Proceedings of the 18th Conference on Computer Science and Intelligence Systems pp. 689–697
DOI: 10.15439/2023F4564
ISSN 2300-5963 ACSIS, Vol. 35

An Enhancement of Reinforcement Learning by Scheduling with Learning Effects

Radosław Rudek
General Tadeusz Kościuszko Military University of Land Forces
Czajkowskiego 109, 51-147 Wrocław, Poland
Email:rudek.radoslaw@gmail.com

***Abstract*—This paper present results, which reveal that approaches obtained for scheduling problems with learning effects can be successfully used to improve the quality of machine learning methods. It is illustrated by modelling some aspects of Q-learning agents as scheduling problems with the learning effect, and constructing sequencing and dispatching algorithms, which take into account the existence of learning. Their application to determine the sequence of tasks processed by Q-learning agents can visibly speed up their convergence to an optimal strategy. Furthermore, we show that a dispatch of tasks according to the longest processing time algorithm for parallel computing can be replaced by a more efficient procedure, if agents can learn. The numerical analysis reveals that our approach is efficient, robust and only marginally dependents on a learning model and an accurate approximation of task processing times.**

I. Introduction

THE LEARNING effect takes place in typical human activity environments or in automatized manufacturing, where a human support for machines is needed during activities such as operating, controlling, setup, cleaning, maintaining, failure removal, etc. It was also observed that performances (objectives) of an industrial system can be essentially improved if the learning ability is utilized (see [1]). It can be done by determining the sequence of processed jobs, which takes into consideration not only the given objectives, but also the presence of learning, i.e., decreasing processing times/costs of jobs. Thereby, time/cost objectives (e.g., the maximum completion time) can be additionally improved (in a specified range) by the sequence (schedule) of jobs (e.g., [2], [3], [4]). In other words, additional benefits from learning can be gained. It is worth highlighting this scheduling approach does not interfere a system nor require any changes of its structure. Therefore, it is a significant advantage, which makes this non-invasive method universal and applicable to improve (optimize) different systems, where learning is present and a sequence of processed jobs can be (at least partially) controlled.

Although scheduling problems with the learning effect have attracted particular attention in research society (e.g., [1], [5], [6], [7], but surprisingly, there was no attempt to apply them to enhance machine learning algorithms, except our idea. However, from the perspective of the discussed utilization of learning by scheduling, there is no difference if the reduction in time or cost required to process a job is the result of human learning or machine learning. Thereby, the mentioned additional benefits from learning can be also gained for dynamically changing systems, which use machine learning. Such illustrative example will be shown in this paper.

Machine learning methods or intelligent agents ([8]) very often act like human, especially in the context of adaptation, self-improvement and autonomous learning, which is present not only during training stages, but first and foremost during their regular exploitation (working stages). In particular, it refers to reinforcement learning algorithms (RL), where an autonomous agent can learn to choose actions in an environment to achieve its goals (see [9], [10]). Such behavior results in robustness and high (increasing) efficiency of these approaches, thereby they have attracted particular attention in different domains (e.g., [11]).

Although there are lots of studies devoted to this group of algorithms, they usually focus on improving them in fields such as learning policies, representation of states, exploration and exploitation strategies and others (e.g., [9], [10]). However, there is lack of research on methods that are able to additionally increase the efficiency of RL by the utilization of its learning ability, without interfering a structure of an algorithm. Therefore, as the preliminary result in this domain, we will show that the speed of convergence of RL can be visibly improved by processing tasks according to a schedule that takes into account the existence of learning (an iterative improvement). It is depicted on the example of finding the shortest path in 2D mesh topology environment by RL. We choose Q-learning based algorithm, which is comprehensible and representative, since the considered relations also hold for other techniques, e.g., temporal difference learning (TD), state-action-reward-state-action (SARSA) (for more details see [10]).

In this paper, we model some aspects of Q-learning as scheduling problems with the learning effect. On this basis, the efficiency of the algorithm can be improved if tasks are processed according to a given sequence following from the analysis of scheduling problems. Furthermore, we show that a dispatch of tasks according to the longest processing time (LPT) algorithm (see [12]) for parallel computing can be replaced by a more efficient strategy, if agents can learn. Our approach does not interfere a structure of machine learning methods and its computation overhead is negligible. It is showed to be efficient, robust and only marginally dependents on a learning model and an accurate approximation of

values of task processing times. It is especially crucial for the application of reinforcement learning methods in varying environments, where they should adapt quickly, whereas deterministic algorithms cannot be used. Therefore, the contribution of this paper is not to make new better learning agents, but to reveal that approaches obtained for scheduling problems with learning effects can be successfully used to improve the quality of machine learning methods. The presented promising preliminary results open new perspectives for the application of scheduling theory.

The remainder of this paper is organized as follows. The illustrative application of the reinforcement learning is given in the next section, which includes the formal definition of the shortest path problem and the Q-learning algorithm dedicated to solve it. Next, the considered issue is expressed as scheduling on parallel processors with the learning effect and on this basis scheduling (dispatching) algorithms are proposed. Their application to enhance the Q-learning algorithm is analysed numerically. Finally, the last section concludes the paper.

II. Reinforcement learning

In this section, we will describe an environment, which is used to illustrate and analyse the application of the algorithms constructed for scheduling with the learning effect to improve the efficiency of reinforcement learning methods.

A. The shortest path problem

The shortest path problem is well known and its deterministic cases can be solved *inter alia* by Dijkstra's algorithm or A* search. However, it also constitutes an excellent environment for comprehensive analysis of various other methods also with learning effects (see [13]). It will be used for a similar purpose in this paper.

There is given a graph $G = (V, E)$, where V is the set of vertices (nodes) and E is the set of edges (links). Each edge (u, w) has an associated length (weight) $l(u, w)$, where $u, w \in V$; G is undirected, thereby $l(u, w) = l(w, u)$. We consider the 2D mesh topology, interconnecting in a grid fashion, where each node has at most four neighbors, thus, the following notation can be used to describe nodes: $V' = \{(x_u, y_u) : 1 \leq x_u \leq X \wedge 1 \leq y_u \leq Y, u \in V\}$ and X and Y are the number of vertical and horizontal nodes in the mesh, respectively; such mesh is called $X \times Y$ size. The objective is to find the shortest path (SP) between each of n source nodes from the set $S = \{s_1, \ldots, s_n\} \subset V$ to one destination node $d \in V$ ("hot spot").

B. Q-learning

In this section, we will briefly describe a reinforcement learning algorithm that is based on a typical Q-learning method (see [10], [14]). For mode details concerning similar applications of Q-learning see [15] or [16]. The discussed algorithm is a model free reinforcement learning technique, which was chosen for the purpose of this paper, since it is transparent for analysis and the idea behind our approach can be clearly illustrated. Its behaviour will be depicted on the example of solving in an adaptive manner the shortest path problem. On this basis, we will show that properties obtained for a scheduling problem with the learning effect can be applied to improve the efficiency of Q-learning.

In the considered approach, the Q-learning agent on the basis of the current node (say u) and the state-action function Q chooses the next node (say w), subsequently it receives the reward depending on the distance between these nodes. This process is repeated until the destination node (say d) is reached. The Q-function is represented by a set $\{Q_1, \ldots, Q_u, \ldots Q_{|V|}\}$ of $|V|$ tables, further called Q-tables. The quality of a state-action function (table) $Q_u(d, w)$ defined for each node $u \in V$ is the total expected discounted reward received by selecting (in node u) the next node w on the way to the destination node d. Using the RL nomenclature, the state is represented by the pair $[u, d]$, whereas w is the action taken in this state.

The determination of a next node and the update of the Q-function for each current node $u \in V$ proceed according to the following steps:

1) The next node w is determined according to a greedy strategy that always chooses a node with the highest Q-value, i.e.,
$$w = \arg \max_{w' \in \mathcal{N}(u)} \{Q_u(d, w')\},$$
where u is the current node, d is the destination node and $\mathcal{N}(u)$ is the set of neighbor nodes of u. The greedy strategy is simple and transparent, but together with the optimistic initialization of Q-tables with zeros and the application of negative rewards, it still drives exploration.
2) The Q-value for node u is updated according to the following rule:
$$\begin{aligned} Q_u(d, w) &= (1 - \alpha) Q_u(d, w) + r(u, w) \\ &+ \gamma \max_{a'} \{Q_w(d, a')\}, \end{aligned}$$
where $\alpha \in [0, 1]$ is a learning rate, $\gamma \in [0, 1]$ is a discount factor, $r(u, w) = r(w, u) = -l(u, w)$ is the reward equal to the negative value of the length between nodes u and w and $\max_{a'}\{Q_w(d, a')\}$ is the best expected Q-value in node w to reach the destination node d. In a typical Q-learning implementation, terms $[r(u, w) + \gamma \max_{a'}\{Q_w(d, a')\}]$ are multiplied by α. Although it is omitted in the presented approach to avoid potential deadlocks, Q-values is convergent (since $r < 0$).
3) To improve learning the forward update of Q-value of node w is proceeded:
$$\begin{aligned} Q_w(d, u) &= (1 - \alpha) Q_w(d, u) + r(u, w) \\ &+ \gamma \max_{a' \in \mathcal{N}(u)} \{Q_u(d, a')\}. \end{aligned}$$

The above procedure is applied starting from the source node $s_j \in S$ until the destination node d is reached (i.e., starting u is equal to s_j). It is repeated for the given

pair (s_j, d) until the same value of the path is obtained for the given number of succeeding iterations (denoted by $TerminateCondition$). This process of finding the shortest path for (s_j, d) will be called task j; for convenience, we will also use the following notation $j \in S \equiv s_j \in S$. Since in the considered example, the destination node d is the same for all tasks, then it does not need to be implemented a part of a state.

It can be observed that the application of distributed or parallel computing can improve efficiency of algorithms and it is commonly used nowadays, therefore, we present a parallel version of the considered approach. Thus, a set $A = \{A_1, \ldots, A_m\}$ of m Q-learning agents are applied. Each agent A_i calculates the shortest paths for the source nodes (tasks) from the set $S_i \subset S$, where S_i are disjoint sets such that $S_1 \cup \ldots \cup S_m = S$. Due to relatively long latency access to shared Q-tables (caused by mutexes, communication, etc.) in reference to very fast calculation times of the Q-values, each agent has its own set of Q-tables. The formal description of the Q-learning agent is given by Algorithm 1.

The objective of the Q-learning agents is not only to find the shortest paths for all given nodes, but to minimize the time of finding them. Let t_j be the processing time of task j, i.e., time of calculating the shortest path from the source node s_j to the destination node d, whereas $C(A_i) = \sum_{j \in S_i} t_j$ be the calculation time taken by Q-learning agent A_i to find shortest paths for all nodes from the related set S_i, i.e., to process tasks from this set. Thus, the time objective is expressed as the minimization of calculation time of the shortest paths for all source nodes (processing of all tasks), which is the maximum calculation time $t_{\max}$ among all Q-learning agents: $t_{\max} = \max_{i=1,\ldots,m}\{C(A_i)\}$.

Since there are multiple agents, then the calculations are distributed among them, i.e., the set S of source nodes (tasks) is partitioned into subsets $\{S_1, \ldots, S_i, \ldots, S_m\}$, which are assigned to related agents $\{A_1, \ldots, A_i, \ldots, A_m\}$. Thus, an assignment algorithm has an essential impact on the minimization of the objective value $t_{\max}$. To construct such methods, which are efficient, we will express the considered problem as scheduling on identical parallel processors. On the basis of its properties, we will propose scheduling algorithms, which will be applied to handle calculations (tasks) by Q-learning agents (i.e., to determine the assignment of calculations and their processing sequences).

Furthermore, we will show that calculations can be speeded up (i.e., $t_{\max}$ can be further minimized) if the allocation algorithms takes into consideration not only the potential time required to find a solution, but also learning (an iterative improvement), which is an inner nature of Q-learning.

III. Scheduling with learning effects

At first, we will present the general concept how the time minimization of finding the shortest paths by the Q-learning agents can be perceived as scheduling on parallel identical processors or in other words parallel machine scheduling. Next, the related scheduling problem will be formally defined.

Algorithm 1 Q-learning agent A_i

```
Determine the set of the source nodes S_i ⊆ S
  and their processing sequence
Initialize Q−tables: {Q_1, ..., Q_|V|}
for each s_j ∈ S_i do
  distance := ∞
  distance* := ∞
  distance_prev := ∞
  counter := 0
  s := s_j
  while counter < TerminateCondition do
    distance := 0
    counter := counter + 1
    u := s
    while u ≠ d do
      w := arg max_{w'∈N(u)} {Q_u(d, w')}
      distance := distance + l(u, w)
      r(u, w) := −l(u, w), r(w, u) := −l(w, u)
      Q_u(d, w) := (1 − α)Q_u(d, w) + r(u, w)
                   +γ max_{w'∈N(u)} {Q_w(d, w')}
      if w ≠ d then
        Q_w(d, u) := (1 − α)Q_w(d, u) + r(w, u)
                     +γ max_{u'∈N(u)} {Q_u(d, u')}
      end if
      u := w
    end while
    if distance < distance* then
      distance* := distance
    end if
    if distance ≠ distance_prev then
      counter := 0
    end if
    distance_prev := distance
  end while
end for
{Q_1, ..., Q_|V|} are the Q−tables containing strategy
  to find the shortest paths for
  the considered pairs (s_j, d), s_j ∈ S_i
```

Each Q-learning agent can be represented in a scheduling problem by a processor, i.e., $A_i \equiv P_i$, whereas a task (say j) that is finding the shortest path (such that $TerminateCondition$ is true) from a source node (say $s_j \in S$) to the destination node d by a Q-learning agent is called a job (also j).

The processing time t_j of task j, i.e., the time required to find (calculate) the shortest path (s_j, d), can depend on the number of nodes from a source node to the destination node (including). Hence, we model it by the processing time p_j of job j, which is given as follows:

$$p_j = |x_d - x_{s_j}| + |y_d - y_{s_j}|, \qquad \forall s_j \in S. \tag{1}$$

Note that p_j is not an exact value of the real processing time t_j required to find the related shortest path, but it only models

this parameter in scheduling domain, i.e., it should reflect the relations such that $t_j < t_k$ implies $p_j < p_k$.

For a better comprehension, we will analyse the following computational example.

Example 1

Let us verify the accuracy of model (1) for the following settings generated from the uniform distribution: graph $G(V, E)$ of size 100×100, lengths of edges $l(u, w) \in \{1, \ldots, 5\}$, where $(u, w) \in E$, 100 random source nodes $s_j = (x_{s_j}, y_{s_j})$, where $x_{s_j} \in \{2, \ldots, 99\}$, $y_{s_j} \in \{2, \ldots, 99\}$, and destination node $d = (x_d, y_d) = (99, 99)$. On this basis, we ran implemented Q-learning algorithm and measured (in milliseconds) the time of finding the shortest path by a single agent from a source node s_j to the destination node d, i.e., processing time t_j of task j. Such analysis was done for each source node $s_j \in S$, but to measure relevant times t_j, Q-tables were cleared before processing each task. Thus, each of them was processed as the only task by a Q-learning agent. We refer each measured t_j to the modelled value – the job processing time p_j – obtained according to (1). The result of this analysis is shown in Figure 1, where related pairs (p_j, t_j) are presented according to the non-decreasing order of p_j. It can be seen that there are some bias such that $t_j < t_k$ not always refers to $p_j < p_k$, but the general tendency holds as required (and expected).

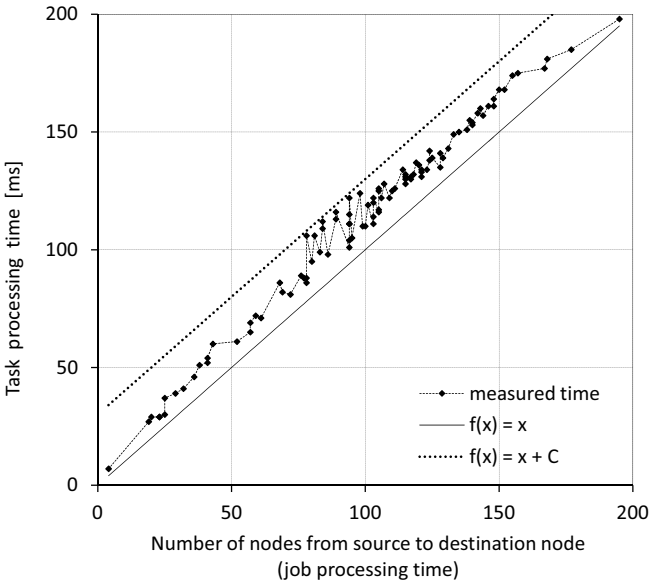


Fig. 1: Relations between measured task processing times t_j and modelled job processing times p_j

It can be seen in Figure 1 that model p_j approximates measured times t_j (in a constant range) if an agent processes only one task. However, due to the learning ability of an agent, the time required to perform a task decreases with the number of previously processed tasks. Let us consider the following example.

Example 2

Given a task related with node $s_1 = (x_{s_1}, y_{s_1}) = (2, 2)$, for which its processing times are measured (in milliseconds) depending on the number of previously processed tasks. Namely, an agent performs tasks from the following groups (1), $(2, 1)$, $(2, 3, 1)$, ..., $(2, 3, \ldots, 100, 1)$ such that $j = 1$ is always processed as the last and Q-tables are cleared before a next group is processed. Other settings are the same as in the earlier example. The result of the analysis is shown in Figure 2, which illustrates a learning curve of an agent, which is similar to learning curves observed in industrial systems (see [17], [18]). The model of decreasing task processing times (learning curve) will be proposed subsequently.

On the basis of the above observations, let us define formally the related scheduling problem. There are given a set $J = \{1, \ldots, n\}$ of n jobs that model tasks S and a set $P = \{P_1, \ldots, P_m\}$ of m identical processors referring to Q-learning agents $\{A_1, \ldots, A_m\}$. Each processor is continuously available and can process at most one job at a time. Once it begins processing a job it continues until this job is finished and there are no precedence constraints between jobs. Each job j is characterized by the processing time $\tilde{p}_j(v_i)$ dependent on the number of previously processed jobs v_i by processor P_i. It models a variable task processing time (see Figure 2), which depends (due to learning) on the number v_i of previously processed tasks by agent A_i. In general, it can be describe by the following:

$$\tilde{p}_j(v_i) = p_j \cdot f(v_i), \tag{2}$$

where p_j is the normal processing time of job j defined as the job processing time of a job without influence of learning, which is calculated on the basis of source and destination nodes related with a task and given by (1); it models a processing time of a task, which is processed by an agent as the first one, i.e., without influence of learning (see Figure 1). Moreover, a function $f(v_i)$ is a learning curve that describes decreasing processing times dependent on the number of processed jobs v_i (see Figure 2).

Note that the only assumption concerning f is that the function is non-increasing for the considered scheduling model, but it is only a model, which does not have to precisely describe the decreasing task processing times. Nevertheless, we will show that even such imprecise and general model (following (1) and (2)) is sufficient to derive properties, which allow us construct a task assignment algorithm that improves Q-learning.

The schedule of jobs can be unambiguously defined by their sequences $\pi = \{\pi_1, \ldots, \pi_i, \ldots, \pi_m\}$, where π_i denotes the sequence of jobs on P_i and $\pi_i(k)$ is the index of the kth job in π_i for $i = 1, \ldots, m$. Moreover, n_i is the number of jobs assigned to P_i, i.e., the cardinality of π_i. Note that π_i refers to the sequence of tasks from set S_i assigned to be processed by agent A_i. For each job $\pi_i(k)$, we can determine its completion

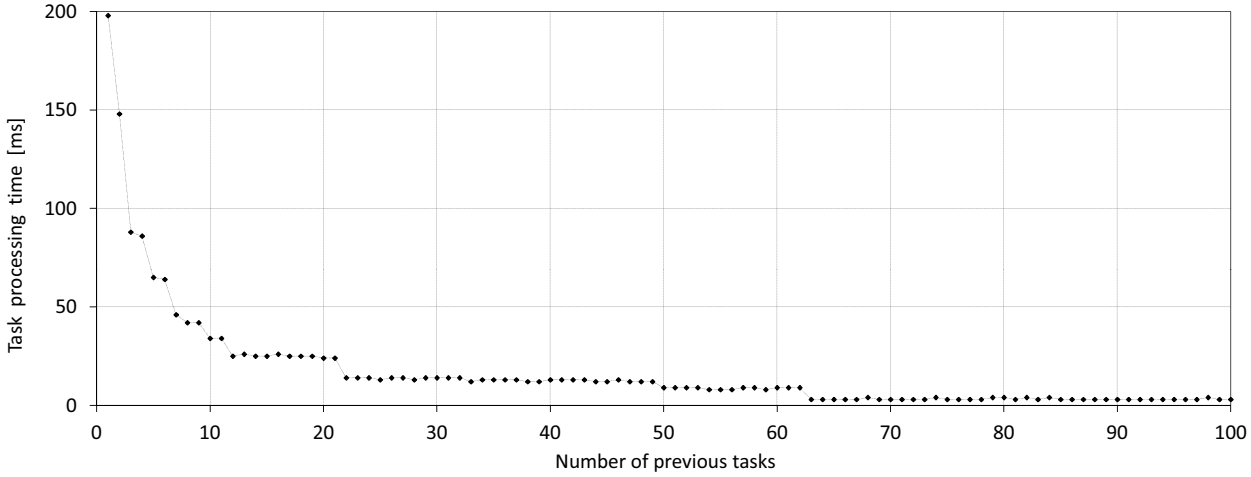


Fig. 2: A measured task processing time depending on the number of previously processed tasks

time $C^{(i)}_{\pi_i(k)}$ on processor P_i:

$$C^{(i)}_{\pi_i(k)} = \sum_{k=1}^{n_i} \tilde{p}_{\pi_i(k)}(k). \tag{3}$$

On this basis $C^{(i)}_{\pi_i(n_i)}$ models the calculation time $C(A_i)$ taken by Q-learning agent A_i to find shortest paths for all nodes from the related set S_i, i.e., to process tasks from this set. Since the objective of Q-learning agents is to minimize the time of finding all shortest paths $t_{\max}$, then the criterion value (we use a similar symbol) for the scheduling problem is defined by $C_{\max}(\pi) = \max_{i=1,\ldots,m}\{C^{(i)}_{\pi_i(n_i)}\}$.

Formally, the objective is to find such a schedule $\pi^* = \{\pi^*_1, \ldots, \pi^*_m\}$ of jobs (i.e., assignment of jobs to processors and their sequences) that minimizes the maximum completion time (makespan):

$$\pi^* \triangleq \text{minarg}_{\pi\in\Pi} \left\{ \max_{i=1,\ldots,m} \left\{ C^{(i)}_{\pi_i(n_i)} \right\} \right\}, \tag{4}$$

where Π is the set of all schedules π.

For convenience and to keep an elegant description of the considered problem we will use the three field notation scheme $X \mid Y \mid Z$ (see [19]), where X describes the processor environment, Y describes job characteristics and constraints and Z represents the minimization objectives. According to this notation the scheduling problem with model (2) will be denoted as $Pm|LE|C_{\max}$, whereas its case with constant job processing times ($\tilde{p}_j(v_i) = p_j \ \forall(j, v_i)$) will be denoted by $Pm||C_{\max}$.

On the basis of the formulated scheduling problem, we will propose algorithms that determine the assignment of tasks to agents and their processing sequences.

IV. Job scheduling algorithms

In this section, we focus on the job scheduling problem with the learning effect. The considered criterion is the minimization of the maximum job completion time (makespan). In the classical version of this problem $Pm||C_{\max}$, the job processing times are constant, thereby their sequence does not affect the criterion value, and thus, the objective is to find the allocation of jobs to the processors to minimize the makespan.

Let us analyse job assignment (dispatching) algorithms for the problem $Pm||C_{\max}$. One of the primary and efficient methods is a list scheduling, which assigns jobs to the first available processor according to an order defined by a list (see [12]). If the sequence of jobs on the list is random, then the worst case of the algorithm (further denoted by RND) is $\frac{C_{\max}(RND)}{C^*_{\max}} = 2 - \frac{1}{m}$ and its computational complexity is $O(nm)$ (see [20], [12]), where $C^*_{\max}$ denotes an optimal criterion value. However, its efficiency can be improved if jobs are assigned according to their longest processing times (LPT), then its worst case is $\frac{C_{\max}(LPT)}{C^*_{\max}} \leq \frac{4}{3} - \frac{1}{3m}$, whereas its computational complexity is $O(n \log n + mn)$ (see [12]).

Note that RND can be straightforwardly applied to the problem of finding shortest paths by Q-learning agents without any definition of a scheduling problem nor any model of task processing times. It only assigns each task to the first available agent. On the other hand, the application of LPT requires a model of task processing times, which has to be sufficient only to determine the non-decreasing relation between tasks during the assignment process.

However, unlike a typical approaches to parallel computations, we will reveal that in case of learning algorithms (such as Q-learning), not only the assignment of tasks is important, but their processing sequences by an agent seem to be essential. It will be shown on the basis of the formulated scheduling problem with learning. At first, we will provide a property for a single processor case $1|LE|C_{\max}$ (referring to one Q-learning agent), which holds for the problem with different learning models (e.g., [5], [6], [7]).

Property 1: The problem $1|LE|C_{\max}$ can be solved optimally by scheduling jobs according to their shortest normal processing times p_j (SNPT rule).

If job processing times are constant, then the sequence of jobs is immaterial for the makespan minimization on a single processor $1||C_{\max}$. However, following Property 1, it is no longer valid if job processing times can vary. This is a premise for a more efficient processing of tasks by a Q-learning agent, which will be analysed numerically in the next section. Furthermore, it can be easily extend to a scheduling on parallel processors (tasks processed by multiple agents).

Property 2: There exists an optimal solution to the considered problem $Pm|LE|C_{\max}$ such that jobs on each processor are scheduled according to their shortest normal processing times p_j (SNPT).

Based on Property 2, we propose a list scheduling algorithm, where jobs on the list are sequenced according to their shortest processing times (further denoted by SNPT). Such conclusion is not only opposite to the popular and efficient LPT algorithm often used for parallel computing ([12]), but it does not follow from any other combined analysis on machine learning algorithms and parallel computing.

It is worth noticing that our theoretical analysis showed that SNPT rule overwhelms LPT in the domain of scheduling problems with the learning effect. It is completely against the existing approaches to dispatch tasks for parallel learning algorithms according to LPT or RND (randomly).

Recall that a job is equivalent to a task, thereby a schedule π for jobs (obtained on the basis of LPT, RND or SNPT) straightforwardly determines the assignments (dispatching) of tasks to Q-learning agent and their processing sequences by each agent.

V. Numerical analysis

In this section, we will use scheduling algorithms LPT, RND and SNPT in the domain of the Q-learning and the shortest path problem. Namely, we will analyse how Q-learning agents can improve their efficiency (running times) if they process tasks according to sequences determined by the application of scheduling algorithms LPT, RND and SNPT. In other words, we run implemented Q-learning agents and measure (in milliseconds) the real time of finding the shortest paths depending on different sequences of processed tasks (i.e., LPT, RND, SNPT).[1] The impact of scheduling algorithms on running times of Q-learning is analysed for the following settings.

2D-mesh (environment):

- mesh (graph) size $X \times Y$: 100×100,
- lengths (weights) of links between nodes $l(u, w)$: generated from the uniform distribution over the integers in the following ranges of values $\{1, \ldots, 5\}$ and $\{1, \ldots, 100\}$, where $u, w \in V$,
- destination node d ("hot spot"): $(x_d, y_d) = (X-1, Y-1)$,
- size $|S|$ of the set S of source nodes (number of tasks): 1000, 2000 and 4000,
- source node coordinates (x_{s_j}, y_{s_j}): $x_{s_j} \in \{2, \ldots, X-1\}$, $y_{s_j} \in \{2, \ldots, Y-1\}$, $s_j \in V$, where $j = 1, \ldots, |S|$.

[1]Q-learning and scheduling algorithms were coded in C++ and simulations were run on PC, CPU Intel® Core™i7-2600K 3.40 GHz and 8GB RAM.

Q-learning (agent):

- $m \in \{1, 2, 10, 20\}$ agents, $\alpha = 0.9$, $\gamma = 0.9$, $TerminateCondition = 5$,
- the applied Q-learning always provided optimal solution (the shortest path) for the considered settings, which was verified by Dijkstra's algorithm.

Job scheduling algorithms:

- $m \in \{1, 2, 10, 20\}$ processors,
- job processing times (the model of task processing times) $p_j = |x_d - x_{s_j}| + |y_d - y_{s_j}|$ for $j = 1, \ldots, |S|$,
- algorithms LPT, RND, SNPT.

For each combination of the defined mesh parameters, a set I_Q of random instances (replications) were generated, where its cardinality is equal to $|I_Q| = 100$. On their basis, the impact of scheduling algorithms on running times of Q-learning is evaluated (as percentage speed up δ) in reference to the running times obtained with LPT as follows for each instance I

$$\delta(TS(I)) = \frac{t_{\max}(LPT(I)) - t_{\max}(TS(I))}{t_{\max}(TS(I))} \times 100\%,$$

where $TS \in \{LPT, RND, SNPT\}$ and $t_{\max}(TS(I))$ is the measured (in milliseconds) maximum calculation time (running time) among all Q-learning agents (equivalent to the makespan) that processed tasks according to the applied task scheduling algorithm TS for the instance $I \in I_Q$.

Thus, for each instance I, the running times (in milliseconds) of finding the shortest paths by Q-learning agents are measured, which processed tasks according to LPT, RND and SNPT, respectively. The results concerning mean, minimum, and maximum speed up δ (in percents) and running times $t_{\max}$ (in milliseconds) of Q-learning are given in Table I, where the related are calculated for each set I_Q of instances as follows: $\delta(TS, min) = \min_{I \in I_Q}\{\delta(TS(I))\}$, $\delta(TS, max) = \max_{I \in I_Q}\{\delta(TS(I))\}$, $\delta(TS, mean) = \sum_{I \in I_Q} \delta(TS(I))/|I_Q|$, and $t_{\max}(TS, min) = \min_{I \in I_Q}\{t_{\max}(TS(I))\}$, $t_{\max}(TS, max) = \max_{I \in I_Q}\{t_{\max}(TS(I))\}$, $t_{\max}(TS, mean) = \sum_{I \in I_Q} t_{\max}(TS(I))/|I_Q|$. Note that the higher value of δ then better, whereas it is oppositive for $t_{\max}$. Moreover, only mean values of δ and $t_{\max}$ are correlated, whereas min and max are informative. For a better comprehension, let us consider the first row of Table I for minimum values of RND. We have the minimum speed up $\delta = -2\%$ and minimum running time $t_{\max} = 204$ ms of Q-learning. However, -2% does not refers to 204, it only means that there was a running time of Q-learning using RND for which the minimum speed up comparing to the approach using LPT was -2%, i.e., LTP caused a result 2% faster than RND. On the other hand, minimum $t_{\max} = 204$ means that the smallest running time of Q-learning using RND was 204 ms.

In the previous studies on machine learning, a sequence of processing tasks has not been taken into account as an

TABLE I: Speed up and running times of Q-learning depending on sequences determined by scheduling algorithms in reference to results obtained by LPT

n	m	$l(u,w)$		LPT			RND			SNPT		
				mean	min	max	mean	min	max	mean	min	max
500	1	5	$\delta[\%]$	0	0	0	5	-1	11	18	9	22
			$t_{\max}$	[225]	[220]	[238]	[213]	[205]	[238]	[192]	[190]	[206]
		100	$\delta[\%]$	0	0	0	5	0	9	14	10	16
			$t_{\max}$	[225]	[218]	[231]	[212]	[201]	[226]	[196]	[192]	[198]
	2	5	$\delta[\%]$	0	0	0	4	0	9	15	11	17
			$t_{\max}$	[221]	[217]	[224]	[208]	[201]	[220]	[192]	[189]	[195]
		100	$\delta[\%]$	0	0	0	5	-1	10	14	11	16
			$t_{\max}$	[215]	[210]	[218]	[204]	[198]	[217]	[188]	[185]	[194]
	10	5	$\delta[\%]$	0	0	0	4	0	9	13	8	17
			$t_{\max}$	[209]	[204]	[216]	[202]	[196]	[211]	[184]	[181]	[195]
		100	$\delta[\%]$	0	0	0	4	-3	9	15	9	19
			$t_{\max}$	[228]	[222]	[237]	[220]	[211]	[231]	[198]	[194]	[211]
	20	5	$\delta[\%]$	0	0	0	2	-1	8	12	8	17
			$t_{\max}$	[213]	[209]	[222]	[206]	[196]	[216]	[189]	[184]	[195]
		100	$\delta[\%]$	0	0	0	3	-1	12	14	11	22
			$t_{\max}$	[219]	[213]	[250]	[212]	[203]	[224]	[192]	[188]	[204]
1000	1	5	$\delta[\%]$	0	0	0	6	0	12	17	14	19
			$t_{\max}$	[262]	[257]	[266]	[244]	[236]	[258]	[224]	[220]	[226]
		100	$\delta[\%]$	0	0	0	6	0	10	12	11	16
			$t_{\max}$	[248]	[245]	[253]	[234]	[226]	[248]	[220]	[218]	[224]
	2	5	$\delta[\%]$	0	0	0	5	0	9	15	12	17
			$t_{\max}$	[240]	[234]	[245]	[228]	[219]	[241]	[207]	[205]	[212]
		100	$\delta[\%]$	0	0	0	6	-1	10	16	14	19
			$t_{\max}$	[241]	[238]	[245]	[228]	[219]	[245]	[207]	[206]	[209]
	10	5	$\delta[\%]$	0	0	0	4	-1	9	17	14	20
			$t_{\max}$	[234]	[229]	[242]	[222]	[215]	[236]	[197]	[195]	[202]
		100	$\delta[\%]$	0	0	0	3	-2	8	17	15	19
			$t_{\max}$	[231]	[229]	[237]	[222]	[213]	[235]	[197]	[194]	[199]
	20	5	$\delta[\%]$	0	0	0	3	0	7	13	10	17
			$t_{\max}$	[219]	[215]	[227]	[212]	[206]	[221]	[192]	[190]	[198]
		100	$\delta[\%]$	0	0	0	3	-1	7	15	8	18
			$t_{\max}$	[224]	[221]	[231]	[217]	[209]	[227]	[195]	[191]	[206]
4000	1	5	$\delta[\%]$	0	0	0	5	1	8	14	12	17
			$t_{\max}$	[333]	[331]	[342]	[316]	[307]	[329]	[292]	[288]	[295]
		100	$\delta[\%]$	0	0	0	4	-1	8	11	8	14
			$t_{\max}$	[329]	[325]	[332]	[313]	[303]	[331]	[292]	[290]	[301]
	2	5	$\delta[\%]$	0	0	0	5	0	10	15	12	18
			$t_{\max}$	[286]	[284]	[291]	[272]	[260]	[286]	[249]	[246]	[254]
		100	$\delta[\%]$	0	0	0	4	0	8	13	11	16
			$t_{\max}$	[282]	[279]	[286]	[267]	[258]	[282]	[247]	[244]	[250]
	10	5	$\delta[\%]$	0	0	0	4	0	7	15	13	19
			$t_{\max}$	[238]	[235]	[246]	[227]	[221]	[242]	[204]	[202]	[208]
		100	$\delta[\%]$	0	0	0	3	0	8	14	9	19
			$t_{\max}$	[226]	[224]	[234]	[217]	[212]	[227]	[197]	[195]	[211]
	20	5	$\delta[\%]$	0	0	0	3	0	8	16	12	18
			$t_{\max}$	[218]	[216]	[227]	[212]	[204]	[219]	[188]	[185]	[193]
		100	$\delta[\%]$	0	0	0	3	-1	9	15	12	19
			$t_{\max}$	[219]	[215]	[228]	[211]	[205]	[220]	[188]	[186]	[194]

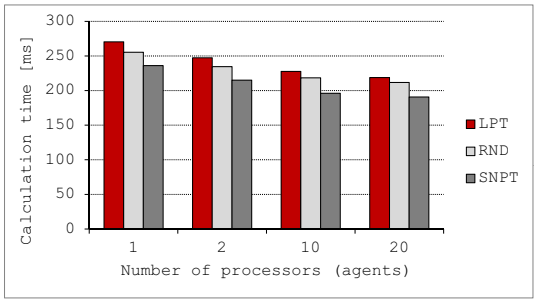
approach of speeding up these methods (in particular Q-learning). Similarly for computing them in parallel, it might be expected that the most efficient among task assignment (scheduling) algorithms is LPT, next RND and SNPT is the worst, or more likely that the differences in running times of machine learning methods are negligible for different sequences of processed tasks. However, the numerical analysis following our theoretical research reveals that it does not have to be so (in fact it is completely opposite to the popular approach).

It can be seen in Table I that a proper sequence of processed tasks (see SNPT) can significantly speed up Q-learning (even 22% and not less than 8%). The running times of Q-learning $t_{\max}(LPT)$ using LPT are 204–342 ms depending on the number of tasks n and agents m, whereas the impact of values of lengths (weights) $l(u,w)$ is negligible. Similar relations hold for RND and SNPT. The best running times are obtained for Q-learning that processed tasks according to SNPT, which is consistent with the theoretical results. It can be seen that SNPT always speeds up Q-learning, i.e., 11–18% (mean), but at least 8–15% (min) and even 14–22% (max); see also Figure 5. For RND, we have speed up about 2–6% (mean) and 7–12% (max), whereas its minimum speed up of RND in reference to LPT is -3–1% (min), where the negative values mean that for some cases LPT is slightly better.

Let us analyse running times of a single agent and multiple agents. For $m = 1$ and $n = 500$, we have the following average running times $t_{\max}(LPT) = 225$ ms, $t_{\max}(RND) = 213$ ms, $t_{\max}(SNPT) = 192$ ms, thereby SNPT is about 30 ms and 20 ms faster than LPT and RND, respectively. Thus, SNPT is visible better not only than LPT, but also than random sequence. It shows that our approach can significantly speed up Q-learning even for a single agent. On the other hand, for $m = 20$ and $n = 4000$ for which LPT should

be (according to other studies) an efficient task assignment method for parallel computing, the mean running times are as follows: $t_{\max}(LPT) = 219$ ms, $t_{\max}(RND) = 211$ ms, $t_{\max}(SNPT) = 188$ ms. Once again, LPT is the worst and a random sequence is better, but they are overwhelmed by SNPT for all analysed instances. It can be seen that the proposed approach is robust, it is slightly affected by the number of agents (Figure 3) or tasks (Figure 4). Moreover, the mean speed up is over 15% in reference event to LPT (well known to be very good algorithm for such cases).

(a) Calculation time (lower – better)



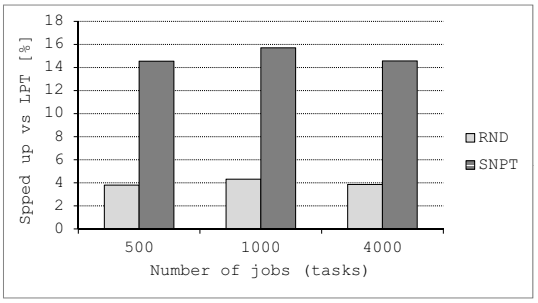
(b) Speed-up vs LPT (higher – better)

Fig. 3: Mean calculation times and speed-ups of algorithms for different numbers of processors (agents); following Table I

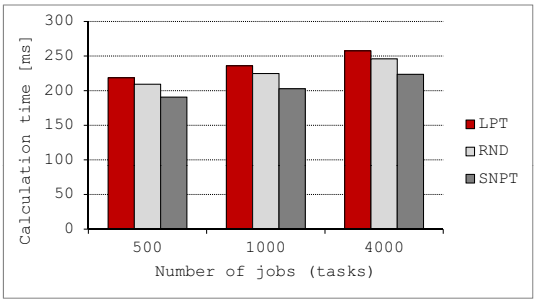
Finally, the time required by considered scheduling algorithms (in particular SNPT) to determine sequences of processed tasks does not exceed 1 ms (even for greater instances $n = 4000$ and $m = 20$), which is negligible. Hence application of our approach significantly speeds up Q-learning (about 15%) without additional effort, it does not interfere Q-learning structure nor cause time overhead. It is especially crucial for varying environments, where Q-learning should adapt quickly. Thus, it can be used in a similar way for other related machine learning methods.

VI. Conclusions

Following dependencies observed in production and manufacturing systems related with a human factor (learning), we modelled relations, which occur between tasks processed by machine learning algorithms. Namely, we presented preliminary results, which revealed that approaches obtained for scheduling problems with learning effects (known from production and manufacturing) can be successfully used to

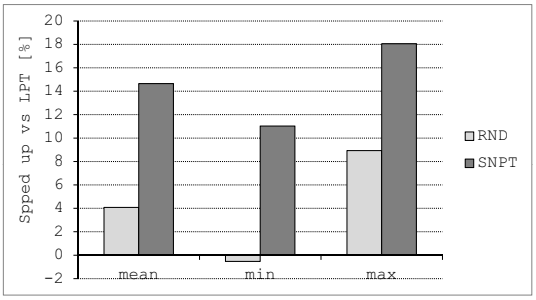


(a) Calculation time (lower – better)

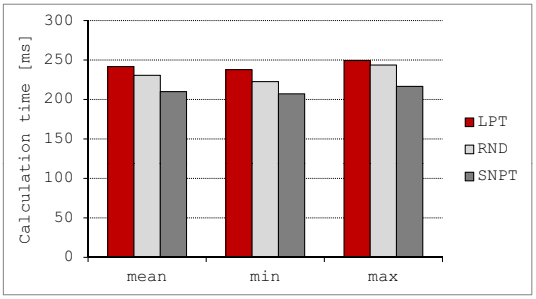


(b) Speed-up vs LPT (higher – better)

Fig. 4: Mean calculation times and speed-ups of algorithms for different numbers of jobs (tasks); following Table I



(a) Calculation time (lower – better)



(b) Speed-up vs LPT (higher – better)

Fig. 5: Mean, minimum and maximum values of calculation times and speed-ups; following Table I

improve the quality of machine learning methods. It was illustrated by modelling some aspects of Q-learning as scheduling problems with the learning effect. The previous studies on machine learning had not taken into consideration that the existence of learning can be utilized and bring additional

benefits. On the basis of our approach, we claimed that the efficiency of Q-learning algorithms can be improved (running times) if tasks are processed according to a given sequence. Furthermore, we showed that for a parallelized Q-learning an assignment of tasks by SNPT is significantly more efficient than LPT, which is somehow in opposition to an intuition following from the previous studies on the sole LPT rule without the presence of learning (e.g., [12], [21]). The numerical analysis revealed that the dependency between running times and applied task processing (scheduling) algorithm is not incidental, but it is a rule.

Thus, our approach is efficient and robust, since it does not depend on a learning curve model nor accurate values of the normal job (task) processing times, but requires only their non-decreasing relation. In the same time, it does not interfere a structure of machine learning methods and its computation overhead is negligible. On the other hand, it is also the main limitation of the presented approach, since the SNPT rule cannot gain additional optimization benefits from knowing the exact model of learning curves describing the reduction of task processing times.

Our future research will focus on the application of our approach to other machine learning methods as well as on the analysis of different criteria, thereby development of other efficient algorithms.

ACKNOWLEDGEMENT

This work was supported by the Polish National Science Centre under grant no. DEC-2020/37/B/HS4/03235.

REFERENCES

[1] D. Biskup, "A state-of-the-art review on scheduling with learning effects," *European Journal of Operational Research*, vol. 188, pp. 315–329, 2008.

[2] R. Rudek, "Scheduling on parallel processors with varying processing times," *Computers & Operations Research*, vol. 81, pp. 90–101, 2017.

[3] J. Xu, C.-C. Wu, Y. Yin, C. Zhao, Y.-T. Chiou, and W.-C. Lin, "An order scheduling problem with position-based learning effect," *Computers & Operations Research*, vol. 74, pp. 175–186, 2016.

[4] C. Zhao, J. Fang, T. Cheng, and M. Ji, "A note on the time complexity of machine scheduling with DeJong's learning effect," *Computers & Industrial Engineering*, vol. 112, pp. 447–449, 2017.

[5] J. Pei, X. Liu, P. M. Pardalos, A. Migdalas, and S. Yang, "Serial-batching scheduling with time-dependent setup time and effects of deterioration and learning on a single-machine," *Journal of Global Optimization*, vol. 67, pp. 251–262, 2017.

[6] R. Rudek, "A fast neighborhood search scheme for identical parallel machine scheduling problems under general learning curves," *Applied Soft Computing*, vol. 113, pp. 108 023.1–16, 2021.

[7] C.-H. Wu, W.-C. Lee, P.-J. Lai, and J.-Y. Wang, "Some single-machine scheduling problems with elapsed-time-based and position-based learning and forgetting effects," *Discrete Optimization*, vol. 19, pp. 1–11, 2016.

[8] M. I. Jordan and T. M. Mitchell, "Machine Learning: Trends, Perspectives, and Prospects," *Science*, vol. 349, pp. 255–260, 2015.

[9] I. Grondman, L. Buşoniu, G. Lopes, and R. Babuška, "A survey of actor-critic reinforcement learning: standard and natural policy gradients," *IEEE Transactions On Systems, Man, And Cybernetics - Part C: Applications And Reviews*, vol. 42, pp. 1291–1307, 2012.

[10] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*. Cambridge: MIT Press, 1998.

[11] S. Whiteson and P. Stone, "Adaptive job routing and scheduling," *Engineering Applications of Artificial Intelligence*, vol. 17, pp. 855—-869, 2004.

[12] M. Pinedo, *Scheduling: Theory, Algorithms and Systems (5rd ed.)*. New York: Springer, 2016.

[13] Y. Wang, X. Li, and R. Ruiz, "An exact algorithm for the shortest path problem with position-based learning effects," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, pp. 3037–3049, 2017.

[14] C. Watkins, "Q-Learning," *Machine Learning*, vol. 8, pp. 279–292, 1992.

[15] W. Y. Kwon, I. H. Suh, and S. Lee, "SSPQL: stochastic shortest path-based Q-learning," *International Journal of Control, Automation, and Systems*, vol. 9, pp. 328–338, 2011.

[16] A. Konar, I. G. Chakraborty, S. J. Singh, L. C. Jain, and A. K. Nagar, "A deterministic improved Q-learning for path planning of a mobile robot," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 43, pp. 1141–1153, 2013.

[17] Z. S. Givi, M. Y. Jaber, and W. P. Neumann, "Modelling worker reliability with learning and fatigue," *Applied Mathematical Modelling*, vol. 39, pp. 5186–5199, 2015.

[18] C. H. Glock and Y. M. Jaber, "Learning effects and the phenomenon of moving bottlenecks in a two-stage production system," *Applied Mathematical Modelling*, vol. 37, pp. 8617–8628, 2013.

[19] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," *Annals of Discrete Mathematics*, vol. 5, pp. 287–326, 1979.

[20] R. Graham, "Bounds for certain multiprocessing anomalies," *Bell System Technical Journal*, vol. 45, pp. 1563–1581, 1966.

[21] W. Li and J. Yuan, "LPT online strategy for parallel-machine scheduling with kind release times," *Optimization Letters*, vol. 10, pp. 159–168, 2016.