Available online at www.sciencedirect.com

ScienceDirect

IFAC PapersOnline CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 52-13 (2019) 1403–1408

A Permutation-Based Heuristic Method for the Blocking Job Shop Scheduling Problem

Julia Lange * **Frank Werner** **

* *Otto-von-Guericke-University, Magdeburg, Germany (e-mail: julia.lange@ovgu.de).*
** *Otto-von-Guericke-University, Magdeburg, Germany (e-mail: frank.werner@ovgu.de)*

Abstract: In the manufacturing of highly customized goods and the operation of automatic logistics systems, efficient schedules constitute an everyday challenge. Therefore, the job shop problem is established as a standard model in scheduling research. While classical variants are well studied, the involvement of practically relevant conditions, such as the absence of intermediate buffers and customer-oriented optimization criteria, shows a lack of theoretical understanding. This work provides a study in this research direction by examining the applicability of a scheduling-tailored heuristic search method to the blocking job shop problem with total tardiness minimization. Permutation-based encodings are used to represent a schedule. Appearing redundancy and feasibility issues are discussed. Two well-known neighborhood structures for sequencing problems are applied and an advanced repairing technique to construct feasible blocking job shop schedules is proposed. The computational results obtained by embedding the components in a simulated annealing framework highlight advantages of the heuristic solution approach against existing general-purpose methods.

Keywords: Scheduling, Heuristic Search, Permutation Algorithm, Blocking Constraints, Total Tardiness

1. INTRODUCTION

By theoretically implementing the processing of customized orders with individual sequences of manufacturing steps, the job shop problem has become one of the standard models in scheduling research. While even classical variants of the problem are proven to be NP-hard, see Garey et al. (1976), real-world planning situations demand for the integration of practically relevant constraints such as release dates of jobs and a lack of storage capacity in the system. The production of huge items and the scheduling of trains in single-track networks represent two applications which imply so-called blocking constraints in the job shop environment. Due to the absence of intermediate buffers, a job blocks the current machine posterior to its processing as long as the consecutively required machine is not idle. Furthermore, customer-oriented optimization criteria constitute an important issue in practice. Accounting for aspects like contractually agreed due dates and predefined schedules, the total tardiness is considered in this paper as a sparsely studied objective function.

The *blocking job shop scheduling problem with total tardiness minimization (BJSPT)* is known to be one of the hardest combinatorial problems, cf. Lawler et al. (1982) and Balas and Vazacopoulos (1998). Numerical experiments on the performance of exact solution techniques indicate the boundaries of solvability of the BJSPT by mixed-integer programming methods in problem size and computation time, see Lange and Werner (2018a) and Lange and Werner (2018b). Therefore, the development and the analysis of heuristic solution methods for complex job shop problems are pursued by many researchers.

Oddi et al. (2012) apply an iterative improvement scheme including a relaxation and a reconstruction phase to the blocking job shop problem with makespan minimization. Regarding the same planning situation, an iterated greedy algorithm based on an alternative graph representation and similarly using destruction and construction is proposed by Pranzo and Pacciarelli (2016). Tabu search approaches, which involve job reinsertion techniques to construct feasible neighboring solutions, are presented for this job shop setting by Mati et al. (2001) and Groeflin and Klinkert (2009). Considering the classical job shop problem without blocking constraints and evaluating tardiness-related objective functions, Mattfeld and Bierwirth (2004) propose a hybrid genetic algorithm which relies on a permutation-based encoding. Zhang and Wu (2011) apply a simulated annealing heuristic to obtain good schedules for this problem with regard to total tardiness. An extended version of the well-known interchange-based neighborhood structure introduced by Van Laarhoven et al. (1992) is used. Different widely applied neighborhoods for the standard job shop setting are compared according to their performance on tardiness-based optimization criteria by Kuhpfahl and Bierwirth (2016). Considering the BJSPT, as it is regarded here, Bürgy (2017) proposes a job reinsertion-based tabu search heuristic for instances including setup and transfer times.

10.1016/j.ifacol.2019.11.395

All these studies consistently show that the occurrence of blocking constraints causes significant feasibility issues in the construction of neighboring schedules and that the guidance of a heuristic search procedure is more difficult for tardiness-based objectives compared to the makespan. This work examines the capability of classical scheduling-tailored mechanisms in planning situations involving both real-world properties. An *advanced repair procedure (ART)* is described to determine feasible schedules from an initial solution and a given desired change. With this, a simulated annealing (SA) metaheuristic is applied using permutation-based encodings as well as interchange- and shift-based neighborhood operators. Promising computational results on two structurally different sets of benchmark instances are presented and compared to the best solutions found by a mixed-integer programming solver.

The remainder of the article is organized as follows. The BJSPT is theoretically introduced in Section 2, which additionally provides a description of the applied permutation-based representations of a schedule. Section 3 includes remarks on redundancy and feasibility together with a brief outline of the ART. In Section 4, two neighborhood structures and their embedding into SA are explained. The computational experiments and results are reported and discussed in Section 5. Section 6 concludes with the main findings and further research perspectives.

2. PROBLEM DESCRIPTION

The considered job shop scheduling problem is defined by a set of jobs $\mathcal{J} = \{J_i \mid i = 1, \ldots, n\}$, which is required to be processed on a set of machines $\mathcal{M} = \{M_k \mid k = 1, \ldots, m\}$. For every job, an individual technological route is given, whereby recirculation (recrc) is allowed. The operation $O_{i,j}$ denotes the j-th processing step of job J_i. The set $\mathcal{O}$ contains all operations. A processing time $p_{i,j} \in \mathbf{Z}_{>0}$ is predefined for each operation $O_{i,j}$ as well as a release date $r_i \in \mathbf{Z}_{>0}$ for each job J_i. As a solution to the problem, a *schedule* is to be determined by the starting times $s_{i,j}$ for all $O_{i,j} \in \mathcal{O}$. A schedule is feasible, if the processing sequences and the technological routes of all jobs are correctly implemented and the following *blocking constraint* is fulfilled for all pairs of consecutively processed operations $O_{i,j}$ and $O_{i',j'}$ requiring the same machine M_k provided that $O_{i,j+1}$ exists.

$$s_{i',j'} \geq s_{i,j+1} \quad \text{for } O_{i,j} \rightarrow O_{i',j'} \tag{1}$$

To evaluate feasible schedules according to a customer-oriented objective function, a due date d_i is given for every job $J_i \in \mathcal{J}$. A schedule is denoted as optimal, if it minimizes total tardiness $\sum T_i$, where $T_i = \max\{C_i - d_i, 0\}$ describes the tardiness value of a job J_i with regard to its completion time C_i. Since this objective function is non-decreasing in the total completion time of all jobs, an optimal solution can always be determined by the earliest possible starting times of all operations. Hence, the sequences of operations on the machines can equivalently be used as a unique representation of a schedule. Following the well-known three-field notation, the BJSPT can be expressed by

$$Jm \mid r_i, recrc, block \mid \sum T_i.$$

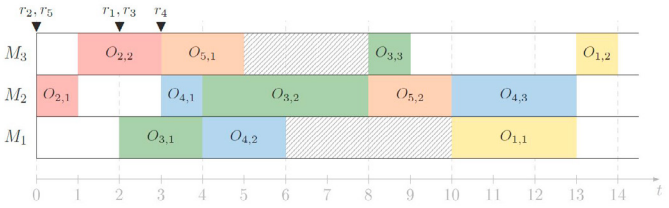

Fig. 1. Gantt chart of a feasible schedule s for the BJSPT

Two types of permutations are used to describe a schedule. The operation-based representation $s^{op} = [\ldots]$ depicts a solution as a single list of all operations $O_{i,j} \in \mathcal{O}$. Such a permutation is always assumed to be feasible with regard to the processing sequences and technological routes of the jobs. The machine-based representation $s^{ma} = [[\ldots], [\ldots], \ldots]$ corresponds to a nested list, which includes the operation sequences on the machines $M_1, \ldots, M_m$. Different aspects of redundancy and feasibility appearing while using these encodings are discussed in the subsequent section.

3. REDUNDANCY AND FEASIBILITY

To highlight several issues in representing schedules by permutations, an exemplary instance involving three machines and five jobs is regarded, cf. Lange (2019). A feasible schedule for this problem is shown in Figure 1 together with the release dates of the jobs $r_1, \ldots, r_5$ indicated by black triangles at the top. Note that the jobs $J_1, \ldots, J_5$ involve different numbers of operations with diverse processing times, feature individual technological routes and the job J_4 requires machine M_2 for its first and third processing step. Time periods, in which a machine is blocked by a job after processing, are marked by shading.

The operation-based representation

$$s^{op} = [O_{2,1}, O_{2,2}, O_{3,1}, O_{4,1}, O_{3,2}, O_{4,2}, O_{5,1}, O_{3,3}, O_{5,2}, O_{4,3}, O_{1,1}, O_{1,2}]$$

as well as the machine-based representation

$$s^{ma} = [[O_{3,1}, O_{4,2}, O_{1,1}], [O_{2,1}, O_{4,1}, O_{3,2}, O_{5,2}, O_{4,3}], [O_{2,2}, O_{5,1}, O_{3,3}, O_{1,2}]]$$

constitute proper encodings of the given schedule s. Since unique expressions of feasible schedules are desirable in operating a structured heuristic search technique, the parallel usage of both representations is necessary.

Operation-based lists feature a critical redundancy aspect. In many cases, there exist several different encodings describing the same schedule. Considering the permutation s^{op} given above, among others the pairs of operations $\{O_{2,2}, O_{3,1}\}$ and $\{O_{3,1}, O_{4,1}\}$ can be interchanged without causing a change in the actual schedule s. Generally, redundant permutations for blocking job shop schedules are constructible by interchanging pairs of operations of different jobs, which require different machines, are not connected by a blocking constraint and do not implement a swap of jobs on machines, see Lange (2019). To avoid treating redundant permutations as different solutions, the corresponding machine-based representations are used, since the operation sequences on the machines uniquely describe a schedule.

Unfortunately, a machine-based list may depict a schedule, which is infeasible with regard to blocking constraints. Consider as an example the encoding s^{ma} given above and interchange the operations $O_{2,1}$ and $O_{4,1}$ on machine M_2. The individual operation sequences do still satisfy the processing sequences and the technological routes of all jobs, but their simultaneous implementation is not possible. Job J_4 will block machine M_2 after the first processing step. It requires operation $O_{3,2}$ to be processed on the same machine as a direct successor due to the ordering $O_{3,1} \rightarrow O_{4,2}$ on machine M_1. Such a blocking and swapping behavior can only be detected and the violation of constraints can only be overcome by examining and adapting the corresponding operation-based representation of the schedule.

As a consequence, both types of representation are used in the proposed heuristic method. Transformations of one encoding into the other can be done in polynomial time, see Lange (2019). A machine-based list s^{ma} is easily constructed from a permutation s^{op} by taking the operations one by one from the first to the last position and stepwise expanding the operation sequences of the corresponding machines. In turn, transferring a machine-based representation s^{ma} to a permutation s^{op} needs a mechanism to account for redundancy. By assigning priorities to the operations in the list s^{ma} in accordance to their initial position in a given permutation s^{op}, a structurally equivalent permutation can be set up. For detailed explanations on this priority-based transformation scheme, the reader is referred to Lange (2019).

4. HEURISTIC SOLUTION METHOD

This section provides an overview of the proposed solution algorithm. First, the neighborhood structures are explained as key components of every heuristic search method. Two well-known permutation-based operators are applied to generate neighboring solutions of a currently regarded schedule. Moreover, the feasibility issue mentioned in the previous section necessitates the introduction of a repairing technique to operate an efficient search process. Consequently, two different neighborhoods are combined in SA to obtain good solutions for various instances of the considered problem.

4.1 Permutation-Based Neighborhood Structures

Since the BJSPT constitutes a special sequencing problem, interchanges and shifts of operations are widely applied transition schemes in permutation-based heuristics, cf. Anderson et al. (1997). Pursuing the ideas of generating close neighbors to assure intensification and a structured search process on the one hand, and guiding the search towards diverse, promising areas of the solution space on the other hand, the following moves are performed, cf. Lange (2019).

Definition 1. An **API move** denotes the pairwise interchange of two adjacent operations $O_{i,j}$ and $O_{i',j'}$ of different jobs requiring the same machine M_k in the machine-based representation of a schedule. A pair of operations is called **adjacent**, if there is no idle time between the preceding operation leaving the machine and the beginning of processing of the succeeding operation.

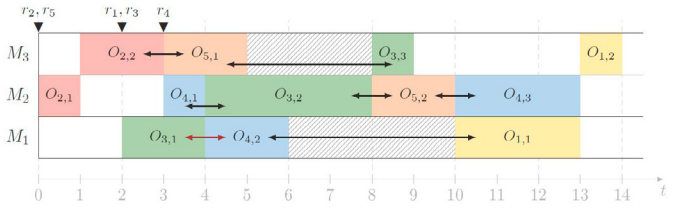

Fig. 2. Possible API moves in the feasible schedule s for the BJSPT

Definition 2. A **TJ move** denotes the random leftward shift of all operations of a tardy job J_i in the operation-based representation of a schedule (with preservation of the processing sequence $O_{i,1}, \ldots, O_{i,n_i}$).

Figure 2 shows all realizable API moves in the given schedule s for the exemplary instance. Note that the consecutively processed operations $O_{2,1}$ and $O_{4,1}$ on machine M_2 as well as $O_{3,3}$ and $O_{1,2}$ on machine M_3 cannot be interchanged in s, since there exists an idle time between these processing steps. The strict definition of adjacency applied here does not generally limit the search process. For most pairs of subsequently processed but non-adjacent operations, there exists a sequence of applicable API moves, which finally realizes an interchange. Only if an idle time between two operations is caused by extraordinarily different release dates of the corresponding jobs, an impossible API might exist. However, with regard to total tardiness, such an API move is unlikely to be beneficial. Thus, the connectivity property cannot be shown for the neighborhood structure purely based on API moves, but it may still be efficient for general and practically relevant instances, cf. Lange (2019).

Considering the API move of the operations $O_{3,1}$ and $O_{4,2}$ indicated in red in Figure 2 together with the operation-based representation s^{op} given above, the question arises whether to implement the transition by shifting operation $O_{3,1}$ rightwards to the position after $O_{4,2}$ or by shifting operation $O_{4,2}$ leftwards to the position before $O_{3,1}$. Evidently, the transfer of an API move from the machine-based representation to the single list of operations may lead to two different neighboring solutions. Here, the resulting permutations encode the same neighboring schedule s' after correcting the processing sequences of the jobs due to the simplicity of the example. Computational studies on the benchmark instances introduced in the subsequent section show that the right shift transformation is favorable against the left shift transformation, since it causes less feasibility issues and yields structurally closer neighbors, see Lange (2019) for details on these results.

A more critical aspect of this neighborhood structure is the appearing infeasibility of a vast majority of the generated neighboring solutions. As mentioned above, an API move in the machine-based representation s^{ma} may lead to a permutation s^{op} that does not satisfy the blocking constraints. Therefore, an ART is introduced to construct the operation-based encoding $s^{op'}$ of a feasible neighboring schedule s' from a potentially infeasible encoding *perm* of the incumbent schedule s involving the given transfered API move. This transition is not trivial, since the API ordering determines a partial schedule, which is to be completed to a neighbor s' by doing as few structural

Algorithm 1. (Advanced Repairing Technique).

Input: permutation *perm*, partial sequence resulting from API move $O_{a,b} \rightarrow O_{c,d}$

Output: feasible permutation $s^{op'}$ involving $O_{a,b} \rightarrow O_{c,d}$

```
 1: while perm is not empty do
 2:     O_{i,j} ← first operation in perm
 3:     if required machine is idle then
 4:         determine s_{i,j} of O_{i,j}
 5:         expand s^{op'} by O_{i,j} and delete O_{i,j} from perm
 6:     else
 7:         add O_{i,j} to queue
 8:         O_{i',j'} ← required operation to fulfill blocking constraint
 9:         while O_{i',j'} is not in queue do
10:             if O_{i',j'} = O_{c,d} then
11:                 shift O_{a,b} leftwards on its machine
12:                 (= modification of initial perm)
13:                 define additional, fixed ordering O_{a,b} → ...
14:                 restart procedure
15:             else
16:                 if required machine is not idle then
17:                     add O_{i',j'} to queue
18:                     O_{i',j'} ← required op. to fulfill blocking constr.
19:                 else
20:                     determine starting times for all op. in queue
21:                     expand s^{op'} and delete operations from perm
22:                 end if
23:             end if
24:         end while
25:         if queue is not empty then
26:             determine equivalent starting time for all op. in queue
27:             expand s^{op'} and delete operations from perm
28:         end if
29:     end if
30: end while
31: return s^{op'}
```

changes as possible in the schedule s. Algorithm 1 includes a schematic outline of the proposed repairing scheme, see Lange (2019).

The main strategy of the ART is to sequentially schedule the operations according to the possibly infeasible operation-based representation *perm*, which includes the given API move. If a machine required by operation $O_{i,j}$ is blocked by an operation $O_{i',j'-1}$ (line 6), an unsatisfied blocking constraint $s_{i',j'} \leq s_{i,j}$ is detected and the corresponding operation $O_{i',j'}$ is shifted from its posterior position in the permutation *perm* to the currently considered position. In case that this leftward shift contradicts a given API ordering $O_{a,b} \rightarrow O_{i',j'} = O_{c,d}$ (line 10), the fixed predecessor $O_{a,b}$ is shifted further to the left in the operation sequence on its machine to assure the implementation of the determined move. This corresponds to an additional structural change in the part of the encoding *perm* that has already been considered. Since the feasibility of the schedule $s^{op'}$ under construction can only be guaranteed, if the previously determined starting times of operations remain unchanged, the procedure needs to be restarted from the beginning after such a required modification of the permutation (line 14). In Lange (2019), the four different variants of a necessary adaptation summarized here in the lines 11 to 13 are comprehensively discussed. Furthermore, it is shown that due to a finite number of operations in the problem and irreversible shifting structures, the ART returns a feasible encoding $s^{op'}$ involving the given API ordering in polynomial time. Consequently, API moves can be applied to set up a neighborhood for the BJSPT.

The TJ move described in Definition 2 constitutes a random search mechanism to explore diverse areas of the solution space. A job J_i is chosen from the set of tardy jobs in the incumbent schedule s and all involved operations $O_{i,j}$ are shifted in the operation-based representation s^{op}, whereby the new position index is restricted to be at most half of the initial position index and the processing sequence of the job is preserved. Following the arguments given above, the resulting permutation is not necessarily feasible with regard to blocking constraints and, in contrast to the API move, a TJ move simultaneously defines several partial orderings. D'Ariano et al. (2007) show that it is not guaranteed that a given partial sequence with more than two operations can be completed to a feasible blocking job shop schedule. Even the corresponding decision problem is proven to be NP-complete. For this reason, a potentially infeasible encoding *perm* including a TJ move is transformed to a feasible schedule for the BJSPT by a simplified ART without any given partial sequence. Thus, the **if**-condition in line 10 of Algorithm 1 is discarded and line 16 is directly operated. Accepting the rarely occurring fact that the resulting feasible neighbor s' is equivalent to the initially given schedule s, TJ moves are applicable as a neighborhood structure for the BJSPT.

4.2 Simulated Annealing Metaheuristic

According to the significant feasibility issues discussed before, it can be expected that the guidance of the heuristic search procedure through the set of feasible schedules is not trivial and the acceptance of non-improving solutions might be a beneficial strategy to overcome the early entrapment in local optima. Therefore, an SA metaheuristic is used as a basic framework to combine the transition schemes described. The following steps and characteristics are involved.

(1) The *initial solution* is obtained as the best schedule returned by several priority rules.
(2) The applied *neighborhood* is chosen in every iteration according to the probability distribution: API move-based 0.9 and TJ move-based 0.1.
(3) An implemented API move is transfered by a left shift and by a right shift transformation to two separate operation-based lists. If the resulting permutations do not encode the same schedule, the superior one is chosen as the candidate to become the new incumbent solution.
(4) A *geometric cooling scheme* is used. Initial and terminal temperatures as well as the cooling factor are chosen as promising values from preliminary experiments in relation to the magnitude of the objective function values for the considered problem.
(5) The temperature level is repeatedly decreased after a finite number of iterations, which is dependent on the total number of operations and machines involved in the considered instance.
(6) The algorithm terminates if a given terminal temperature level or an overall time limit is reached and returns the best solution found.

Table 1. Computational results of SA for the train scheduling-inspired instances

Inst.	MIP	#(It.)	$\overline{\sum T_i}$	Best	Time
ts01	138*	33264	140.2	138*	384.99
ts02	90*	31878	94.6	91	394.39
ts03	72*	33264	74.2	72*	396.98
ts04	41*	34650	41.8	41*	437.78
ts05	71*	31878	71.4	71*	404.85
ts06	88*	57519	121.6	107	883.49
ts07	172*	58905	195.4	189	880.14
ts08	163*	57519	184.2	179	820.17
ts09	153	58905	178.8	168	921.90
ts10	97*	56133	114.8	97*	767.11
ts11	366	80388	406.4	390	1430.03
ts12	419	81774	428.2	412	1360.35
ts13	452	83160	462.6	448	1497.99
ts14	459	81774	462.8	418	1398.62
ts15	418	83160	419.4	401	1390.91

5. COMPUTATIONAL EXPERIMENTS

Numerical tests are conducted on two structurally different sets of benchmark instances. First, the well-known Lawrence instances la01 to la40, which feature arbitrary technological routes without recirculation and random processing times, cf. Lawrence (1984), are examined. Since these problems are created for the classical job shop setting, several modifications, namely the involvement of blocking constraints as well as the generation and consideration of release dates and due dates for all jobs, are required. The instances include 5 to 15 machines processing 10 to 30 jobs. To comparatively perform computational tests on real-world-inspired instances with interna structures, a set of randomly generated train scheduling instances ts01 to ts15 introduced by Lange and Werner (2018a) is used. These problems are based on a railway network, involve recirculation and implement trains of different speed levels traveling on tracks of different predetermined length. The track sections are transfered to 11 machines, while 10 to 20 trains define the set of jobs. For all benchmark instances, the release dates of the jobs are generated so that processing steps are supposed to overlap and the due dates are determined with a tight due date factor of 1.2, see Lange and Werner (2018a).

The computational experiments are performed on a notebook operating an Intel Dual Core i5 processor (2.20 GHz) with 8 GB RAM. The SA algorithm involving the permutation-based encodings, the neighborhood structures and the ART is implemented in Python 3. To precisely evaluate the performance of the scheduling-tailored heuristic against an advanced general-purpose method, the instances are modeled as mixed-integer programs using pairwise precedence variables, see Lange and Werner (2018a), and solved by IBM ILOG CPLEX 12.8.0 with a time limit of two hours. The same terminal criterion is also given to the SA algorithm. 表1和表2分别显示了混合整数规划(MIP)和SA对列车调度实例和劳伦斯实例的结果。

Tables 1 and 2 show the results obtained by mixed-integer programming (MIP) and SA for the train scheduling-inspired instances and the Lawrence instances, respectively. The second column displays the best objective function value determined by MIP. An asterisk marks a solution with proven optimality. The third column contains the total number of iterations #(It.) performed by SA. Columns four and five display the average total tardiness value $\overline{\sum T_i}$ reached within five independent runs of SA and the objective function value of the best found schedule. The average computation time of SA in seconds is given in the last column.

Table 2. Computational results of SA for the Lawrence instances

Inst.	MIP	#(It.)	$\overline{\sum T_i}$	Best	Time
la01	762*	11080	787.4	773	147.22
la02	266*	11080	283.4	266*	354.58
la03	357*	11080	357.0*	357*	118.16
la04	1165*	11080	1217.2	1165*	205.85
la05	557*	11080	557.0*	557*	138.57
la06	2516	18005	2790.0	2616	239.06
la07	1677*	18005	1942.2	1869	259.52
la08	1829*	18005	2335.0	1905	204.66
la09	2851	18005	3275.2	3161	202.80
la10	1841*	18005	2178.2	2069	203.90
la11	6534	24930	6186.2	5704	349.30
la12	5286	24930	5070.0	4859	400.51
la13	7737	24930	7850.6	7614	329.37
la14	6038	24930	6616.8	5714	379.36
la15	7082	24930	7088.6	5626	331.12
la16	330*	22160	395.8	335	428.88
la17	118*	22160	144.2	120	341.19
la18	159*	22160	229.4	159*	331.01
la19	243*	22160	306.6	243*	302.85
la20	42*	22160	55.6	42*	330.91
la21	1956	36010	2847.2	2101	675.21
la22	1455	36010	2052.8	1773	670.79
la23	3436	36010	3692.6	3506	650.58
la24	560*	36010	966.8	761	669.17
la25	1002	36010	1557.4	1289	667.03
la26	7961	49860	9275.8	8475	1383.19
la27	8915	49860	7588.0	6596	1304.75
la28	2226	49860	3430.8	2876	1253.97
la29	2018	49860	2948.0	2432	1273.96
la30	6655	49860	7621.6	6775	1328.15
la31	20957	77560	18921.8	17984	4174.65
la32	23150	77560	21991.4	20401	4779.26
la33	none	77560	22494.2	19750	4432.09
la34	none	77560	20282.8	18633	4770.32
la35	none	77560	21895.0	18778	4789.15
la36	675	54015	1856.0	1711	2403.28
la37	1070	54015	1774.2	1621	2154.29
la38	489*	54015	760.4	645	2298.98
la39	754	54015	1573.0	1391	2110.06
la40	407*	54015	1008.6	613	2163.05

The results indicate that the permutation-based heuristic method is capable of finding optimal and near-optimal solutions for small instances of the BJSPT. Among the problems ts01-ts05 ($m = 11, n = 10$), la01-la05 ($m = 5, n = 10$) and la16-la20 ($m = n = 10$), SA found 11 of 15 proven optimal solutions. For instances of medium size such as ts06-ts10 ($m = 11, n = 15$), la06-la10 ($m = 5, n = 15$) and la21-la25 ($m = 10, n = 15$), the heuristic method cannot compete with MIP in solution quality but still obtains good solutions in reasonable runtime. Considering instances involving even more machines and jobs, the

general-purpose solver suffers from weak lower bounds and the exponentially increasing number of possible branch and cut decisions. For the instances ts11-ts15, la11-la15 and la26-la34, MIP terminates after two hours of computation time with unsatisfactory feasible schedules. Regarding la33-la35 especially, there is no feasible solution found at all (indicated by 'none' in Table 2). Here, the scheduling-tailored search technique clearly outperforms general MIP with a guaranteed construction of feasible schedules and improved solution quality. For 15 out of 30 instances, for which an optimal solution is not yet known, SA determines the overall best schedule.

A comparison of the average and the minimal total tardiness values obtained by SA reveals structural issues, which reason the difficulties in tackling the BJSPT. The average objective function values differ significantly from the minimal values. This implies that the heuristic search procedure is highly dependent on random choices made in the construction of neighboring solutions. Thus, empirical evidence is given for a considerable ruggedness of the search space caused by the restrictedness of the problem and the tardiness-based optimization criterion. As a consequence, guiding the heuristic method according to observed objective function values and predictions might lead to an early entrapment in unsatisfactory locally optimal solutions, see also Lange and Werner (2018b). Furthermore, it is remarkable that the distances between neighboring schedules constructed by applying an API move are considerably larger for feasible schedules of the BJSPT than for schedules fitting the classical job shop setting, cf. Lange (2019). Doubtless, this is induced by the resolution of the feasibility issues described above. Feasible schedules fulfilling blocking constraints include strongly interdependent structures, which are hard to construct, destruct and recover. This is why in the generation of the majority of the neighboring schedules, a significant number of additional interchanges of operations is required to regain feasibility. For these reasons, the BJSPT remains a challenging combinatorial optimization problem.

6. CONCLUSION

Two different sets of benchmark instances of the BJSPT are solved by a scheduling-tailored heuristic search method, which involves permutation-based encodings of the schedules, interchange- and shift-based neighborhood structures and a complex repairing technique as key components. Computational results highlight the capability of the proposed heuristic method in finding optimal solutions for small instances and providing good feasible schedules for problems of larger size.

The search procedure suffers from the ruggedness of the search space and the required repeated application of non-trivial transformation and repairing schemes. Therefore, the design of hybrid solution approaches combining beneficial mechanisms of heuristic and MIP techniques can be named as a promising future research direction for complex job shop scheduling problems.

REFERENCES

Anderson, E.J., Glass, C.A., and Potts, C.N. (1997). *Machine Scheduling*, chapter 11, 361–414. Wiley Chichester, UK.

Balas, E. and Vazacopoulos, A. (1998). Guided local search with shifting bottleneck for job shop scheduling. *Management Science*, 44(2), 262–275.

Bürgy, R. (2017). A neighborhood for complex job shop scheduling problems with regular objectives. *Journal of Scheduling*, 20(4), 391 – 422.

D'Ariano, A., Pacciarelli, D., and Pranzo, M. (2007). A branch and bound algorithm for scheduling trains in a railway network. *European Journal of Operational Research*, 183(2), 643–657.

Garey, M.R., Johnson, D.S., and Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2), 117–129.

Groeflin, H. and Klinkert, A. (2009). A new neighborhood and tabu search for the blocking job shop. *Discrete Applied Mathematics*, 157(17), 3643–3655.

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.

Lange, J. (2019). *Solution Techniques for the Blocking Job Shop Scheduling Problem with Total Tardiness Minimization*. Ph.D. thesis, Otto-von-Guericke-Universität Magdeburg. Under submission.

Lange, J. and Werner, F. (2018a). Approaches to modeling train scheduling problems as job-shop problems with blocking constraints. *Journal of Scheduling*, 21(2), 191–207.

Lange, J. and Werner, F. (2018b). A permutation-based neighborhood for the blocking job-shop problem with total tardiness minimization. In *Operations Research Proceedings*, 581–586. Springer International Publishing.

Lawler, E.L., Lenstra, J.K., and Rinnooy Kan, A.H.G. (1982). *Recent Developments in Deterministic Sequencing and Scheduling: A Survey*, 35–73. Springer Netherlands, Dordrecht.

Lawrence, S. (1984). Supplement to resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques. *GSIA, Carnegie Mellon University, Pittsburgh, PA*.

Mati, Y., Rezg, N., and Xie, X. (2001). A taboo search approach for deadlock-free scheduling of automated manufacturing systems. *Journal of Intelligent Manufacturing*, 12(5-6), 535–552.

Mattfeld, D.C. and Bierwirth, C. (2004). An efficient genetic algorithm for job shop scheduling with tardiness objectives. *European journal of operational research*, 155(3), 616–630.

Oddi, A., Rasconi, R., Cesta, A., and Smith, S.F. (2012). Iterative improvement algorithms for the blocking job shop. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling*.

Pranzo, M. and Pacciarelli, D. (2016). An iterated greedy metaheuristic for the blocking job shop scheduling problem. *Journal of Heuristics*, 22(4), 587–611.

Van Laarhoven, P.J., Aarts, E.H., and Lenstra, J.K. (1992). Job shop scheduling by simulated annealing. *Operations Research*, 40(1), 113–125.

Zhang, R. and Wu, C. (2011). A simulated annealing algorithm based on block properties for the job shop scheduling problem with total weighted tardiness objective. *Computers & Operations Research*, 38(5), 854–867.