Computer-aided solution of difficult combinatorial optimization problems

Theses of Ph.D. dissertation

Written by: Gyula Ábrahám

Supervisors: Professor György Dósa, Professor Ágnes Werner-Stark

University of Pannonia
Faculty of Information Technology
Doctoral School of Information Science and Technology

2022

1. Background and motivation

In the dissertation I have dealt with three problems related to the areas of scheduling and bin packing. All three areas have many applications in practice, including industry, economy or scheduling. In the solution of scheduling problem (unrelated machine scheduling with precedence constraints) I have used a well-known algorithm from the field of reinforcement learning as a basis for my solution to determine the order of tasks. During the solution of bin packing problems I have used so called pre-processing algorithms. Furthermore, I have dealt with a relatively new area which is called bin covering with delivery (BCD for short). In this area, on the one hand, I have solved benchmark problems with well-known algorithm from literature and compared the solutions with the results of a new, flexible algorithm I have developed. All three considered problems are quite difficult, complex combinatorial optimization tasks.

1.1. Basically in a scheduling problem tasks (jobs) and resources (machines) are given. During a scheduling it is determined which task is executed by which machine and when. The number of tasks is given. The resources are used to execute the tasks. The goal is to assign these tasks to resources to optimize an objective function. In the case I have examined the goal was to minimize the makespan. To solve the scheduling problem an algorithm was developed motivated by Q-learning which I have named **Q-Learning Motivated algorithm**, **QLM** for short. The algorithm consists of two parts: a greedy scheduler and a procedure based on Q-Learning that determines the order of tasks.

A detailed overview of scheduling can be found, for example, in [1]. The book discusses theoretical models related to scheduling and various scheduling problems in great detail. The field of scheduling has a very great literature, here I mention two basic works by Ronald L. Graham [2, 3]. In these articles, he defined the famous LS (*List Scheduling*) algorithm, which can be considered the first online scheduling algorithm; and its ordered version, the LPT (*Longest Processing Time*) algorithm.

In the field of scheduling I have dealt with unrelated machine scheduling with precedence constraints. For the solution the Q-Learning algorithm was used from the field of reinforcement learning. To the best of my knowledge, there is no such solution to the above-mentioned scheduling problem in the literature, however, there are examples of the application of reinforcement learning in other areas of scheduling. Orhean et al. [4] introduced a scheduling method for distributed cloud systems based on reinforcement learning. The goal was to optimize the performance of a system by resource scheduling. Aydin and Öztemel [5] developed an agent-based scheduling method, in which the agent selects rules under various conditions, based on which the scheduling takes place. An improved version of Q-learning was used to train the agent. Stefán [6] applied the Q-learning algorithm to a permutation flow shop problem, where the goal was to minimize the idle time of the machines. In his dissertation, Stefan [7] gave a more detailed description of the algorithm, which was designed to solve the flow shop type problem. The article [6] and the dissertation [7] helped me to see the problem I have dealt with from the perspective of reinforcement learning. Gabel and Riedmiller [8] also applied Q-learning, but they applied it to a job shop type problem in which the Q-function was approximated using a neural network. Shahrabi et al. [9] used reinforcement learning to improve a procedure developed for a job shop type problem. More examples of the application of reinforcement learning in scheduling can be found in [10–12]. In [13], for example, a version of Q-learning combined with neural networks, called Deep Reinforcement Learning, was used.

1.2. In the case of bin packing we want to pack items into bins so that the number of used bins is minimal and the size of the items packed into the same bin don't exceed the capacity of the bin. This problem is NP-hard [14, 15]. The bin packing problem was defined and started to be investigated at the beginning of the 70's. The so-called approximation algorithms were developed in this area. An algorithm is called an approximation algorithm if it is not expected to necessarily provide an optimal solution to a problem, but on the one hand it is fast (with polynomial time), and on the other hand the solution it provides is guaranteed to be "not too far" from the optimum value.

D. S. Johnson's dissertation [16] on bin packing and Graham's work [2] are among the early works that started and formed the study of approximation algorithms and set the direction for further research. In the area of bin packing, we distinguish between online and offline cases. In the online case, the details of the data are not known in advance, but in the offline case, they are.

In this area, I have dealt with a new kind of approach to bin packing problems. The idea of this approach is that before solving bin packing problems, a so-called pre-processing step is performed to simplify the problem to be solved. If there is a bin packing problem to be solved optimally, it can be difficult. But this does not mean that finding the optimal solution for every single input is difficult. In the new approach, I have tried to determine the optimal solution for the given input by a greedy algorithm; if this is successful, we are ready. If we choose another, more complex algorithm. The essence of using the greedy algorithm is to optimally solve as many inputs as possible within the given class with a simple algorithm. Hence, the number of inputs decreases by the solved ones. To solve the inputs, I have created two algorithms named **REM WS** and **FU**, whose effectiveness was tested on known and freely accessible benchmarks (Schwerin and Falkenauer_U).

1.3. Bin covering with delivery is a relatively new area. Similar to the bin packing problem, in this case too the items are packed into bins but when they become covered they are closed and delivered. The objective function is determined based on the number of covered and delivered bins. That is, we receive money for each delivered bin and the goal is to maximize profit. The problem was introduced first in [17]. An offline version of this problem is covered in [18], and some related problems are presented in [19].

In the online version of the BCD problem, the items are not known in advance and arrive one after the other. The incoming item must be immediately packed into a bin. The objective function changes depending on the number of open bins. The more bins are open at the same time, the more the value of the objective function decreases. The goal is to maximize the objective function, i.e., total profit. The offline and online versions of the bin packing and bin covering problem are dealt with, among others, in the review articles [14], [20] and [21].

During the research, I have implemented natural algorithms that are already known from the literature (**DNF**, **H(K)** and **SH(K)**) and developed a new, flexible, parametric algorithm I have named **MMask**. To test the algorithms, I have used the previously applied Schwerin and Falkenauer_U classes and a new problem class which was created by me called Large Range (abbreviated: LR). I have defined three profit functions: a slowly decreasing version, a quadratically decreasing version, and a strongly decreasing version. I have made modifications to the problem classes: I have broken the ordering, normalized the inputs and connected the problem classes with the profit functions.

2. Methods and Tools

2.1. Scheduling

2.1.1. The scheduling problem

In the thesis, I have dealt with the unrelated machine scheduling with precedence constraints problem, which can be given as follows:

$$R_m|prec|C_{max}. \tag{1}$$

Here R_m is the set of machines (m unrelated machines), $prec$ denotes that there are precedence constraints between tasks and $C_{max} = max(C_1, \dots, C_n)$ means the completion time of the latest activity in the system, which is minimized. The goal is to minimize the makespan. One input of the problem

$$\langle \mathcal{T}, \mathcal{M}, G \rangle \tag{2}$$

can be described by an ordered triple, where $\mathcal{T} = \{task_1, \dots, task_n\}$ is the set of all tasks, $\mathcal{M} = \{m_1, \dots, m_m\}$ is the set of machines and $G = (V, E)$ is a graph, where V is a finite set of vertices, $E \subseteq V \times V$ is the set of edges. We write an edge of the graph in the form $(v_i, v_j) \in E$ and has to fulfill the following constraints:

- directed graph, i.e., $E \subseteq V \times V$ is the set of ordered pairs formed by vertices,
- simple graph, i.e., $(v_i, v_j) \in E$ for $i \neq j$ (loop-free) and $(v_j, v_i) \notin E$ (no multiple edges) for $\forall i, j$,
- union of disjoint paths and isolated points.

2.1.2. The proposed method

The algorithm can be divided into two components. One is a greedy algorithm that schedules tasks in a given order. The other is the component supported by Q-learning, which generates the permutation of the tasks. The goal is for the algorithm to find a permutation according to which, when greedily scheduled, the makespan is minimal. The algorithm tries to find the best order, but does not guarantee an optimal solution. The greedy algorithm always assigns that machine to the next task with which the makespan achieved so far increases the least while also taking into account the precedence constraints.

In the first step, the QLM algorithm creates a random order among the tasks. The order is random in the first step because the elements of the Q table are zero, so the selection probability is random. During the process, the values of the Q table are continuously updated according to rule (3).

$$Q_{t+1}(S_t, A_t) = (1 - \alpha_t)Q_t(S_t, A_t) + \alpha_t(R_t + \gamma_t \max_a Q_t(S_{t+1}, a)), \tag{3}$$

where S_t and A_t are the current state and the currently selected action. α_t is the learning rate, which controls the speed of learning, and γ_t is the discount rate determines the importance of future rewards. R_t is the value of the reward received in the current state S_t, after selection of the action A_t.

The strategy is initially exploratory, which means that the selection is random, then as it starts to converge more and more towards the solution, the strategy becomes more exploitative. The strategy used is called the Boltzmann discovery strategy, which determines the probability of the next selected element, in our case, task in the permutation, based on the calculated Q values. The algorithm performs the greedy schedule based on the computed permutation and compares it with previous results. Based on the quality of the new result, it determines a reward R_t, which is part of the update rule (3) and affects the change of Q values.

2.1.3. Results

In order to test the effectiveness of the developed algorithm, I have solved small problem presented in the articles [22, 23], and also generated four problem classes based on the parameters of the problems in the articles.

The machine times of the small problem were available, the optimal solution of the problem was 13. The solution of the algorithm published in [22] was 15, and the solution of the QLM algorithm I have developed was 13. For further testing of QLM, I have chosen three of the 33 problems published in [22] and one of the problems published in [23]. Keeping the numbering of the problems used in [22], #1, #2 and #5 were selected. I have chosen a large problem, number #28 from [23]. Based on this data, I have created four classes. n is the number of activities, m is the number of machines, and NC denotes the number of precedence constraints.

- *Class* #1 $\rightarrow n = 14$, $m = 8$ és $NC = 5$
- *Class* #2 $\rightarrow n = 28$, $m = 7$ és $NC = 8$
- *Class* #3 $\rightarrow n = 27$, $m = 4$ és $NC = 1$
- *Class* #4 $\rightarrow n = 74$, $m = 19$ és $NC = 10$

I have shown that the QLM algorithm is capable of solving the scheduling problem presented in this chapter. The algorithm was specially developed for those scheduling problems where the execution time of the activities depends on the machine, the execution cannot be preempted, and there may be precedence constraints between the tasks. I haven't find any publication on the subject of reinforcement learning to solve the problem I have specified. The solution given in [22] and [23] could have been a basis for comparison, but the details of the problems solved here are not known. So basically, I have tested the QLM algorithm with problems generated by myself, and also solved the same problems with CPLEX. Based on the results, it can be seen that QLM found the optimal solution in all cases where CPLEX did. In the other cases, we do not know for sure that QLM found an optimal solution because CPLEX could not confirm this. Based on this, it can be seen that, limited to the investigated problems classes, the QLM algorithm is efficient and competes with the CPLEX solver in solving the problems, since the results provided by QLM are at least as good as the results provided by CPLEX, but QLM is much faster than CPLEX.

2.2. Bin packing

2.2.1. The bin packing problem

In case of bin packing problem, items are packed into bins. Each item has a size w_i and each bin has a capacity C. The packing is performed so that the total size of the items packed into same bin does not exceed the capacity of the bin, and the number of used bins is minimum. This problem is NP-hard.

2.2.2. The proposed method

The purpose of the pre-processing is to optimally solve as many of the instances in the problem classes as possible with a simple algorithm. To solve the Schwerin and the Falkenauer_U benchmark classes, I have created two greedy algorithms called REM SW and FU. The two algorithms were developed specifically for these two classes. The REM SW algorithm selects the value of $0 \leq k \leq 6$ so that the k largest items and the $6-k$ smallest items are placed into one bin. After packing the largest k items, fill the remaining space with the $6-k$ items as best as possible. For the latter, it uses the pathfinding slave algorithm. The FU algorithm works with four slave algorithms that pack pairs, triplets, and quadrets by taking into account a value r that is the lower bound of the remaining (reserve) space in the bin. In the last step, the algorithm packs the remaining items with the FFD algorithm.

In the case of both classes, I have examined the characteristics of the instances included in them and what advantages can be gained from them in terms of the solution. Based on the discovered properties, all tasks of the Schwerin class and 91% of the tasks of the Falkenauer_U class were solved optimally. Based on the results, it can be seen that the pre-processing in the case of the investigated classes helps.

2.2.3. Results

Limiting ourselves to the Schwerin and the Falkenauer_U classes, it can be stated that the developed algorithms achieved very good results. In the case of the Schwerin class, all of the 200 tasks were optimally solved by the REM SW algorithm, i.e., the ratio of optimal solutions is 100%. The FU algorithm also performed very well on the tasks of the Falkenauer_U class, as it provided an optimal solution for 73 of the 80 tasks, which represents a 91% ratio for optimal solutions.

The introduced greedy algorithms are fast, simple and find the optimal solution in most cases.

The general applicability of the described algorithms (Rem SW and FU) is different. Schwerin algorithm takes advantage of the fact that the item sizes are very close to each other and there are 5 or 6 items in each bin. However, the algorithm can also be used in cases where the item sizes are still between 150 and 200, but the bin capacity is not 1000, but for example 1200. Then 6 or 7 items will be packed into each bin. (8 items can be packed into one bin if and only if each of them has a size of 150, but the chance of this is very small.)

The FU algorithm can be good in general if the size of the items is between 0 and C. It packs pairs, threes and fours. To do this packing in such a way that the items fill the bin well. And the rest are packed with FFD.

2.3. Bin covering

2.3.1. The bin covering problem

The bin covering problem is similar to the bin packing problem, but with a few additions. Items arrive one by one. In order, the size of item i. is $w_i > 0$ and we assume that an infinite number of bins with the same capacity C are available. Furthermore, a positive integer $K > 0$ is given, which specifies how many bins can be open at the same time. It means that the packing algorithm can only open a new bin if the number of open bins is less than K.

A bin is considered covered if the total size of the items packed into the bin is at least equal to the capacity C. Also given an objective function G for which

$$G : \{1, \ldots, K\} \to R^+. \tag{4}$$

If $1 \leq k \leq K$ bins are open at a given time, and one of them becomes covered and is delivered, then the realized profit is $G(k)$. After each covered and delivered bin the value of k decreases by one, but a new bin can be opened at any time if and only if there are no more open bins than K. The function G is a monotonically decreasing, positive-valued function. The goal is to maximize the profit we get after the bins are covered and delivered.

2.3.2. The proposed method

During the investigation, I have implemented already existing algorithms (DNF, H(K), SH(K)), and also created a new, parametric algorithm called MMask. To test the algorithms, similar to the bin packing topic, I have used the Schwerin and the Falkenauer_U task classes, and in addition I have created a new problem class called Large Range. As a first step, I have prepared the problems as follows:

(a) **Disordering.** I have randomly mixed the items based on their sizes.

(b) **Falkenauer class normalization.** I have changed the capacity of the bin to $C = 1000$, so the size of each item had to be multiplied by $\frac{1000}{150}$.

(c) **New problem class.** I have created a new class where the item sizes are from the interval $[1, 1000]$. There are a total of 400 tasks in this class, each with a maximum of 1000 items.

(d) **Profit functions.** No profit function is specified for the original benchmarks. Hence, I have created three functions:

- $G1(k) = 10,1 - 0,1 \times k$
- $G2(k) = 11 - k$
- $G3(k) = 10,05 - 0,05 \times k^2$

(e) **Connection of item types and profit functions.** Finally, problem classes were combined with the three profit functions using the following notations:

- $SiGu$
- $FjGu$
- $LRjGu$

where $i = 1, 2$, $j = 1, 2, 3, 4$ és $u = 1, 2, 3$.

I have implemented some algorithms which are already known from literature. These are DNF, H(K) and SH(K). Each algorithm works in a very simple way:

- **DNF:** The next item is always packed into the currently open bin. If the bin becomes covered, the algorithm closes it and opens a new one for the next item. If there are no more items, the algorithm stops.

- **H(K):** The items whose size falls in the interval $I_k = \left(\frac{1}{k+1}, \frac{1}{k}\right]$ are packed into the bin of type k., where $k = 1, \ldots, K - 1$. The smallest items, i.e., those whose size falls within the interval $I_K = \left(0, \frac{1}{K}\right]$) are packed into the bin of type K. If a bin becomes covered, the algorithm closes it. If no such bin is open, it will open one of this type. If there are no more items, the algorithm stops.

- **SH(K):** If the following item can cover one or more bins, it is packed into the bin with the lowest load. The algorithm then closes the bin. In other cases, the SH(K) algorithm works as H(K) algorithm.

The newly developed algorithm called MMask has three parameters: K is a positive integer defining the maximum number of open bins at the same time, α is a K-dimensional nonnegative vector, and β is a positive integer. MMask is based on an accept-reject policy. The algorithm accepts the packing of the next item if, after packing, the load of the given bin falls within the acceptance range. Otherwise, it will reject to pack the item.

- The accepting area in case of k^{th} bin ($1 \leq k \leq K$): $[0; C - \alpha_k] \cup [C; C + \beta]$
- The rejecting area in case of k^{th} bin ($1 \leq k \leq K$): $(C - \alpha_k; C) \cup (C + \beta; \infty)$

MMask: If the next item can be packed into a bin that makes the bin covered in the accepting area, then the current item will be packed into that bin. If there are several such bins, then the bin is chosen with smallest load. If the next item can be packed into a bin whose load falls within the acceptance area after packing, but will not be covered, then the item will be packed into this bin. If there are more of this kind of bin, the algorithm chooses one at random. If $k < K$, the algorithm opens a new bin. The bin type is the smallest k value for which does not have an open bin, and packs the item here. If $k = K$ is satisfied, the current item is packed into the bin with the smallest load. If the bin is covered, the algorithm closes it. If there are no more items, the algorithm stops.

An algorithm based on local search is developed for setting the parameters of the MMask. The neighborhood structure can be defined in a natural way between the different MMask parameter settings. The neighbor of a specific parameter setting is obtained by modifying one of the parameters K, α, β. Let Δ be a small positive constant, then parameters α_i and β will be increased or decreased by the value of Δ so that the new value remains positive and is smaller than the bin capacity. The value of K also changes by 1 unit in a negative or positive direction. The direction of change is random.

2.3.3. Results

Based on the results of the DNF, H(K) and SH(K) they gave the same results for the S1 subclass. Although the K value increased, the profit did not increase (160). This means that, in the case of S1, the algorithms behaved as DNF even with the $K > 1$ setting. In the case of F1 and F4, the SH(K) algorithm achieved the highest profit for $K = 2, 3, 4, 5$ values. In the case of the LR1 and LR4 classes, the SH(K) algorithm gave the best profit values with the $K = 4$ setting, in two cases it improved the profit with the $K = 5$ setting compared to the $K = 4$ setting.

In addition to the manual setting of the parameters, the MMask algorithm did not improve the profit for any task of S1, for F1 and F4 it improved the value of the profit for all tasks, and for LR1 and LR4 task classes it also improved the results, except for three cases.

Applying the parameter settings chosen by local search, for the tasks of F1 and F4 classes, except for one case, the profit was improved everywhere compared to the previous algorithms and the manual setting. For classes LR1 and LR4, it also improved the profit everywhere, except for three cases.

Based on the tests, it can be said that a "good parameter setting" can be found with the help of the local search. Of course, even better settings could probably be achieved with more advanced algorithms, but this is a future work option at this time.

The local search can be used to find better parameter settings, i.e., it is capable of automatic search of parameters in such a way that it achieves better results than the results of previous algorithms and manually set parameters. This is a big help, as it eliminates the manual setting of the parameters. Furthermore, the fusion of MMask and local search as a parameter optimization procedure proved to be effective.

3. New scientific results

Thesis 1 *I have created a new algorithm to solve the problem of scheduling of unrelated machines with precedence constraints ($R_m|prec|C_{max}$). The developed algorithm determines the sequence of tasks using an algorithm based on Q-learning known from the field of reinforcement learning, and then schedules the tasks in this order with a greedy scheduler. For the comprehensive examination of the algorithm, I have created four new benchmark classes. The problem classes are varied based on their structure and difficulty.*

1.1 I have compared the obtained results with the results provided by the CPLEX solver. Based on the evaluation, it can be seen that the QLM algorithm I have created finds the optimal solution in all the examined problem classes when CPLEX does. In the other cases, QLM produced at least as good results as CPLEX. Limiting ourselves to the problem classes, it can be stated that the QLM algorithm is efficient and competes with the CPLEX solver in solving the problems, but it is much faster.

The corresponding publications are: [P1], [P4], [P5], [P6], [P7], [P8], [P9]

Thesis 2 *I have dealt with the bin packing problem from a new perspective, from the side of pre-processing. I have used the Schwerin and the Falkenauer_U classes for the research. I have created two algorithms called REM SW and FU. I have designed the REM SW algorithm for Schwerin and the FU algorithm for solving the problems of the Falkenauer_U class.*

2.1 In the case of both classes of problems, I have examined the characteristics of the instances included in them and what advantages can be gained from them in terms of the solution. Based on the discovered properties, all tasks of the Schwerin class and 91% of the instances of the Falkenauer_U class were solved optimally.

2.2 I have examined the algorithms not only according to how many instances of the given class can be solved optimally, but also compared the my algorithms with the HEA algorithm. The HEA algorithm is one of the most efficient (and one of the newest) general metaheuristic algorithms for solving this and other benchmark classes. Comparing the REM SW algorithm with the HEA [24] algorithm, the average running times for REM SW were much better. For Schwerin 1 it was 0.0038 (HEA - 0.34 seconds) seconds, while for Schwerin 2 it was 0.004 (HEA - 0.47 seconds) seconds. The running time of the FU algorithm is also significantly shorter than the running time of the HEA even for 80 problems of the Falkenauer_U class.

The corresponding publication is: [P2]

Thesis 3 *I have done detailed investigations related to problem called "bin covering with delivery", as part of it I have used the Schwerin and the Falkenauer_U classes, and also created new benchmark class (LR). I have created a new, parametric algorithm called MMask, which implements an accept-reject policy during the packing of the items, and also automatically sets its parameters by local search.*

3.1 The solvability of the benchmarks was partially investigated with naturally occurring algorithms known from the literature (DNF, H(K), SH(K)). Based on the evaluation of the results, I have found that these algorithms effectively solve the elements of the problem classes in the vast majority of cases.

3.2 I have introduced MMask. Furthermore, I have found that with the right choice of parameters, the MMask algorithm was able to achieve better results than the previous results. To optimize the parameters, I have created a local search-based heuristic that determined the parameter settings that achieve the best results. The optimizer was able to find even better settings than manual settings.

The corresponding publication is: [P3]

4. Publications related to the thesis

The described results have been presented in journals, international conferences, and conference proceedings. I list them below.

[P1] **Gy. Ábrahám**, P. Auer, Gy. Dósa, T. Dulai and Á. Werner-Stark. A reinforcement learning motivated algorithm for process optimization. Periodica Polytechnica Civil Engineering, 63(4):961 970, 2019. (**IF: 1.34**)

[P2] **Gy. Ábrahám**, Gy. Dósa, T. Dulai, Zs. Tuza and Á. Werner-Stark. Efficient Pre-Solve Algorithms for the Schwerin and the Falkenauer_U Bin Packing Benchmark Problems for Getting Optimal Solutions with High Probability. Mathematics, 9(13):1540, 2021. (**IF: 2.592**).

[P3] **Gy. Ábrahám**, P. Auer, Gy. Dósa, T. Dulai, Zs. Tuza and Á. Werner-Stark. The bin covering with delivery problem, extended investigations for the online case. Central European Journal of Operations Research, pages 1-27, 2022. (**IF: 2.345**)

[P4] Á. Werner-Stark, T. Dulai and **Gy. Ábrahám**. Modeling of an agent system to support the management of cooperating and rival resources for business workflows. In 2014 4th International Conference On Simulation And Modeling Methodologies, Technologies And Applications (SIMULTECH), pages 407-412. IEEE, 2014.

[P5] T. Dulai, Á. Werner-Stark and **Gy. Ábrahám**. Support of Efficient resource allocation of technological processes by a heuristic solution and agent technology. Data Envelopment Analysis and its Applications, page 153, 2016.

[P6] T. Dulai, Á. Werner-Stark and **Gy. Ábrahám**. Improvement of resource allocation in workflows by stochastic method. In 10th International Conference on Applied Informatics, 2017.

[P7] **Gy. Ábrahám**, T. Dulai, Á. Werner-Stark and Gy. Dósa. Reinforcement learning in process scheduling. In 13th Miklós Iványi International PhD DLA Symposium - Abstract Book: Architectural, Engineering and Information Sciences, pages 15-15, 2017.

[P8] **Gy. Ábrahám**, T. Dulai, Á. Werner-Stark and Gy. Dósa. Learning the parameters of a reinforcement learning algorithm for process optimization. In Veszprém Optimization Workshop, pages 11-11, 2019.

[P9] **Gy. Ábrahám**, T. Dulai, Á. Werner-Stark and Gy. Dósa. Parameter optimization of q-learning motivated algorithm in process scheduling. In Proceedings of the Pannonian Conference on Advances in Information Technology, pages 17-24, 2020.

The listed publications has received the following references until the date of submission of the thesis: [25], [26]

5. Further research and development opportunities

5.1. Regarding to the scheduling problem, in addition to the presented solutions, the subject of further investigation may be in certain special cases (for example, the case of $m = 2$ machines, or the case of only two types of execution time) whether we could obtain better lower bounds and check how the QLM algorithm or some modified version of it works in this case. Furthermore, it is worth to examine other different structures, e.g., tree structure, how our method works. Of course, new benchmark classes would be needed for this case.

5.2. Regarding to bin packing, I present some additional options that can be used to improve the FU algorithm in the future. In the case of the Schwerin class, since all the tasks were solved optimally, no further research is needed. However, in the case of the Falkenauer_U class, the rate was 91%, so there is place for further investigation:

1. Testing other parameters, which can be determined manually, based on experience, or automated with some search algorithm. We finally select the best one from the results given by the different settings.

2. Introduction of additional parameters, e.g., we not only examine the existence of large items, but also take into account their numbers.

What happens when the number of items increases is also an unanswered question at the moment. Let's assume that the capacity of the bin, i.e., the value of C is a fixed integer, so consequently the size of the items comes from the interval $[1, C]$. Then, according to Lenstra's result [27], the problem can be solved in polynomial time, where n is the number of objects and n can be arbitrarily large. Despite being solvable in polynomial time, the number of steps of such algorithm would be very large, since the exponent of n would be very large, and the coefficient of the $O(.)$ expression would also be large. For these reasons, such an algorithm may not be usable in practice, perhaps after some kind of pre-processing.

Furthermore, this theorem does not take into account the fact that the size of the items is randomly drawn from a given interval with a uniform distribution. Based on these, we can make the following conjecture.

Conjecture: *Let $1 \leq a < b \leq C$ be fixed integers, where C is the size of the bin. Let us assume that the sizes of the items are drawn randomly and uniformly distributed from integers $a, a+1, \ldots, b$. Then there exists an algorithm whose running time can be estimated from above with a low-order polynomial and the coefficient of the expression $O(.)$ is small, and the probability of the optimal solution of the problem approaches 1 when $n \to \infty$.*

For example, an algorithm with a running time of $20n^4$ and a probability of 0.9 to find the optimal solution for $n = 1000$ can already be interesting. I have formulated my previous expectation as a conjecture because we actually know little about what can be done in such cases. This would require further investigations.

5.3. In the field of bin covering, I give some possibilities for further development of the MMask algorithm.

- Some of the presented parameter settings were set manually. We may get much better results with other settings.

- It has turned out that in the case of the LR class, the SH(4) algorithm is very efficient in several cases. Based on this, the fusion of SH(4) and the MMask algorithm should be considered (we run both algorithms separately and choose the better result).

- In its 2nd step, the MMask algorithm could choose arbitrarily from the available suitable bins. It is possible that if it chooses a bin "smarter", then the performance of MMask will be improved.

- Another modification option for step 2 of the algorithm was the following. Let $B_1, B_2, \ldots, B_t$ be the bins the current items can be packed into. If $t = 1$, then it is clear in which bin the current item went, since only one bin is open. If, on the other hand, $t > 1$, the algorithm can choose from several bins. For each corresponding bin, I have calculated the average size of the items in them, denoted by $x_1, x_2, \ldots, x_t$, and the size of the current item by x. Then, for each bin, I have computed the value of the expression $(x - x_i)^2$ and chosen the smallest one, which specified that it should be packed into the bin B_i, where $1 \leq i \leq t$. Unfortunately, this strategy did not prove to be effective. In some cases it improved the result, in other cases it made it worse, but overall the procedure did not improve.

References

[1] M. L. Pinedo. *Scheduling: Theory, Algorithms and Systems.* Springer Publishing Company, Incorporated, Boston, MA, USA, 4th edition, 2012.

[2] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[3] R. L. Graham. Bounds on multiprocessing timing anomalies. *Siam Journal on Applied Mathematics*, 17(2):416–429, 1969.

[4] A. I. Orhean, F. Pop, and I. Raicu. New scheduling approach using reinforcement learning for heterogeneous distributed systems. *Journal of Parallel and Distributed Computing*, 117:292–302, 2018.

[5] M. E. Aydin and E. Öztemel. Dynamic job-shop scheduling using reinforcement learning agents. *Robotics and Autonomous Systems*, 33(2):169–178, 2000.

[6] P. Stefan. Flow-shop scheduling based on reinforcement learning algorithm. *Production Systems and Information Engineering*, 1(1):83–90, 2003.

[7] P. Stefan. *Combined Use of Reinforcement Learning And Simulated Annealing: Algorithms and Applications.* PhD thesis, University of Miskolc, Miskolc, 2003.

[8] T. Gabel and M. Riedmiller. Adaptive reactive job-shop scheduling with reinforcement learning agents. *International Journal of Information Technology and Intelligent Computing*, 24(4):14–18, 2008.

[9] J. Shahrabi, M. A. Adibi, and M. Mahootchi. A reinforcement learning approach to parameter estimation in dynamic job shop scheduling. *Computers & Industrial Engineering*, 110:75–82, 2017.

[10] W. Zhang and T. G. Dietterich. A reinforcement learning approach to job-shop scheduling. In *IJCAI*, volume 95, pages 1114–1120. Citeseer, 1995.

[11] M. Zweben, E. Davis, B. Daun, and M. J. Deale. Scheduling and rescheduling with iterative repair. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(6):1588–1596, 1993.

[12] Z. Huang, W.M.P. van der Aalst, X. Lu, and H. Duan. Reinforcement learning based resource allocation in business process management. *Data & Knowledge Engineering*, 70(1):127–145, 2011.

[13] Y. Ye, X. Ren, J. Wang, L. Xu, W. Guo, W. Huang, and W. Tian. A new approach for resource scheduling with deep reinforcement learning. *CoRR*, abs/1806.08122, 2018.

[14] E. G. Coffman Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: A survey. *Approximation algorithms for NP-hard problems*, pages 46–93, 1996.

[15] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, volume 174. W. H. Freeman and Co., San Francisco, 1979.

[16] D. S. Johnson. *Near-optimal bin packing algorithms.* PhD thesis, Massachusetts Institute of Technology, 1973.

[17] A. Benkő, Gy. Dósa, and Zs. Tuza. Bin packing/covering with delivery, solved with the evolution of algorithms. In *2010 IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA)*, pages 298–302. IEEE, 2010.

[18] A. Benkő, Gy. Dósa, and Zs. Tuza. Bin covering with a general profit function: approximability results. *Central European Journal of Operations Research*, 21(4):805–816, 2013.

[19] Gy. Dósa and Zs. Tuza. Bin packing/covering with delivery: Some variations, theoretical results and efficient offline algorithms. *arXiv preprint arXiv:1207.5672*, 2012.

[20] G. Galambos and G. J. Woeginger. On-line bin packing—a restricted survey. *Zeitschrift für Operations Research*, 42(1):25–45, 1995.

[21] J. Csirik and G. J. Woeginger. On-line packing and covering problems. *Online Algorithms*, pages 147–177, 1998.

[22] J. Herrmann, J. Proth, and N. Sauer. Heuristics for unrelated machine scheduling with precedence constraints. *European Journal of Operational Research*, 102(3):528–537, 1997.

[23] C. Liu and S. Yang. A heuristic serial schedule algorithm for unrelated parallel machine scheduling with precedence constraints. *J. Softw.*, 6:1146–1153, 2011.

[24] I. Borgulya. A hybrid evolutionary algorithm for the offline bin packing problem. *Central European Journal of Operations Research*, 29(2):425–445, 2021.

[25] B. M. Kayhan and G. Yildiz. Reinforcement learning applications to machine scheduling problems: a comprehensive literature review. *Journal of Intelligent Manufacturing*, pages 1–25, 2021.

[26] T. Wagner. *Petri Net-based Combination and Integration of Agents and Workflows.* PhD thesis, Staats-und Universitätsbibliothek Hamburg Carl von Ossietzky, 2017.

[27] A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische annalen*, 261:515–534, 1982.