https://doi.org/10.3311/PPci.14295
Creative Commons Attribution

Periodica Polytechnica Civil Engineering, 63(4), pp. 961–970, 2019

A Reinforcement Learning Motivated Algorithm for Process Optimization

Gyula Ábrahám[1*], Peter Auer[2], György Dósa[1], Tibor Dulai[1], Ágnes Werner-Stark[1]

[1] Faculty of Information Technology, University of Pannonia,
Egyetem str. 10., 8200, Veszprém, Hungary
[2] Department of Mathematics and Information Technology, University of Leoben,
Franz-Josef str. 18, 8700, Leoben, Austria
[*] Corresponding author, e-mail: abraham.gyula@virt.uni-pannon.hu

Received: 30 April 2019, Accepted: 14 August 2019, Published online: 27 September 2019

Abstract
In process scheduling problems there are several processes and resources. Any process consists of several tasks, and there may be precedence constraints among them. In our paper we consider a special case, where the precedence constraints form short disjoint (directed) paths. This model occurs frequently in practice, but as far as we know it is considered very rarely in the literature. The goal is to find a good resource allocation (schedule) to minimize the makespan. The problem is known to be strongly NP-hard, and such hard problems are often solved by heuristic methods. We found only one paper which is closely related to our topic, this paper proposes the heuristic method HH. We propose a new heuristic called QLM which is inspired by reinforcement learning methods from the area of machine learning. As we did not find appropriate benchmark problems for the investigated model. We have created such inputs and we have made exhaustive comparisons, comparing the results of HH and QLM, and an exact solver using CPLEX. We note that a heuristic method can give a "near optimal" solution very fast while an exact solver provides the optimal solution, but it may need a huge amount of time to find it. In our computational evaluation we experienced that our heuristic is more effective than HH and finds the optimal solution in many cases and very fast.

Keywords
process scheduling, reinforcement learning, scheduling, resource allocation

1 Introduction

We deal with an optimization problem that occurs e.g. in an office. A typical example is a municipal, immigration, or tax office where many documents are processed. We divide the documents into two types. For the first type of documents, the processing is easy, and any officer can do it. The second type of documents is processed by some officer, but then in a second phase need to be checked by the officer herself or by another officer.

Another typical example is the construction of a building. The construction can be split into several activities like foundation, walling and engineering work, etc., and there are precedence relations between them. Several groups of workers may work on the construction and the efficiency of these groups may vary for the different activities. The activities should be distributed among the working groups so that the construction is finished as soon as possible.

A similar example is the reconstruction of a family house. A team of workers with different skills comes to the house for the reconstruction. The reconstruction company wants to assign activities to the workers such that the work is finish as soon as possible.

We model the problem as scheduling tasks on unrelated machines with precedence constraints. In the first mentioned application (i.e. office management) the officers or workers correspond to the machines, and the processing steps of the documents or the building activities correspond to the tasks. For example, for the second type of documents, there are two tasks, the processing and the checking, and there is a precedence relation between them. The machines are unrelated, which means that the officers can handle the documents with different efficiencies.

Our problem belongs to the area of process scheduling. In process scheduling problems, we consider several processes, each consisting of several tasks, and a set of resources. The resources need to be assigned to the tasks of the processes such that several constraints are satisfied. In our study, the objective is to minimize the overall execution time of all processes (the makespan).

Process scheduling is a computationally hard problem. Such hard problems are often solved by heuristic methods. In this paper, we introduce a method that is motivated by reinforcement learning. In the following section, we introduce the relevant issues of scheduling to provide context for our research. Then we briefly introduce reinforcement learning and in particular Q-learning in Section 2. In Section 3, we define the scheduling problem we are considering. Our proposed algorithm is presented in Section 4, and it is computationally evaluated in Section 5. We finish the paper with our conclusions in Section 6.

1.1 Scheduling

In this section we give a very short overview about the scheduling problem we are considering. Generally, scheduling problems consist of two main components: the resources and the tasks (jobs). The goal is to allocate the resources to the tasks during certain time periods such that some objective is optimized.

The resources can be machines, processing units, humans, and so on. The tasks are the operations that must be completed using the resources. The tasks may have priority levels, earliest start times or due dates. These parameters influence the scheduling. There are several types of objectives, for example minimizing the lateness of tasks. An excellent overview of scheduling can be found in [1]. We mention here also the seminal works of Graham [2, 3].

1.1.1 Scheduling unrelated machines with precedence constraints

Scheduling unrelated machines minimizing the makespan is a classical problem denoted by $R||C_{\max}$. Here job j on machine M_i has processing time p_{ij}. Lenstra et al. [4] present a polynomial-time 2-approximation algorithm for this problem. The result is slightly improved by Shchepin and Vakhania [5] who gave a $(2 - 1/m)$-approximation for m machines. However, it is also proven in [4] that one cannot get any worst-case ratio better than 3/2 unless P = NP.

In this paper, we consider the case where there are also precedence constraints between some jobs. These precedence constraints are given by a directed graph G. The nodes of the graph correspond to the tasks and an oriented edge means that the predecessor must be finished before the successor can be started. We will consider only such precedence constraints that are represented by a chain graph, where the in-degree and the out-degree of any vertex is at most one. This means that the graph is a union of directed disjoint paths and isolated points.

Hermann et al. [6] and Liu and Yang [7] are the only papers we are aware of that deal with unrelated machine scheduling with precedence constraints represented by chain graphs. In [6] three kinds of lower bounds (LB1, LB2, and LB3) and several heuristics for this problem are introduced. A case study on 33 inputs compares the solutions of the heuristics to the maximum of the lower bounds.

Liu and Yang [7] propose a more efficient heuristic, and their method is also applicable to more general problems than those are considered here.

There are also several other papers on the topic, but we think [6] is the only one which is comparable to our paper, since only this paper considers chain-type precedence constraints (and provides results). In [7] the investigated problem is more general but the paper doesn't provide experimental results for precedence constraints of this type.

Recently, there is considerable interest in new models for scheduling unrelated machines. We list several such models below.

In [8] the authors consider a difficult model called Team Work Scheduling where from a set of workers we can create working teams and under several constraints it is given that the team is how effective to make some kind of job. For any job we choose a team. This choice of the team determines the processing time of the job, e.g. a bigger team can execute the job faster. Any team can work only one job at any time, and any job must be processed by some team. The teams should be chosen so that the given collection of jobs is made as soon as possible. This model generalizes the unrelated scheduling model.

Another related model is called Multiprofessor Scheduling [9]. In this model there are professors and instructors and some teaching activities should be assigned to them satisfying several constraints. The problem is a generalization of scheduling of unrelated machines since these professors have different abilities to deliver the lessons. For example, one is expert in Algebra and another professor is expert in Geometry. Moreover, for some lectures it is required that some instructors must be also present (to learn how to teach this subject). This case is very similar to a construction where for example, for painting the walls a master and a utility worker are needed. The paper gives complexity results and approximation algorithms.

A special model of Multiprofessor Scheduling is considered in [10], where restricted assignment scheduling is investigated with the presence of several kinds of resource constraints. In [10] paper also complexity results and approximation algorithms are given.

1.1.2 Related work using reinforcement learning for scheduling problems

Orhean et al. [11] introduced a reinforcement learning based scheduling method for a distributed cloud system. The goal was to optimize the performance of the cloud system by resource scheduling. Aydin and Öztemel [12] developed a dynamic scheduling technique that is trained by reinforcement learning. Stefán [13] applied the Q-learning method for a permutation flow shop problem, where the goal is to minimize the sum of idle times of the machines. In [14], Stefán gave a detailed description of the algorithm for the flow shop problem. We built our algorithm partly on his ideas. Gabel and Riedmiller [15] used Q-learning for a job shop scheduling problem by applying a neural network to approximate the Q-function. Shahrabi et al. [16] used reinforcement learning to enhance the performance of a dynamic job shop scheduling method. Further examples for applying reinforcement learning to scheduling problems are [17, 18, 19]. We note that Q-learning is rarely applied to scheduling problem, one such paper is [20].

2 Reinforcement learning

Reinforcement learning (RL) [21] is an area within machine learning which considers learning mid- to long-term policies. A policy is a mapping from the observed state of an agent to an action that the agent should perform. A distinguishing property of reinforcement learning problems is that the benefit of an action is typically not immediately observable but is received as a delayed reward. The objective of an optimal policy is to maximize the sum of received rewards.

Formally, a reinforcement problem is described by a Markov Decision Process (MDP) with a set of states S, a set of actions A, transition probabilities $p(s'|s, a)$ and a reward function $r(s, a)$. If action a is taken in state s, then reward $r(s, a)$ is received and the state transitions into state s' with probability $p(s'|s, a)$. The goal is to find a policy $\pi: S \rightarrow A$, that selects an action for each state such that the sum of expected discounted rewards is maximized. This sum of expected discounted rewards is the value of the policy, and for a state it can be calculated as

$$V^{\pi(s)} = r\left(s,\pi\left(s\right)\right) + \gamma \sum_{s'} p\left(s' \mid s,a\right) V^{\pi(s')}, \tag{1}$$

where $\gamma \in [0, 1)$ is the discount factor that determines the importance of future rewards. If the discount factor is 0, then the objective is greedy as it considers only the immediate reward. However, if the discount factor is close to 1, then the objective is to maximize the long-term rewards.

To optimize the policy the optimal Q-function is helpful,

$$Q^*\left(s,a\right) = r\left(s,a\right) + \gamma \sum_{s'} p\left(s' \mid s,a\right) V^*\left(s'\right), \tag{2}$$

$$V^*\left(s\right) = \max_a Q^*\left(s,a\right). \tag{3}$$

Here $V^*(s)$ is the value of the optimal policy, and Q^* is the expected value, if in state s at first action a is chosen, and then the optimal policy is followed. From the optimal Q-function the optimal policy is easily constructed by choosing in state s the action a that maximizes $Q^*(s, a)$. Thus, many reinforcement-learning algorithms try to estimate the optimal Q-function. An important algorithm is Q-learning [22], which can be derived as an iterative approximation scheme from Eq. (2) and Eq. (3).

2.1 Q-learning

Let (s_1, a_1, r_1), (s_2, a_2, r_2) be the sequence of observed states s_t, the chosen actions a_t, and the received rewards r_t, over time steps $t = 1, 2, \ldots$. Then Eqs. (1, 2) suggest the following update rule for an approximation of the optimal Q-function,

$$\begin{aligned} &Q_{t+1}\left(s_t,a_t\right) = \left(1-\alpha_t\right) Q_t\left(s_t,a_t\right) \\ &+\alpha_t\left(r_t + \gamma_t \max_a Q_t\left(s_{t+1},a\right)\right), \end{aligned} \tag{4}$$

$$Q_{t+1}\left(s,a\right) = Q_t\left(s,a\right), \tag{5}$$

where α_t is a suitable learning rate. The learning rate determines to what extent the newly acquired information overwrites the previous one. If $\alpha_t = 0$, then the agent does not learn anything; if $\alpha_t = 1$, then no previous information is kept. The discount factor $\gamma \in [0, 1)$ determines the importance of future rewards.

It can be shown [22], that with this update rule Q_t converges to Q^* if $\sum_t \alpha_t = \infty$, $\sum_t \alpha_t^2 = \infty$, and each state-action pair (s, a) appears infinitely often in the sequence (s_1, a_1), (s_2, a_2), … . The last condition is not controlled by the Q-learning update rule, but needs to be guaranteed by an appropriate exploration strategy that selects the actions a_t. There is a significant number of such exploration strategies, and a popular one is the ε-greedy strategy, which in each time step selects with probability $1 - \varepsilon_t$ the action a_t that maximizes $Q_t(s_t, a)$, and with probability ε_t selects an action chosen uniformly at random. Another popular strategy is Boltzmann exploration.

2.2 Boltzmann exploration

Boltzmann exploration selects action randomly according to the probabilities

$$p_t(a \mid s_t) = \frac{e^{\frac{Q(s_t,a)}{\tau_t}}}{\sum_a e^{\frac{Q(s_t,a)}{\tau_t}}}, \tag{6}$$

where τ_t is a decreasing temperature parameter. For large τ_t the action selection is almost uniformly random, whereas for τ_t close to 0 the selection is almost deterministic. Our algorithm will use Boltzmann exploration to try out different schedules.

2.3 Episodic Q-learning

An MDP is episodic if it is restarted after each episode $(s_1, a_1, r_1), (s_2, a_2, r_2), \ldots, (s_n, a_n, r_n), s_{n+1}$ where s_{n+1} is the terminal state of the episode. The end of an episode is determined either by the number of steps taken or by a stopping condition. For episodic MDPs the Q-learning update can be performed only once after each episode, and it is particularly simple if the states visited in an episode are all distinct.

Then for $k = 1, \ldots, n$, $Q_{t+1}(s_k, a_k) = (1 - \alpha_t)\, Q_t(s_k, a_k)$
$+ \alpha_t(r_t + \gamma_t \max_a Q_t(s_{k+1}, a))$
and
$Q_{t+1}(s, a) = Q_t(s, a)$ for (s, a) not among the (s_k, a_k).
Our algorithm will use episodic updates of the Q-function.

3 The considered problem

Below we define our scheduling problem in detail. Given are n tasks (called also jobs) denoted by $T_1, \ldots T_n$ and m machines (also called resources) denoted by $R_1, \ldots R_m$. The machines are unrelated, such that the processing time of a task depends on the machine it is assigned to. Let p_{ir} denote the processing time of task T_i if it is processed by machine R_r. It is possible that a machine is not suitable to process some job, in this case the processing time is infinite. There are precedence constraints among the jobs, which are defined by a directed graph G. We restrict our attention to problems where G is a chain graph partitioned into vertex disjoint paths. These paths we call processes. Some processes may contain only one task.

At any time, any machine can process at most one task, and the execution of a task cannot be preempted. The objective of a schedule is to minimize the makespan.

As illustration we give a small example in Fig. 1: There are two processes ($Proc_1$, $Proc_2$) and four machines (R_1, R_2, R_3, R_4). $Proc_1$ consists of tasks T_1 and T_2, while $Proc_2$ consists of tasks T_3, T_4 and T_5. Fig. 1 also shows which machines can execute the individual tasks.

In Fig. 2, the tasks are scheduled on the machines in certain time slots such that the precedence constraints are satisfied. The processing times of the tasks on the respective machines ($p_{1,1} = 3, p_{2,2} = 8, p_{3,3} = 6, p_{4,2} = 2, p_{5,4} = 10$) are given in parentheses after the names of the tasks.

The schedule in Fig. 2 results in a makespan of $3 + 8 + 2 + 10 = 23$ time units. In Fig. 3 we show another schedule with a reduced makespan of $6 + 2 + 10 = 18$ time units, which is in fact an optimal schedule.

In the next section we introduce a method which schedules the tasks one by one in a greedy fashion according to some order of the tasks. A good order of the tasks is calculated by a method inspired by reinforcement learning.

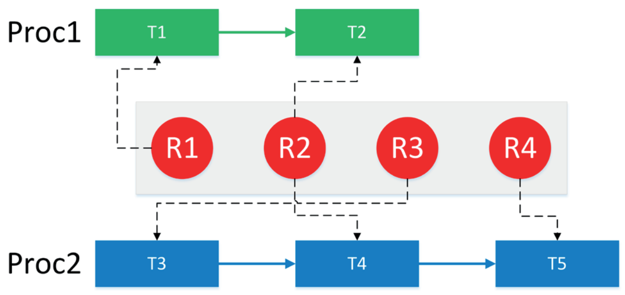

Fig. 1 Example of processes, tasks, and machines

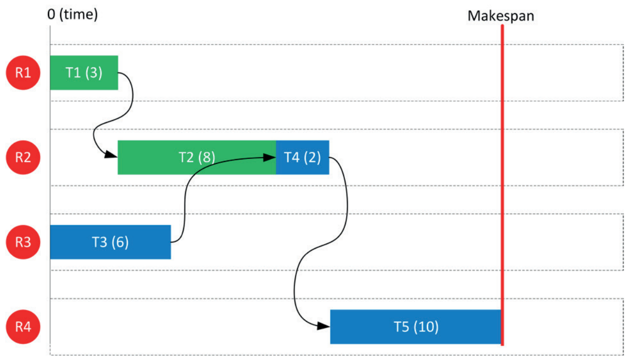

Fig. 2 A feasible schedule for the processes

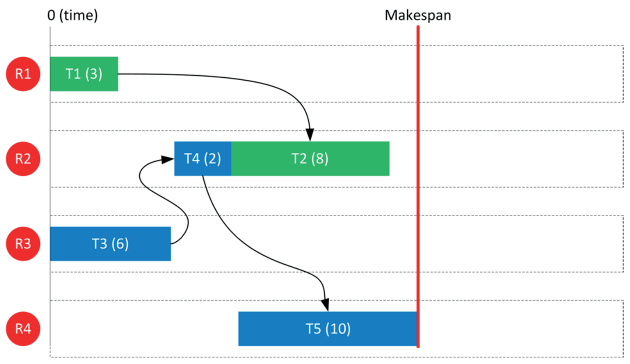

Fig. 3 Optimal schedule for the processes

4 The proposed method

For the scheduling problem described in the previous section, we propose the new heuristic QLM (Q-Learning Motivated heuristic). We want to find a "good" permutation (or order) of tasks such that allocating the tasks greedily in this order to the machines will result in a small makespan. This greedy algorithm works as follows. The next task will always be assigned to the machine that will finish the task the earliest. This is the appropriate version of Graham's list scheduling algorithm [2, 3]. To find a good ordering of the tasks, we follow the approach of [17] to compute values $Q(i, j)$ for each pair of tasks T_i and T_j that represent how advantageous it is if task T_i is directly followed by task T_j in the list of the tasks. For calculating the values $Q(i, j)$ we devise an algorithm that is motivated by reinforcement learning, in particular Q-learning [22].

Our method can be summarized as follows:

(1) determine a good order for the tasks based on the Q-values;

(2) schedule the tasks greedily according to the determined order;

(3) calculate the makespan of the schedule;

(4) calculate the reward based on the current value of the makespan and the best makespan so far;

(5) update the Q-values.

The above (1)–(5) steps are repeated for a certain number of iterations. In the next section we describe the algorithm in detail.

4.1 Generating a task sequence from the Q-values

We assume that values $Q(i, j)$ have been calculated for all tasks T_i and T_j. For notational convenience we assume that there is a task T_0 with processing time 0 which will be always the first task in the order. Then the $Q(0, i)$ can be used to select the first task of a sequence. A task sequence $L_t = (i_0, i_1, \ldots, i_n)$, $i_0 = 0$, is built incrementally and randomly based on the values $Q(i, j)$ using Boltzmann exploration. Let $A = \{j_1, \ldots, j_l\}$ be the set of indices j such that, according to the precedence constraints, task T_j is eligible for execution after completion of tasks $T_{i_1}, \ldots, T_{i_k}$. Then i_{k+1} is selected among the $j \in A$ with probability

$$p_j = \frac{e^{\frac{Q(i_k, j)}{\tau}}}{B}, B = \sum_{i \in A} e^{\frac{Q(i_k, j)}{\tau}}.$$

4.2 Calculating the Q-values

Our heuristic QLM (Q-Learning Motivated heuristic) is motivated by considering the sequential selection of an ordering as a reinforcement learning problem. The states would be initial orderings of tasks $T_{i_1}, \ldots, T_{i_k}$, and the possible actions would be the tasks that are eligible as the next task $T_{i_{k+1}}$, obeying the precedence constraints.

Since such a state space would be exponential in the number of tasks, we use a crude approximation of the states by representing each initial ordering $T_{i_1}, \ldots, T_{i_k}$ only by its last task T_{i_k}. While a lot of information is lost by using such approximate states, our experiments indicate that still very good orderings of the tasks can be found. The obvious reward would be the negative makespan, thereby punishing schedules with large makespan. However, since the makespan becomes available only after the last task is scheduled, the Q-learning algorithm would receive a (negative) reward only for scheduling the last task. Then the Q-learning updates would propagate these rewards to the other states, which would take many update iterations. Therefore, we instead use episodic Q-learning, where the Q-values are updated all together only after a schedule is finished, and we use a modified reward signal to accelerate convergence. Since we are not interested in the exact Q-values but use only the relative order of Q-values to select the next task in the schedule, we define the rewards relative to the current best makespan.

Our algorithm proceeds in episodes $t = 1, 2, \ldots,$ where in each episode t a sequence L_t of the tasks and a corresponding schedule is calculated, based on the current Q-values $Q_t(i, j)$. For the first episode all Q-values are initialized to 0, i.e. $Q_1(i, j) = 0$. For each episode t the algorithm proceeds as follows:

- Calculate a task sequence $L_t = (i_{t,0}, i_{t,1}, \ldots, i_{t,n})$ based on the Q-values Q_t as in Section 3.1.
- Calculate a schedule by greedily scheduling the tasks in the order of sequence L_t.
- Obtain the makespan z_t of this schedule.
- The reward is calculated as follows. Let $Z = \max_{s<t} z_t$. Let $r_t = 10$ if $z_t < Z$; $r_t = 0$, if $z_t = Z$; $r_t = -1$ if $z_t > Z$.
- Update the Q-values: for $k = 1, \ldots, n$ let $Q_{t+1}(i_{t,k-1}, i_{t,k})$ $= (1-\alpha)\, Q_t(i_{t,k-1}, i_{t,k}) + \alpha(r_t + \gamma \cdot \max_{i \in B_k} Q_t(i_{t,k}, i))$,

where B_k is the set of (indices of) tasks that can directly follow task $T_{i_{t,k}}$. There is no change in the other Q-values.

We give the flowchart of the algorithm on Fig. 4.

The result of the algorithm is the schedule with the minimum makespan observed over all iterations. In our computational experiments we used the parameters $\alpha = 0.8$ and $\gamma = 0.7$. For the selection of a sequence L_t by Boltzmann exploration we use the parameter $\tau_t = \tau_0 \times d_t$ for some constant $0 < d < 1$. The method is not very sensitive in respect to the choice of d, we set $d = 0.99$.

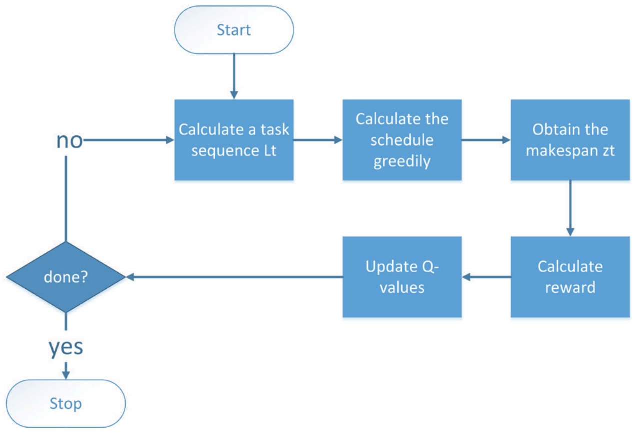

Fig. 4 Flowchart of the algorithm

Table 1 Processing times

Processing times	T_1	T_2	T_3	T_4
M_1	5	1	8	7
M_2	2	10	4	3

Table 2 The Q-values after 299 iterations

Q_{299}	T_0	T_1	T_2	T_3	T_4
T_0	-	–3.333	5.810	14.036	-
T_1	-	-	4.979	–1.692	7.673
T_2	-	–2.102	-	0.065	1.996
T_3	-	5.893	12.448	-	–1.000
T_4	-	10.000	-	–1.000	-

4.3 A toy example

For illustration we give a small example for the calculations in the $(t + 1)$-th iteration of the algorithm. We consider two machines and four tasks, the processing times are given in Table 1. There is only one precedence constraint: $T_2 \rightarrow T_4$ which means that T_2 must be finished before T_4 starts.

We applied our algorithm to this data. After $t = 299$ iterations we arrived at the Q-values given in Table 2.

The best makespan so far is $Z = 7$, the current makespan is $z_{299} = 8$, and the current order of the tasks is $L_{299} = (1, 2, 3, 4)$.

We show the calculation for iteration $t + 1 = 300$. First, we determine the new list: $A^1_{300} = \{1, 2, 3\}$. Task T_4 has a predecessor and is not contained in this set. Boltzmann exploration uses the values of $Q_{299}(0, 1)$, $Q_{299}(0, 2)$, $Q_{299}(0, 3)$ for the calculation. Let us assume that T_2 is chosen as the first element of the list.

Now $A^2_{300} = \{1, 3, 4\}$. Boltzmann exploration uses $Q_{299}(2, 1)$, $Q_{299}(2, 3)$, $Q_{299}(2, 4)$ for the calculation, and we assume that T_3 is chosen as the second element of the list.

For the choice of the third element, $A^3_{300} = \{1, 4\}$, $Q_{299}(3, 1)$, $Q_{299}(3, 4)$ are used for the calculation, and T_4 may be chosen as the third element of the list.

Finally, $A^4_{300} = \{1\}$. Thus, the generated list is $L_{300} = (2, 3, 4, 1)$. We create the corresponding schedule, and the makespan will be $z_{300} = 7$. Since the value of Z is also 7, $r_{300} = 0$.

During the update procedure the Q-values are updated: from the $Q_{299}(i, j)$ the next $Q_{300}(i, j)$ are calculated, $Q(0, 2)$, $Q(2, 3)$, $Q(3, 4)$, $Q(4, 1)$.

4.4 Some notes

In our calculations we used the parameters $\alpha = 0.8$ and $\gamma = 0.7$; these values were efficient in the calculations. We experimented also with other parameter values without a noticeable difference in efficiency. Only if both parameters were set to 1 the algorithm did not converge. We ran the algorithm 10 times for 2000 iterations in each and report the best result. For the Boltzmann exploration the initial value of τ was chosen to be 2000.

5 Computational experiments

First, we considered a small example that was solved both in [6] and [7]. In this example there are 7 tasks (T1–T7), 3 precedence constraints (T1 → T3 → T7; T2 → T6), and the corresponding processing times. This small example is optimally solved in [7], the optimal makespan is 13, and the heuristic method of [6] gives a solution with makespan 15. For this example, our Q-Learning based method was also able to find the optimal solution.

For further experiments, we followed [6] and [7]. In [6], another 33 scheduling problems were solved, but unfortunately the complete data of these instances are not provided. In [7], the authors define problem classes by giving the number of machines, the number of tasks, and the number of precedence constraints. The processing times of the tasks are randomly generated from some discrete uniform distribution. However, the generation of the precedence constraints is not given. We combined the two procedures for generating inputs as follows.

We have chosen some of the 33 examples considered in [6], where we have chosen examples that differ significantly from each other. In particular, we have chosen #1, #2, #5 from the small examples in [6], and one of the biggest examples, namely #28. (We kept the original instance numbering.) We considered these examples as input types. That is, we have created four input classes. In the first class (Class #1) the number of tasks is $n = 14$, the number of machines is $m = 8$ and the number of precedence constraints is $NC = 5$. The corresponding data for the other three classes #2, #5, and #28 can be seen in Table 3. For each type the processing times are chosen uniformly at random from {1, 2, …, 10}.We continued our investigations so that first we generated one input from each input class. For each input we ran our heuristic QLM, calculated the lower bounds LB1 and LB2 form [6], and also tried to calculate an optimal schedule using the CPLEX solver. The results are shown in Table 3.

Recall that n is the number of tasks, m is the number of machines and NC denotes the number of precedence constraints. For each input we executed 10 independent runs of QLM and the number of iterations was chosen as 2000 in each run. The total running time of our algorithm always was below 1 second except in case of example #28, where it remained below 4 seconds. Therefore, we do not give the running time of our algorithm QLM in the above table. For determining the optimal solution, we used the mixed integer formulation of [7] and we used the CPLEX solver. We note that for running CPLEX we used the NEOS homepage [23]. CPLEX is a commercial solver for linear and mixed integer programs. CPLEX is considered one of the best such solvers, possibly only outperformed by Gurobi (which is also a commercial solver) in some cases. For more information see the homepages at [24, 25].

We believe that comparing our algorithm with CPLEX is very informative. As we have mentioned above, we have found only two papers ([6] and [7]) that deal with similar scheduling problems like ours. The algorithms of these two papers are called HH and SS. Comparison with these two algorithms would not be completely fair, as they are defined for somewhat different problems. The difference is in the type of the precedence relations, as they deal with more general cases. We deal only with such precedence relations that form disjoint paths. This is a very typical case in some real-life problems, as we discussed in the Introduction. Thus, we believe that the most truthful comparison is with an exact solver. However, we also compare our algorithm to algorithm HH, as discuss below.

Table 3 Numerical examples

Example #	#1	#2	#5	#28
n	14	28	27	74
m	8	7	4	19
NC	5	8	1	10
LB1	10	11	7	4
LB2	4	9	18	5
CPLEX LB	10	11	7	5
CPLEX UB	10	11	18	10
QLM	10	11	18	5
QLM-freq	10/10	3/10	1/10	9/10
QLM-gap	0 %	0 %	0 %	0 %

For each example we allowed 1000 seconds for the run of the CPLEX solver. For some examples this was to enough to find the optimal solution, and in Table 3 we give the lower and upper bound (CPLEX_LB, CPLEX_UB) on the solution that was determined by CPLEX.

For our QLM algorithm we give the best solution found (QLM), in how many of the 10 runs an optimal solution was found (QLM-freq), and the percentage by which the best QLM solution exceeds the optimal solution (QLM-gap).

We mention, as described above, that the input examples we solve are not necessarily the same examples that were solved by the HH algorithm in [6]. Reason is that [6] gives only the number of precedence constraints but not sufficient detail to replicate the experiments. Nevertheless, we note that the gaps of Algorithm HH for some inputs chosen from the four input classes are 0 %, 15 %, 8 %, and 8 %, respectively.

5.1 Evaluations of the results

The first example (i.e. example #1) was easy to solve for both CPLEX and our QLM algorithm. CPLEX could provide the optimal solution in less than one second, and our algorithm also found an optimal solution very fast in all of the 10 runs. Also, algorithm HH could find the optimal solution according to [6].

Example #2 was harder, the gap of algorithm HH (for a similar input from this type) is 15 % while the gap of QLM is 0 %. (If the gap is 0, this means the algorithm could find the optimal solution.) Our algorithm could find the optimal solution in 3 out of 10 runs. The problem was also harder for CPLEX, we note that it found an optimal solution in 11 seconds.

Example #5 is significantly harder. Our algorithm could find the optimal solution only once among the 10 runs. This example also was hard for CPLEX. It found the optimal solution within 1000 seconds but could not verify its optimality, since the lower bound that CPLEX found was smaller than the optimal solution.

Example #28 is a much bigger example. Surprisingly it was not hard for the QLM algorithm (the optimal solution was found in 9 out of 10 runs). This example was really hard for CPLEX. Its best lower bound equals the optimum value (i.e. 5), but the best solution it found was 10, so the gap of CPLEX is big here. Note that the model of the problem (which is given in [7]) has 7030 discrete variables and 412°287 linear constraints. The CPLEX solver would provide a better solution if it is allowed more time. We emphasize that the running time of our algorithm remains very low, 10 runs (2000 iterations in each) needed 4 seconds in total.

5.2 More detailed investigations

For a more investigation we considered the four input classes #1, #2, #5, and #28 again and generated 10 more examples for each class.

For Class #1, our heuristic QLM found an optimal solution for all 10 examples. The optimal makespans are as follows: 9, 5, 9, 8, 8, 4, 9, 17, 7, and 10. For each example QLM found the optimal solution in all 10 out of 10 runs. CPLEX could also find the optimal solution for each example very quickly (in less than 0.1 sec). These experiments completely agree with the results in Table 3. The algorithm finds optimal solution 10/10 times. (There are 10 inputs and for each input we performed the algorithm 10 times.)

Let us consider what are the orders of the tasks in the 10 optimal solutions. These orders are listed below.

1 8 4 5 0 11 10 7 3 6 9 12 13 2
10 2 4 0 11 3 7 8 1 6 9 12 5 13
0 8 7 10 5 1 3 6 2 4 11 9 12 13
1 5 8 10 2 7 11 4 0 3 6 9 12 13
4 11 0 2 10 5 7 3 1 8 6 9 12 13
5 1 4 7 0 11 2 10 8 3 6 9 12 13
5 8 2 0 7 4 1 11 3 6 10 9 12 13
8 11 5 0 1 7 3 4 6 10 2 9 12 13
4 0 2 1 11 3 6 9 7 10 5 12 8 13
4 0 2 1 10 7 3 8 6 11 5 9 12 13

Without taking far-reaching conclusion, we conclude that

- the orders are not the same,
- still there are some similarities like 13 is almost always the last task, and 12 and 13 are often directly after each other, etc.

For Class #2, the best solutions found by algorithm QLM, the frequencies of QLM finding the best solutions, and all other data are shown is Table 4.

For example, for the first input within Class #2 among the 10 runs the makespan of the schedule was 9 six times and in the remaining four times it was weaker (i.e. it was 10). For the same input LB1 = 8 and LB2 = 7. This means that the optimal value is at most 9 and at least 8. Still remained a gap with the previous information we cannot decide what is the optimum. The same result was provided by CPLEX, its best solution has value 9 and the lower bound provided by CPLEX is 8. *The consequence is that QLM provided the same result like CPLEX (in case of the first input within Class #2) but much faster.* Since this is the most relevant information we wrote this to rows by bold letters.

The lower bound of CPLEX is worse than LB in the next cases: 2-th, 3-rd, 5-th, 7-th, 9-th. In the case of the 6-th input the upper and also the lower bounds equal to 7 which means that the found solutions are optimal. If we compare the result of QLM with LB, we can realize that our solution is also optimal in the 7-th and 9-th case, but this could not be verified by CPLEX. Regarding Class #3 we provide the appropriate results in Table 5.

We can verify the optimality of QLM (comparing it to LB) for the 5-th and 7-th inputs. CPLEX never gave better solution than QLM but it provided slightly worse result for the 3-rd input. It is worth to mention that the gaps between the lower and upper bounds of CPLEX are really huge. For example, for the first input these values are 5 and 18 while LB is 16, a much better lower bound.

We conclude that QLM gives the same or better bounds than CPLEX and much faster. These investigations confirm our first impression about this input class which we gave in Table 3. In Class #4 the results are in Table 6.

Table 4 Numerical results for Class #2

	1	2	3	4	5	6	7	8	9	10
QLM	9	8	8	9	12	7	7	8	7	11
QLM-freq	6	10	10	3	6	2	6	7	9	6
LB	8	7	7	8	9	7	7	6	7	9
CPLEX LB	8	6	5	8	7	7	6	6	5	9
CPLEX UB	9	8	8	9	12	7	7	8	7	11

Table 5 Numerical results for Class #3

	1	2	3	4	5	6	7	8	9	10
QLM	18	20	17	19	16	20	17	19	21	17
QLM-freq	10	9	7	8	2	10	1	8	9	10
LB	16	18	15	18	16	19	17	18	19	16
CPLEX LB	5	8	5	7	6	8	5	8	6	6
CPLEX UB	18	20	18	19	16	20	17	19	21	17

Table 6 Numerical results for Class #4

	1	2	3	4	5	6	7	8	9	10
QLM	6	5	6	6	6	5	6	6	5	5
QLM-freq	6	1	7	10	10	1	10	8	1	3
LB	6	5	5	5	5	5	5	5	5	5
CPLEX LB	6	4	4	5	5	4	3	5	5	4
CPLEX UB	13	9	11	9	9	15	9	9	8	10

Here QLM provides the same solution like LB in half of the cases, in the other half cases it is bigger only by one. So, *we can verify the optimality of QLM in 5 cases* out of 10 cases, but *this input type is really hard for CPLEX as it provided very weak upper bounds*. This conclusion is very similar to that we got for the input class in Table 3.

Finally, we mention that all data (processing times and precedence relations) and all results for QLM and CPLEX are gladly provided by the authors via e-mail.

6 Conclusions

In this study we investigate a special case of a scheduling problem that exists in the literature: we deal with scheduling unrelated machines with precedence constraints. The specialty is in the type of precedence constraints: they form short disjoint paths. We argue that this special case is relevant and interesting, e.g. in civil engineering. There are only a few algorithms in the literature that are relevant for related models. Our algorithm is the first one for this specific case and thus a valuable contribution.

We have shown how a Q-learning motivated method can be applied for solving this process scheduling problem. It is not trivial to make Q-learning applicable in this setting, since the large state space makes a generic application of reinforcement learning infeasible.

With our simplification of the state space our method is shown to be quite powerful. It is an interesting open question how this technique can be applied to other scheduling problems.

Currently there are no results regarding the worst-case approximation ratio for the type of scheduling problems we have considered in this paper. This is also an interesting research topic for future work.

Acknowledgement

The authors acknowledge the financial support of Széchenyi 2020 under EFOP-3.6.1-16-2016-00015, and also Stiftung Aktion Österreich-Ungarn 99öu5. Moreover Dósa acknowledges the financial support of NKFIH under the grant SNN 129364.

References

[1] Pinedo, M. L. "Scheduling. Theory, Algorithms and Systems", 4th ed., Springer, Boston, MA, USA, 2012. https://doi.org/10.1007/978-1-4614-2361-4

[2] Graham, R. L. "Bounds for certain multiprocessing anomalies", The Bell System Technical Journal, 45(9), pp. 1563–1581, 1966. https://doi.org/10.1002/j.1538-7305.1966.tb01709.x

[3] Graham, R. L. "Bounds on Multiprocessing Timing Anomalies", SIAM Journal on Applied Mathematics, 17(2), pp. 416–429, 1969. https://doi.org/10.1137/0117039

[4] Lenstra, J. K., Shmoys, D. B., Tardos, E. "Approximation algorithms for scheduling unrelated parallel machines", Mathematical Programming, 46(1–3), pp. 259–271, 1990. https://doi.org/10.1007/BF01585745

[5] Shchepin, E. V., Vakhania, N. "An optimal rounding procedure gives a better approximation for scheduling unrelated machines", Operations Research Letters, 33(2), pp. 127–133, 2005. https://doi.org/10.1016/j.orl.2004.05.004

[6] Herrmann, J., Proth, J.-M., Sauer, N. "Heuristics for unrelated machine scheduling with precedence constraints", European Journal of Operational Research, 102(3), pp. 528–537, 1997. https://doi.org/10.1016/S0377-2217(96)00247-0

[7] Liu, C., Yang, S. "A heuristic serial schedule algorithm for unrelated parallel machine scheduling with precedence constraints", Journal of Software, 6(6), pp. 1146–1153, 2011. https://doi.org/10.4304/jsw.6.6.1146-1153

[8] Dosa, Gy., Kellerer, H., Tuza, Zs. "Team Work Scheduling", In: MATCOS-16, Middle-European Conference on Applied Theoretical Computer Science, Koper, Slovenia, 2016, pp. 80–82. [online] Available at: http://matcos16.iam.upr.si/en/resources/files//published-material/is2016volumeh---matcos.pdf [Accessed: 10 August 2019]

[9] Dosa Gy., Tuza, Zs. "Multiprofessor scheduling", Discrete Applied Mathematics, 234, pp. 195–209, 2018. https://doi.org/10.1016/j.dam.2016.01.035

[10] Dosa, Gy., Kellerer, H., Tuza, Zs. "Restricted assignment scheduling with resource constraints", Theoretical Computer Science, 760, pp. 72–87, 2019. https://doi.org/10.1016/j.tcs.2018.08.016

[11] Orhean, A. I., Pop, F., Raicu, I. "New scheduling approach using reinforcement learning for heterogeneous distributed systems", Journal of Parallel and Distributed Computing, 117, pp. 292–302, 2018. https://doi.org/10.1016/j.jpdc.2017.05.001

[12] Aydin, M. E., Öztemel, E. "Dynamic job-shop scheduling using reinforcement learning agents", Robotics and Autonomous Systems, 33(2–3), pp. 169–178, 2000. https://doi.org/10.1016/S0921-8890(00)00087-7

[13] Stefán, P. "Flow-shop scheduling based on reinforcement learning algorithm", Production Systems and Information Engineering, 1, pp. 83–90, 2003. [online] Available at: http://www.ait.iit.uni-miskolc.hu/files/2003/07-PSAIE2003-Stefan-83-90.pdf [Accessed: 10 August 2019]

[14] Stefán, P. "Combined Use of Reinforcement Learning And Simulated Annealing: Algorithms and Applications", PhD Thesis, University of Miskolc, 2003. [online] Available at: http://midra.uni-miskolc.hu:80/?docId=5607 [Accessed: 10 August 2019]

[15] Gabel, T., Riedmiller, M. "Adaptive reactive job-shop scheduling with reinforcement learning agents", International Journal of Information Technology and Intelligent Computing, 24(4), pp. 14–18, 2008. Available at: http://tgabel.de/cms/fileadmin/user_upload/documents/Gabel_Riedml_ITIC-07.pdf [Accessed: 10 August 2019]

[16] Shahrabi, J., Adibi, M. A., Mahootchi, M. "A reinforcement learning approach to parameter estimation in dynamic job shop scheduling", Computers & Industrial Engineering, 110, pp. 75–82, 2017. https://doi.org/10.1016/j.cie.2017.05.026

[17] Zhang, W., Dietterich, T. G. "A reinforcement learning approach to job-shop scheduling", In: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, Montreal, QC, Canada, 1995, pp. 1114–1120. [online] Available at: https://www.ijcai.org/Proceedings/95-2/Papers/013.pdf [Accessed: 10 August 2019]

[18] Zweben, M., Davis, E., Daun, B., Deale, M. J. "Scheduling and rescheduling with iterative repair", IEEE Transactions on Systems, Man and Cybernetics, 23(6), pp. 1588–1596, 1993. https://doi.org/10.1109/21.257756

[19] Huang, Z., van der Aalst, W. M. P., Lu, X., Duan, H. "Reinforcement learning based resource allocation in business process management", Data & Knowledge Engineering, 70(1), pp. 127–145, 2011. https://doi.org/10.1016/j.datak.2010.09.002

[20] Ye, Y., Ren, X., Wang, J., Xu, L., Guo, W., Huang, W., Tian, W. "A New Approach for Resource Scheduling with Deep Reinforcement Learning", Cornell University, Ithaca, NY, USA, 2018. [pdf] Available at: https://arxiv.org/abs/1806.08122v1 [Accessed: 10 August 2019]

[21] Sutton, R. S., Barto, A. G. "Reinforcement learning, An introduction", 2nd ed., The MIT Press, Massachusetts, USA, 2018.

[22] Watkins, C. J. C. H., Dayan, P. "Q-learning", Machine Learning, 8(3–4), pp. 279–292, 1992. https://doi.org/10.1007/BF00992698

[23] Wisconsin Institutes for Discovery "NEOS Server" [online] Available at: https://neos-server.org [Accessed: 10 August 2019]

[24] Wikipedia "CPLEX" [online] Available at: https://en.wikipedia.org/wiki/CPLEX [Accessed: 10 August 2019]

[25] Gurobi Optimization, LLC "Gurobi Optimizer" [online] Available at: https://www.gurobi.com/ [Accessed: 10 August 2019]