternative would be large neighborhood search. Finally, another interesting direction is to use a similar GNN architecture to learn branching heuristics to guide the search of CP algorithms.

Acknowledgements

We thank the anonymous reviewers for their useful comments which helped to improve the paper. This work has received funding from the European Union's Horizon Europe Research and Innovation program under the grant agreement TUPLES No 101070149.
F. Teichteil-Königsbuch, G. Povéda and G. González de Garibay were additionally supported by Airbus R&T, and S. Thiébaux by ARC grant DP220103815.

References

Applegate, D.; Bixby, R.; Chvátal, V.; and Cook, W. 1995. Finding cuts in the TSP (a preliminary report). Technical Report 05, DIMACS.

Artigues, C.; Demassey, S.; and Neron, E. 2008. *Resource-constrained project scheduling*. Wiley Online Library.

Artigues, C.; Michelon, P.; and Reusser, S. 2003. Insertion techniques for static and dynamic resource-constrained project scheduling. *European Journal of Operational Research*, 149(2): 249–267.

Ayodele, M.; McCall, J.; and Regnier-Coudert, O. 2017. Estimation of distribution algorithms for the multi-mode resource constrained project scheduling problem. In *Proc. IEEE Congress on Evolutionary Computation (CEC)*.

Bengio, Y.; Lodi, A.; and Prouvost, A. 2021. Machine learning for combinatorial optimization: A methodological tour d'horizon. *Eur. J. Oper. Res.*, 290(2): 405–421.

Bronstein, M. M.; Bruna, J.; Cohen, T.; and Veličković, P. 2021. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges. *ArXiv*, abs/2104.13478.

Chong, E.; Givan, R.; and Chang, H. S. 2000. A framework for simulation-based network control via hindsight optimization. In *Proc. IEEE Conference on Decision and Control (CDC)*, 1433–1438.

Dai, H.; Dai, B.; and Song, L. 2016. Discriminative Embeddings of Latent Variable Models for Structured Data. In *Proc. International Conference on Machine Learning (ICML)*, volume 48 of *Proceedings of Machine Learning Research*, 2702–2711. PMLR.

Deudon, M.; Cournut, P.; Lacoste, A.; Adulyasak, Y.; and Rousseau, L.-M. 2018. Learning Heuristics for the TSP by Policy Gradient. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 170–181. Springer International Publishing.

Fan, W.; Ma, Y.; Li, Q.; He, Y.; Zhao, E.; Tang, J.; and Yin, D. 2019. Graph Neural Networks for Social Recommendation. In *Proc. World Wide Web Conference (WWW)*, 417–426. New York, NY, USA. ISBN 9781450366748.

Fey, M.; and Lenssen, J. E. 2019. Fast Graph Representation Learning with PyTorch Geometric. *CoRR*, abs/1903.02428.

Ganian, R.; Hamm, T.; and Mescoff, G. 2020. The Complexity Landscape of Resource-Constrained Scheduling. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, 1741–1747.

Gasse, M.; Chételat, D.; Ferroni, N.; Charlin, L.; and Lodi, A. 2019. Exact Combinatorial Optimization with Graph Convolutional Neural Networks. In *Proc. Annual Conference on Neural Information Processing Systems (NeurIPS)*, 15554–15566.

Hartmann, S.; and Kolisch, R. 2000. Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. *Eur. J. Oper. Res.*, 127: 394–407.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Deep Residual Learning for Image Recognition. *CoRR*, abs/1512.03385.

Joshi, C. K.; Cappart, Q.; Rousseau, L.-M.; and Laurent, T. 2022. Learning the travelling salesperson problem requires rethinking generalization. *Constraints*, 27(1-2): 70–98.

Khalil, E. B.; Dai, H.; Zhang, Y.; Dilkina, B.; and Song, L. 2017. Learning Combinatorial Optimization Algorithms over Graphs. In *Proc. Annual Conference on Neural Information Processing Systems (NeurIPS)*, 6348–6358.

Khalil, E. B.; Morris, C.; and Lodi, A. 2022. MIP-GNN: A Data-Driven Framework for Guiding Combinatorial Solvers. In *Proc. AAAI Conference on Artificial Intelligence (AAAI)*, 10219–10227.

Kolisch, R. 1994. Serial and parallel resource-constrained projekt scheduling methodes revisited: Theory and computation. Technical Report 344, Universität Kiel, Institut für Betriebswirtschaftslehre; ZBW – Leibniz Information Centre for Economics.

Kolisch, R.; and Sprecher, A. 1997. PSPLIB - A project scheduling problem library: OR Software - ORSEP Operations Research Software Exchange Program. *European Journal of Operational Research*, 96(1): 205–216.

Kurin, V.; Godil, S.; Whiteson, S.; and Catanzaro, B. 2020. Can Q-Learning with Graph Networks Learn a Generalizable Branching Heuristic for a SAT Solver? In *Proc. An-*

nual Conference on Neural Information Processing Systems (NeurIPS), 9608–9621.

Laborie, P. 2018. An Update on the Comparison of MIP, CP and Hybrid Approaches for Mixed Resource Allocation and Scheduling. In *Proc. International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR)*, volume 10848 of *Lecture Notes in Computer Science*, 403–411. Springer.

Mercado, R.; Rastemo, T.; Lindelöf, E.; Klambauer, G.; Engkvist, O.; Chen, H.; and Bjerrum, E. J. 2021. Graph networks for molecular design. *Machine Learning: Science and Technology*, 2(2): 025023.

Nair, V.; Bartunov, S.; Gimeno, F.; von Glehn, I.; Lichocki, P.; Lobov, I.; O'Donoghue, B.; Sonnerat, N.; Tjandraatmadja, C.; Wang, P.; Addanki, R.; Hapuarachchi, T.; Keck, T.; Keeling, J.; Kohli, P.; Ktena, I.; Li, Y.; Vinyals, O.; and Zwols, Y. 2020. Solving Mixed Integer Programs Using Neural Networks. *CoRR*, abs/2012.13349.

Özdamar, L.; and Ulusoy, G. 1995. A survey on the resource-constrained project scheduling problem. *IIE Transactions*, 27: 574–586.

Park, J.; Chun, J.; Kim, S. H.; Kim, Y.; and Park, J. 2021. Learning to schedule job-shop problems: representation and policy learning using graph neural network and reinforcement learning. *International Journal of Production Research*, 59(11): 3360–3377.

Pellerin, R.; Perrier, N.; and Berthaut, F. 2020. A survey of hybrid metaheuristics for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 280(2): 395–416.

Perron, L.; and Furnon, V. 2022. OR-Tools v9.4. https://developers.google.com/optimization/. Accessed: 2023-04-06.

Regnier-Coudert, O.; and Povéda, G. 2021. An Empirical Evaluation of Permutation-Based Policies for Stochastic RCPSP. In *Proc. GECCO Genetic and Evolutionary Computation Conference Companion*, GECCO '21, 1451–1458.

Schutt, A.; Feydy, T.; Stuckey, P. J.; and Wallace, M. G. 2013. Solving RCPSP/max by lazy clause generation. *J. Sched.*, 16(3): 273–289.

Shen, W.; Trevizan, F. W.; and Thiébaux, S. 2020. Learning Domain-Independent Planning Heuristics with Hypergraph Networks. In *Proc. International Conference on Automated Planning and Scheduling (ICAPS)*, 574–584.

Shi, Y.; Huang, Z.; Feng, S.; Zhong, H.; Wang, W.; and Sun, Y. 2021. Masked Label Prediction: Unified Message Passing Model for Semi-Supervised Classification. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, 1548–1554.

Song, W.; Chen, X.; Li, Q.; and Cao, Z. 2022. Flexible Job Shop Scheduling via Graph Neural Network and Deep Reinforcement Learning. *IEEE Transactions on Industrial Informatics*, 1–11.

Van Peteghem, V.; and Vanhoucke, M. 2014. An experimental investigation of metaheuristics for the multi-mode resource-constrained project scheduling problem on new dataset instances. *European Journal of Operational Research*, 235(1): 62–72.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention is All you Need. In *Proc. Annual Conference on Neural Information Processing Systems (NeurIPS)*, 5998–6008.

Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; and Yu, P. S. 2021. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1): 4–24.

Yoon, S.; Fern, A.; Givan, R.; and Kambhampati, S. 2008. Probabilistic Planning via Determinization in Hindsight. In *Proc. AAAI Conference on Artificial Intelligence (AAAI)*, volume 2, 1010–1016.