INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2011, VOL. 57, NO. 2, PP. 223–228
Manuscript received April 20, 2011; revised May 2011. DOI: 10.2478/v10177-011-0031-x

Parallel Computation Approaches to Optimize Learning Systems

Tomasz Czyż, Radosław Rudek, and Henry Selvaraj

***Abstract*—This paper is devoted to the total tardiness minimization scheduling problem, where the efficiency of a processor increases due to its learning. Such problems model real-life settings that occur in the presence of a human learning (industry, manufacturing, management) and in some computer systems. However, the increasing growth of significant achievements in the field of artificial intelligence and machine learning is a premise that the human-like learning will be present in mechanized industrial processes that are controlled or performed by machines as well as in the greater number of multi-agent computer systems. Therefore, the optimization algorithms dedicated in this paper for scheduling problems with learning are not only the answer for present day scheduling problems (where human plays important role), but they are also a step forward to the improvement of self-learning and adapting systems that undeniably will occur in a new future. To solve the analysed problem, we propose parallel computation approaches that are based on NEH, tabu search and simulated annealing algorithms. The numerical analysis confirm high accuracy of these methods and show that the presented approaches significantly decrease running times of simulated annealing and tabu search and also reduce the running times of NEH.**

***Keywords*—parallel computation, metaheuristic, scheduling, learning.**

I. Introduction

THE development of artificial intelligence (AI) methods and techniques provides new possibilities and opens new perspectives for industry, manufacturing, economy and businesses sectors as well as for computer and network systems and especially for various web services. The application of artificial intelligence makes systems more autonomous and allows them to adapt to the changes of their working environments and to improve their efficiency by continuous learning (see [1], [2]). Thereby AI unlocks an access to the attributes such as learning or adaptation that have been perceived to be reserved only for human or living beings.

Research considering human learning usually focuses on different aspects of its acceleration. On the other hand, in many cases learning that is the result of repeating similar operations (learning-by-doing) is autonomous (see [3]). Therefore, in economy, industry and manufacturing a model that describes "learning effect" in a quantitative form is crucial, since it enables for more efficient estimation of a variable production time and/or cost caused by learning, thus, allows to improve lot-sizes, worker management, energy/resource consumption, etc. (e.g., [3], [4]). Moreover, learning (even autonomous) provides additional control abilities (i.e., the sequence of processed tasks) that allow to improve the optimized system objectives by the utilization of human learning (see [3]).

Similarly as in case of human learning, research devoted to AI methods and their applications focuses mostly on the acceleration of the learning process and its accuracy (e.g., [1], [5]), whereas the analysis considering learning models and the estimation of the impact of AI learning on processing times of tasks is marginal. Therefore, advantages related with the utilization (exploitation) of learning (known from human activity domains) are also marginal for AI (e.g. [6]). Although human learning can be accelerated by additional training courses, it is difficult for AI systems, especially if they work on-line and improve themselves during interactions with their environments (e.g. [1], [5]). The reductions in time objectives of such systems result from autonomous learning, however, they can be further improved by the sequence of processed tasks as in case of human learning (see [3]). For example, the decreasing of processing times can be taken into account during sequencing tasks to minimize the total tardiness of tasks or the number of late tasks. Thus, such approach does not influence learning (that anyway in many cases is impossible) nor interferes with the structure of the optimized system. Thereby it can support already working systems and requires only minor changes in (or the introduction of) their modules, which determine the sequence of processed tasks taking into consideration learning of such systems.

Nevertheless, the determination of the sequences of processed tasks to optimize given learning system objectives usually belongs to hard computation problems (see [7]), hence in practice the first sequence with an acceptable objective value is search or such with the best objective value that is found in the given period of time. For this purpose heuristic or metaheuristic methods are often applied that are characterized by short running times and relatively high accuracy. Since in practice calculation time is often crucial, thus, additional techniques that can speed up heuristic and metaheristic algorithms are essential. It is especially significant for metaheuristic algorithms that can search greater number of potential solutions in the restricted period of time.

Therefore, we propose the parallel computation approaches to improve the computation time of the well known NEH algorithm [8], tabu search [9] and simulated annealing [10] that determine the sequence of tasks in a learning system to optimize its time objectives.

T. Czyż is with Wrocław University of Technology, Janiszewskiego 11/18, 50-352 Wrocław, Poland (e-mail: tomasz.czyz@gmail.com).

R. Rudek is with Wrocław University of Economics, Komandorska 118/120, 53-345 Wrocław, Poland (http://www.radoslawrudek.pl).

H. Selvaraj is with Department of Electrical and Computer Engineering, University of Nevada, Las Vegas, 4505 Maryland Parkway Las Vegas, NV 89154-4026, USA (e-mail: Henry.Selvaraj@unlv.edu).

The contribution of this paper is the analysis of the total tardiness minimization scheduling problem with learning that models many real-life problems occurring in computer and industrial systems. Next, we provide parallel algorithms that efficiently solve this problem, furthermore, they can be easily tuned for other combinatorial optimization problems.

The remainder of this paper is organized as follows. The next section contains problem formulation. Further, the description of the proposed parallel computation approaches is presented and followed by the analysis of their efficiency. Finally, the last section concludes the paper.

II. Problem Formulation

In this section, a formal description of an autonomous learning system (ALS) will be provided. Next we will show that the optimal control problem for the considered system with the given objective can be expressed in the context of the scheduling theory. It follows from the fact that this theory offers efficient and convenient methods of mathematical analysis and supports the design of control strategies.

The ALS consists of a single processor (e.g., an algorithm, a web service, an intelligent agent, a group of cooperating agents, services or even a human) and a set $J = \{1, \ldots, n\}$ of n tasks that have to be performed by the processor in the given period of time; there are no precedence constraints between tasks. The processor is continuously available and can process at most one task at a time. Once it begins processing a task it will continue until this task is finished.

Each task j is characterized by its release time (when it comes to the system or can be processed), a due date d_j (when it should be completed) and its normal processing time p_j (i.e., the time required to processes a taks if no learning occurs). Due to the learning effect the (actual) processing time of task j is described by the learning curve $\tilde{p}_j(v)$ that defines decreasing of the time required to perform this task depending on the number of already processed tasks v. Following approaches presented in [3] and [6], the processing time of task j if it is performed as the vth in a sequence is given as:

$$\tilde{p}_j(v) = p_j v^{\alpha}, \tag{1}$$

where $\alpha < 0$ is the learning index that describes the learning curve (characteristic) of the considered system.

As it was mentioned in the previous section, the objectives of ALS can be controlled by the sequence of processed tasks. Therefore, let us define control variables formally.

Let $\pi = \langle \pi(1), ..., \pi(i), ..., \pi(n) \rangle$ denote the sequence of tasks (permutation of the elements of the set J), where $\pi(i)$ is the task processed in position i in this sequence. By Π we will denote the set of all such permutations. For the given sequence (permutation) π, we can easily determine the completion time $C_{\pi(i)}$ of a task placed in the ith position in π from the following recursive formulae:

$$C_{\pi(i)} = \max\{C_{\pi(i-1)}, r_{\pi(i)}\} + \tilde{p}_{\pi(i)}(i), \tag{2}$$

where $C_{\pi(0)} = 0$. Usually it is required for ALS and also other systems that the processing of a task should be completed before its due-date otherwise such task is late. Therefore, in this paper, the objective is to find such control decision, i.e., sequence π of tasks on the single processor, which minimizes the total tardiness:

$$TT(\pi) = \sum_{i=1}^{n} T_{\pi(i)}, \tag{3}$$

where $T_{\pi(i)} = \max\{0, C_{\pi(i)} - d_{\pi(i)}\}$. Formally the optimal control (sequence) $\pi^* \in \Pi$ for the considered minimization objective is defined as follows $\pi^* \triangleq argmin_{\pi \in \Pi}\{TT(\pi)\}$. For convenience and to keep an elegant description, we denote the minimization of the Total Tardiness Problem in a learning system as TTP.

III. Parallel Computation Approaches

In this section, we propose parallel computation approaches to solve TTP that are based on the well known algorithms NEH [8], tabu search (TS) [9] and simulated annealing (SA) [10].

A. Parallel NEH

The standard NEH algorithm for TTP is based on the method introduced in [8]. It starts with an initial sequence that determines the order of tasks that are inserted subsequently into the resulting solution. Namely, in each iteration the algorithm gets the first task from the initial sequence and adds it to the recent partial sequence of tasks such that it is inserted into a position in this partial sequence that minimizes the criterion value and next this task is removed from the initial sequence. This new partial sequence is the starting sequence in the next iteration of NEH. This process is continued until the initial sequence is empty. The computational complexity of this algorithm is $O(n^3)$.

The fundamentals of the parallel NEH (PNEH) are based on NEH. The main process (MP) gets tasks from the initial sequence and insert them into the new partial sequence to minimize its criterion value. For each such task and the partial sequence MP generates *moves*, i.e., all possible insertion of this task into the sequence, where the parameters of each move are the current task index and its position in a new sequence, i.e., (i_k, j_k). Each move pair is put into the *move queue*, from where the threads get them to calculate their corresponding criterion values and after that to write the move pair and its criterion value in *solution queue*, i.e., $(i_k, j_k, value_k)$. When the move queue is empty and the criterion value for all moves of the current task are calculated, MP chooses the best insertion (pair) and the corresponding sequence becomes the new partial sequence for the next iteration. The general idea of PNEH is given in Fig. 1.

B. Parallel Tabu Search (PTS)

Tabu search (TS) algorithm [9] uses local search with a short term memory, called tabu list, that stores forbidden moves, hence allows TS to escape from local minima. In the implemented algorithm *move* is defined as the insertion of a task being in position i in a sequence into position j. The applied tabu list stores pairs (i, j) of forbidden moves. If (i, j) is in the tabu list then for any sequence the insertion

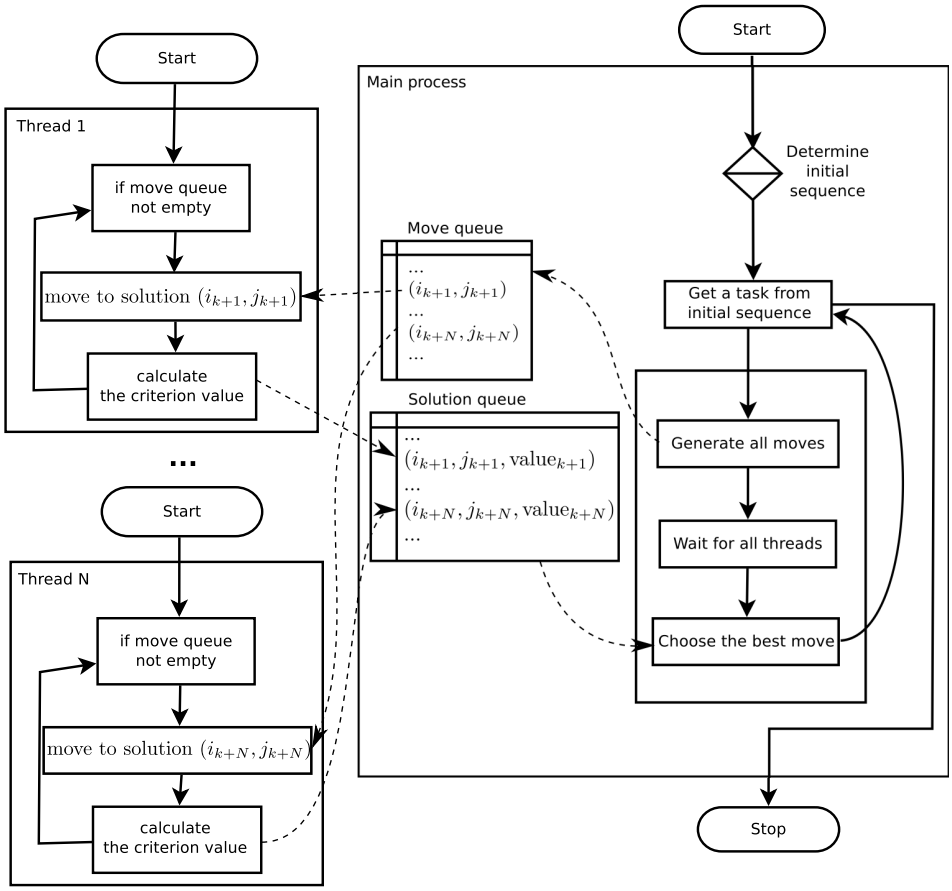

Fig. 1. Parallel NEH (PNEH).

of a task from position i into position j is forbidden. The tabu list is organized as FIFO (First In First Out), thereby, if the list is full then the new pair (i, j) is added at its beginning overriding the previous pair occupying this position. The size of the tabu list is fixed and denoted by $|TabuList|$. If the criterion value for the current sequence (for which the neighbourhood is searched) is not changed for the given number of iterations, then the new current sequence is diversified by choosing a random solution. Note that the best found sequence (solution) among all visited solutions is always stored. The computational complexity of the applied TS is $O(Iterations \cdot n^3)$.

In the parallel tabu search (PTS), the main process (MP) for each sequence determines all possible moves and puts parameters of each move k, i.e., pair (i_k, j_k), into the *move queue*. The threads get the pairs from the front of the move queue such that each pair is processed by one thread only and each thread calculates the criterion value for such move (sequence), e.g., Thread 1 calculates the criterion value for move (i_{k+1}, j_{k+1}), Thread 2 for (i_{k+2}, j_{k+2}), etc. After that each thread puts triple $(i_k, j_k, value_k)$ in the *solution queue* and among them MP chooses the new best sequence according to the same rules as TS. When the queue with pairs is empty, then MP waits for all threads to finish their calculations and to send them to the solution queue to verify if they can become the best solution in this neighbourhood. Next the whole process is repeated according to the same rules as for TS. The general idea of PTS is given in Fig. 2.

C. Parallel Simulated Annealing (PSA)

The primary simulated annealing algorithm [10] in each iteration generates a new permutation π' based on the current sequence (solution) π by interchanging of random two tasks in π. The new solution π' replaces π with the probability $P(T, \pi', \pi) = \min\{1, exp(-\frac{TT(\pi') - TT(\pi)}{T})\}$, where T is the temperature that decreases in a logarithmical manner $T = \frac{T}{1+\lambda T}$. The values of the initial temperature T_0 and of the parameter λ are chosen empirically. The algorithm results the solution π^* that is the best found criterion value in all iterations. Since the algorithm stops after $Iterations$ steps, then its overall computational complexity is $O(Iterations \cdot n)$.

Since in each iteration of SA a new solution π' is generated randomly, then we distribute calculations to each thread that is called the parallel simulated annealing (PSA). Therefore, each thread works according to SA independently and searches for a solution with the smallest criterion value. When all threads finish calculations the main process chooses the best found solution among the results provided by the threads. From the perspective of searching of the solution space, PSA is similar to running SA, however, the parallel calculations (PSA) made by threads speeds up such process. The general idea of PSA is given in Fig. 3.

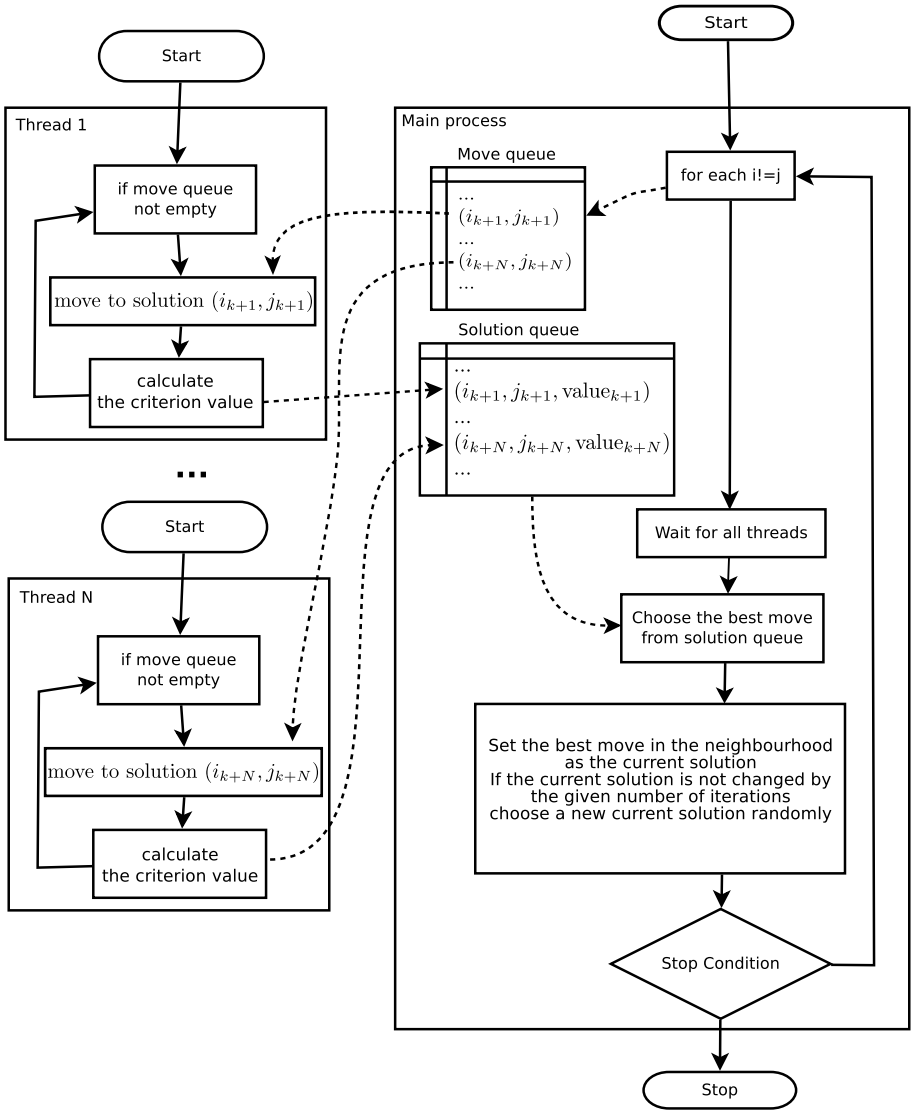

Fig. 2. Parallel tabu search (PTS).

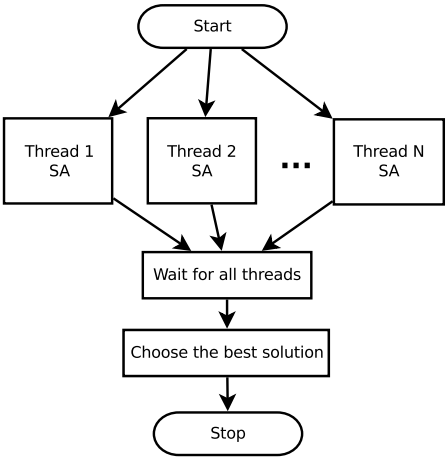

Fig. 3. Parallel simulated annealing (PSA).

IV. Computational Experiment

The numerical analysis of the algorithms covers two aspects, namely their efficiency in finding solution with the minimal criterion value and in decreasing computation times depending on the number of applied threads.[1]

Let us introduce the useful notation, where PNEHi, PTSi and PSAi denote the algorithms with the given number i of the applied threads.

First we focus on finding solution with the minimal criterion value and we compare only standard versions of the algorithms with one thread only, i.e., PNEH1, PTS1 and PSA1. It follows from the fact, that the standard and the parallel versions of NEH and TS (for the fixed number of $Iterations$) search the same solution space, but the parallel algorithms do it faster. It is only slightly different for SA since each thread starts with the initial value of the temperature. The considered algorithms were evaluated for different problem sizes $n \in \{10, 50, 100, 150, 200\}$. For each value of n, 100 random instances are generated from the uniform distribution in the following ranges of parameters: $p_j \in [1, 100]$, $r_j \in$

[1]The algorithms were coded in C++ and simulations were run on PC, Dual-Core AMD Opteron^{TM} Processor 2218 and 8GB RAM.

$[1, \sum_{i=1}^{n} p_j]$, $d_j = r_j + p_j$ (it models settings, where tasks have to be processed just after they are ready to release) and $\alpha = -0.322$ (that corresponds to the slope of the most popular 80% learning curve, see [3]). However, to avoid infinity relative errors (i.e., to guarantee that at least one task is late), we set for one task $d_1 = r_1$. Values of the parameters of the algorithms were chosen empirically as follows:

- PTS: $|TabuList| = 10$ and diversification is after 7 iterations,
- PSA: $T_0 = 1000000$ and $\lambda = 0.01$,

where the stop condition for PTS1 and PSA1 is the fixed running time (i.e., the number of iterations is immaterial). Therefore, their efficiency in finding the best solution for different problem sizes can be compared for the same running times of the algorithms.

The considered algorithms are evaluated, for each instance I, according to the relative error $\delta_A(I)$ that is calculated in the following way: $\delta_A(I) = \left(\frac{TT(\pi_I^A)}{TT(\pi_I^*)} - 1\right) \cdot 100\%$, where $TT(\pi_I^A)$ denotes the criterion value provided by algorithm $A \in \{$PNEH1, PTS1, PSA1$\}$ for instance I and $TT(\pi_I^*)$ is the optimal solution of instance I (if $n = 10$) or the best found solution of instance I (if $n \geq 10$) provided by the considered approximation algorithms. The optimal solution is provided by extensive search algorithm.

The results concerning the mean total tardiness $\bar{TT}$, the percentage values of mean relative errors $\bar{\delta}$ provided by the analysed algorithms as well as the number of instances $best$ for which each algorithm found the best criterion value (per 100 instances) and their running times $\bar{t}$ are presented in Table I.

TABLE I
THE MEAN TOTAL TARDINESS $\bar{TT}$, PERCENTAGE VALUES OF MEAN RELATIVE ERRORS OF ALGORITHMS $\bar{\delta}$, THE NUMBER OF INSTANCES $Best$ FOR WHICH EACH ALGORITHM FOUND THE BEST CRITERION VALUE (PER 100 INSTANCES) AND THEIR RUNNING TIMES $\bar{t}$

Algorithm \ n		10	50	100	150	200
PNEH1	$\bar{t}$[s]	0.00	0.03	0.39	1.15	1.89
	$\bar{TT}$	70	518	1309	2214	3516
	$\bar{\delta}$[%]	29.98	380.52	1 157.33	2 145.24	3 262.25
	best	31	2	0	0	0
PSA1	$\bar{t}$[s]	6.00	30.00	60.00	60.00	60.00
	$\bar{TT}$	13	7	2	1	2
	$\bar{\delta}$[%]	0.02	0.03	0.02	0.00	0.00
	best	91	93	96	100	100
PTS1	$\bar{t}$[s]	6.00	30.00	60.00	60.00	60.00
	$\bar{TT}$	13	7	2	1	4
	$\bar{\delta}$[%]	0.00	0.00	0.00	0.00	1.73
	best	100	100	100	100	94

It can be seen in Table I that the proposed PSA1 and PTS1 are characterized with very low mean relative errors. Among them the best results are provided by PTS1 for $n < 200$, where the mean total tardiness did not exceed 13, i.e., average tardiness of a job is less than 0.2. The results provided by PSA1 are similar, however, PSA1 is better than PTS1 for $n \geq 200$. Moreover, PNEH1 revealed to be fast algorithm that finds solutions, which mean relative errors are high, however, the average tardiness of a job does not exceed 18 even for $n = 200$. Therefore, it can be use to provide a good initial solution for other algorithms or it can be applied individually in learning systems if the running (computation) time is crucial.

The second part of the numerical analysis focus on decreasing running times of the algorithms if parallel computation approaches are applied. The parameters of the algorithms are the same as in the previous experiment, only the iterations are different. PTSi (where $i \in \{1, 2, 4\}$) have the same number of iterations for each i, $Iterations_{PTS} = 30$, whereas for PSAi the number of iterations depends on the number of threads, i.e., $Iterations_{PSA}/i$ for PSAi, where $Iterations_{PSA} = 1800000$. Therefore, PTS$i$ search the same solution space. It is similar for PSA with a small difference concerning the value of initial temperature for each thread. The running times of PSA, PNEH and PTS depending on the number of working threads are presented in Fig. 4–9 for the problem with $n = \{150, 200\}$ tasks.

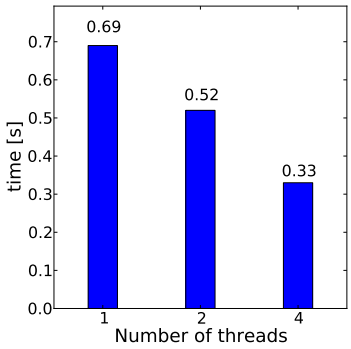

Fig. 4. Running times of PNEH ($n = 150$).

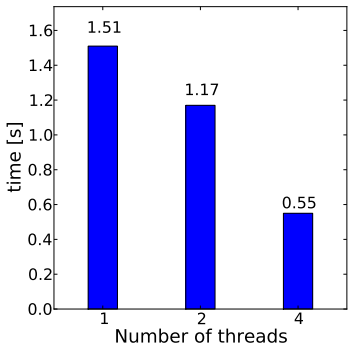

Fig. 5. Running times of PNEH ($n = 200$).

It can be seen in the figures that the applied parallel computation approach has the smallest impact of on the running times of PNEH and in case of PSA the decreasing of the running times is almost proportional to the number of applied threads that it is also similar for PTS.

At first let us analyse PSA. In this case, the most of the calculations are done by threads, therefore, the relation between the running time and the number of applied threads is almost proportional (Fig. 8). The number of tasks n does not affect the efficiency of the parallel computation approach on the running times of PSA (see Fig. 8 and Fig.9).

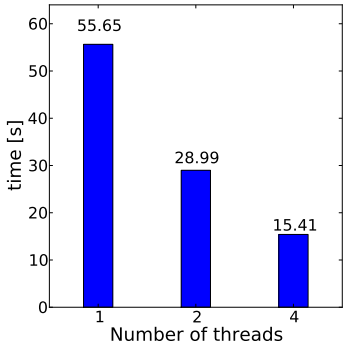

Fig. 6. Running times of PTS ($n = 150$).

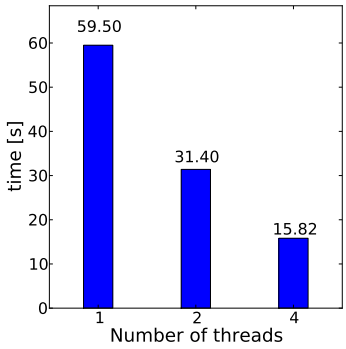

Fig. 7. Running times of PTS ($n = 200$).

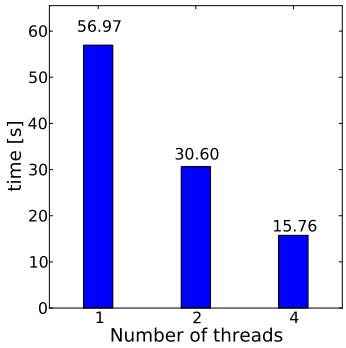

Fig. 8. Running times of PSA ($n = 150$).

For PTS the number of calculations in the main process is negligible in reference to the calculations done by threads, since for each iteration of the main process, the threads have to calculate $O(n^2)$ moves. Therefore, the efficiency of the parallel computation approach for PTS increases with the number n of tasks (see Fig. 6 and Fig.7) and the relation between the running time and the number of threads is also almost proportional for $n = 200$.

However, for PNEH, the number of calculations done by threads is not so significantly greater than calculations done by the main process, since for each iteration of the main process the threads calculate $O(n)$ moves (insertions) and corresponding criterion values. Thus, the impact of the analysed parallel computation approaches in the running times of PNEH is smaller than in case of PTS (see Fig. 4 and Fig.5).

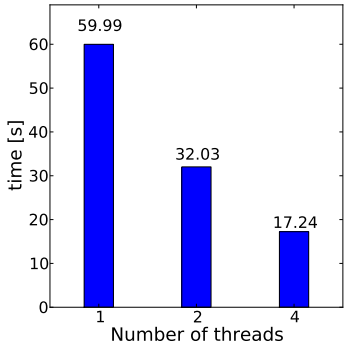

Fig. 9. Running times of PSA ($n = 200$).

V. Conclusions

In this paper, we pointed out that learning that is present in many systems can be additionally utilized by the sequence of processed tasks such that the system objectives are improved. Namely, we analysed the minimization of the total tardiness problem in the learning system, where tasks have different release times. To solve it we proposed parallel computation approaches that are based on NEH, tabu search and simulated annealing. The numerical analysis confirmed high accuracy of these methods, furthermore, we showed that the parallel computation approaches significantly decrease running times of simulated annealing and tabu search and also reduces the running times of NEH. Furthermore, the presented algorithms can be easily tuned to solve other combinatorial optimization problems.

References

[1] L. Buşoniu, R. Babuška, and B. De Schutter, "A comprehensive survey of multiagent reinforcement learning," *IEEE Transactions On Systems, Man, And Cybernetics - Part C: Applications And Reviews*, vol. 38, pp. 156–172, 2008.

[2] A. Grzech and P. Świątek, "Modeling and optimization of complex services in service-based systems," *Cybernetics and Systems*, vol. 40, pp. 706–723, 2009.

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

[4] H. Kerzner, *Project management: a system approach to planning, scheduling, and controlling*. New York: John Wiley & Sons, Inc., 1998.

[5] P. Świątek, A. Grzech, and P. Rygielski, "Adaptive packet scheduling for requests delay guaranties in packet-switched computer communication network," *Systems Science*, vol. 36, pp. 7–12, 2010.

[6] A. Janiak and R. Rudek, "A note on the learning effect in multi-agent optimization," *Expert Systems with Applications*, vol. 38, pp. 5974—-5980, 2011.

[7] ——, "Experience based approach to scheduling problems with the learning effect," *IEEE Transactions on Systems, Man, and Cybernetics - Part A*, vol. 39, pp. 344–357, 2009.

[8] M. Nawaz, J. E. E. Enscore, and I. A. Ham, "A heuristic algorithm for m-machine, n-jobs Flow-shop sequencing problem," *Omega*, vol. 11, pp. 91–95, 1983.

[9] F. Glover and M. Laguna, *Tabu search*. Norwell, MA: Kluwer, 1997.

[10] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, 1983.