Ikonen, Teemu J.; Heljanko, Keijo; Harjunkoski, Iiro

Reinforcement learning of adaptive online rescheduling timing and computing time allocation

Published in:
Computers and Chemical Engineering

DOI:
10.1016/j.compchemeng.2020.106994

Published: 04/10/2020

Document Version
Peer-reviewed accepted author manuscript, also known as Final accepted manuscript or Post-print

Reinforcement learning of adaptive online rescheduling timing and computing time allocation

Teemu J. Ikonen[a], Keijo Heljanko[b,c], Iiro Harjunkoski[a,d,*]

[a]*Aalto University, Department of Chemical and Metallurgical Engineering, PO Box 16100, 00076 Aalto, Finland*
[b]*University of Helsinki, Department of Computer Science, PO Box 68, 00014 University of Helsinki, Finland*
[c]*Helsinki Institute for Information Technology (HIIT), Helsinki, Finland*
[d]*ABB Power Grids Research, Kallstadter Str. 1, 68309 Mannheim, Germany*

Abstract

Mathematical optimization methods have been developed to a vast variety of complex problems in the field of process systems engineering (e.g., the scheduling of chemical batch processes). However, the use of these methods in online scheduling is hindered by the stochastic nature of the processes and prohibitively long solution times when optimized over long time horizons. The following questions are raised: When to trigger a rescheduling, how much computing resources to allocate, what optimization strategy to use, and how far ahead to schedule? We propose an approach where a reinforcement learning agent is trained to make the first two decisions (i.e., rescheduling timing and computing time allocation). Using neuroevolution of augmenting topologies (NEAT) as the reinforcement learning algorithm, the approach yields, on average, better closed-loop solutions than conventional rescheduling methods on three out of four studied routing problems. We also reflect on expanding the agent's decision-making to all four decisions.

Keywords: online scheduling, rescheduling procedures, reinforcement learning, decision-making, timing, computing resource allocation

1. Introduction

The operation of a large-scale production plant or a supply chain involves an enormous number of real-time decisions. This decision-making, made in limited time and often with limited information, affects not only the profitability of the system but also its environmental impact via the consumption of electricity, fuel, and raw materials, and the rate of equipment degradation. The development of technologies for the optimal operation of such systems belongs to the field of process systems engineering (PSE). During the past 50 years, mathematical optimization techniques have been developed and extensively applied to problems emerging in PSE (Grossmann & Harjunkoski, 2019). The most significant of these methods are mathematical programming and heuristic optimization algorithms. Today, these techniques are valuable analytical decision-making tools for system operators, capable of handling significantly more decision variables than a human while providing the solution with proven global optimality.

However, there are limitations on the applicability of mathematical optimization techniques. First, the length of the time horizon, for which the optimal solution can be formed, is limited. The reason is that the longer the time horizon, the larger is the number of decision variables and, consequently, the greater the required computing capacity. The iterative procedure of solving scheduling problems in subsequent, and typically partly overlaid, time horizons is referred to as the *rolling horizon approach* (Harjunkoski et al., 2014). Second, industrial processes are inherently stochastic. As processes are subject to changes and disturbances, the longer is the time allocated to an optimization procedure, the weaker is the correspondence of the optimized solution to the reality. These limitations raise two main questions: 1) When should

*Corresponding author. E-mail address: iiro.harjunkoski@de.abb.com (I. Harjunkoski) rescheduling procedures be started and 2) what is the optimal length of the time horizon in each operating situation?

While in the industry such decisions are often made by (human) operators, the methods reported in the literature for the timing of rescheduling procedures can be categorized into periodic (e.g., Gupta et al. (2016), Dong et al. (2018)) and event-triggered (e.g., Touretzky et al. (2017), Katragjini et al. (2013)) rescheduling, as well as their hybrids (e.g., Pattison et al. (2017)). In periodic methods, optimization procedures are repeatedly performed at a predefined frequency. In event-triggered methods, a new optimization procedure is started when new information is obtained. In this context, the integration, or at least interfacing, of scheduling and process control is important, in order to communicate the information of process level disturbances to the scheduling level (Baldea & Harjunkoski, 2014). Subramanian et al. (2012) proposed a state-space model for production scheduling, which enhances the representation of the current process state at the scheduling level.

Gupta & Maravelias (2016) report that, based on their results, minimum values exist for both the rescheduling frequency and the scheduling horizon length, below which the quality of the generated schedules is significantly compromised. Nevertheless, in the literature, the influence of the scheduling horizon length on the goodness of the solutions is inadequately studied and, to our knowledge, no adaptive methods have been reported for this purpose. It is also worth mentioning that certain fields of optimization methods, such as stochastic programming (Birge & Louveaux, 2011) and robust optimization (Ben-Tal et al., 2009), enable the anticipation of uncertainties in optimization parameters. However, the required computing capacity of these methods is considerably higher than that of deterministic methods (discussed above). For this reason, when using deterministic methods, rescheduling procedures can be performed more frequently, and with longer time horizons, than with stochastic optimization methods that anticipate uncertainty (Subramanian et al., 2012; Harjunkoski et al., 2014).

In an earlier conference contribution (Ikonen & Harjunkoski, 2019), we proposed a framework, where a reinforcement learning (RL) agent (Sutton & Barto, 2018) makes the following decisions on rescheduling procedures:

1. The timing of rescheduling procedures;
2. The allocated computing time;
3. The optimization strategy (mathematical programming or a heuristic algorithm); and
4. The length of the scheduling horizon.

The RL agent is trained in a simulated operating environment. Decisions 1, 3 and 4 are among the critical design aspects for online scheduling methods outlined by Gupta et al. (2016). Regarding Decision 3, heuristic algorithms are rule-based methods that can complement mathematical programming. Out of the two, mathematical programming can, in general, find a better solution and prove the global optimality of the solution. On the other hand, heuristic algorithms are often computationally lighter than mathematical programming. For these reasons, arguably, keeping both methods in the 'repertoire' of the agent enhances its ability to react to changes in the operating environment.

RL is one of the three main categories of machine learning, along with supervised and unsupervised learning. RL has a significant potential to automate and enhance decision-making traditionally made by humans. In RL, an active goal-seeking agent interacts with a passive environment. Following its action(s), the agent receives feedback from the environment, including the updated state of the environment and the reward of the action(s) (Fig. 1).

A recent example of the effectiveness of RL is the AlphaZero algorithm by DeepMind that can achieve, via self-play, higher ELO rating[1] than Stockfish 8 in chess and a superhuman level in two other board games shogi and Go (Silver et al., 2018). The reason for the recent development in RL is in the deep neural networks, which allow high-dimensional state and action spaces. Deep RL algorithms, such as deep Q-networks (DQN) (Mnih et al., 2015) and trust region policy optimization (TRPO) (Schulman et al., 2015), have been successfully applied to large-scale decision-making problems that were intractable in the past.

[1]The ELO rating, often used in chess, describes the skill level of a player relative to other players.

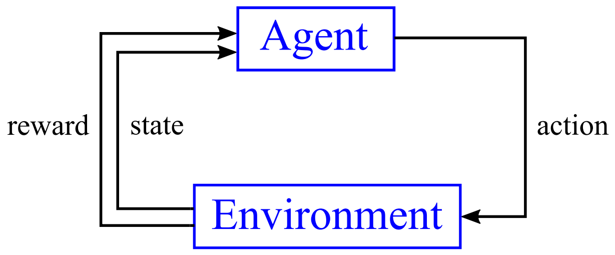


Figure 1: A reinforcement learning agent interacting with the environment at time t.

Another approach to RL is to train the agents by neuroevolution algorithms, such as neuroevolution of augmenting topologies (NEAT) (Stanley & Miikkulainen, 2002) and hypercube-based NEAT (HyperNEAT) (Stanley et al., 2009). The fundamental difference between deep RL and neuroevolution is that the former is based on the gradient-based backpropagation method (which is used to update the node weights) and a single agent, whereas the latter is based on evolutionary optimization and a population of agents. A key feature of neuroevolution is that the evolution process can be initiated from a very simple neural network topology, the complexity of which then incrementally increases during the evolution. The feature reduces unnecessary complexity of the final neural network. Despite a general belief that gradient-based optimization methods are more efficient than those based on evolutionary algorithms, Such et al. (2017) showed that neuroevolution is a competitive alternative for the deep RL methods. For extensive reviews of deep RL and neuroevolution, the reader may wish to consult papers by Arulkumaran et al. (2017) and Stanley et al. (2019), respectively.

Both deep RL and neuroevolution methods have been successfully used to train agents to play Atari 2600 video games (Mnih et al., 2013; Hausknecht et al., 2014), in which the controls are to move a joystick with two degrees of freedom and to press a button. In decision-making related to rescheduling procedures, the ability of the agent to influence the environment is at a somewhat similar level of complexity as in the game mentioned above. What is critical is the timing of these actions.

Regarding scheduling, RL has been used to train an agent to explicitly decide scheduling actions in a real or simulated environment (Šemrov et al., 2016; Atallah et al., 2018), to repair outdated schedules (Palombarini & Martinez, 2012), as well as to learn dispatching rules suitable for a specific scheduling environment (Priore et al., 2014; Aydin & Öztemel, 2000). Recently, Shin et al. (2019) reviewed the implications of reinforcement learning to process control, and highlighted the development of a hierarchical modular structure, in which RL operates at the higher level and mathematical optimization at the lower level, as one of the future research directions. Shin & Lee (2019) then proposed a modeling framework for efficient integration of multi-timescale decisions, which combines mathematical programming and RL.

While the earlier applications of RL to scheduling optimization have mainly aimed at learning explicit scheduling decisions or suitable dispatching rules for a certain operating environment, the framework proposed in Ikonen & Harjunkoski (2019) operates at a higher level. It utilizes the strengths of mathematical programming and heuristic algorithms at finding the optimal, or a nearly optimal, solution (a result of significant research effort during the past 50 years), and focuses on the decision-making related to the optimization procedures in a changing operating environment.

In this paper, we propose an approach where a RL agent is trained to make the first two decisions of the framework (i.e., the rescheduling timing and computing time allocation). As the RL algorithm, we use NEAT[2] because of its desirable feature of yielding simple neural network topologies, facilitating the interpretation of the final solution. A description of the algorithm is given in Section 4.1.

The structure of this paper is the following. In Section 2, we further elaborate the original framework involving four decisions on the rescheduling procedures. In Section 3, we present a set of simple dynamic routing problems, suitable to serve as test cases for the approach. In Section 4, we train RL agents to decide

[2]Regarding NEAT, our contributions do not involve any improvements or modifications to the algorithm. We use the implementation of the algorithm by McIntyre et al. (2019).

the *rescheduling timing* and the *allocated computing time* on these routing problems. Thus, the decisions on the *optimization strategy* and the *scheduling horizon length* are not involved in the test cases, but are topics of the future work.

2. Elaborating the framework proposed in Ikonen & Harjunkoski (2019)

In the proposed framework, the RL agent makes the four rescheduling decisions (see the introduction), which only affect the execution of the rescheduling procedures, and not the process itself. Therefore, the process and the *optimizer*, which is the module that performs the rescheduling procedures, together form the environment for the RL agent (Fig. 2). The optimizer may use either mathematical programming or a heuristic algorithm, and has access to predetermined computing resources. The scheduling decisions, determined by the optimizer, control the operation of the process. Vice versa, new information of the process is updated to the optimizer so that rescheduling procedures are performed with the latest information of the process parameters[3].

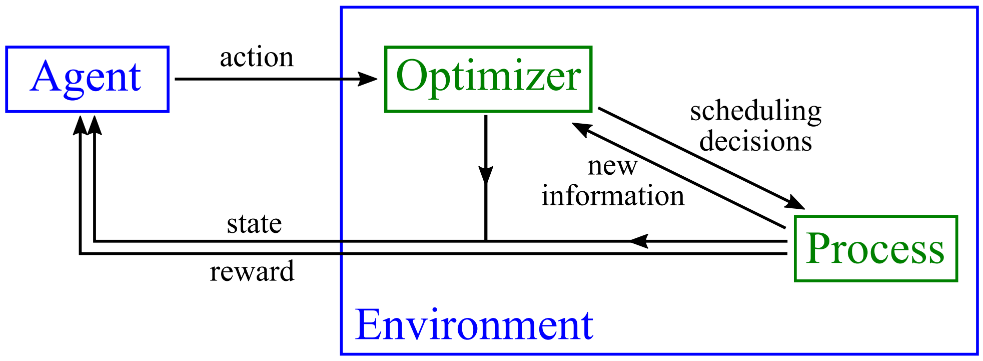


Figure 2: The proposed approach, in which the optimizer and the process together form the environment for a RL agent.

Figure 3 shows generic action and state spaces of the agent. Each signal in these spaces may receive a value in the range of $[0, 1]$ The agent decides the actions at a frequency f_{a}, significantly lower than a typical rescheduling frequency. The four rescheduling decisions, stated above, are encoded into five action signals. The first three define whether a rescheduling procedure is executed, and if positive, whether the optimizer is commanded to use mathematical programming or a heuristic algorithm (Decisions 1 and 3). The action to be executed is that of the three signals receiving the highest value. The fourth and fifth signal determine the allocated computing time (Decision 2) and horizon length (Decision 4), respectively. For these decisions, the signal value is mapped into a predefined range of allowable values.

We divide the state space signals into three categories. Category 1 signals describe the deviations in the optimization parameters that have occurred in the process after the initiation of the previous rescheduling procedure. Examples of these are deviations in processing times or material yields. A signal value of less than 0.5 indicates a decrease and a value of more than 0.5 an increase in the parameter value. Category 2 signals describe discrete changes in the process environment after the initiation of the previous rescheduling procedures. Examples of these are new orders and equipment breakdowns[4]. Such changes may be significant for the operation of the process. Considering the examples, the n most urgent orders can be associated with state signals, such that the signal value indicates the due date of the order with respect to the current scheduling horizon. If equipment breakdowns are considered, each equipment can be associated with a

[3]Thus, the interaction between the optimizer and the process follows a standard procedure used, for example, in periodic and event-triggered rescheduling.

[4]An equipment breakdown means an unexpected occurrence of a technical problem, which stops the operation of the equipment.

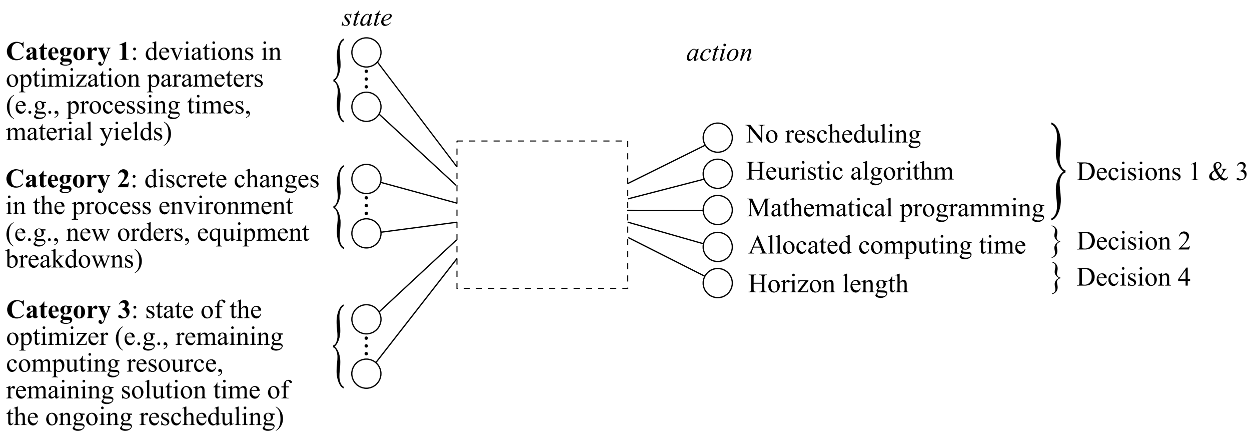

state
Category 1: deviations in optimization parameters (e.g., processing times, material yields)
Category 2: discrete changes in the process environment (e.g., new orders, equipment breakdowns)
Category 3: state of the optimizer (e.g., remaining computing resource, remaining solution time of the ongoing rescheduling)
action
No rescheduling
Heuristic algorithm
Mathematical programming
Allocated computing time
Horizon length
Decisions 1 & 3
Decision 2
Decision 4

Figure 3: State and actions spaces of the RL agent, making the rescheduling decision. Regarding the first three action signals (Decisions 1 & 3), the one receiving the highest value is executed.

signal, which receives the value of 1 if the state of the unit (functioning or failed) is the same as at the initiation of the previous rescheduling, and the value of 0 if opposite. Category 3 signals describe the state of the optimizer. Examples of quantities in this category are the computing resources remaining, the remaining time allocated for the ongoing rescheduling procedure, and the current optimality gap (the last one applies only to mathematical programming).

The framework expands the scope of conventional rescheduling methods, i.e. periodic and event-triggered rescheduling, and their hybrids. Instead of only deciding the timing of rescheduling, it also decides the allocated computing time, the solution algorithm, and the horizon length.

3. Dynamic routing problem

In this section, we describe the dynamic routing problem, suitable to serve as a simple case study for the proposed approach with two decisions. As we have stated in Section 2, in our approach the RL environment consists of both the process and the optimizer. Therefore, we define here an optimization problem where information is also obtained during the timespan of the problem (i.e., the process), as well as an optimization method and an available computing budget (i.e. the optimizer).

As the optimization problem, let us consider a square region, having dimensions 1000×1000 m, and a vehicle traveling in the region at a constant speed of $v = 10$ m/s. The purpose of the vehicle is to visit $n_{\text{init}} + n_{\text{new}}$ sites in the region before site-specific due dates $t_{\text{due}} \in \{0, 1 \ldots t_{\text{span}}\}$, where t_{span} is the timespan of the problem. Each site has an order date t_{ord}, at which the vehicle receives the location and due date of the site. The order dates of n_{init} sites are known at the beginning of the timespan ($t_{\text{ord}} = 0$), and those of n_{new} sites are received between the start of the timespan and the due date ($t_{\text{ord}} \in \{0, 1 \ldots t_{\text{due}}\}$). At $t = 0$, the vehicle is at site 0. The objective of the optimization problem is to minimize the delay sum of visiting the $n_{\text{init}} + n_{\text{new}}$ sites. Since scheduling decisions must be made with limited information, finding the optimal solution to the problem requires rescheduling.

As the optimization method, we use ant colony optimization (ACO) (Dorigo & Gambardella, 1997), which is a probabilistic metaheuristic search method of finding good paths in a graph. In the method, ants communicate by laying pheromone on paths between nodes. The laid pheromone concentration of an ant is proportional to the objective function value of the route the ant traveled. When a new ant is at node i, it chooses the path to node j with the probability

$$p_{ij} = \frac{\phi_{ij}^{\alpha} d_{ij}^{\beta}}{\sum_{i,j=1}^{n} \phi_{ij}^{\alpha} d_{ij}^{\beta}}, \tag{1}$$

where ϕ_{ij} and d_{ij} are the pheromone level and desirability[5] of the path (i, j), respectively, and α and β are influence parameters[6]. We use a population size of 200. We have implement ACO using the Python module ACOpy (Grant, 2018). In all experiments of this paper, the computing budget for the rescheduling procedures during the timespan is restricted to 50 CPU seconds. Each rescheduling procedure is associated with information time t_{info} and execution time t_{exe}, the difference of which is the allocated computing time t_{comp}. The rescheduling is conducted using the information available at t_{info}, whereas t_{exe} is the planned start time of the new schedule.

Ideally, the vehicle should receive an updated route before reaching the end of the previously scheduled route. However, this may not always happen. As an example, in periodic rescheduling, such incidents would occur if the horizon length is defined to be shorter than the rescheduling interval. Therefore, we define that, if the vehicle arrives at the end of the scheduled route, it starts to follow the route of a greedy algorithm, in which it proceeds to the closest site that has not yet been visited, until it receives a new optimized route. If all ordered sites are visited, the vehicle is on standby at the last scheduled site until the end of the timespan t_{span}. We generate the initial route at $t = 0$ also by the greedy algorithm.

In the following two sections, we present an introductory example (Section 3.1) of this dynamic routing problem, as well as larger test cases (Section 3.2), to which we will later apply the RL-based approach.

3.1. Introductory example with 9 sites

The introductory example involves $n_{\text{ord}} = 7$ sites, the orders (i.e., the location and the due date) of which are known at $t = 0$, and $n_{\text{new}} = 2$ sites, the orders of which are obtained during the timespan (Table 1). The timespan of the problem is $t = 500$ s. Figures 4(a)-4(e) show the locations of the sites.

Table 1: Order and due dates, t_{ord} and t_{due}, of the sites in the introductory example.

site	t_{ord} [s]	t_{due} [s]
0	-	-
1	0	46
2	0	198
3	0	343
4	0	14
5	0	279
6	0	400
7	0	346
8	4	43
9	185	439

Here, we use periodic rescheduling with an interval of 50 s and a horizon length of 500 s (i.e. the entire timespan of the problem). The computing budget of 50 s is distributed evenly between the rescheduling procedures. Therefore, each procedure is allocated a computing time of $t_{\text{comp}} = 5$ s. Figure 4(a) shows the initial route (determined by a greedy search) for the vehicle. Further, Figures 4(b)-4(d) visualize representative open-loop routes during the process[7]. In these figures, the locations of the vehicle at t_{info} and t_{exe} are indicated by the grey and black circles, respectively.

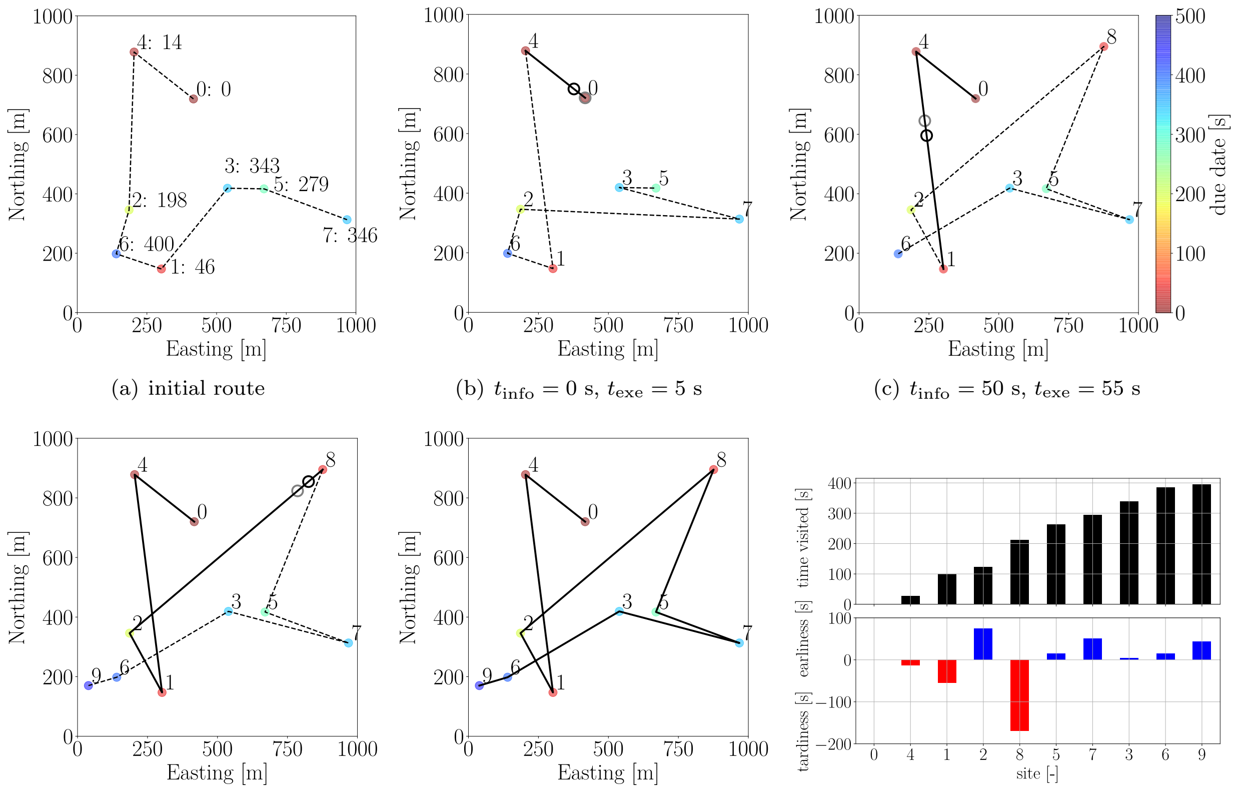
At $t_{\text{exe}} = 5$ s, the first optimized route is obtained, in which the visit to urgent site 1 is scheduled before visits to sites 6 and 2 (which is the opposite to the initial route). At $t_{\text{exe}} = 55$ s and $t_{\text{exe}} = 205$ s, the optimizer includes visits to sites 8 and 9, respectively, to the open-loop route. The locations and due dates of these sites were not known at $t = 0$. When examining the open-loop routes, we can see that obtaining the order of site 8 causes a complete rescheduling of the route (cf. Figs. 4(b) and 4(c)), whereas, when the order of site 9 is obtained, it is simply appended to the open-loop route (cf. Figs. 4(c) and 4(d)). In fact,

[5]In this work, we use the distance of a path as its desirability measure.

[6]In this work, we use values $\alpha = 1$ and $\beta = 3$.

[7]Rescheduling procedures were also triggered at times $t_{\text{info}} = \{100, 150, 250, 300, 350\}$ s, but these procedures did not change the open-loop route. At $t_{\text{info}} = \{400, 450\}$ the vehicle was on standby at site 9. As no new orders are obtained, the rescheduling procedures at these times were omitted.

the rescheduling was not necessary in the case of the latter. The same closed-loop solution would have been obtained even if no rescheduling procedures were triggered after $t_{\text{info}} = 50$. This highlights the importance of carefully assessing the obtained new information. Figure 4(e) shows the final closed-loop route of the vehicle. The delay sum of the final route is 235.30 s, which is caused by the vehicle failing to meet the due dates at sites 1, 4 and 8 (see Fig. 4(f)).



(a) initial route

(b) $t_{\text{info}} = 0$ s, $t_{\text{exe}} = 5$ s

(c) $t_{\text{info}} = 50$ s, $t_{\text{exe}} = 55$ s

(d) $t_{\text{info}} = 200$ s, $t_{\text{exe}} = 205$ s

(e) final closed-loop route

(f) Visiting times and the tardiness/earliness of the vehicle at the sites

Figure 4: Representative open-loop routes and the final closed-loop route for the introductory example with nine sites. The locations and due dates of sites 8 and 9 are received at $t = 4$ s and 185 s, respectively, whereas the other orders are already known at $t = 0$. The continuous path represents the realized route of the vehicle, and dashed path the scheduled open-loop route. The grey and black circles indicate the location of the vehicle at t_{info} and t_{exe}, respectively, for the open-loop routes.

3.2. Test cases with 40-60 sites

In this section, we define four test cases that are similar to, but larger than, the introductory example in the previous section. We define Test cases 1-3 to involve 50 sites but, in order to vary the amount of new information obtained during the timespan, to have varying numbers of initially known and new orders (Table 2). The number of new orders varies from 0 (i.e., no new orders) to 20. However, in reality, the frequency at which new orders are received fluctuates between time frames. In order to mimic this fluctuation, we define the number of new orders, n_{new}, to be randomly drawn from the uniform distribution of $\{0, 1 \dots 20\}$ in Test case 4.

Further, we define the parameters for the test cases randomly. The x- and y-coordinates of each site are randomly drawn from a uniform distribution of [0, 1000] m, and the due dates t_{due} from a discrete uniform distribution of $\{0, 1 \dots t_{\text{span}}\}$, where the timespan $t_{\text{span}} = 1000$ s. Accordingly, the order dates t_{ord} of the n_{new} sites are randomly drawn from a discrete uniform distribution of $\{0, 1 \dots t_{\text{due}}\}$. We use this randomization method to create different instances of the test cases. It is worth noticing that, because of

Table 2: The numbers of initially known and new orders, n_{init} and n_{new}, respectively, in the four test cases.

Test case	n_{init} [-]	n_{new} [-]
1	50	0
2	40	10
3	30	20
4	40	randomly drawn from $\{0, 1 \ldots 20\}$

the definition, the probability distribution of the order dates t_{ord} in $\{0, 1 \ldots t_{\text{span}}\}$ is not uniform, but biased towards the beginning of the time span (Fig. 5). We can express the probability of the k^{th} time point in $\{0, 1 \ldots t_{\text{span}}\}$ to be chosen as the order date as

$$p_k = \frac{1}{n_{\text{t}}} \sum_{i=k}^{n_{\text{t}}} \frac{1}{i}, \tag{2}$$

where $n_{\text{t}} = |\{0, 1 \ldots t_{\text{span}}\}|$.

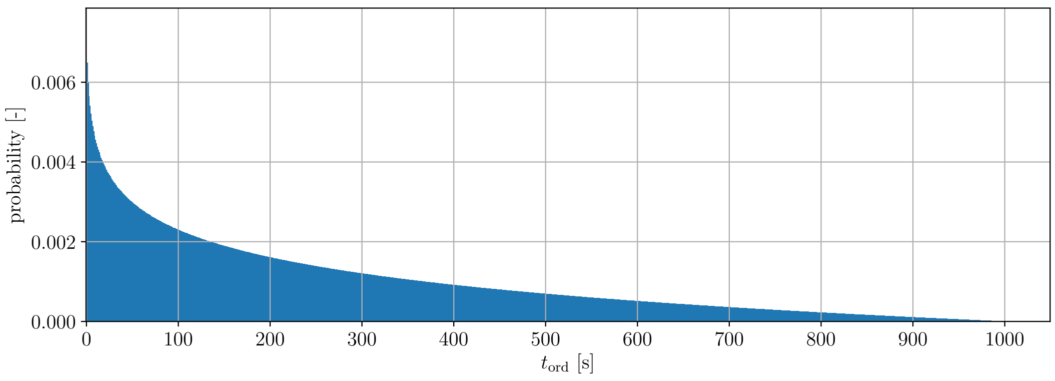


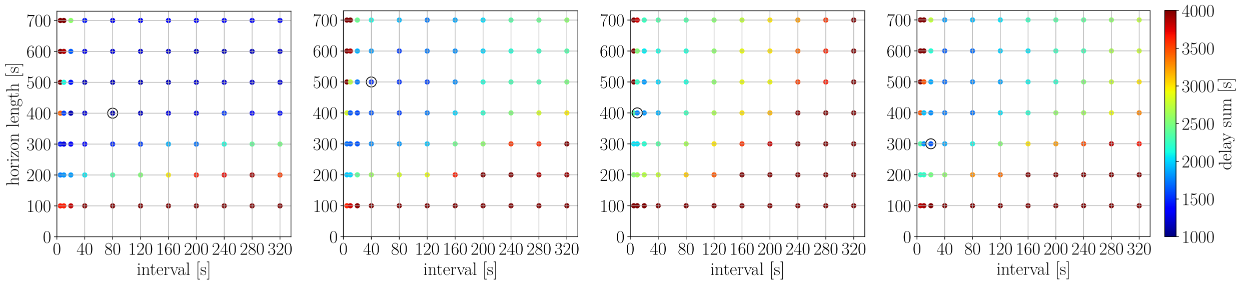
Figure 5: Probability distribution of the order date $t_{\text{ord}} \in \{0, 1 \ldots t_{\text{span}}\}$.

In Section 4, we will use these problems as test cases for the RL-based rescheduling approach. In the remainder of this section, we tune the parameters of periodic and event-triggered rescheduling, as well as their hybrid, to each of the four test cases using a grid search, as the optimal values of these parameters are problem-dependent. The parameters to be tuned of periodic rescheduling are the rescheduling interval and horizon length. In event-triggered rescheduling, we tune the horizon length and the allocated computing time per rescheduling procedure, and, in hybrid rescheduling, the rescheduling interval and the allocated computing time per rescheduling procedure. The resulting rescheduling methods with tuned parameters will be benchmark methods for our RL-based approach.

The parameter tuning of these rescheduling methods is equivalent to the training of the agent in the RL-based approach. In order to facilitate a fair comparison between the methods, we train the models on the same randomly generated instances of each test case. More precisely, we use the first five instances (seeds $\{0, 1 \ldots 4\}$) in the training, and the next 30 (seeds $\{5, 6 \ldots 34\}$) in testing the methods. The latter provides an indication of performance of the methods on unseen instances of the test cases. We evaluated each parameter combination on the five training instances of each test case. We ran the experiment on a cluster of 20 Intel(R) Xeon(R) Gold 6148 CPUs, each having 4.4 GB of RAM, such that only one CPU is assigned to each of the parameter combination evaluations.

In the periodic rescheduling, the tested parameter values for the rescheduling interval are $\{5, 10, 20, 40, 80, 120, 160, 200, 240, 280, 320\}$ s and for the horizon length $\{100, 200, \ldots, 700\}$ s, yielding a total of 77 parameter combinations, which we study using a grid search. Figures 6(a)-6(d) show the average delay sums for

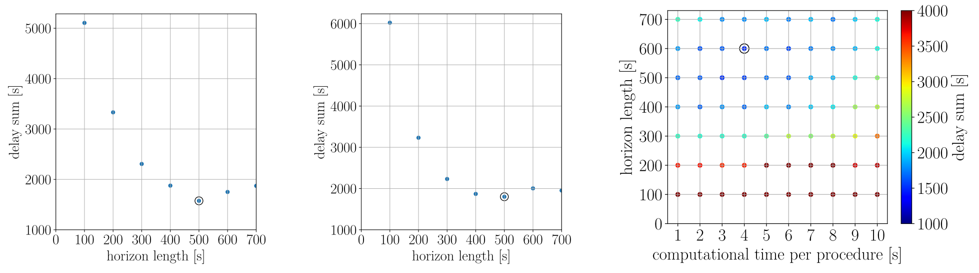
Test cases 1 to 4, respectively. The following behavior is visible in all results. The combinations lying near the lower right corner of the plots have poor performance because, in this region, the rescheduling interval is longer than the horizon length, forcing the vehicle occasionally to switch to the greedy algorithm. The combinations lying in the other extreme (i.e., near the top left corner of the plot) also have a poor performance. The reason is that in these combinations rescheduling procedures are performed very frequently with short computing times and long horizon lengths. The optimizer does not have enough time to find good open-loop solutions to these relatively large problems. The combinations lying in between these two regions have good performance.



(a) Test case 1 ($n_{\text{new}} = 0$) (b) Test case 2 ($n_{\text{new}} = 10$) (c) Test case 3 ($n_{\text{new}} = 20$) (d) Test case 4 (n_{new} is varied)

Figure 6: Tuning of periodic rescheduling. The reported delay sums are the averages of five problem instances. The best parameter combination is highlighted by a circle. Darkest red points exceed the scale.

The event-triggered rescheduling is only applicable to Test cases 2-4, as no new orders are obtained in Test case 1. The tested parameter values for the rescheduling horizon length are $\{100, 200, \ldots, 700\}$ s. In Test cases 2 and 3, we know precisely how many rescheduling procedures will be triggered[8], as the number of new orders, n_{new}, is fixed. Thus, in these test cases, we distribute the computing budget of 50 s evenly for the resulting $n_{\text{new}} + 1$ rescheduling procedures (taking also the rescheduling at time $t = 0$ into account). In Test case 4, the number of new orders varies from 0 to 20. Therefore, as we cannot allocate the computing time per procedure optimally a priori, we adopt a similar grid search strategy as with the periodic rescheduling. The tested values for the computing time per procedure are $\{1, 2, \ldots, 10\}$.



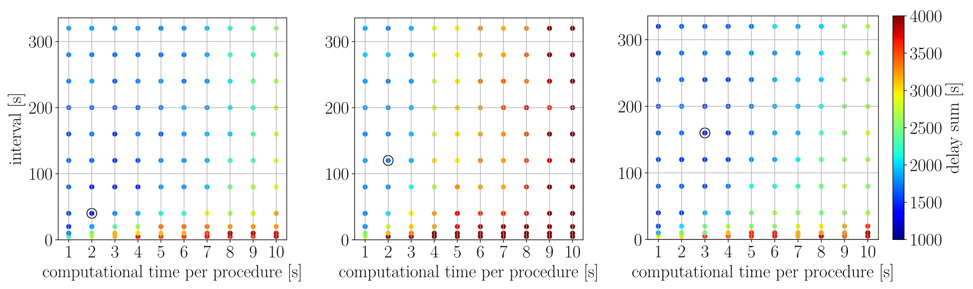
(a) Test case 2 ($n_{\text{new}} = 10$) (b) Test case 3 ($n_{\text{new}} = 20$) (c) Test case 4 (n_{new} is varied)

Figure 7: Tuning of event-triggered rescheduling. In Test cases 2 and 3, the computing budget is distributed evenly between the rescheduling procedures based on the number of new orders, n_{new}, which is known a priori. In Test case 4, the computing time per procedure is a tuning parameter. Event-triggered rescheduling is not applicable to Test case 1.

The hybrid rescheduling also is not applicable to Test case 1. The parameters we test in Test cases 2 to 4 are the rescheduling intervals of $\{5, 10, 20, 40, 80, 120, 160, 200, 240, 280, 320\}$ s and the allocated computing

[8]With the caveat that new rescheduling procedures cannot be triggered if the previous rescheduling is ongoing (the same applies later to the RL agent).

times of $\{1, 2, \dots, 10\}$ s. In the hybrid rescheduling, we do not know a priori how many rescheduling procedures will be required. The horizon length is also a key parameter but, in order to keep the size of the grid search manageable, we choose here to use the horizon length of 400 s, which is a representative value in periodic and event-triggered rescheduling.



(a) Test case 2 ($n_{\text{new}} = 10$) (b) Test case 3 ($n_{\text{new}} = 20$) (c) Test case 4 (n_{new} is varied)

Figure 8: Tuning of hybrid rescheduling. The best parameter combination is highlighted by a circle. Darkest red points exceed the scale. Hybrid rescheduling is not applicable to Test case 1.

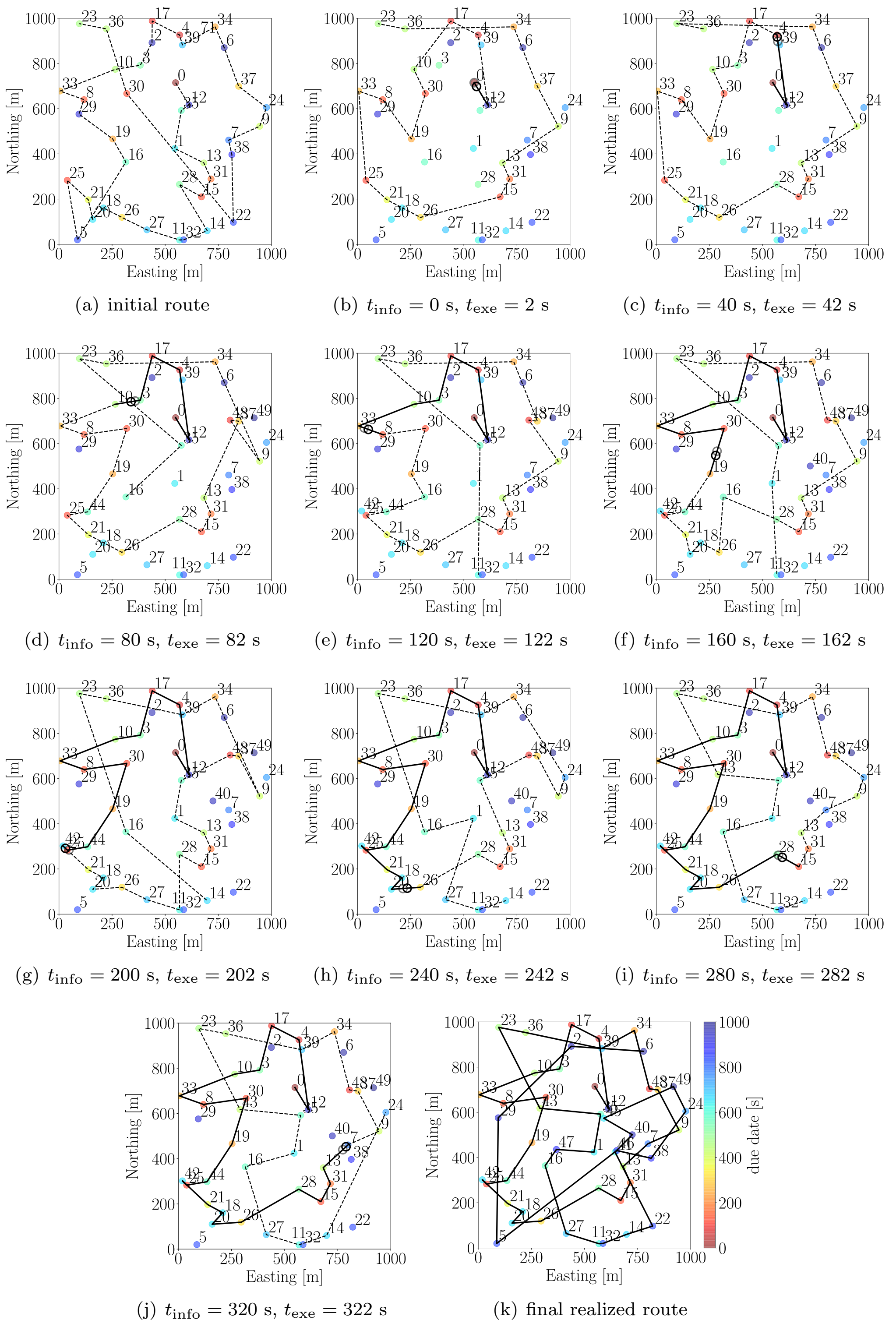
The best-performing parameter combinations for the conventional rescheduling methods, as well as the corresponding average delay sums, are listed in Table 3. In these combinations, the horizon length varies between 300 and 600 s. In Test cases 1-3, the rescheduling interval of periodic rescheduling seems to be inversely proportional to the number of new orders n_{new}. This result makes sense from the practical point of view – the more frequently new information is obtained, the sooner the optimized schedule becomes outdated. In Test cases 1-3, the average delay sum seems to be proportional to the number of new orders n_{new}, showing the difficulty of finding good solutions with limited information. When applicable (i.e., in Test case 2 to 4), the hybrid rescheduling yields the shortest average delay sums. The results reported in Table 3 are the benchmarks for our RL-based approach in Section 4.

Table 3: The best performing parameters for the periodic, event-triggered, and hybrid rescheduling, as well as the corresponding delay sums, in Test cases 1-4.

Test case [-]	method	interval [s]	horizon length [s]	computing time per procedure [s]	average delay sum [s]
1[i]	periodic	80	400	3.84[ii]	1120.7
2	periodic	40	500	2.00[ii]	1504.9
	event-triggered	-	500	4.54[ii]	1574.7
	hybrid	40	400[iii]	2.00	1419.0
3	periodic	10	400	0.50[ii]	1793.0
	event-triggered	-	500	2.38[ii]	1801.8
	hybrid	120	400[iii]	2.00	1699.7
4	periodic	20	300	1.00[ii]	1638.6
	event-triggered	-	600	4.00	1515.5
	hybrid	180	400[iii]	3.00	1463.2

[i] Event-triggered and hybrid rescheduling are not applicable to Test case 1. [ii] Not explicitly tuned. The value is determined by distributing the total computing budget evenly between the rescheduling procedures, the number of which is known a priori. [iii] Not explicitly tuned. The value is chosen as a representative value of tuned parameters in periodic and event-triggered rescheduling.

Finally, Figure 9 presents representative initial, first nine open-loop, and the final closed-loop route of the periodic rescheduling on the first instance of Test Case 2. Figure 10 shows the corresponding visiting times and the tardiness/earliness of the vehicle at the sites. Despite being a solution to a larger scale problem, the same features are also present here as in the introductory example (Section 3.1): sites with urgent due dates are prioritized and, as the time proceeds, new orders are included in the scheduled route.



(a) initial route (b) $t_{\text{info}} = 0$ s, $t_{\text{exe}} = 2$ s (c) $t_{\text{info}} = 40$ s, $t_{\text{exe}} = 42$ s

(d) $t_{\text{info}} = 80$ s, $t_{\text{exe}} = 82$ s (e) $t_{\text{info}} = 120$ s, $t_{\text{exe}} = 122$ s (f) $t_{\text{info}} = 160$ s, $t_{\text{exe}} = 162$ s

(g) $t_{\text{info}} = 200$ s, $t_{\text{exe}} = 202$ s (h) $t_{\text{info}} = 240$ s, $t_{\text{exe}} = 242$ s (i) $t_{\text{info}} = 280$ s, $t_{\text{exe}} = 282$ s

(j) $t_{\text{info}} = 320$ s, $t_{\text{exe}} = 322$ s (k) final realized route

Figure 9: Visualization of the vehicle routing in the first instance of Test case 2 ($n_{\text{new}} = 10$). In addition to the initial **(a)** and the final closed-loop route **(k)**, the first nine open-loop routes are shown **(b)**-**(j)**.

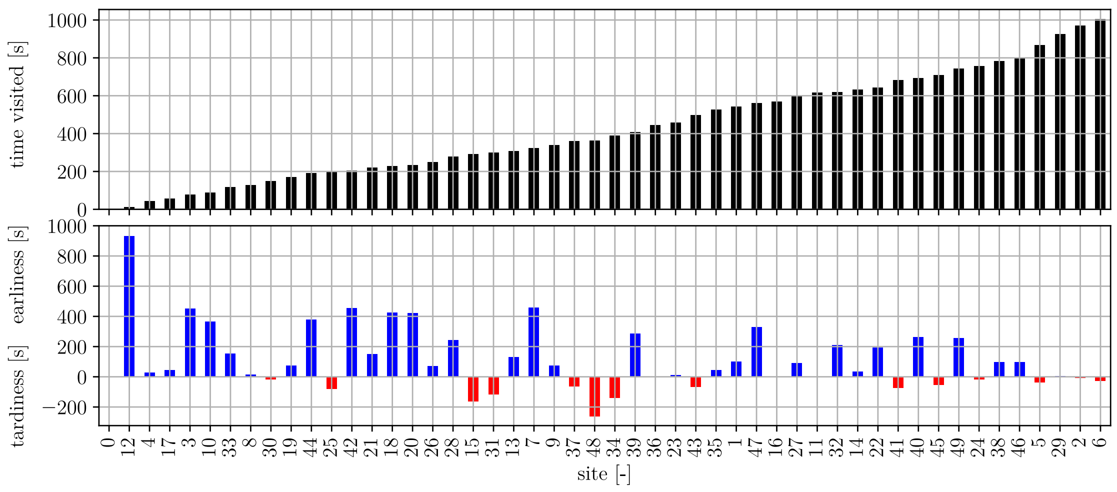


Figure 10: Visiting times and the tardiness/earliness of the vehicle at the sites, corresponding to the final route shown in Fig. 9(k).

4. Reinforcement learning of rescheduling timing and computing time allocation

Let us now solve the dynamic routing problems by the proposed approach, considering the first two of the four rescheduling decisions, listed in Section 1. As the RL algorithm, we use neuroevolution of augmenting topologies (NEAT). We describe NEAT in the next section. In Section 4.2, we describe the state and action spaces, as well as the reward function, of the simplified approach.

4.1. Neuroevolution of augmenting topologies (NEAT)

NEAT is an evolutionary algorithm that simultaneously evolves the topology and weights of a neural network (Stanley & Miikkulainen, 2002). The freedom to control the topology allows NEAT to scale the complexity of the neural network to a level appropriate for the problem. For simple problems, the optimized neural network topology may consist of only a few nodes and connections, whereas, in complex problems, also the complexity of the neural network is increased. In the applications of the algorithm, the largest neural networks are composed of connections, the number of which is in an order of hundreds or thousands (Stanley et al., 2019).

NEAT is based on three main features. First, the *genes* of the individuals are tracked with historical markers, in order to enable crossover between individuals with different topologies. Here, a gene contains the genetic information of a node or connection in a neural network. When structural mutation introduces a new node or connection into an individual, the gene of the new structure, is assigned a marker, referred to as the *innovation number*. In crossover, the genes with the same innovation number are aligned. An offspring inherits each aligned gene randomly from one of its parents and the non-aligned genes from the more fit parent.

Second, the population is divided into species based on a measure, referred to as the *genetic distance*. Between two individuals, the genetic distance is determined based on the number of non-aligned genes and the average weight differences in aligned genes. Individuals primarily compete within their own species. This feature protects new innovations, which perhaps have not yet reached their potential, from being prematurely eliminated from the population.

Third, NEAT starts the evolution process from a topologically uniform population, having minimal complexity. During the evolution, mutations introduce topological variations, which incrementally increase the complexity of the neural networks, and topological diversity of the population. This strategy minimizes the dimensionality of the search space. Stanley & Miikkulainen (2002) concluded, based on a series of ablation studies, that the three main features are interdependent and, thus, necessary for the algorithm to reach a fast learning rate. In this work, we use a Python implementation of the algorithm, neat-python (McIntyre et al., 2019).

4.2. State and action spaces, and the reward function

The two rescheduling decisions we consider in this work are 1) the timing of rescheduling procedures and 2) the allocated computing time. Figure 11 shows the state and action spaces for the agent, interacting with the environment.

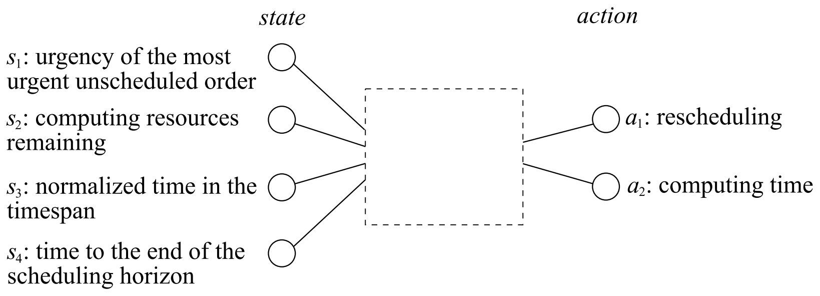

state
action
s_1: urgency of the most urgent unscheduled order
s_2: computing resources remaining
s_3: normalized time in the timespan
s_4: time to the end of the scheduling horizon
a_1: rescheduling
a_2: computing time

Figure 11: State and action spaces for the dynamic routing problem.

State space. The state space signals **s** are the means of the agent to sense the environment. In our test cases, these signals describe the status of the process (i.e., the dynamic vehicle routing) and the optimizer (i.e., ACO and the status of the computing resources). We use four simple state space signals.

Signal s_1 describes the urgency of the most urgent unscheduled order as

$$s_1 = \begin{cases} 1, & \text{if } \check{t}_{\text{due}} - t > h \\ \frac{\check{t}_{\text{due}} - t}{h}, & \text{if } 0 \le \check{t}_{\text{due}} - t \le h \\ 0, & \text{otherwise,} \end{cases} \tag{3}$$

where $\check{t}_{\text{due}}$ is the due date of the most urgent unscheduled order, t is the current time, and h is the length of the scheduling horizon. The most urgent unscheduled order can belong to the n_{init} initially known orders or the n_{new} new orders. Signal s_2 describes the level of computing resources remaining as

$$s_2 = \frac{c_{\text{rem}}}{c_{\text{budget}}}, \tag{4}$$

where c_{rem} is the used computing time and c_{budget} is the computing time budget, allocated for the timespan. While the test cases have a finite timespan, signal s_2 could in continuous online scheduling describe the remaining computing budget of a cloud computing facility, allowing more parallel processors to be used simultaneously but with a predefined quota for a certain timespan. Signal s_3 is the current time, normalized with respect to the timespan t_{span}, defined as

$$s_3 = \frac{t}{t_{\text{span}}}. \tag{5}$$

Finally, signal s_4 indicates how far in the future is the end of the previously used scheduling horizon. This signal is defined as

$$s_4 = \begin{cases} 1, & \text{if } \hat{t}_{\text{scheduled}} - t > h \\ \frac{\hat{t}_{\text{scheduled}} - t}{h}, & \text{if } 0 \le \hat{t}_{\text{scheduled}} - t \le h \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

where $\hat{t}_{\text{scheduled}}$ is the visiting time of the last site in the current scheduled route.

Based on the categorization of state space signals in Section 2, signal s_1 belongs to Category 2, and signals s_2, s_3, and s_4 to Category 3. If we would model the speed v of the vehicle to fluctuate[9], for example due to traffic conditions, we could include also a Category 1 signal in the state space. This signal would describe the deviation in the progress of the vehicle with respect to the scheduled visiting times. Alternatively, we could model the due dates of the known orders to change in time. In this case, Category 1 signals could describe the deviations of the due dates with respect to the parameters used in the previous rescheduling procedure.

Action space. We encode the two rescheduling decisions into an action space $\mathbf{a}$ of two signals. Signal a_1 encodes the decision of timing the rescheduling procedures. At each time point, the signal determines whether a new rescheduling procedure is triggered, as follows:

$$\begin{cases} \text{If } a_1 \leq \frac{1}{2} & \rightarrow \text{no rescheduling} \\ \text{If } a_1 > \frac{1}{2} & \rightarrow \text{rescheduling.} \end{cases} \tag{7}$$

In Section 2, we encoded 1) the timing of rescheduling procedures and 2) the optimization strategy into three action space signals (see Fig. 3). As here the optimization strategy always is the heuristic algorithm, ACO, the timing of rescheduling procedures can be encoded using only one signal. Signal a_2 encodes the allocated computing time c for the rescheduling procedure, as follows:

$$c = c_{\min} + (c_{\max} - c_{\min})a_2, \tag{8}$$

where $c_{\min}$ and $c_{\max}$ are the predefined minimum and maximum computing times for a rescheduling procedure. If signal $a_1 \leq \frac{1}{2}$, signal a_2 is ignored.

The reward function. The agent seeks to maximize its reward from the environment. In Section 3, we defined the objective of dynamic routing problem to be to minimize the delay sum of the vehicle at the sites. In the beginning of Section 4, we defined that we train the agents on five instances of each of the three test cases. Therefore, we now define the reward function to be the average delay sum in the five instances multiplied by -1.

4.3. Results

The parameters we use in this case study to define the basic behavior of the agent are listed in Table 4. We use the same parameters for all four test cases. We use a fixed horizon length of $h = 400$ s, as this is a representative value in the tuned parameter combinations of the periodic and event-triggered rescheduling. In Section 3.2, the allocated computing time per procedure with the conventional rescheduling methods varies between 0.5 and 4.54 s[10] in the test cases. Here, we have chosen such minimum and maximum computing times, $c_{\min}$ and $c_{\max}$, that this interval is captured. We define the agent to receive the state signals and act at a frequency of $f_{\mathrm{a}} = 1$ Hz. However, if a rescheduling procedure is ongoing, the agent cannot terminate it and start a new one.

Regarding the NEAT algorithm, we use a population of 80 individuals, and evolve the population for 80 generations. For other parameters of the algorithm, we have chosen to use the same values as in the single-pole-balancing example in neat-python (McIntyre et al., 2019). We have listed these parameters in Appendix A.

The training and testing were performed on the same computing facility, which we used in Section 3.2. As already mentioned in Section 3.2, we use five training and 30 test instances of each test case. The computational cost to train an agent is dominated by the total computing time budget c_{budget} and the number of evaluated training instances of the test case, n_{training}. With the parameters we have chosen here, the training involves evaluation of 6400 candidate agents, and each evaluation requires, at most,

[9]For the sake of simplicity, we have defined the speed of the vehicle to be constant.

[10]The lower bound corresponds to the periodic rescheduling in Test case 3, in which the tuned rescheduling interval was 10 s. This means 100 rescheduling procedures during the time span of 1000 s, each of which is allocated a computing time of 0.5 s. The upper bound is from the event-triggered rescheduling in Test case 2.

Table 4: The parameter defining the basic behavior of the agent and its training.

Parameter	Symbol	Value
minimum computing time per procedure	$c_{\min}$	0.5 s
maximum computing time per procedure	$c_{\max}$	5 s
horizon length	h	400 s
action frequency	f_{a}	1 Hz
number of training instances	n_{training}	5
population size (NEAT)	n_{pop}	80
number of generations (NEAT)	n_{gen}	80

$n_{\mathrm{training}} \times c_{\mathrm{budget}} = 250$ s of computing time on the ACO algorithm. Although, in reality, the average evaluation time is a bit lower, as all candidate agents do not use the whole computing time budget. Thus, on our computing facility, consisting of 20 CPUs, the required time to train an agent to an environment is around 20 hours.

Table 5 shows the average delay sums in Test cases 1-4 when the rescheduling decisions are determined by the optimized agent, obtained by the NEAT algorithm, and the conventional rescheduling methods with the tuned parameters listed in Table 3. We report the average delay sums separately for the training and test instances. For the conventional rescheduling methods, the training results are the same as in Table 3, and the test results we have generated accordingly using the tuned parameters. We report the relative differences with respect to that conventional rescheduling method yielding the best results on the test instances. The event-triggered and hybrid rescheduling are not applicable to Test case 1, so we report the relative differences with respect to the periodic rescheduling. When looking at the training instances, the agents, obtained by the NEAT algorithm, make rescheduling decisions that lead to closed-loop routes with smaller average delay sums than those obtained by the conventional rescheduling methods on all four test cases.

Table 5: Comparisons of close-loop results for Test cases 1 to 4 when using the NEAT agents and the conventional rescheduling methods to trigger the rescheduling procedures. The comparisons are made separately on $n_{\mathrm{training}} = 5$ training and $n_{\mathrm{test}} = 30$ test instances.

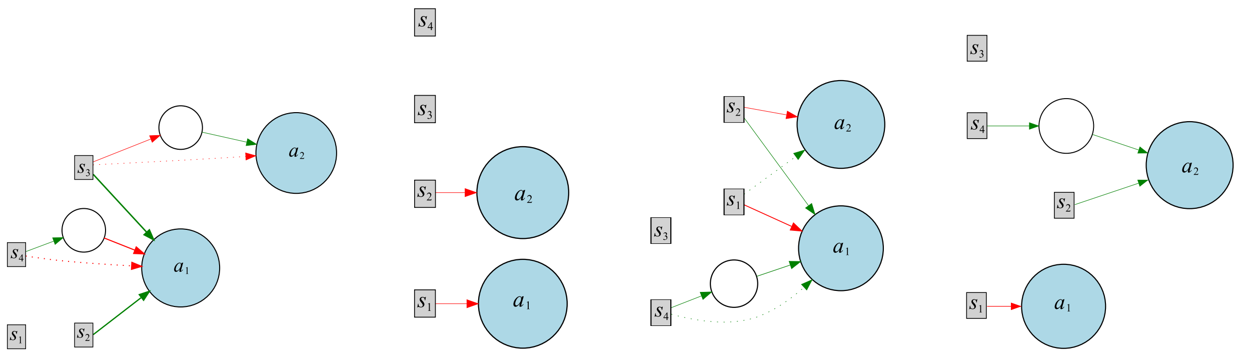
		Average delay sum [s]		Relative difference [%]	
Test case	Rescheduling method	training instances	test instances	training instances	test instances
1[i]	periodic	1120.7	1275.4	0.00	0.00
	NEAT agent	1061.3	1235.1	-5.30	-3.16
2	periodic	1504.9	1604.8	-4.43	3.82
	event-triggered	1574.7	1545.8	0.00	0.00
	hybrid	1419.0	1550.8	-9.89	0.32
	NEAT agent	1365.9	1530.1	-13.26	-1.02
3	periodic	1793.0	2128.6	-0.49	7.10
	event-triggered	1801.8	1987.5	0.00	0.00
	hybrid	1699.7	2032.1	-5.67	2.24
	NEAT agent	1635.2	1980.1	-9.25	-0.37
4	periodic	1638.6	1743.5	11.99	8.26
	event-triggered	1515.5	1723.1	3.57	6.99
	hybrid	1463.2	1610.5	0.00	0.00
	NEAT agent	1423.0	1674.4	-2.75	3.97

[i] Event-triggered and hybrid rescheduling are not applicable to Test case 1.

When looking at the corresponding results on the unseen test instances, the NEAT agents yield, on average, better results than the conventional rescheduling methods on three out of four test cases (i.e., Test cases 1 to 3). The margins to the best-performing conventional rescheduling method are 0.37 to 3.16%. On Test case 4, the hybrid rescheduling yielded, on average, the best results by a margin of 3.97% to those obtained by the NEAT algorithm, which yielded the second best results. It is worth noticing that substantial effort has been made in tuning the conventional rescheduling methods to the test cases (see Section 3.2).

Let us next examine the neural network topologies of the agents we obtained by the NEAT algorithm.

Figure 12 presents the topologies of the agents that are trained to the four different test cases. Signal s_1 describes the urgency of the most urgent unscheduled order. It is particularly meaningful when a new order arrives, as the new order may well become the most urgent order. In Test case 1, no new orders arrive during the timespan. In the corresponding neural network (Fig. 12(a)), signal s_1 is not connected to either of the action signals. The reason is that, in this case, signals s_1 and s_4 (i.e., the remaining time in the previously used scheduling horizon) describe nearly the same information of the environment. As signal s_4 has connections to a_1, signal s_1 is redundant. In Test cases 2 to 4, new orders are obtained during the timespan. In the corresponding neural networks (Figs. 12(b) to 12(d)), signal s_1 is connected to action signal a_1 by a negative connection weight. Therefore, the smaller is the value of signal s_1 (meaning greater urgency), the more it stimulates the agent to trigger a rescheduling procedure. In fact, the information of s_1 is very relevant for the agents operating in environments where new orders arrive, which we will show later when discussing Fig. 13.



(a) Test case 1 (b) Test case 2 (c) Test case 3 (d) Test case 4

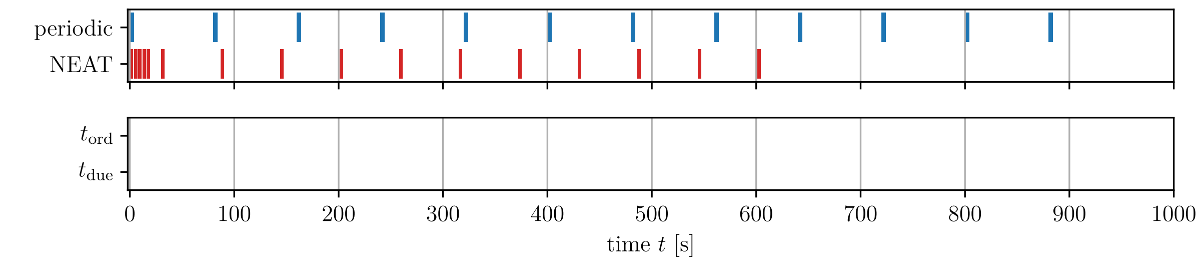
Figure 12: Neural network topologies of the optimized agents to Test cases 1-4. State signal s_1 is the urgency of the most urgent unscheduled order, s_2 is the computing resources remaining, s_3 is the normalized time in the timespan, and s_4 is the time to the end of the scheduling horizon. Action signals a_1 and a_2 determine whether the rescheduling procedure is initiated and the allocated computing time, respectively. The red and green arrows indicate connections with negative and positive weights, respectively. The figures are created by a visualization function in neat-python, which uses Graphviz (Ellson et al., 2003).

Signal s_2 describes the computing resources remaining. In Figs. 12(a) and 12(c)), it has a positive connection to signal a_1. The more computing resources remain, the more it stimulates the agent to triggering a rescheduling procedure. However, in Figs. 12(b) and 12(c), signal s_2 has also a negative connection to signal a_2, which means that the smaller are the computing resources remaining, the more computing time is allocate to rescheduling procedures. We would expect the agent to become more frugal when the remaining computing resources are reduced (like in Fig. 12(d)). Therefore, this negative connection seems counter-intuitive (we will return to this later in this section).

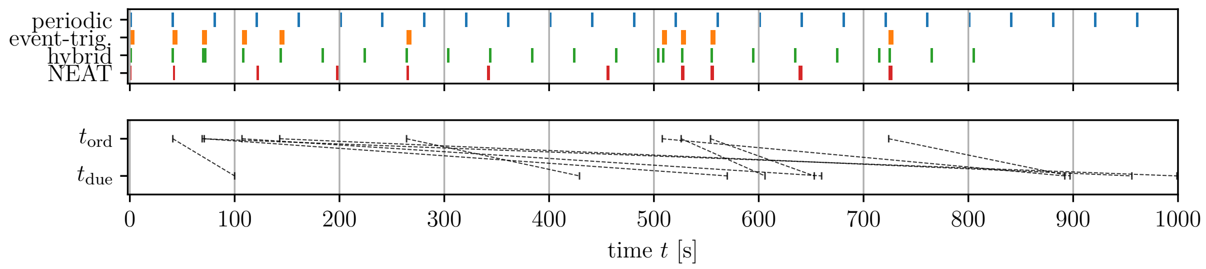
Signal s_3 is the normalized time with respect to the timespan. In Test cases 2 to 4, the signal seems to have low importance, as it is not connected to the action signals in Figs. 12(b) to 12(d). In these test cases, new orders are obtained during the timespan. In contrast, in the neural network of Test case 1 (with no new orders), signal s_3 seems to have a high importance, as it has connections to both a_1 and a_2.

Figure 13 shows the timing and duration of the rescheduling procedures, as well as the order dates t_{ord} of the n_{new} new orders and their corresponding due dates t_{due}. The figures correspond to the first training instance of Test cases 1-4. The rescheduling intervals of the periodic rescheduling were 80, 40, 10, and 20 s in these test cases, respectively. Some periodic rescheduling procedures were omitted at the end of the timespan. The reason is that at these time points the open-loop optimization problems include less than two sites, the order of which can be varied. Therefore, as the solution is trivial, the procedures are omitted.

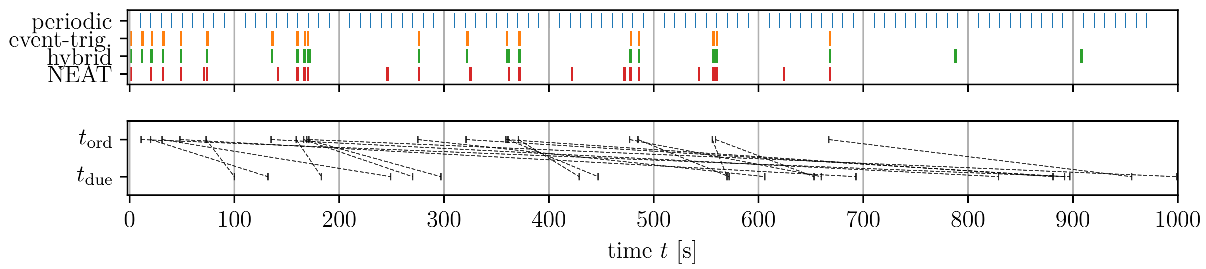
The behavior of the trained agents, obtained by the NEAT algorithm, is closer to that of the event-triggered rescheduling than that of the periodic rescheduling. In contrast to the periodic rescheduling, the



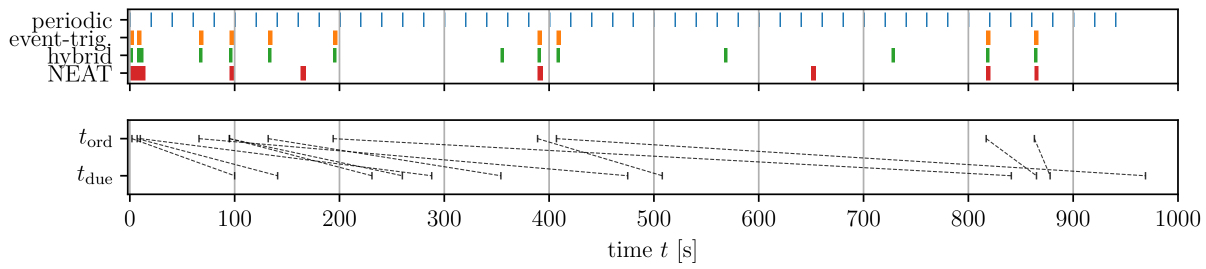
(a) Test case 1 (no new orders)



(b) Test case 2



(c) Test case 3



(d) Test case 4

Figure 13: The timings and allocated computational times.

agents, obtained by the NEAT algorithm, bias the rescheduling procedures towards the beginning of the timespan. This behavior is the clearest by the agent trained to Test case 1 (Fig. 13(a)). At the very beginning of the timespan, this agent triggers a series of five rescheduling procedures at times $t = \{0, 4, 8, 12, 16\}$ s, each of which having an allocated computing time of 3.27 s. Therefore, between $t = 0 \dots 16$ s, the agent triggers always when possible. The agent has learned that, as no new orders occur in this test case, the beginning of the timespan is critical. Investing a significant fraction (32.7%) of computing time budget already during the first 16 s of the timespan is a part of a behavior that yields a good reward. After $t = 30$ s, the agent triggers rescheduling procedures at an interval of around 57 s. It triggers the last rescheduling procedure at $t = 601$ s, when the remainder of the timespan fits into a single scheduling horizon of $h = 400$ s.

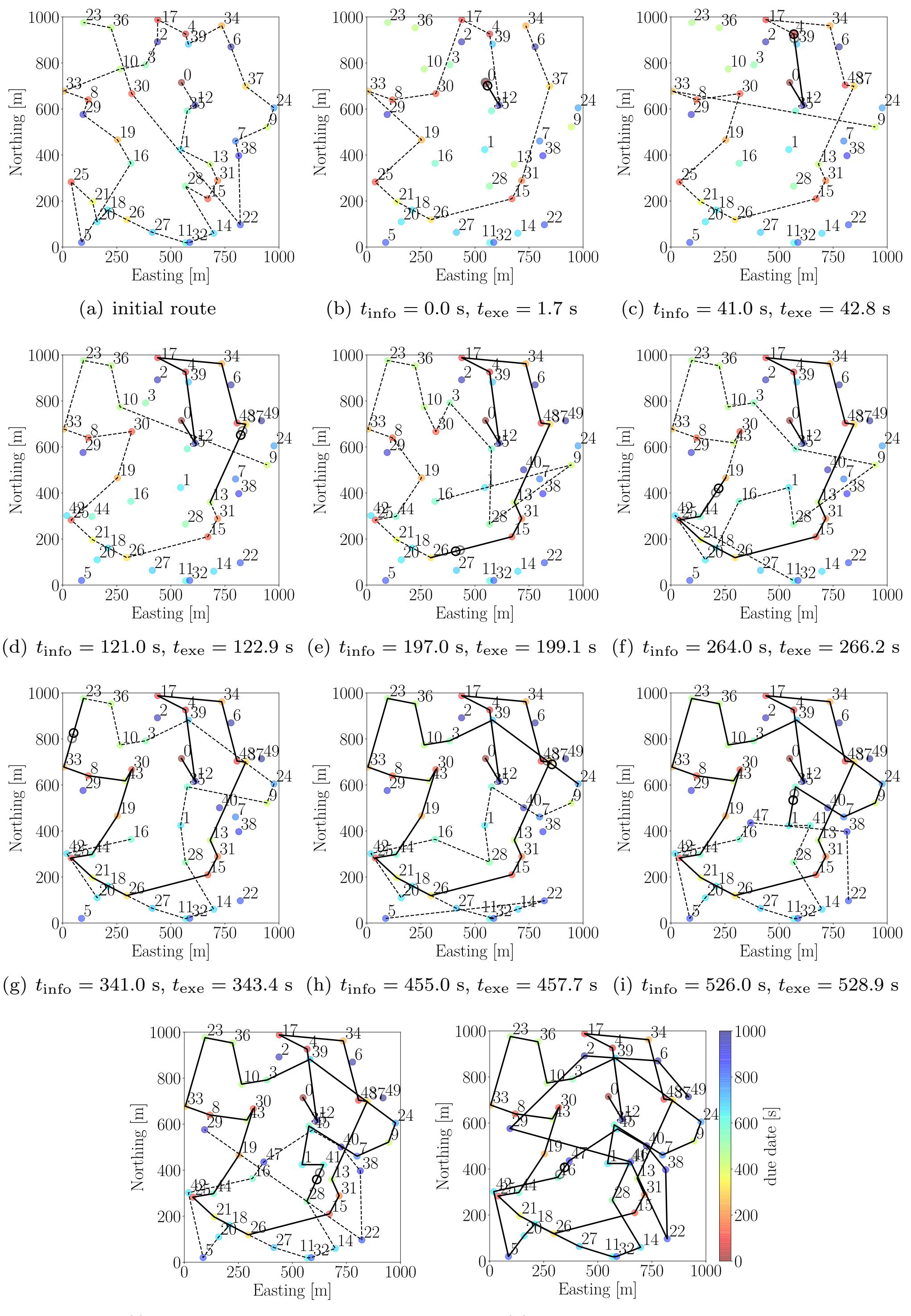
The agents trained to Test cases 2 to 4 (Figs. 13(b) and 13(c)) also have biased the rescheduling procedures towards the beginning of the timespan, but the bias is not as strong as by the agent trained to Test case 1. These rescheduling procedures follow the arrivals of new orders[11] – particularly those with high urgency. The level of urgency is indicated by the slope of the line connecting the order date t_{ord} to its due date t_{due}. This makes sense from a practical point of view: If a new urgent order arrives, an immediate rescheduling seems like a reasonable action. In Fig. 13(b), five orders with the steepest slope occur at times $t = \{41, 264, 526, 554, 724\}$ s. At these time points, the agent triggers immediate rescheduling procedures. The order with the sixth steepest slope occurs at time $t = 508$ s. The agent does not immediately react to this, or to the other orders with gentler slopes. The agents of Test cases 3 and 4 have a similar behavior. This ability to track the arrivals of new orders is due to signal s_1, as signals s_2 and s_3 are senseless to the order and due dates of the sites, and signal s_4 is senseless to sites that are not yet scheduled.

Earlier we noticed that, counter-intuitively, signal s_2 has a negative connection to signal a_2 in the agents trained to Test cases 2 and 3 (see Figs. 12(b) and 12(c)). This negative connection causes the allocated computing time per procedure, c (Eq. 8), to gradually increase during the timespan from 1.66 to 3.72 s in (the shown instance of) Test case 2, and from 1.77 to 2.38 s in Test case 3. Looking at Figs. 12(b) and 12(c), this behavior seems now more justifiable. The randomization method we used to generate the instances biases the order dates towards the first half of the timespan (see Fig. 5). The agent has learned that in the beginning of the timespan new orders are frequent, and it should react to these by frequent rescheduling procedures. As the resulting schedules will anyway be changed soon, it is not worth to invest long computing times to these procedures. In the end of the timespan, new orders are less likely (in Test cases 2 and 3, last orders arrive at $t = 724$ and $t = 667$ s, respectively). Therefore, it is worth investing longer computing times in the corresponding rescheduling procedures. Despite the counter-intuitiveness of the negative connection between signals s_2 and a_2, this was the way the agent was able to adapt to the biased order dates. In contrary, in Test case 4, the number of new orders is uncertain, to which the agent reacts by reducing the allocated computing time per procedure from 5.0 to 3.56 s during the timespan, in order to reduce the risk of prematurely running out of the computing time.

Finally, let us return to the visualized instance of the routing procedure by the periodic rescheduling, shown in Fig. 9, and compare it with the corresponding routing by the agent trained to Test case 2 (Fig. 14). The timings of rescheduling procedures, as well as the order and due dates, t_{ord} and t_{due}, of these two routings were shown in Fig. 13(b). The initial routes (Fig. 9(a) and 14(a)) are the same because they are generated by the same deterministic greedy search algorithm. The routes resulting from rescheduling procedures triggered at $t = 0$ s (Fig. 9(b) and 14(b)) have features similar to each other: The visits to urgent sites 4 and 17 are prioritized, and then the remaining sites in the scheduling horizon are visited by routes forming slightly zigzagging counter-clockwise spirals. It is worth noticing that, here, the periodic rescheduling has the scheduling horizon of $h = 500$ s and the scheduling procedures decided by the agent the scheduling horizon of $h = 400$ s. Therefore, sites 9, 10, 13, 23, and 36, the due dates of which are between $t = 400 \dots 500$, are included in the open-loop route in Fig. 9(b), but not to that of Fig. 14(b).

The next pair of open-loop routes in Fig. 9(c) and 14(c) are drastically different. These are the routes from rescheduling procedures that occurred at $t_{\text{info}} = 40$ (the periodic rescheduling) and $t_{\text{info}} = 41$ s (the NEAT agent). The periodic rescheduling triggers a rescheduling procedure at $t = 40$ because it belongs to the predefined plan. The agent, on the other hand, does not reason a rescheduling procedure necessary at

[11]The probability distribution of the order dates is biased towards the beginning of the timespan, see Fig. 5)



(a) initial route

(b) $t_{\text{info}} = 0.0$ s, $t_{\text{exe}} = 1.7$ s

(c) $t_{\text{info}} = 41.0$ s, $t_{\text{exe}} = 42.8$ s

(d) $t_{\text{info}} = 121.0$ s, $t_{\text{exe}} = 122.9$ s

(e) $t_{\text{info}} = 197.0$ s, $t_{\text{exe}} = 199.1$ s

(f) $t_{\text{info}} = 264.0$ s, $t_{\text{exe}} = 266.2$ s

(g) $t_{\text{info}} = 341.0$ s, $t_{\text{exe}} = 343.4$ s

(h) $t_{\text{info}} = 455.0$ s, $t_{\text{exe}} = 457.7$ s

(i) $t_{\text{info}} = 526.0$ s, $t_{\text{exe}} = 528.9$ s

(j) $t_{\text{info}} = 554.0$ s, $t_{\text{exe}} = 557.2$ s

(k) final realized route

Figure 14: Visualization of the vehicle routing in the first instance of Test case 2 ($n_{\text{new}} = 10$). The instance is the same is in Fig. 9, which was generated by the periodic rescheduling.

$t = 40$, and also has not reasoned before. At $t = 41$ s, a new urgent order to site 48 arrives (cf. the slope of the first new order in Fig. 13(b)). This new order causes a rapid decrease in state signal s_1, which then stimulates the agent to trigger a rescheduling procedure at the same time point. Site 48 receives a high priority in the new open-loop route – and changes the general layout of the route into a zigzagging *clockwise* spiral. In the periodic rescheduling, the order at site 48 is included in a rescheduling procedure the first time at $t_{\text{info}} = 80$ s. However, at this point, the vehicle has already moved further away from site 48, and the visit there is included in the route at a later stage. As a result, the tardiness of the vehicle at site 48 is 263.4 s (Fig. 10), which is the greatest single contribution to a total delay sum of 1128.8 s. On the other hand, in the routing where the decisions are made by the agent, the tardiness at site 48 is only 13.6 s (Fig. 15), and the total delay sum is 1041.1 s. The opposite traveling directions of the vehicle are still clearly visible in the open-loop routes in Figs. 9(j) (counter-clockwise) and 14(j) (clockwise).

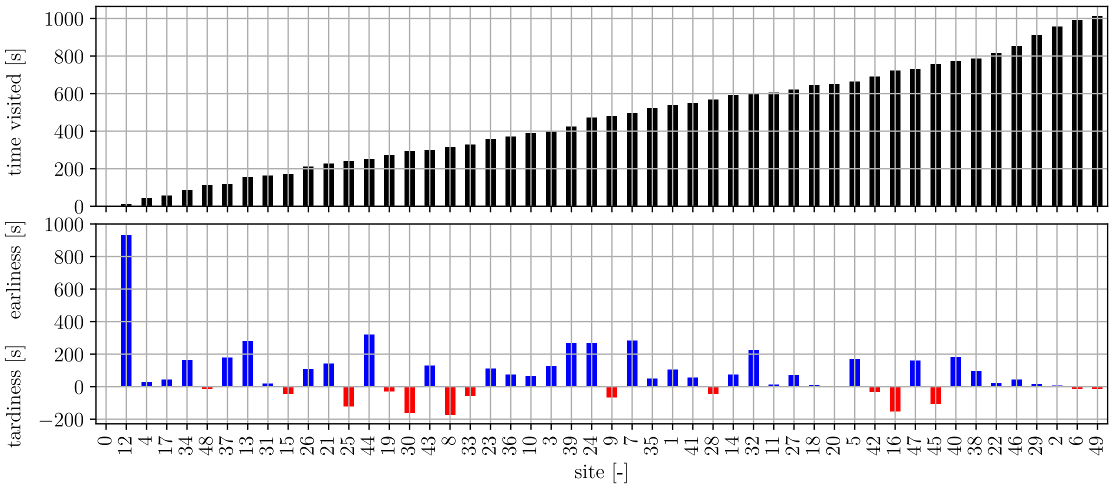


Figure 15: Visiting times and the tardiness/earliness of the vehicle at the sites, corresponding to the final route shown in Fig. 14(k).

5. Discussion

In this work, we have proposed an approach where a RL agent makes decisions on the timing and allocated computing time of rescheduling procedures. In Section 4, we trained agents to four test cases of a dynamic routing problem, in which the optimizer is based on ACO. The obtained results are, on average, better than those by the periodic and event-triggered rescheduling. Presumably, if the agent is given the freedom to also decide the horizon length and the optimization strategy[12] (see Section 2), we would obtain a further improvement in the closed-loop solutions (with the same computing budget). The argument for this is that, when a new decision is assigned to the agent, the resulting action space is a generalization of the previous action space.

We trained the agents to four test cases, each having different rates of arriving new orders, using the same settings. The resulting agents are still very different from each other both in terms of their neural network topology (Fig. 12) and behavior (Fig. 13). Each of these agents was able to adapt to their test case. The agents were also able to find favorable behavior that, to humans, may seem counter-intuitive (cf. the negative connection between signals s_2 and a_2). Although the test cases are simple, and yet artificial, similar adaption without prejudice could be seen in more complex industrial applications. A natural choice

[12]Assuming that the alternative strategy is more efficient or effective in some open-loop optimization problems than the used ACO algorithm.

for the next application would be a real online vehicle routing problem, of which the studied test case is a simplification, or a sequencing problem of a single-stage process with sequence-dependent changeover times. Future work should also evaluate the suitability of the approach to online planning and/or scheduling problems.

A challenge with the proposed approach is the computational cost of training the agent to an environment. The training of each agent, the results of which we reported in Section 4.3, required a computing time of around 20 hours on the scientific computing cluster, consisting of 20 CPUs. If we would have considered a larger computing budget[13] for the timespan, even higher computational cost would have been incurred. Thus, the applicability of the approach to problems with a larger computing budget is currently limited. The future work involves investigation of ways to mitigate the computational cost of training the agents. Possible ways include, first, the use of alternative reward functions, the value of which the agent receives more frequently. With the current reward function, an agent receives feedback of its actions only once, after interacting with the environment for the timespan of t_{span}. An alternative reward function could be, for example, the improvement in the objective function of the open-loop problem divided by the invested computing time. The agent would receive the value of this reward function after each rescheduling procedure. Second, despite neuroevolution is indicated, in the literature, to be a competitive alternative for deep RL (Such et al., 2017), the suitability of deep RL methods to the proposed approach should be investigated.

In this work, we used the NEAT algorithm to evolve the topology and node weights of the neural networks. We acknowledge that most likely some more recent neuroevolution (see the review paper by Stanley et al. (2019)) or deep RL methods are more suitable for the proposed approach. However, the NEAT algorithm yields, in general, neural network topologies with low complexity. This feature facilitates the interpretation of relationships between the state and action signals in the optimized neural network topology, which we did in Section 4.3. Arguably, the interpretation would not have been this convenient if the training was conducted by a deep RL method.

In Sections 1 and 2, we listed mathematical programming and a heuristic algorithm as the two alternative optimization strategies. The 'repertoire' of the agent can be tailored to a process, and may alternatively include two different mathematical programming models or two different heuristic algorithms. A key element in choosing the strategies is that they represent different trade-offs between *effectiveness* and *efficiency*. By effectiveness, we mean the ability of an optimization method to find highly competitive solutions and, by efficiency, the ability to find good solutions quickly. Alternatively, this pair of strategies could include one deterministic approach (e.g., a deterministic mixed-integer programming model) and another considering parameter uncertainty (e.g. a stochastic programming model). In this case the trade-off is between efficiency and the level of uncertainty anticipation. Further, the 'repertoire' could also comprise more than two optimization strategies, in the case of which each additional strategy expands the action space by one new signal (see Fig. 3).

6. Conclusions

We propose an online rescheduling approach, in which the decision-making of rescheduling timing and computing time allocation are made by a reinforcement learning (RL) agent. We demonstrate the approach on four dynamic routing problems, using the RL algorithm NEAT. When comparing the results on 30 randomized test instances of each problem, the proposed approach yields closed-loop solutions with smaller average delay sums than the periodic, event-triggered, and hybrid rescheduling (the key parameters of which have been tuned) in three out of four test cases by margins of 0.37 to 3.16%. In one test case, the hybrid rescheduling yields, on average, better results than the proposed approach by a margin of 3.97%. The future work will investigate the expansion of the RL agent's decisions to also include the optimization strategy and the length of the scheduling horizon.

[13]Here, were used a computing budget of $c_{\text{budget}} = 50$ s.

Acknowledgements

Financial support from the Academy of Finland, through project SINGPRO (decision numbers 313466 and 327533), is gratefully acknowledged. In addition, the authors would like to thank CSC – the Finnish IT Center for Science – for providing the computing resource for the project.

Appendix A. Parameters of the NEAT algorithm

This appendix lists the parameters we used in the neuroevolution of augmenting topologies (NEAT) algorithm in Section 4.3. The parameters, shown in the format of the neat-python configuration file (McIntyre et al., 2019), are:

```
[NEAT]
fitness_criterion                    = max
fitness_threshold                    = 0
pop_size                             = 80
reset_on_extinction                  = False

[DefaultGenome]
num_inputs                           = 4
num_hidden                           = 1
num_outputs                          = 2
initial_connection                   = partial_direct 0.5
feed_forward                         = True
compatibility_disjoint_coefficient   = 1.0
compatibility_weight_coefficient     = 0.6
conn_add_prob                        = 0.2
conn_delete_prob                     = 0.2
node_add_prob                        = 0.2
node_delete_prob                     = 0.2
activation_default                   = sigmoid
activation_options                   = sigmoid
activation_mutate_rate               = 0.0
aggregation_default                  = sum
aggregation_options                  = sum
aggregation_mutate_rate              = 0.0
bias_init_mean                       = 0.0
bias_init_stdev                      = 1.0
bias_replace_rate                    = 0.1
bias_mutate_rate                     = 0.7
bias_mutate_power                    = 0.5
bias_max_value                       = 30.0
bias_min_value                       = -30.0
response_init_mean                   = 1.0
response_init_stdev                  = 0.0
response_replace_rate                = 0.0
response_mutate_rate                 = 0.0
response_mutate_power                = 0.0
response_max_value                   = 30.0
response_min_value                   = -30.0
weight_max_value                     = 30
```

```
weight_min_value             = -30
weight_init_mean             = 0.0
weight_init_stdev            = 1.0
weight_mutate_rate           = 0.8
weight_replace_rate          = 0.1
weight_mutate_power          = 0.5
enabled_default              = True
enabled_mutate_rate          = 0.01

[DefaultSpeciesSet]
compatibility_threshold      = 3.0

[DefaultStagnation]
species_fitness_func         = max

max_stagnation               = 20

[DefaultReproduction]
elitism                      = 2
survival_threshold           = 0.2
```

These parameters are, in general, the same as in a single-pole-balancing example in neat-python. However, we have 1) changed the population size (see ‘pop_size’) to 80 (instead of 250 in the example), 2) the number of action signals (see ‘num_outputs’) to 2, since we have two signals in the action space, and 3) the fitness threshold of termination to 0, which corresponds to the delay sum of 0 s. For further information on these parameters, the reader is referred to the original paper by Stanley & Miikkulainen (2002) and the user manual of neat-python (McIntyre et al., 2019).

References

Arulkumaran, K., Deisenroth, M. P., Brundage, M., & Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, *34*, 26–38. doi:`https://doi.org/10.1109/MSP.2017.2743240`.

Atallah, R. F., Assi, C. M., & Khabbaz, M. J. (2018). Scheduling the operation of a connected vehicular network using deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, *20*, 1669–1682. doi:`https://doi.org/10.1109/TITS.2018.2832219`.

Aydin, M. E., & Öztemel, E. (2000). Dynamic job-shop scheduling using reinforcement learning agents. *Robotics and Autonomous Systems*, *33*, 169–178. doi:`https://doi.org/10.1016/S0921-8890(00)00087-7`.

Baldea, M., & Harjunkoski, I. (2014). Integrated production scheduling and process control: A systematic review. *Computers & Chemical Engineering*, *71*, 377–390. doi:`https://doi.org/10.1016/j.compchemeng.2014.09.002`.

Ben-Tal, A., El Ghaoui, L., & Nemirovski, A. (2009). *Robust optimization* volume 28. Princeton University Press.

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

Dong, Y., Maravelias, C. T., & Jerome, N. F. (2018). Reoptimization framework and policy analysis for maritime inventory routing under uncertainty. *Optimization and Engineering*, *19*, 937–976. doi:`https://doi.org/10.1007/s11081-018-9383-8`.

Dorigo, M., & Gambardella, L. M. (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on evolutionary computation*, *1*, 53–66. doi:`https://doi.org/10.1109/4235.585892`.

Ellson, J., Gansner, E. R., Koutsofios, E., North, S. C., & Woodhull, G. (2003). Graphviz and dynagraph static and dynamic graph drawing tools. In *Graph drawing software* (pp. 127–148). Springer-Verlag. doi:`https://doi.org/10.1007/978-3-642-18638-7_6`.

Grant, R. (2018). ACOpy [accessed on the 15th of october, 2018]. `https://github.com/rhgrant10/acopy`.

Grossmann, I. E., & Harjunkoski, I. (2019). Process systems engineering: Academic and industrial perspectives. *Computers & Chemical Engineering*, *126*, 474–484. doi:`https://doi.org/10.1016/j.compchemeng.2019.04.028`.

Gupta, D., & Maravelias, C. T. (2016). On deterministic online scheduling: Major considerations, paradoxes and remedies. *Computers & Chemical Engineering*, *94*, 312–330. doi:`https://doi.org/10.1016/j.compchemeng.2016.08.006`.

Gupta, D., Maravelias, C. T., & Wassick, J. M. (2016). From rescheduling to online scheduling. *Chemical Engineering Research and Design*, *116*, 83–97. doi:`https://doi.org/10.1016/j.cherd.2016.10.035`.

Harjunkoski, I., Maravelias, C. T., Bongers, P., Castro, P. M., Engell, S., Grossmann, I. E., Hooker, J., Méndez, C., Sand, G.,

& Wassick, J. (2014). Scope for industrial applications of production scheduling models and solution methods. *Computers & Chemical Engineering*, *62*, 161–193. doi:https://doi.org/10.1016/j.compchemeng.2013.12.001.

Hausknecht, M., Lehman, J., Miikkulainen, R., & Stone, P. (2014). A neuroevolution approach to general atari game playing. *IEEE Transactions on Computational Intelligence and AI in Games*, *6*, 355–366. doi:https://doi.org/10.1109/TCIAIG.2013.2294713.

Ikonen, T. J., & Harjunkoski, I. (2019). Decision-making of online rescheduling procedures using neuroevolution of augmenting topologies. In *Computer Aided Chemical Engineering* (pp. 1177–1182). Elsevier volume 46. doi:https://doi.org/10.1016/B978-0-12-818634-3.50197-1.

Katragjini, K., Vallada, E., & Ruiz, R. (2013). Flow shop rescheduling under different types of disruption. *International Journal of Production Research*, *51*, 780–797. doi:https://doi.org/10.1080/00207543.2012.666856.

McIntyre, A., Kallada, M., Miguel, C. G., & da Silva, C. F. (2019). neat-python [accessed on the 17th of june, 2019]. https://github.com/CodeReclaimers/neat-python.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, .

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G. et al. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*, 529. doi:https://doi.org/10.1038/nature14236.

Palombarini, J., & Martinez, E. (2012). Smartgantt–an interactive system for generating and updating rescheduling knowledge using relational abstractions. *Computers & Chemical Engineering*, *47*, 202–216. doi:https://doi.org/10.1016/j.compchemeng.2012.06.021.

Pattison, R. C., Touretzky, C. R., Harjunkoski, I., & Baldea, M. (2017). Moving horizon closed-loop production scheduling using dynamic process models. *AIChE Journal*, *63*, 639–651. doi:https://doi.org/10.1002/aic.15408.

Priore, P., Gómez, A., Pino, R., & Rosillo, R. (2014). Dynamic scheduling of manufacturing systems using machine learning: An updated review. *AI EDAM*, *28*, 83–97. doi:https://doi.org/10.1017/S0890060413000516.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015). Trust region policy optimization. In *International conference on machine learning* (pp. 1889–1897).

Šemrov, D., Marsetič, R., Žura, M., Todorovski, L., & Srdic, A. (2016). Reinforcement learning approach for train rescheduling on a single-track railway. *Transportation Research Part B: Methodological*, *86*, 250–267. doi:https://doi.org/10.1016/j.trb.2016.01.004.

Shin, J., Badgwell, T. A., Liu, K.-H., & Lee, J. H. (2019). Reinforcement learning–overview of recent progress and implications for process control. *Computers & Chemical Engineering*, *127*, 282–294. doi:https://doi.org/10.1016/j.compchemeng.2019.05.029.

Shin, J., & Lee, J. H. (2019). Multi-timescale, multi-period decision-making model development by combining reinforcement learning and mathematical programming. *Computers & Chemical Engineering*, *121*, 556–573. doi:https://doi.org/10.1016/j.compchemeng.2018.11.020.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T. et al. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, *362*, 1140–1144. doi:https://doi.org/10.1126/science.aar6404.

Stanley, K. O., Clune, J., Lehman, J., & Miikkulainen, R. (2019). Designing neural networks through neuroevolution. *Nature Machine Intelligence*, *1*, 24–35. doi:https://doi.org/10.1038/s42256-018-0006-z.

Stanley, K. O., D'Ambrosio, D. B., & Gauci, J. (2009). A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, *15*, 185–212. doi:https://doi.org/10.1162/artl.2009.15.2.15202.

Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation*, *10*, 99–127. doi:https://doi.org/10.1162/106365602320169811.

Subramanian, K., Maravelias, C. T., & Rawlings, J. B. (2012). A state-space model for chemical production scheduling. *Computers & chemical engineering*, *47*, 97–110. doi:https://doi.org/10.1016/j.compchemeng.2012.06.025.

Such, F. P., Madhavan, V., Conti, E., Lehman, J., Stanley, K. O., & Clune, J. (2017). Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. arXiv:1712.06567.

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

Touretzky, C. R., Harjunkoski, I., & Baldea, M. (2017). Dynamic models and fault diagnosis-based triggers for closed-loop scheduling. *AIChE Journal*, *63*, 1959–1973. doi:https://doi.org/10.1002/aic.15564.