Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 180 (2021) 534–544

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Industry 4.0 and Smart Manufacturing

Real-life scheduling with rich constraints and dynamic properties – an extendable approach

Michael Bögl[a]*, Anna Gattinger[a], Ionela Knospe[a], Manuel Schlenkrich[a], Roman Stainko[a]

[a]*RISC Software GmbH, Softwarepark 35, 4232 Hagenberg, Austria*

Abstract

The industries' demand for appropriate solution approaches to solve complex, dynamic scheduling problems with a large amount of jobs (and operations) is steadily increasing. To meet these expectations, we developed a flexible and extendable optimization approach to deal with a variety of real-life, dynamic scheduling problems. The goal of the solution approach is to calculate feasible, good solutions for large problem instances in a short amount of time. In this paper, we describe the current state of the data model and algorithms to solve large scheduling problems with complex constraints and dynamic properties, e.g., machine capacity types, breaks on machines, setup times, etc., and give a short overview of the developed solution concept. Finally, we discuss further extensions of the given approach as well as future research directions.

Keywords: production planning, dynamic and real-life scheduling, large-scale optimization, decision support system

1. Introduction

Digitalization in industry is an ongoing process in recent years and affects all industrial sectors. It oftentimes triggers a review of the companies' planning problems and reveals the need for tools to solve them. Therefore, the demand for specialized tools increases. In this paper we focus on planning problems in the area of scheduling. This type of problem arises in many companies in different industries, e.g., in manufacturing companies, companies that must schedule their workforce, etc. Companies often deal with various constraints and objectives. Those real-life

* Corresponding author. Tel.: +43-7236-3343-284
E-mail address: michael.boegl@risc-software.at

planning problems may differ significantly in practice. Additionally, a planning approach, to be of any help for practitioners, must solve the very problem at hand instead of just a variant of it. Therefore, an integrated, uniform, adaptable model to solve these planning problems is of great interest. According to the no free lunch theorem (Wolpert and Macready 1997), there is no generic algorithm that performs well on all classes of problems. Moreover, and more importantly in terms of practical applicability, the cost of abstraction should be low, e.g., if the customer problem does not require handling setup times, then this data model aspect should not impact the performance of the search, i.e., the computational effort.

This paper presents a real-life scheduling problem with rich constraints and dynamic properties and proposes an extendable and configurable solution concept that can easily be adapted to various scheduling problem settings. It is not the goal to show that the developed solution concept can outperform approaches specifically tailored to solve certain scheduling problems. Also, it is not the goal to calculate provably optimal solutions. The developed solution concept has proven to be adaptable to different problem settings and it can solve real-life instances up to 20,000 and more operations.

The paper is structured as follows: In Section **Errore. L'origine riferimento non è stata trovata.** we introduce the problem and give an overview of the current data model. Section **Errore. L'origine riferimento non è stata trovata.** gives an overview of related literature. Our proposed approach is described in Section **Errore. L'origine riferimento non è stata trovata.**. Finally, in Section **Errore. L'origine riferimento non è stata trovata.** we summarize the key findings and give an outlook on further developments and research directions.

2. Problem statement / data model

Generally, in this paper we deal with the problem of scheduling a set J of n jobs ($J = \{j_1, j_2, \ldots, j_n\}$) (and their operations) on a set M of k machines ($M = \{m_1, m_2, \ldots, m_k\}$) so that a set of given constraints is satisfied, and a given objective function is optimized. This generic formulation is explained in more detail in the following: First, we describe the different types of machines (or resources) that are available. Then, an overview of the data model of the jobs and their respective operations is given.

2.1. Machines

Currently, four different types of machines are available. They differ in two aspects: (a) utilization and (b) calculation of the duration of an operation. These machine types can be used to model the actual machines/resources. However, in case none of the available machine types can appropriately model the real-world, a new type can be implemented easily. Generally, the available machine types have two different possibilities to handle an unavailability:

1. Ongoing operations are stretched over any unavailability (referred to as *breaks*)
2. Ongoing operations are not allowed during any unavailability

The difference between these situations is shown in

Figure 1 top: ongoing operations are stretched over an unavailability (case 1); bottom: ongoing operations are not allowed during an unavailability (case 2).

. In both cases, the given machine is not available during the time interval [10, 20). An operation o_1 with a duration of 40 must be planned. The start of the planning horizon, i.e., the earliest possible starting time, is 0. At the top, case 1 is depicted. The operation can start at time 0 as it is stretched over the break; the gross duration is 50. At the bottom, the operation is not allowed to start before the unavailability, because it cannot be finished within the interval [0, 10). Therefore, the earliest start time is 20 and it is finished at 60. Operations must not start during a break and cannot end during a break, as the break is non-productive time.

2.1.1. Single operation machine

For machines of this type, at most one operation can allocate the machine at any given time. It is equivalent to the machines in the job-shop scheduling problem (JSP). An example is given in **Errore. L'origine riferimento non è**

stata trovata., top. Two operations, o_1 and o_2, with a duration of 40 are assigned to the machine. As this capacity type only allows a single operation at any time, the operations are scheduled sequentially. Generally, this machine is a special case of the varying capacity, fixed duration machine (see section 2.1.2). This type is implemented separately for performance reasons.

2.1.2. Varying capacity, fixed duration machine

Machines of this type have a varying available capacity, e.g., in the interval [0, 10) the available capacity is 2, in the interval [10, 30) the capacity is 1, and in the interval [30, ∞) the capacity is 2. Every operation has a defined, fixed capacity demand for every machine it can be assigned to (it may be different for each machine), see Section **Errore. L'origine riferimento non è stata trovata.**. An operation can only be assigned to a machine if enough remaining capacity is available at the machine throughout the entire duration of the activity.

An example is given in **Errore. L'origine riferimento non è stata trovata.**, middle. Two operations, o_1 and o_2, with a duration of 40 and a capacity demand of 1, are assigned to the machine. First, o_1 is scheduled, and it is assigned to the interval [0, 40). Then o_2 is scheduled. It cannot start at time 0, because the capacity of the machine drops to 1 at time 10 and the available capacity is too low. Consequently, operation o_2 can start at 30 at the earliest, because sufficient machine capacity is available then. An example for this type of machine is machinery and their operators, where the number of operators may be reduced during night or weekend shifts.

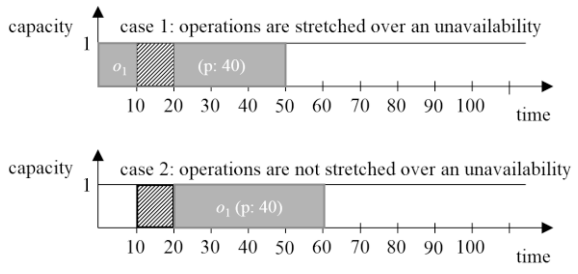

capacity
case 1: operations are stretched over an unavailability
1
o1
(p: 40)
10 20 30 40 50 60 70 80 90 100
time
capacity
case 2: operations are not stretched over an unavailability
1
o1 (p: 40)
10 20 30 40 50 60 70 80 90 100
time

Figure 1 top: ongoing operations are stretched over an unavailability (case 1); bottom: ongoing operations are not allowed during an unavailability (case 2).

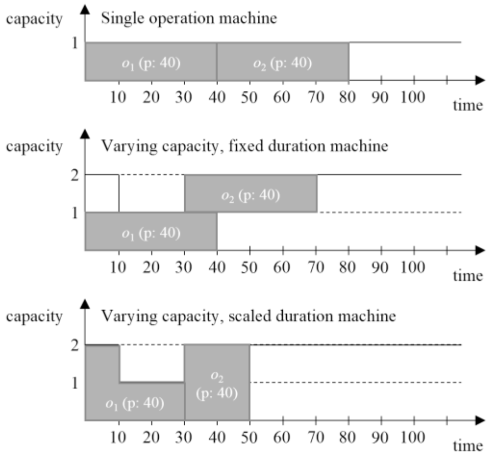


Figure 2 different machine capacity types; top: a single operation can be assigned to the machine; middle: operations can be assigned to machines simultaneously as long as enough capacity is available; bottom: a single operation can be assigned to the machine and the execution time scales with the capacity of the machine.

2.1.3. Varying capacity, scaled duration machine

The third capacity type is a scaled machine type. Again, the capacity of the machine may change over time. In this case, only a single operation can be assigned to a machine, but the duration of the operation scales with the available capacity of the machine. Here, it is assumed that the capacity demand of all operations and machine capacities are normalized to 1. Typically, workforce is modeled using this type of machine.

An example is given in **Errore. L'origine riferimento non è stata trovata.**, bottom. Two operations, o_1 and o_2, with a duration of 40 and a capacity demand of 1 are scheduled on the machine. First, o_1 is scheduled. The machine has a capacity of 2 in the interval [0, 10), therefore a fraction $(2 \cdot 10 / 40)$ of the overall duration is scheduled in the first interval. The remaining duration of operation o_1 is 20, which is scheduled in the interval [10, 30), as the capacity of the machine is 1. Operation o_2 is then scheduled in the interval [30, 50) as the available capacity is 2 in the interval $[30, \infty)$.

2.1.4. Unlimited capacity

Additionally, a machine type with unlimited capacity is available. An arbitrary number of operations can be scheduled at the same time on this machine. This type of machine allows for the representation of external manufacturers or production steps.

2.2. Jobs and Operations

Jobs and their respective operations are scheduled on the machines. Besides the machines, the operations are a central element in the data model. Operations are interconnected by precedence constraints and must not be preempted by other operations. But they can be interrupted in case of breaks (see Section 2.1). Operations have the following data associated:

1. earliest starting time and latest finish time
2. a set of compatible machines
3. machine-dependent capacity demand
4. machine-dependent production duration
5. machine-dependent production costs
6. teardown duration
7. machine- and sequence-dependent setup times
8. machine-dependent setup costs
9. a set of predecessors
10. a set of successors
11. relation type of predecessor/successor (overlaps between operations)
12. waiting time
13. transportation time
14. requirement of (multiple) additional machines
15. current start of execution and end of execution time

An operation must not be scheduled before the earliest start time (i.e., release date). The latest finish time is used in the objective function to determine lateness (1). For every operation, a set of allowed machines is given (2) and for each of these machines, a capacity demand (3), a production duration (4), production costs (5), and a teardown duration (6) are given. Furthermore, sequence-dependent setup times (7) and machine-dependent setup costs (8) are given for every operation and machine.

An operation may have (multiple) predecessor and successor relations to other operations. These relations are not limited to operations within a job (9 and 10) but must be acyclic over all operations.

If two operations have a direct precedence relation, then it is possible to define an overlap (11). Normally, an operation can start at the earliest after all its predecessors are finished (adding waiting times and transportation times to the end time of the preceding operation). Special relations can be defined, e.g., in a time-based relation, the successor can start at a certain time after its predecessor has started. Additionally, it is possible to define an amount-based relation, i.e., the successor can start as soon as the given amount has been produced by the predecessor. An example is shown in **Errore. L'origine riferimento non è stata trovata.**, where o_1 precedes o_2 and o_2 can start as soon as m_1 produced 30 pieces for o_1. In this example, production speed is 1 piece per time unit on both machines.

Operations can have a defined waiting time (12), which is considered after the operation is finished on the machine. The machine is not occupied during this time. When operations transfer from one machine to another, transportation times can arise (13). Every machine must belong to a location and a distance matrix of transfer times between all locations must be given. An operation can have a demand for several additional machines/resources during the execution (setup, processing, and teardown) of this operation (14). For example, an operation on a drilling machine is executed on the drilling machine and needs an operator. This is modeled by introducing an operation for the drilling machine and a simultaneous operation for the operator in addition. In case a plan already exists, current start and end times (for setup, production and teardown) are given for operations (15). This information is used for rolling horizon planning or in case operations are fixed on a machine, e.g., their execution has already started.

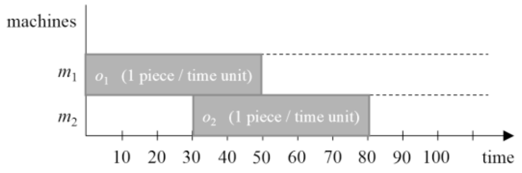
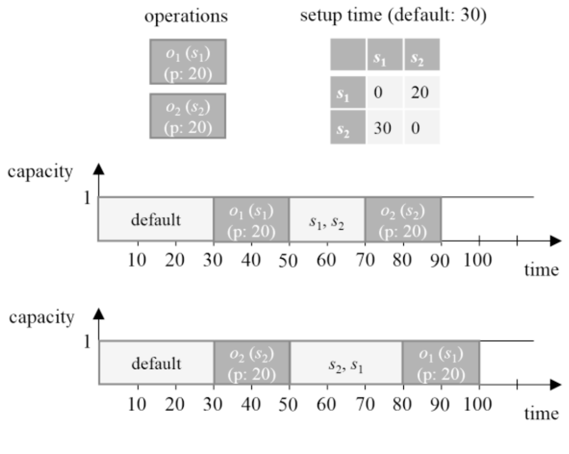


Figure 3 Amount based overlap: o_1 and o_2 produce 50 pieces with a production speed of 1 piece per time unit and o_1 is the only predecessor of o_2; in this example, after o_1 produced 30 pieces on m_1, the operation o_2 can start on m_2 at time 30.

Errore. L'origine riferimento non è stata trovata. shows an example of sequence-dependent setup times. Two operations o_1 and o_2 are scheduled on the same machine, both operations have a production duration of 20. Operation o_1 has setup attribute s_1 and o_2 hast setup attribute s_2. Setup information for this machine is given in the table at the top right-hand corner. The default setup duration is 30, which is used in case no known predecessor is on the machine, or the predecessor has no setup attribute. The figure depicts two scenarios: at the top, operation o_1 is scheduled before o_2, and on the bottom, operations are scheduled in reverse order. As we see, due to the given data, scheduling o_1 before o_2 has a shorter overall setup time.



	s_1	s_2
s_1	0	20
s_2	30	0

Figure 4 Example of sequence-dependent setup times for two operations. Light grey rectangles are setup durations. Dark grey areas are production durations.

2.3. Objectives

Since the data model comprises a range of diverse aspects, it allows for a large set of objective functions. Each company might pursue different objectives. The actual objective, however, may depend on the given planning situation. Possible objectives are:

1. Minimize the makespan
2. Minimize the total weighted (order) tardiness
3. Minimize the total setup time / cost
4. Minimize the production time / cost
5. Minimize the order makespan

Cost-based objectives are basically calculated by multiplying the duration with a machine-dependent cost factor. Two additional objective functions are available, which are used to minimize the deviation from a given plan. They penalize operations which are assigned to a different machine or start at different times when compared to the input data. Those objectives are used in case of rolling horizon planning or disruption management.

2.4. User-specific functions and decision logic

Considering all these data model aspects, a lot of questions on the details of handling certain conditions arise. For example, in case an operation requires multiple machines, how are breaks handled? Or in case of amount-based overlaps: What happens if the successor operation would end before the preceding operation? Is the start of the successor operation delayed, or is it artificially stretched to exceed the end of the preceding operation?

There is no single correct answer to these questions because it is customer-specific and therefore a matter of definition. For rich data models, the number of such situations is huge.

Our approach to cope with this problem is to provide an extendable problem-specific approach. From the perspective of implementation, specifics of the customer can be modeled using defined interfaces and callbacks. Therefore, the customer-specific logic on how to handle certain aspects of the planning problem is implemented in customer-specific functions.

2.5. Summary

Due to properties of the machines, i.e., changing available capacity over time, scaling duration as a function of the available capacity, breaks on the machine, amount-based overlaps, etc., the problem has dynamic properties. The calculation of the exact start and end times of the operations requires consideration of all capacity changes during the execution of the operation. However, all data is deterministic. Typical practical problem sizes are in the range of a few hundred operations up to 20,000 and even more. The planning horizon varies from a few weeks to multiple months.

3. Related literature

Two different categories of related literature were identified: (a) generic, framework-based approaches to solve different optimization problems and (b) problem-specific scheduling approaches. Solutions that provide generic (meta-)heuristic optimization frameworks are related but are intended for different purposes, e.g., a survey is given in [14]. The goal of such frameworks is mostly to provide different algorithms for problems and not to provide extendable problem models.

In the literature, only a few generic approaches exist which provide a framework to model specific scheduling problems. An early approach is described in [11], where a specific problem is described by a schedule description language. However, as stated in the paper, it is not possible to “tailor the internal representation of the problem to suit a specific problem.” This is a major goal of our approach.

In [16] a generic library of solving methods for scheduling applications described. They designed an ontology to describe tasks (operations) and their constraints using a knowledge modeling framework. However, dynamic data is not included in the ontology. It is not clear from their description if such aspects could be flexibly implemented in their framework.

From an algorithmic point of view, relevant literature for this type of problem can be found in the fields of job shop scheduling (JSP) and resource-constrained project scheduling (RCPSP) and their respective extensions and variants. First, we review relevant papers of the JSP domain, followed by a short review of relevant papers in the area of RCPSP. Exact approaches, which can provably calculate the optimal solution for the given constraints and objective, were developed for scheduling problems ([16] and [17]). Due to the intractability of the non-trivial instances of scheduling problems [1], we focus on heuristic and metaheuristic approaches to solve this type of problem. Ivens and Lambrecht use a variant of the shifting bottleneck procedure to solve real-life problems [7]. They use a longest tail heuristic to solve the machine sequencing subproblem. Braune, Wagner, and Affenzeller point out the importance of a very good and fast re-optimization method for the shifting bottleneck procedure and state that partial enumeration and even metaheuristics are too time-consuming [2]. Gonçalves, Mendes, and Resende propose a hybrid genetic algorithm to solve the JSP [5]. They translate the chromosomes into an active schedule by using a schedule generator procedure, which generates a parameterized active schedule, and finally apply a local search procedure.

The tabu search proposed in [4] was very successful and able to achieve very good solutions. It uses a special initial solution construction heuristic, a variable length tabu list, which is adapted according to the current search progress, and neighborhoods to exchange operations on critical arcs. The neighborhood operators generate only feasible solutions. Kuhpfahl and Bierwirth propose additional neighborhoods based on critical arcs, which perform larger changes in the solution but still yield feasible solutions [9].

In the RCPSP, activities must be scheduled such that an objective function is optimized. According to Brucker, Drexl, Möhring, Neumann, and Pesch machine scheduling is a special case of RCPSP. They propose a classification scheme and review methods to solve these types of problems [3]. Hartman and Briskorn discuss different extensions to the basic model in [6], i.e., setup times, multiple modes, multiple projects, etc.

Heuristic methods often use schedule generation schemes to calculate schedules from the list of activities. Metaheuristic approaches work with a list of operations and translate this list into a schedule by employing a schedule generation scheme [8].

A tabu search approach for the multi-mode RCPSP with schedule-dependent setup times is proposed in [12]. They refer to schedule-dependent setup times, because they consider transfer times between resources as setup times, too. They use three neighborhood operators: activity swap, which swaps two operations; mode change, which changes the mode, i.e., machine, for a given operation; location change, moving an operation to two neighboring locations (resources are grouped into locations which impact the setup times).

Neither approach handles scaled processing times and breaks and neither of them fully reflects the complexity and dynamic aspects of the given problem.

4. Solution approach

The presented solution approach was designed with the following considerations:

1. configurable for special problem instances
2. modular and extendable for user specific needs
3. handling of very large problems (up to tens of thousands of operations and more)

In order to meet these criteria, a modular solution approach was designed, adapting different concepts from literature.

4.1. Solution representation

Our approach does not directly work on the schedule or on the disjunctive graph representation but uses an activity list-based solution representation, as proposed in the literature (see, e.g., [8] or [13]). This activity list represents the sequence of operations and can be viewed as an encoded solution.

A schedule generation scheme translates the given sequence into a schedule. A schedule contains all start and end times and the selected machines for all operations, and an objective function can evaluate the quality. In fact, the schedule contains more information, e.g., for amount-based overlaps of operations it must be known when the minimum overlap amount is produced on the machine, which depends on the given capacity and breaks during the production time. **Errore. L'origine riferimento non è stata trovata.** illustrates the process of translating the activity list into a schedule using the schedule generation scheme and the evaluation of the solution to determine the final solution quality.

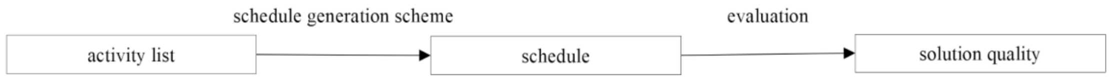


Figure 5 the generation of a schedule using a schedule generation scheme (based on the activity list) and the evaluation of the schedule.

Even though this architecture adds some overhead in terms of abstraction, it has significant advantages over directly working with the schedule. Due to the dynamics of machines and operations (e.g., capacity changes, breaks on machines, overlaps of operations, …), changes made to the schedule (e.g., by neighborhood operators) involve computationally expensive calculations because the schedule must possibly be recalculated for all successors.

Furthermore, the calculation of the schedule and the (neighborhood) operations on the schedule would be tightly coupled. To overcome this problem, schedule generation schemes are used. These schemes can be tailored to the specific problem at hand, e.g., if multiple parallel machines/resources are not required, the schedule generation scheme does not need the logic to handle them.

Currently, two different schedule generation schemes are implemented:

1. serial schedule generation scheme: operations are scheduled as sequenced in the activity list and as soon as possible with respect to precedence relations, i.e., they may be scheduled between other operations on a machine if time and capacity constraints can be satisfied.
2. strict sequence-based schedule generation scheme: operations are scheduled as sequenced in the activity list, but they may not fill gaps between other operations on machines, i.e., an operation is always scheduled after the last operation on the machine.

4.2. Initial feasible solution construction

The initial solution will affect the final solution quality. When dealing with very large instances, it is important to start with a good initial solution. Therefore, we implemented a variety of different solution construction methods. All of them work on the activity list and generate sequences of operations, which are then translated into schedules using the schedule generation scheme and are then evaluated. The best solution serves as initial solution.

The following construction algorithms are available:

1. Use the sequence of operations as currently operationally executed (used in case of rolling horizon planning and disruption management).
2. Sort operations according to increasing latest finish and earliest starting time. If the latest finish times of two operations are close, e.g., within 60 minutes, then sort them according to increasing earliest start time (release time).
3. Sort the operations according to a simplified schedule generation scheme: For every schedulable operation, the earliest start time over all machines is calculated. The machine state and capacity are considered.

4. Bidirectional sorting: generates a sequence based on the construction algorithm proposed in [4].

4.3. Tabu search

Different successful solution approaches were proposed in the literature for scheduling problems with rich constraints. We implemented a tabu search algorithm similar to the proposed scheme in [4] because it can easily be extended and adapted to different problem instances. We also use dynamic tabu list management: the tabu list has a given minimum length and a given maximum length. In case the search made an improving move the tabu list length is decreased; in case the search made a deteriorating move, the length of the tabu list is increased.

Further, if no progress is made, i.e., the incumbent solution does not improve for a given (configurable) number of iterations, a restart of the search is performed. The current incumbent solution is replaced by the best-known solution and the search continues from there. As the neighborhood operators generate new solution candidates, different solution acceptance/selection strategies are possible. Currently, the following solution selection schemes are implemented: first improvement: the first improving solution is accepted; best first improvement: the best solution of the first improving solution over all neighborhood operators is selected; best improvement: the best solution over all neighborhood operators is selected (this operator is only applicable for small neighborhoods and appropriately sized problem instances); adaptive neighborhood selection: the best solution of the currently highest-ranking neighborhood operator is chosen. The neighborhood operators are ranked with respect to their current progress potential, which is continuously updated.

Generally, the algorithm is designed to be customizable. Custom code/logic can be executed at defined steps of the algorithm, e.g., at the beginning of a new iteration, whenever a new incumbent solution is found, etc. This code can access the complete current state of the algorithm as well as the solutions and can therefore dynamically control the execution of the algorithm.

4.4. Neighborhood operators

A subset of operators proposed by Dell'Amico and Trubian [4] (N1 and N2) as well as a subset of operators proposed in Kuhpfahl and Bierwirth [9] have been implemented. Those operators change critical arcs in the schedule. However, early experiments with large instances showed that in some cases, especially in the beginning of the search, small local changes lead to minor improvements and the computational overhead is very high. Therefore, we implemented neighborhood operators that perform larger changes in the solution. Those operators are specifically targeted at minimizing tardiness of orders, e.g., moving all operations of a tardy order to earlier positions in the activity list. Another idea is to detect idle machines and move tardy operations to the respective positions in the activity list to utilize the machine during those times. Because an extendable metaheuristic search algorithm is used, the set of neighborhood operators can be extended by implementing new operators in case limitations of the current set of operators are identified.

4.5. Next steps on solving real-life instances

Currently, first tests with this approach are applied to selected real-life instances. The size of those problems ranges from about 850 operations to about 20,000 overall operations and about 50 machines to 110 machines. The high number of operations and machines is due to considering multiple production stages. The next step is to validate the solution approach on real-life and artificial benchmark instances. When solving such problem instances, experience has shown that the structure and quality of the initial solution is very important, because every iteration of the search algorithm is very expensive. In a typical use case, the algorithm is called using the solution currently executed (rolling horizon planning and disruption management). In case no initial solution is given, multiple different construction algorithms are available. Our current assumption is that the structure of real-life data differs from artificial benchmark problems. In real-life, orders are often already planned roughly according to the available production capacity (by enterprise resource planning systems) before they are scheduled. Additionally, the execution of the previous plan is already ongoing, therefore the planning algorithm does not start from scratch, but can use the previous solution and adapt it (using neighborhood operators and the appropriate objective function).

We also experience that many proposed neighborhood operators suggested in the literature perform minor changes which are too small to make progress in large instances with a bad initial solution. Therefore, we implemented operators introducing larger changes to the sequence of operations, e.g., moving all operations of a job in the activity list. Nonetheless, dealing with real-life instances is very difficult due to the problem size and the dynamic aspects that must be considered.

5. Conclusion and further research

The developed solution approach is currently tested on selected real-life scheduling instances. It proves to be a very flexible and extendable framework for customer-specific requirements. On the one hand, this is due to the nature of the used metaheuristic, which can easily be extended. On the other hand, and this is a major factor, customer-based (decision) logic for the problem-specific parts can be customized by providing decision functions for interfaces defined by the framework.

Due to the increasing number of parameters, manual tuning for specific use cases is very time-consuming and requires deep understanding of the problem instance data. Therefore, we are planning to use automatic parameter tuning, e.g., irace [10].

The solution approach can handle dynamic properties of the environment, such as machine capacity changes, breaks, production speed, … However, in case a sudden break-down of a machine happens or new jobs arise, the planning algorithm must be started with the new data and the plan is adapted to the new circumstances. This adaption must currently be initiated by humans. In future research, it may be possible to have a mediating system tracking the execution of the current plan (some type of machine execution system) and initiating the calculation of a new plan as soon as changes in the environment are registered. This way it may be possible to avoid serious disruptions in the execution by continuously adapting the current plan slightly.

References

[1] Baker K., and Su Z.-S. (1974) “Sequencing with due-dates and early start times to minimize maximum tardiness.” *Naval Research Logistics Quarterly* **21**: 171-176.

[2] Braune R., Wagner S., and Affenzeller M. (2007) „Optimization Methods for Large-Scale Production Scheduling Problems.” *11th International Conference on Computer Aided Systems Theory*, 812-819, February 12-16, 2007, Las Palmas de Gran Canaria, Spain.

[3] Brucker P., Drexl A., Möhring R., Neumann K., and Pesch E. (1999) “Resource-constrained project scheduling: Notation, classification, and methods.” *European Journal of Operational Research* **112**: 3-41.

[4] Dell’Amico M., and Trubian M. (1993). “Applying tabu search to the job-shop scheduling problem.” *Annals of Operations Research* **41**: 231-252.

[5] Gonçalves J., Mendes J., and Resende M. (2005) “A hybrid genetic algorithm for the job shop scheduling problem.” *European Journal of Operational Research* **167**: 77-95.

[6] Hartmann S., and Briskorn D. (2010) “A survey of variants and extensions of the resource-constrained project scheduling problem.” *European Journal of Operational Research* **207**: 1-14.

[7] Ivens P., and Lambrecht M. (1996) “Extending the shifting bottleneck procedure to real-life applications.” *European Journal of Operational Research* **90**: 252-268.

[8] Kolisch R., and Padman R. (2001) “An integrated survey of deterministic project scheduling.” *Omega* **29**: 249-272.

[9] Kuhpfahl J., and Bierwirth C. (2016) “A study on local search neighborhoods for the job shop scheduling problem with total weighted tardiness objective.” *Computers & Operations Research* **66**: 44-57.

[10] López-Ibáñez M., Dubois-Lacoste J., Pérez Cáceres L., Stützle, T., and Birattari, M. (2016). “The irace package: Iterated Racing for Automatic Algorithm Configuration.” *Operations Research Perspective* **3**: 43-58.

[11] Mcllhagga M: (1997) “Solving generic scheduling problems with a distributed genetic algorithm.” *Lecture Notes in Computer Science* **1305**: 199-212.

[12] Mika M., Waligóra G., and Węglarz J. (2008) “Tabu search for multi-mode resource-constrained project scheduling with schedule-dependent setup times.” *European Journal of Operational Research* **187**: 1238-1250.

[13] Moumene, K., Ferland, J.A. (2009) “Activity list representation for a generalization of the resource-constrained project scheduling problem.” *European Journal of Operational Research* **199**: 46-54.

[14] Parejo J., Ruiz-Cortés A., Lozano S., and Fernandez P. (2012) “Metaheuristic optimization frameworks: a survey and benchmarking.” *Soft Computing* **16**: 527-561.

[15] Pinedo M. (2016) “Scheduling – Theory, Algorithms, and Systems.” New York:Springer Science+Business Media.

[16] Rajpathak D., Motta E., and Zdrahal Z., Roy R. (2006) "A Generic Library of Problem Solving Methods for Scheduling Applications." *IEEE Transactions on Knowledge and Data Engineering* **18**: 815-828.

[17] Singer M., and Pinedo M. (1998) "A computational study of branch and bound techniques for minimizing the total weighted tardiness in job shops." *IIE Transactions* **30**: 109-118.

[18] Wolpert D., and Macready W. (1997) "No Free Lunch Theorems for Optimization." *IEEE Transactions on Evolutionary Computation* **1**: 67-82.