Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 72 (2018) 1283–1287

www.elsevier.com/locate/procedia

51st CIRP Conference on Manufacturing Systems

Industrial scheduling with Monte Carlo tree search and machine learning

Marco Lubosch[a*], Martin Kunath[a], Herwig Winkler[a]

[a]*Chair of Production and Operations Management, Brandenburg University of Technology Cottbus-Senftenberg, Siemens-Halske-Ring 6, 03044 Cottbus, Germany*

* Corresponding author. Tel.: +49 355 69 4082; fax: +49 355 69 4091. *E-mail address:* Marco.Lubosch@b-tu.de

Abstract

Although scheduling has been studied for decades, no general approach for solving industrial scheduling problems has yet been found. The variety of production systems often requires unique modifications of scheduling algorithms to meet industrial requirements. To address this challenge, an algorithm is proposed that combines Monte Carlo tree search and machine learning. It tries to eliminate the need for problem-specific knowledge while improving the production system's performance. The algorithm has been investigated using a reentrant flow-shop problem. The results show that a combination of random-based search algorithms and machine learning is a promising way to handle complex industrial scheduling problems.

Keywords: Scheduling; Monte Carlo tree search; machine learning; flow-shop

1. Introduction

Scheduling is a decision-making process that deals with the allocation of limited resources to actions [1]. Typical resources for production applications are machines, operators and transportation systems. Typical actions are loading, processing, unloading and transporting products. The aim of scheduling is to fulfil one or more objectives [2]. Frequently used objectives are minimizing the overall makespan and maximizing machine utilization [3, 4]. Scheduling has a direct influence on the performance of production systems. Although scheduling problems have been studied for decades, no general approach for solving industrial scheduling problems has yet been found. The variety of algorithms demonstrates the great effort put into customizing algorithms to solve special cases of scheduling problems [3].

Nomenclature

GBDT	Gradient Boosted Decision Tree
MCTS	Monte Carlo tree search
UCT	Upper Confidence bounds applied to Trees

To address this challenge, a general scheduling algorithm is proposed in this paper. The algorithm tries to eliminate the need for problem-specific knowledge for a given production system. This flexible scheduling algorithm can easily adapt to different types of problems and situations while still finding near-optimal schedules. The algorithm is based on a combination of Monte Carlo tree search (MCTS) and a machine learning algorithm. Due to the complexity of scheduling problems, full decision trees cannot be created efficiently to find the best decisions. Monte Carlo methods allow an evaluation based on random samples, but classical Monte Carlo Simulation lacks the ability to prefer obviously good moves when searching. To address this issue, Coulom (2006) proposed the MCTS, which combines random sampling with tree search [5]. Especially in domains where look-ahead search is required, such as strategic games, MCTS can improve the decision-making process and can lead to superior performance [6]. Similar to strategic games, production scheduling also requires a look-ahead search due to delayed rewards. This paper analyses the possible application of MCTS combined with a machine learning algorithm for industrial scheduling problems. The algorithm is validated, using the Intel Five-Machine Six Step Mini-Fab, which represents a reentrant flow-shop problem.

After a review of related literature, the Intel Five-Machine Six Step Mini-Fab validation example is described. A brief introduction to MCTS and different selection functions is given. An approach to balance the exploration and exploitation ratio during the search process based on the Upper Confidence bounds applied to Trees (UCT) and Gradient Boosted Decision Trees (GBDT) is proposed. The results show that the combination of MCTS and machine learning is a promising way to solve scheduling problems in an industrial environment. It can be used to improve the performance of production systems without problem-specific and process-specific knowledge.

2. Related Work

MCTS was originally proposed by Coulom (2006) as an algorithm to play the two-person zero-sum game with perfect information called Go [5]. However, experienced human players could easily outplay traditional Go-playing algorithms. The main difficulty in developing algorithms for Go is the large state-space complexity of $\sim 10^{360}$ possible game states compared to less complex games such as chess with $\sim 10^{123}$ game states [7]. This large state-space makes complete calculations of all possibilities infeasible and Go has been declared the most challenging game for computers [8].

Silver et al. (2016) introduced an algorithm for the game Go, which later defeated the 18-times world champion Lee Sedol on a full-scale board [6]. The proposed algorithm was based on MCTS and was combined with different machine learning techniques to achieve superior performance. In a first step, it was trained to predict human expert moves with supervised learning and deep learning from 30 million positions of online games. In the second step, reinforcement learning was used to optimize the policy that defines which actions are taken by playing against different versions of itself. In the final step, reinforcement learning was used to create a value function that could measure the potential outcome from every state of the game. In summary, Silver et al. (2016) combined policy and value functions with MCTS to search in a look-ahead manner [6].

Chaslot et al. (2006) were the first to introduce MCTS for solving production related problems. They compared MCTS with Fixed Planning Heuristics and Evolutionary Planning Heuristics to solve production management problems that include the following four elements: a fixed set of products, a fixed set of production actions, constraints and target products. The objective was to maximize the output of the target products under the given constraints. Products can be converted into other products by production actions, which may cost time and money. The optimal solution is found when the optimal sequence of actions is determined. Money and time can be seen as resources that need to be assigned to certain actions. Therefore, the described production management problem can be considered a scheduling problem. For problems with a higher complexity, Chaslot et al. have observed better results with MCTS [9].

Runarsson et al. (2012) investigated MCTS and the Pilot method for a job-shop scheduling problem. They compared both methods for a set of 300 scheduling problems of different sizes. The objective was to find an optimal schedule which minimizes the overall makespan of a production program with different products. The Pilot method used a one-step look-ahead approach to evaluate alternative decisions and preferred the best decision based on this evaluation. The MCTS used the ε-greedy policy for balancing exploration and exploitation. The different problem sizes led to different complexities. For small and medium complexity, the MCTS outperformed the Pilot method. For scheduling problems with a higher complexity (14 jobs, 14 machines) better results were achieved with the Pilot method. Runarsson et al. explain this with the method that compared both approaches and suggest some possible improvements for MCTS and its use in scheduling [10].

Loth et al. (2013) combined MCTS with Constraint Programming to solve a job-shop scheduling problem. The use of Constraint Programming makes it possible to prevent an expansion of the decision tree if it only suggests solutions that are worse than the best known so far. Loth et al. examined different evaluation functions. A version of UCT performed best on their given problem [11].

Furuoka and Matsumoto (2017) used an algorithm based on MCTS to find good schedules for a reentrant scheduling problem. They combined MCTS with different heuristics based on experts' knowledge [12].

3. Description of the proposed approach

MCTS is a best-first search algorithm that is guided by the results of Monte Carlo Simulations [5]. MCTS was introduced by Coulom (2006) to find good decisions in the game Go, but can also be used for different Markov Decision Processes such as production scheduling [9, 11]. With MCTS, the space of possible decisions can be searched efficiently. However, it cannot be guaranteed to find the optimal solution if the search process is aborted before the full decision tree is created. In games like Go or complex scheduling problems, the full decision tree cannot normally be created due to the large number of possible states. MCTS can be used for problems with or without opponents. A major benefit of MCTS is the minimal need for problem-specific knowledge. For MCTS, a simple model that defines possible actions for a particular state and performs state transitions after these actions are executed is sufficient. In scheduling, this allows to solve different problems with the same algorithm if a model of the problem can be created. Especially for company specific objectives, the use of MCTS can be beneficial, even if no suitable planning algorithm is known prior to optimization.

The MCTS process can be subdivided into four steps which are repeated iteratively [13]. Each node in the MCTS decision tree represents a state of the problem [5]. In scheduling, these states can store information about actions and resources. Connections between states symbolize actions that the planning agent can execute. Taking actions usually leads to a different successor state. The decision process always starts at the root node. At the beginning of an MCTS process, the decision tree contains only the root node and is expanded step-by-step towards the full decision tree. The iterative process is typically interrupted after a given number of iterations or after a given period of time [13].

In the first step of an MCTS process, an appropriate leaf node is selected. A leaf node is a node that represents either a final state or a state whose successor states have not yet been added to the decision tree. The selection process always starts at the root node. If no successor states are known, this node is selected. Otherwise, one of the successor states is repeatedly selected until a leaf node is found. The selection of a suitable successor state is made by the selection function, which can estimates a potential for each state. A selection function has to fulfil two objectives. First, it must favour actions that have led to good performance in previous simulations. Second, it must favour actions that have not been tested often. This is known as the exploration-exploitation dilemma [14]. Kocsis and Szepesvári (2006) introduced an algorithm called Upper Confidence Bounds applied to trees (UCT), which tries to balance the two objectives. Formula (1) shows the UCT for a state s_i, where v_i contains the sum of the results of all sampled observations for this state, and n_i is the number of occurrences of this state. The first part estimates the average outcome if the state s_i is selected. The second part grows if n_i is small compared to N. N represents the number of decisions sampled from the predecessor state. If the adjacent states have been tested more often, the second part of the formula will grow and the algorithm will favour an exploration of this state. Constant c is used to balance the exploration and exploitation component and is usually determined empirically [14]. By using negative values, e.g. $-v_i$, the UCT can also be used for minimization problems.

$$UCT(s_i) = \begin{cases} \frac{v_i}{n_i} + c\sqrt{\frac{\ln N}{n_i}}, & if\ n_i > 0 \\ +\infty, & otherwise \end{cases} \tag{1}$$

After calculating the potential of all possible actions, the action with the highest potential is selected. If several states with the highest potential exist, a random selection is made. Another frequently used selection function is the ε-greedy policy, where the best known action is performed with the probability $1 - \varepsilon$ and a random action with probability ε [15]. In the second step of the MCTS process, it is checked whether the selected node can be further expanded. If the selected node is not a terminal node, all successor states are added to the decision tree and a random state is selected from them. Based on the selected state, a Monte Carlo Simulation is carried out in the third step, in which decisions are made randomly until a terminal node is reached. In the last step, the result of the random sample is propagated back to all predecessor states until the root node is reached. During back-propagation, the estimated values of all involved states are updated. After completion of the four-step MCTS process, the next iteration can be started with step 1. The MCTS process is often stopped after a set of specified iterations or after reaching a certain time limit [13].

In order to balance exploration and exploitation in both selection functions, the parameter c or ε must be set appropriately. Usually, different values are manually checked and adjusted. This is a major disadvantage for the integration of the algorithm into a fully automated scheduling system and should be eliminated. The proposed algorithm uses a supervised learning algorithm called Gradient Boosted Decision Trees (GBDT) [16]. It creates an ensemble of simple decision trees to model the relationship between the adjustable parameter and the results of the MCTS scheduling algorithm. First, MCTS is tested with different parameter values in a reasonable interval, e.g. $0 < c < 100$ or $0 < \varepsilon < 1$. Second, with GBDT a model is built that describes the relationship between the parameter value and the overall performance of the MCTS algorithm for the given scheduling problem. Third, for different parameter values the performance of the MCTS algorithm is estimated by the GBDT model and a best parameter value is selected. This approach helps select a parameter value that maximizes the performance of the scheduling algorithm and eliminates the need to manually adjust parameter values.

4. Experimental setup and results

In order to validate the presented approach, a practical example of industrial scheduling should be used. The example should be described in detail to ensure comparability of different implementations and should include typical challenges of industrial scheduling. Kempf (1994) proposed a reentrant flow-shop problem called Intel Five-Machine Six Step Mini-Fab, which meets these requirements [17]. Fig. 1 visualizes the Mini-Fab production. The Mini-Fab represents

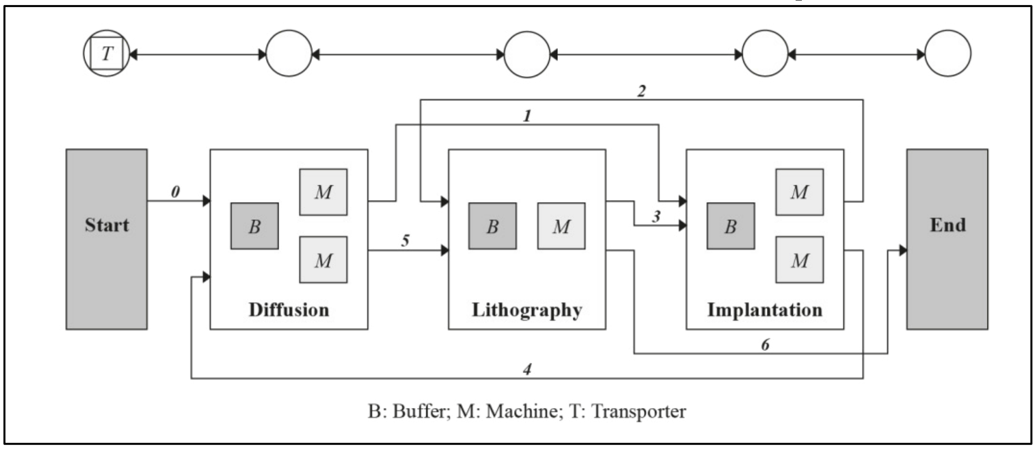

Fig. 1. Product flow through the Mini-Fab production.

the production of microchip wafers and has been used by various authors [18, 19, 20, 21]. The Mini-Fab production consists of six process steps that are carried out on three different machine types. It contains the process steps diffusion, lithography, and implantation. Each process step is executed twice in a different sequence. Each machining area has a separate buffer from which two operators load products into the machines. At the beginning, all products are stored in the start buffer. They can be transported to different buffers from a transporter that has to follow a specific route. Finished products are stored in the end buffer. The objective is to produce as many products as possible within a week. The original Mini-Fab example also includes test products as well as technicians who need to perform preventive and emergency maintenance. Test products and technicians are not implemented in the simulation model used for this analysis. In addition, the objective was changed to minimizing the overall makespan for a fixed output.

Several traditional algorithms have been implemented to validate different MCTS approaches. Every algorithm was tested 100 times with a runtime of 60 seconds on an Intel Xeon E5440 processor with 2,83GHz and single threading. Table 1 gives an overview of the studied scheduling algorithms.

Table 1. Analysed algorithms for scheduling.

Algorithm	Abbr.
Shortest imminent operation time	SI
Uniform Distribution as selection function over all possible actions	UNIF
Monte Carlo Simulation	MCS
MCTS with ε-greedy selection function and GBDT parameter tuning	EPSI
MCTS with negative UCT selection function and GBDT parameter tuning	UCTN
MTCS with negative UCT selection function that only stores the best possible result and GBDT parameter tuning	UCTB

Fig. 2 illustrates the results and their distributions. The SI rule is known to be good for problems where lead times are to be minimized, similar to the validation example [22]. As expected, the SI rule always creates the same schedule, resulting in a constant makespan. The MCS finds slightly worse schedules on average, but has a higher variance like other random-based algorithms. EPSI and UNIF perform slightly better than the SI rule. Surprisingly, the UCTN performs worse compared to MCS. The reason for this is that more simulation runs can be performed with MCS, since no states and intermediate results have to be stored. With a limited number of iterations, UCTN would have performed better than MCS. The best performance could be achieved with the UCTB, which uses the best known outcome of a state as an estimate value instead of the average of known outcomes. This is mainly due to the high degree of determinism in the validation example. Interestingly, UNIF, which used a uniform distribution as a selection rule, performed almost as well as UCTB, but had a higher variance.

The results show that simple dispatching rules such as SI can still be effective if the available planning time is short. In contrast, random-based algorithms such as MCTS are able to find better schedules when the available planning time increases. With increasing planning time, random-based scheduling algorithms find schedules closer to the optimal solution. Although MCTS is a more sophisticated approach than simple MCS, it cannot be concluded that MCTS always finds better solutions for scheduling problems. For the specific case, it has been shown that MCTS with a UCTB selection function performs best. The results also show that different selection functions have a significant influence on the performance of MCTS-based scheduling algorithms.

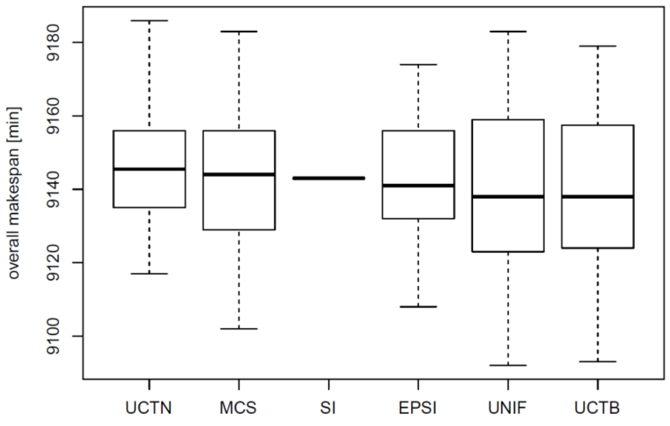

Fig. 2. Results of the analysed algorithms.

5. Conclusions

In this paper a scheduling algorithm based on the combination of MCTS and machine learning is proposed. The algorithm does not use any problem-specific knowledge other than a simulation model and thus reduces the effort of problem-specific customisations. The approach was validated using a simplified version of the Intel Five-Machine Six Step Mini-Fab proposed by Kempf (1994) [17].

For the specific scheduling problem, it could be shown that simple dispatching rules can surpass more sophisticated approaches, if the available planning time is short. If the available planning time increases, random-based scheduling algorithms such as MCTS can find better schedules. It has been shown that the performance of the MCTS approach differs significantly depending on the type of selection function. For the given example, a simple UCT-based selection function that uses the best-known result performed best. The combination of MCTS and machine learning leads to good results for the specific case. However, this approach should be validated for a wider range of scheduling problems in order to analyse and verify the performance of the specific selection functions.

References

[1] Pinedo, Michael. Planning and Scheduling in manufacturing and services. Second Edition. New York: Springer; 2009.

[2] Pinedo M. Scheduling: Theory, Algorithms, and Systems. Fifth Edition. New York: Springer; 2016.

[3] Panwalkar SS, Iksander W. A Survey of Scheduling Rules. Operations Research, 1977;25-61.

[4] Gutenberg E. Grundlagen der Betriebswirtschaftslehre: Die Produktion. Berlin: Springer; 1951.

[5] Coulom R. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. 5th International Conference on Computer and Games. Turin; 2006.

[6] Silver D, Huang A, Maddison CJ, Guez A, Sifre L, Van Den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M, Dieleman S, Grewe D, Nham J, Kalchbrenner N, Sutskever I, Lillicrap T, Leach M, Kavukcuoglu K, Graepel T, Hassabis D. Mastering the game of Go with deep neural networks and tree search. Nature, 2016;529(7587):484-489.

[7] Allis LV. Searching for Solutions in Games and Artificial Intelligence. Maastrich: Ponsen & Looijen; 1994.

[8] Van Den Herik HJ, Uiterwijk JWHM, Van Rijswijck J.Games solved: Now and in the future. Artificial Intelligence, 2002;134(1-2):277-311.

[9] Chaslot GMJB, Uiterwijk SDJJ, Saito JT: Monte-Carlo Tree Search in Production Management Problems. In Proceedings of the 18th BeNeLux Conference on Artificial Intelligence, 2006:91-98.

[10] Runarsson TP, Schoenauer M, Sebag M. Pilot, Rollout and Monte Carlo Tree Search Methods for Job Shop Scheduling. In LION, 2012:160-174.

[11] Loth M, Sebag M, Hamadi Y, Schoenauer M, Schulte C. Hybridizing Constraint Programming and Monte-Carlo Tree Search: Application to the Job Shop Problem. In International Conference on Learning and Intelligent Optimization, 2013:315-320.

[12] Furuoka R, Matsumoto S. Worker's knowledge evaluation with single-player Monte Carlo tree search for a practical reentrant scheduling problem. Artificial Life and Robotics, 2017;22(1):130-138.

[13] Chaslot GMJB, Winands MHM, Van den Herik HJ, Uiterwijk JWHM. Progressive Strategies for Monte-Carlo Tree Search. In Proceedings of the 10th Joint Conference on Information Sciences (JCIS 2007), 2007:655-661.

[14] Kocsis L, Szepesvári C. Bandit based Monte-Carlo Planning. In ECML, 2006;6:282-293.

[15] Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, Petersen S, Beattie C, Sadik A, Antonoglou I, King H, Kumaran D, Wierstra D, Legg S, Hassabis D. Human-level control through deep reinforcement learning. Nature, 2015;518(7540):529-533.

[16] Friedman JH. Greedy function approximation: a gradient boosting machine. Annals of statistics, 2001:1189-1232.

[17] Kempf K. Intel Five-Machine Six Step Mini-Fab Description. Intel/ASU Report, 1994, http://aar.faculty.asu.edu/research/intel/papers/fabspec.html.

[18] El Adl MK, Rodriguez AA, Tsalakis KS. Hierarchical Modeling and Control for Re-entrant Semiconductor Manufacturing Facilities. In Proceedings of the 35th IEEE Conference on Decision and Control, 1996;2:1736-1742.

[19] Tsakalis KS, Flores-Godoy JJ, Rodriguez AA. Hierarchical Modeling and Control for Re-entrant Semiconductor Fabrication Lines: A Mini-Fab Benchmark. In Proceedings ot the 6th International Conference on Emerging Technologies and Factory Automation, 1997:508-513.

[20] van den Berk JPA. Analysis of the Intel Five-Machine Six Step Mini-Fab. 2004.

[21] Heger J. Dynamische Regelselektion in der Reihenfolgeplanung: Prognose von Steuerungsparametern mit Gaußschen Prozessen. Springer Wiesbaden, 2014.

[22] Conway RW, Johnson BM, Maxwell WL. An experimental investigation of priority dispatching. In Journal of Industrial Engineering 1960;11(3):221-229.