Synthesis of scheduling heuristics by composition and recombination

Dominik Mäckel[1Γ], Jan Winkels[2Γ], and Christin Schumacher[3Γ]

[1] TU Dortmund University,
Department of Computer Science 14 – Software Engineering, Dortmund, Germany,
dominik.maeckel@tu-dortmund.de

[2] TU Dortmund University,
Department of Computer Science 14 – Software Engineering, Dortmund, Germany,
jan.winkels@tu-dortmund.de

[3] TU Dortmund University,
Department of Computer Science 4 – Modeling and Simulation, Dortmund, Germany,
christin.schumacher@tu-dortmund.de

Abstract. In many machine scheduling studies, individual algorithms for each problem have been developed to cope with the specifics of the problem. On the other hand, the same underlying fundamentals (e.g. Shortest Processing Time, Local Search) are often used in the algorithms and only slightly modified for the different problems. This paper deals with the synthesis of machine scheduling algorithms from components of a repository. Especially flow shop and job shop problems with makespan objective are considered to solve with Shortes/Longest Processing Time, NEH, Giffler & Thompson algorithms. For these components, the paper includes an exemplary implementation of an agile scheduling system that uses the Combinatory Logic Synthesizer to recombine components of scheduling algorithms to solve a given scheduling problem. Special attention is given to the composition heuristics and the process of recombination to executable programs. The advantages of this componentization are discussed and illustrated with examples. It will be shown that algorithms can be generalized to deal with scheduling problems of different machine environments and production constraints.

1 Introduction

In production, machine scheduling algorithms help to decide automatically when a certain job should be executed on which machine. Many manufacturers have not yet automated their machine scheduling. One reason is that for each machine scheduling problem with its numerous specific characteristics, suitable algorithms have to be selected, adapted, and implemented individually. Each practical scheduling problem can be categorized into a problem class, for which dedicated heuristics are applicable. If a class is a subset of another class, the heuristics of the superset class can often also be applied to the subset class.

Also, relationships and overlapping between categories can be identified which simplifies the transfer of heuristics between problem classes.

The assignment problem which a combination of heuristics or metaheuristics should be chosen for which practical production environment concerning the applicability, solution quality, and computing time represents a combinatorial challenge. The synthesis framework Combinatory Logic Synthesizer ((CL)S) [1] is suitable for the automated solution of this task. The (CL)S can construct software from a collection of individual components and it is possible to specify components semantically, which enables the (CL)S to select the appropriate components. The framework then automatically generates all possible combinations in the form of executable software.

The objective of the paper is to use the (CL)S-Framework to automatically select and combine different algorithms to solve a given scheduling problem. Therefore, we build a (CL)S repository of algorithms for different machine environments, which takes the relationships of the classes into account and automatically composes selected algorithms for instances of these problems.

This paper is structured as follows: First, we present the general classification scheme of machine scheduling problems. In the related work, we discuss algorithms for scheduling of flow shop and job shop problems and present the framework on which our implementation is based on, the (CL)S. The handling of this framework, as well as the generation and composition of algorithms, is shown in the fourth chapter with example runs. In detail, we show the potential of the tool and the resulting possibilities using the Giffler & Thompson's algorithm.

2 Classification of Machine Scheduling Problems

Machine scheduling problems can be specified by a tuple $\alpha|\beta|\gamma$ [2, pp. 288–290][3, pp. 13–21][4, pp. 1–2]. In the following, parameter values are specified which are considered in this paper.
The parameter α defines the amount and arrangement of machines [3, pp. 14–15]:

- 1: Single Maschine, one machine is available for production.
- Fm: flow shop, m machines with one machine per processing stage. All jobs follow the same route through the machines.
- Jm: job shop, m machines with one machine per stage. Each job has a prescribed route through the stages. The route may differ between the jobs.
- Om: Open Shop, m machines, where each job can visit the machines one after the other in an order that is determined by the planner.

Parameter β can contain as many entries as required and describes characteristics and limitations of the production process:

- $prmu$: Permutation, the processing sequence of jobs from the first processing stage through all machines is to be kept consistent [3, p. 17].

– *skip*: skipping stages of jobs is possible (further example, but not applicated in the paper) [5, pp. 1151–1155, 4, p. 13].

γ specifies the objective function:

– C_{max}: Makespan, interval between production start of the first scheduled job and finish time of the last job.

3 Related Work

In the following, important scheduling algorithms for these machine environments and β-constraints in combination with makespan minimization are described, as well as related work according to the (CL)S.

3.1 Maschine Scheduling Algorithms for Flow Shops and Job Shops

In the context of machine scheduling, an enormous number of papers and algorithms are available. Literature overviews for flow shops and job shops can be found in Komaki, Sheikh, and Malakooti [6], Framinan, Gupta, and Leisten [7] (permutation flow shop with makespan minimization) and Zhang et al. [8]. A comparison between commonly used algorithms for constructive flow shop scheduling can be found in Ruiz and Maroto [9]. Different dispatching rules have been studied in Arisha, Young, and El Baradie [10]. In the following, selectively a few algorithms of the overviews are analyzed that dealt with flow shops or job shops to minimize the makespan and are related to our problem classes (see Section 2).

Some of the most commonly used constructive heuristics for flow shops and job shops are Shortest Processing Time First (SPT), Longest Processing Time First (LPT), and the NEH-heuristic (flow shops) and have therefore been considered in this paper. The benefits of dispatching rules like SPT and LPT are low computational complexities and therefore fast calculations, and transparent behavior for production planners. The NEH-Heuristic, firstly published by Nawaz, Enscore, and Ham [11, pp. 92–94] for permutation flow shops and makespan minimization ($Fm|prmu|C_{max}$) produces good results in most cases. Giffler and Thompson [12] published a constructive algorithm that also applies rules like SPT and LPT to job shops.

3.2 Giffler & Thompson algorithm

Using the algorithm by Giffler & Thompson, job shop as well as flow shop problems can be solved. It schedules exactly one job on a machine in each iteration, so the algorithm returns complete schedules after $m * n$ iterations, where m is the number of machines and n the amount of jobs. The heuristic is only parameterized by the applied dispatching rule. In the algorithm, this dispatching rule decides between several competing jobs on the same machine. The implementation of the complete Giffler & Thompson algorithm is shown in Alg. 1.1.

The algorithm consists of four phases where steps 2 to 4 iterate until all jobs are scheduled [13, S. 75-76]. The calculated schedule and the completion times of the scheduled jobs and for all machines are returned.

1 Let Z_i be the completion time of machine i. Initialize $Z_i = 0$ for $i = 0, ..., m$. Select a dispatching rule.
2 Select machine i^* that first can finish a job out of the set of jobs, which are waiting to be processed next on one of the machines and are not scheduled yet.
3 From the set of all jobs waiting to be processed on this machine i^* select one job by the dispatching rule which is initalized in step 1.
4 Schedule selected job on machine i^* and update Z_{i^*}. If there are jobs left to be scheduled, return to step 2.

Algorithm 1.1: Implementation of the Giffler & Thompson algorithm

Alg. 1.1 works as follows. In each iteration (step 2-4), the machine is determined, which can first complete a job. For this purpose, each not yet fully scheduled job is iterated and the end time after scheduling on the next machine to be visited is compared. Up to this point, it is a greedy algorithm that selects a machine according to the earliest completion time on the next machine the job has to be processed on. Once the machine to be scheduled has been determined, in the second phase the job is varied to meet a prioritization on the machine. This is done by determining all jobs that are also to be scheduled next on the selected machine, including the job determined in the previous phase. If two or more jobs are waiting to be scheduled on the selected machine, the jobs get ranked according to the selected dispatching rule. After selecting a job on the determined machine, it gets scheduled and Z_i, as well as the current end time of the job, gets updated.

3.3 Combinatory Logic Synthesizer

Combinatory Logic Synthesizer, short: (CL)S, is a type-based framework for the synthesis of software from a set of components specified in a repository [1]. The framework was developed in the programming language Scala and is used in this paper. In addition to the synthesis, the framework also allows the immediate execution of the synthesis result. Due to the implementation in the Scala programming language, the synthesis results can also access existing Java and Scala libraries. The framework (CL)S was developed at the chair 14 of the faculty for Computer Science at the TU Dortmund University.

The Combinatory Logic Synthesizer ((CL)S) is particularly suitable for handling unpredictable variability, which makes it well suited for the synthesis of machine allocation algorithms in production planning. (CL)S enables the specification of components, their implementation, as well as the modeling of variability and the automatic composition of components under consideration of the

modeled variability rules [14]. All this is uniformly done within the framework. Thus, the framework provides a solid basis for mapping and specifying individual heuristics and algorithms, and is also suitable as a technological basis for the automatic composition of components [15]. The (CL)S has been used in the past for numerous applications of a similar nature. As an example, we mention the automatic configuration of factory planning projects [16], the automatic generation of BPMN processes [17], and the automated configuration of plans in construction projects [18]. The basis for the use of the framework is that within the target domain, results can be composed of specifiable components. In the (CL)S the specification is done by so-called semantic intersection types. How components can be specified and implemented, and which solutions are then generated automatically, is shown in the following chapters using an example.

4 Implementation

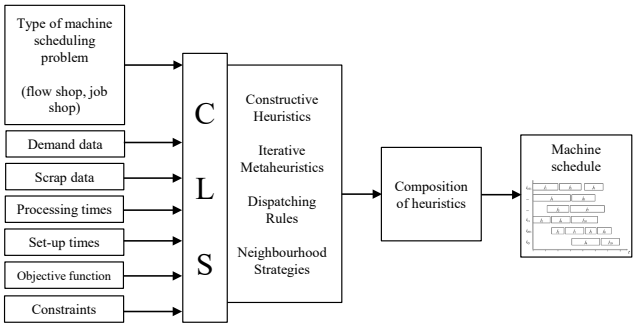

Fig. 1. Concept of schedule generation with (CL)S

The (CL)S-Repository contains all algorithm components as shown in Fig. 1, which can be combined into an executable scheduling system. Through a synthesis request to the (CL)S framework, production characteristics can be used to intersect with the defined types of the algorithm components. The (CL)S only selects those heuristics that are applicable to the given problem class. Available problem classes in this exemplary implementation are flow shop and job shop. After composing the algorithms, they can be utilized to solve the given scheduling problem and produce valid machine schedules. The synthesized algorithms work as transition functions and transfer the given data object into an applicable machine schedule. After scheduling, the makespan is calculated.

Further problem classes can be integrated by adding further possible parameter assignments and therefore extending the intersection types. By specifying additional parameters, further β constraints can be realized, which may exclude further heuristics because they are not applicable for the problem, or

$\Gamma = \{$

Scheduler: (String → String) ∩ (Algorithm ∩ shopClass → Scheduler(shopClass))
NEH: String ∩ (Algorithm ∩ FS)
FSDispatch: (String → String) ∩ (PriorityRule → Algorithm ∩ FS)
GifflerThompson: (String → String) ∩ (PriorityRule → Algorithm ∩ JS ∩ FS)
LPT: String ∩ PriorityRule
SPT: String ∩ PriorityRule

$\}$

Fig. 2. (CL)S repository

include others because they require certain assumptions or additional data such as deadlines.

Our defined (CL)S repository is shown in Fig. 2 and the solution tree calculated by the (CL)S across all combinators of the repository is illustrated in Fig. 3. The repository's first combinator *Scheduler* of Fig. 2 is a wrapping base module, which serves as the common target type for all synthesis requests. Accordingly, it is found on the first level of the solution tree (left square in Fig. 3). As parameter *shopClass* (see Fig. 2) it receives information about the problems' machine environment (α-component). Starting from the base module, the different algorithms for flow shop and job shop problems of the type *Algorithm* are now available according to the parameter *shopClass*. By concretizing the parameter when calling the synthesis, the number of applicable combinators is reduced in such a way that only the algorithms for the corresponding problem class can be used. This is done by using the parameter also as an intersection type of the base module and thus an intersection with combinators of other problem classes is no longer possible.

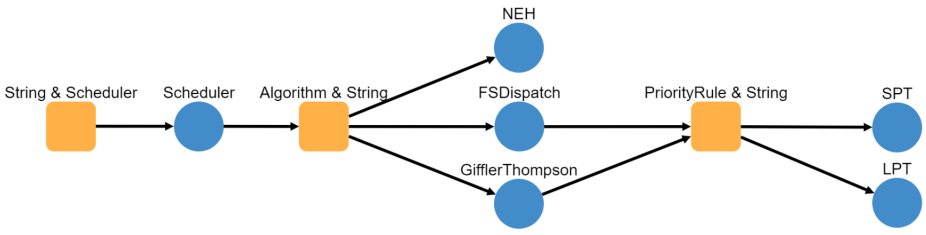

Fig. 3. (CL)S solution tree for flow shops

The first two algorithms *NEH* and *FSDispatch* in our implementation can only be applied to flow shops while the algorithm of Giffler & Thompson can be applied to job shops, which implies that it can also be used for flow shops because flow shops are a real subset of job shops as shown in Fig. 4.

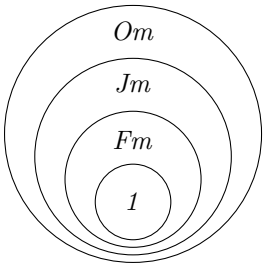

Fig. 4. Relationships between considered scheduling problem classes

The algorithms *FSDispatch* and *GifflerThompson* additionally require a dispatching rule. Fig. 3 shows the reuse of these dispatching rules SPT and LPT for *FSDispatch* and *GifflerThompson*. This shows again one advantage of such a composing method. It is easily possible to integrate and combine new algorithms, heuristics, and dispatching rules into the tool by inserting them into the repository as combinators with corresponding intersection types. New components can reuse already existing ones. Individual components can also be replaced by other possibly better performing components without having to replace them individually at all points. Furthermore, the derivation graph in Fig. 3 shows similarities and differences between algorithms in the sense that the use of similar components is immediately recognizable. The procedure of disassembling an algorithm into reusable components and representing them as (CL)S-combinators is now explained in detail using the example of the Giffler & Thompson algorithm.

5 Results

To show that the same implementation of an algorithm can be effectively used for different machine environments, the Giffler & Thompson algorithm and its implementation is shown in Alg. 1.1 has been applied to a flow shop and a job shop problem. The selection of the dispatching rule takes place inside the dispatching rule combinator that has been selected by CLS and parsed into the program code at this point. The dispatching rule is varied by replacing the code at this point.

To give a concrete example, processing times in Tab. 1 have been randomly generated from a triangular distribution with lower limit 5s, upper limit 15s, and mode 8s. For the job shop problem, also the processing order has been randomized across the stages as shown in Tab. 2. The entry "4" in row "S1" and column "job 1" indicates that job 1 has to be processed on the first stage (S1) in the fourth production step. Before, the job has to visit stage 3, then stage 2 and stage 2 in exactly this sequence. The calculated job shop schedule of the Giffler & Thompson algorithm with LPT-rule is shown in fig. 5.

Since the algorithm was not particularly designed for flow shop problems, it is reasonable to compare its result with the NEH heuristic. The two schedules

Job:	1	2	3	4	5	6	7	8	9	10
S1	6	12	8	9	10	8	9	9	11	7
S2	7	11	7	7	9	7	10	9	10	6
S3	8	12	11	8	9	8	12	13	7	9
S4	9	12	11	7	6	9	10	8	11	10

Table 1. Generated processing times

Job:	1	2	3	4	5	6	7	8	9	10
S1	4	1	4	3	4	4	1	3	2	3
S2	3	3	1	4	1	2	2	2	3	1
S3	1	2	2	1	3	3	4	4	4	4
S4	2	4	3	2	2	1	3	1	1	2

Table 2. Order for job shop production

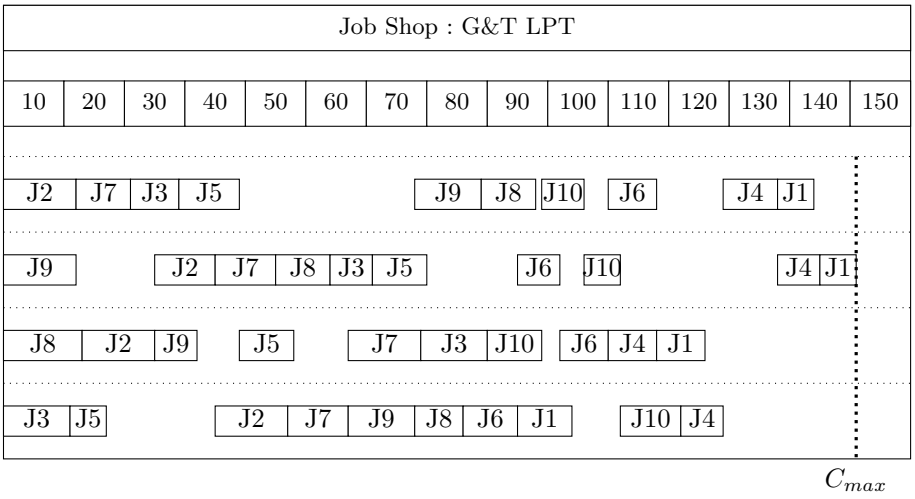

Fig. 5. Jop shop schedule with Giffler & Thompson

are shown in Fig. 6. As expected, the NEH heuristic creates a better schedule than the Giffler & Thompson algorithm. It is worth mentioning that Giffler & Thompson created a valid schedule that can keep up with algorithms specially designed for flow shop algorithms and can therefore be for example used as a starter solution for an iterative algorithm or it can be used if no better solution is available. In addition, Giffler & Thompson algorithm can be executed with different priority rules. To execution of the algorithm with different priority rules as input parameters lead to multiple solutions, the planner team can choose from. The benefit is not having to implement an algorithm for flow shop problems as the job shop algorithm can already handle it.

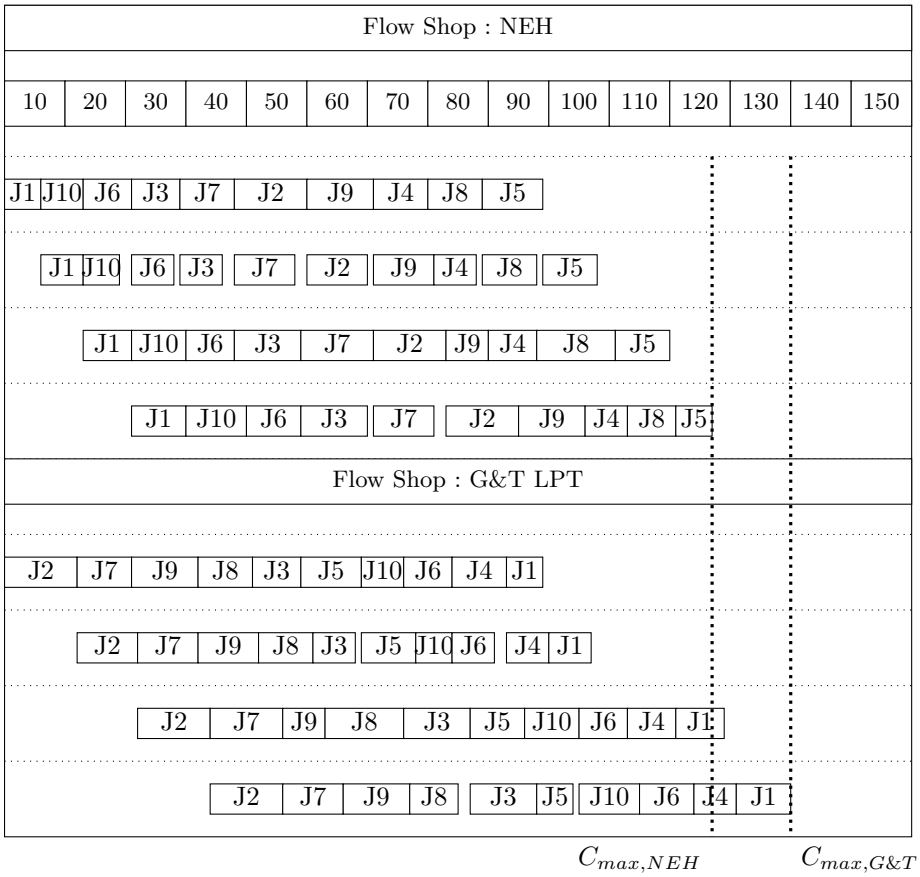

Fig. 6. Comparison of flow shop schedules with NEH and Giffler & Thompson

6 Conclusion

In this paper, we presented a repository for machine scheduling algorithms using the (CL)S, a framework that can generate algorithms automatically and to create solutions that are specially tailored to a previously specified problem. We used this framework for the problem area of machine scheduling in order to solve flow shop and job shop problems with SPT, LPT, NEH and Giffler & Thompson.

We have classified scheduling algorithms and mapped them as components in a (CL)S repository. Through componentization, different algorithms can be integrated into a framework via a uniform interface. This makes it easy to generate different algorithmen to scheduling problems. The recombined algorithms generate valid schedules according to their functionalities. Algorithms can be defined for various problem classes and constraints. According to the synthesis request, only those algorithms are recombined that apply to the current problem.

The shown concept is not limited to constructive algorithm as presented in this study and can also be applied to any iterative metaheuristic in further studies if the given data object already contains a constructive start solution. Concatenations of different constructive and iterative heuristics are conceivable as well. Also, extensions of other objective functions are possible.

References

[1] Jan Bessai et al. “Combinatory Logic Synthesizer”. In: *Leveraging applications of formal methods, verification and validation*. Ed. by Tiziana Margaria-Steffen and Bernhard Steffen. Vol. 8802. LNCS sublibrary. SL 1, Theoretical computer science and general issues. Heidelberg: Springer, 2014, pp. 26–40. ISBN: 978-3-662-45233-2. DOI: 10.1007/978-3-662-45234-9.

[2] R. L. Graham et al. “Optimization and approximation in deterministic sequencing and scheduling: a survey”. In: *Annals of discrete mathematics: Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications of the Systems Science Panel of NATO and of the Discrete Optimization Symposium co-sponsored by IBM Canada and SIAM Banff, Aha. and Vancouver*. Ed. by P. L. Hammer, E. L. Johnson, and Korte B H. Vol. 5. Elsevier, 1979, pp. 287–326. DOI: 10.1016/S0167-5060(08)70356-X.

[3] Michael Pinedo. *Scheduling: Theory, Algorithms, and Systems*. 5th ed. Cham et al.: Springer, 2016. ISBN: 9783319265780. DOI: 10.1007/978-3-319-26580-3. URL: http://dx.doi.org/10.1007/978-3-319-26580-3.

[4] Rubén Ruiz and José Antonio Vázquez-Rodríguez. “The hybrid flow shop scheduling problem”. In: *European Journal of Operational Research* 205.1 (2010), pp. 1–18. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2009.09.024.

[5] Rubén Ruiz, Funda Sivrikaya Şerifoğlu, and Thijs Urlings. “Modeling realistic hybrid flexible flowshop scheduling problems”. In: *Computers & Operations Research* 35.4 (2008), pp. 1151–1175. ISSN: 03050548. DOI: 10.1016/j.cor.2006.07.014.

[6] G. M. Komaki, Shaya Sheikh, and Behnam Malakooti. "Flow shop scheduling problems with assembly operations: a review and new trends". In: *International Journal of Production Research* 57.10 (2019), pp. 2926–2955. ISSN: 0020-7543. DOI: 10.1080/00207543.2018.1550269.

[7] J M Framinan, J N D Gupta, and R Leisten. "A review and classification of heuristics for permutation flow-shop scheduling with makespan objective". In: *Journal of the Operational Research Society* 55.12 (2004), pp. 1243–1255. DOI: 10.1057/palgrave.jors.2601784. eprint: https://doi.org/10.1057/palgrave.jors.2601784. URL: https://doi.org/10.1057/palgrave.jors.2601784.

[8] Jian Zhang et al. "Review of job shop scheduling research and its new perspectives under Industry 4.0: Journal of Intelligent Manufacturing, 30(4), 1809-1830". In: *Journal of Intelligent Manufacturing* 30.4 (2019), pp. 1809–1830. ISSN: 0956-5515. DOI: 10.1007/S10845-017-1350-2.

[9] Rubén Ruiz and Concepción Maroto. "A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime". In: *European Journal of Operational Research* 40 (Sept. 2005), pp. 479–494. DOI: 10.1016/j.ejor.2004.04.017.

[10] Amr Arisha, Paul Young, and Mohie El Baradie. "Flow Shop Scheduling Problem: a Computational Study". In: Sixth International Conference on Production Engineering and Design for Development (PEDD6). Dublin Institute of Technology. Cairo, Egypt, Jan. 1, 2002, pp. 543–557.

[11] Muhammad Nawaz, E. Emory Enscore, and Inyong Ham. "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem". In: *Omega* 11.1 (1983), pp. 91–95. ISSN: 03050483. DOI: 10.1016/0305-0483(83)90088-9. URL: http://www.sciencedirect.com/science/article/pii/0305048383900889.

[12] B. Giffler and G. L. Thompson. "Algorithms for Solving Production Scheduling Problems". In: *Operations Research* 8.4 (1960), pp. 487–503. DOI: 10.1287/opre.8.4.487. URL: http://search.ebscohost.com/login.aspx?direct=true&db=bsu&AN=7687426&site=ehost-live.

[13] Florian Jaehn and Erwin Pesch. *Ablaufplanung: Einführung in Scheduling*. 1st ed. Berlin u.a.: Springer, 2014. ISBN: 978-3-642-54439-2.

[14] Jan Bessai et al. "Combinatory Process Synthesis". In: 9952 (2016), pp. 266–281. DOI: 10.1007/978-3-319-47166-2.

[15] Jan Winkels. *Automatisierte Komposition und Konfiguration von Workflows zur Planung mittels kombinatorischer Logik*. Technische Universität Dortmund. DOI: 10.17877/DE290R-20469.

[16] Jan Winkels et al. "Automatic Composition of Rough Solution Possibilities in the Target Planning of Factory Planning Projects by Means of Combinatory Logic". In: *Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice*. Ed. by Tiziana Margaria and Bernhard Steffen. Vol. 11247. Theoretical Computer Science and General Issues. Cham: Springer International Publishing, 2018, pp. 487–503. ISBN: 9783030034283. DOI: 10.1007/978-3-030-03427-6.

[17] Jan Bessai et al. “Combinatory Process Synthesis”. In: 9952 (2016), pp. 266–281. DOI: 10.1007/978-3-319-47166-2.

[18] Lisa Theresa Lenz et al. “Smart Factory Adaptation Planning by means of BIM in Combination of Constraint Solving Techniques”. In: *Proceedings of the International Council for Research and Innovation in Building and Construction (CIB), World Building Congress 2019 – Constructing Smart Cities* (2019).