Exact and metaheuristic approaches for unrelated parallel machine scheduling

Maximilian Moser1 · Nysret Musliu1 · Andrea Schaerf2 · Felix Winter1

Accepted: 19 October 2021 / Published online: 7 December 2021
© The Author(s) 2021

Abstract
In this paper, we study an important real-life scheduling problem that can be formulated as an unrelated parallel machine scheduling problem with sequence-dependent setup times, due dates, and machine eligibility constraints. The objective is to minimise total tardiness and makespan. We adapt and extend a mathematical model to find optimal solutions for small instances. Additionally, we propose several variants of simulated annealing to solve very large-scale instances as they appear in practice. We utilise several different search neighbourhoods and additionally investigate the use of innovative heuristic move selection strategies. Further, we provide a set of real-life problem instances as well as a random instance generator that we use to generate a large number of test instances. We perform a thorough evaluation of the proposed techniques and analyse their performance. We also apply our metaheuristics to approach a similar problem from the literature. Experimental results show that our methods are able to improve the results produced with state-of-the-art approaches for a large number of instances.

Keywords Unrelated parallel machine scheduling · Multi-objective optimisation · Mixed-integer programming · Metaheuristics · Simulated annealing

1 Introduction
Finding optimised machine schedules in manufacturing is an important and challenging task, as a large number of jobs need to be processed every day. While a manual approach performed by human experts can be used to deal with a small number of jobs, the large-scale requirements of modern factories introduce the need for efficient automated scheduling techniques. In the literature, such scheduling problems that deal with the assignment of jobs to multiple machines which operate in parallel have previously been described as Parallel Machine Scheduling Problems (PMSPs, e.g. Allahverdi et al. 2008; Allahverdi 2015).

Several types of machines can perform different sets of operations in the industry, so that each job can be assigned to a specified set of eligible machines, and varying machine efficiency has to be considered. PMSPs that consider eligible machines are well-known in the literature and have been studied in several publications (e.g. Afzalirad and Rezaeian 2016; Perez-Gonzalez et al. 2019). Similarly, problems with varying machine efficiency have been previously described as Unrelated PMSP (UPMSP) (e.g. Vallada and Ruiz 2011; Avalos-Rosales et al. 2015; Allahverdi 2015). After a job has reached its completion on a machine, in practice it is often required to perform a change of the tooling or a machine maintenance before the next job can be processed. The corresponding changeover times between jobs have been referred to as sequence-dependent setup times in previous publications (e.g. Vallada and Ruiz 2011; Perez-Gonzalez et al. 2019).

In the problem, we investigate in this paper, which emerges from a company in the packaging industry, each job cor-
responds to a customer order with an associated due date. Therefore, the problem’s objective function aims to minimise the total tardiness of all jobs in addition to the total makespan.

In summary, the real-life problem we investigate in this paper can be characterised as UPMSP with sequence-dependent setup times and eligible machines that aims to minimise both tardiness and makespan. Although a large number of different variants of UPMSP that feature all of the mentioned attributes to some extent have been described in the literature, efficient metaheuristic and exact solution methods that consider both tardiness and makespan objectives at the same time remain to be investigated.

We adapt existing mathematical models for related problems and compare several variations for our problem. However, the evaluated exact approaches can solve in reasonable time only small instances. To solve large practical instances, we deeply investigate several variants of Simulated Annealing (SA, Kirkpatrick et al. 1983). The variants we investigate include different cooling schemes such as a dynamic cooling rate and a reheating mechanism. We investigate different search neighbourhoods based on shift and swap moves, which are commonly used in the context of PMSP. Additionally, we propose a novel neighbourhood operator for UPMSP that operates on blocks of consecutively scheduled jobs. To increase the effectiveness of the move generation procedure, we guide the search towards more promising regions of the search space by incorporating domain knowledge into the neighbourhood move selection strategy.

We provide real problem instances which are based on real-life scheduling scenarios that have been provided to us by an industrial partner. Furthermore, we propose a random instance generator to generate diverse datasets that together with the real-world instances form a large pool of instances which we use in our experimental evaluation. Experimental results show that the Simulated Annealing approach is able to obtain high-quality solutions for both randomly generated and real-life instances.

To show the robustness of our method we compare to the approach from Perez-Gonzalez et al. (2019) that was proposed recently for a similar problem. The problem specification provided by Perez-Gonzalez et al. (2019) uses the same set of constraints of the problem that we investigate in this paper and also aims to minimise total tardiness. However, the authors do not consider the minimisation of the total makespan. As the minimisation of total tardiness is incomparably more important than the makespan in our problem specification, we can easily use our solution methods to approach the problem instances provided by Perez-Gonzalez et al. (2019). Our comparison on a huge set of instances that have been provided by Perez-Gonzalez et al. (2019) shows that our approach produces improved results for the majority of the instances.

The structure of the paper is as follows: In Sect. 2, we describe our problem and give an overview of the existing related literature. We provide eight different mixed-integer programming (MIP) formulation variants for the problem in Sect. 3. Our metaheuristic algorithms are presented in Sect. 4. Our instance generator and our datasets are described in Sect. 5. The results of our computational experiments are presented in Sect. 6. Finally, conclusions and remarks on possible future research are given in Sect. 7.

2 Problem statement and literature review

The problem we investigate in this paper can be characterised as an Unrelated Parallel Machine Scheduling Problem with Sequence-Dependent Setup Times and Machine Eligibility Constraints with the objective of minimising both total tardiness as well as makespan. A Parallel Machine Scheduling Problem (PMSP) aims to assign jobs to a given number of machines to create an optimised production schedule, where every machine is continuously available and can process one job at a time. The execution of jobs cannot be interrupted and resumed later, and there are no precedence constraints between jobs. Furthermore, every machine is assumed to start immediately the execution of its assigned schedule without any break or interruption.

Instances of the PMSP are stated using a set of machines \( M \) and a set of jobs \( J \). For every job \( j \in J \), we have a due date \( d_j \) denoting its latest acceptable completion time and a set of eligible machines \( M_j \subseteq M \) on which the job can be processed. The processing time \( p_{jk} \) of each job \( j \in J \) depends on the machine \( k \in M \) on which it is executed. Further, setup times \( s_{ijk} \) are defined for any pair of jobs \( i,j \) that are consecutively scheduled on a machine \( k \). Additionally, an initial setup time \( s_{0jk} \) is required before the execution of job \( j \) can begin when it is scheduled as the first job on machine \( k \). Analogously, a clearing time \( s_{j0k} \) is required after the execution of job \( j \), if it is the last scheduled job on machine \( k \).

A solution to a PMSP assigns a schedule for every machine \( k \), which is represented by a permutation of a subset of all jobs \( J \). If a job \( i \) occurs directly before another job \( j \) in the schedule for machine \( k \), then \( i \) is called the predecessor of \( j \) (and \( j \) is the successor of \( i \)). Since the solution representation of the schedule is sequence-based, the completion time \( C_j \) of a job \( j \) is the sum of the predecessor’s completion time \( C_i \), the appropriate setup time \( s_{ijk} \), and the job’s processing time \( p_{jk} \). If a job is the first in its schedule, its completion time is defined by the initial setup time \( s_{0jk} \) plus its own processing time \( p_{jk} \). Furthermore, the tardiness of a job is defined to be the difference between its completion time and due date \( (T_j := \max(0, C_j - d_j)) \). The machine span \( O_k \) for a machine \( k \) is set to the completion time of the job which is scheduled last plus the final clearing time \( s_{j0k} \). Note that the
final clearing times only affect the machine spans, but not the completion times of jobs. Finally, the makespan $C_{\text{max}}$ is defined to be equal to the maximum of all machine spans.

Graham et al. (1979) proposes a three-field $\alpha/\beta/\gamma$ notation to categorise machine scheduling problems, where $\alpha$, $\beta$ and $\gamma$ describe the machine environment, additional constraints and the solution objectives, respectively. The machine environment for PMSPs typically consists of identical machines ($P_m$) or unrelated machines ($R_m$). Given that we are dealing with an environment consisting of unrelated machines, the problem we investigate can be characterised as $R_m | s_{ijk}, M_j | \text{Lex}(\Sigma_j(T_j), C_{\text{max}})$. The objective function $\text{Lex}(\Sigma_j(T_j), C_{\text{max}})$ in this case means that both tardiness and makespan should be minimised, with the tardiness being incomparably more important than the makespan (i.e. the objectives are lexicographically ordered). This particular problem variant can be seen as a generalisation of the basic PMSP with identical machines ($P_m | C_{\text{max}}$), which has been shown to be NP-hard even with only two machines (see Garey and Johnson 1979). Furthermore, even the minimisation of only tardiness has been shown to be NP-hard by Du and Leung (1990).

PMSPs have been the subject of thorough research in the past, and two surveys by Allahverdi et al. (2008) and Allahverdi (2015) give an overview of the related literature. Sequence-dependent setup times on unrelated machines have been described for many problems that have been studied in the literature. A well-known problem in this area is for example the Unrelated Parallel Machine Scheduling Problem with Sequence-Dependent Setup Times, aiming to minimise the makespan ($R_m | s_{ijk} | C_{\text{max}}$). Among the first to investigate this problem variant were Al-Salem (2004), who propose a Partitioning Heuristic as solution method and Rabadi et al. (2006) propose a Parti-

ditional machines ($P_m | C_{\text{max}}$) or unrelated machines ($R_m$). Given that we are dealing with an environment consisting of unrelated machines, the problem we investigate can be characterised as $R_m | s_{ijk}, M_j | \text{Lex}(\Sigma_j(T_j), C_{\text{max}})$. The objective function $\text{Lex}(\Sigma_j(T_j), C_{\text{max}})$ in this case means that both tardiness and makespan should be minimised, with the tardiness being incomparably more important than the makespan (i.e. the objectives are lexicographically ordered). This particular problem variant can be seen as a generalisation of the basic PMSP with identical machines ($P_m | C_{\text{max}}$), which has been shown to be NP-hard even with only two machines (see Garey and Johnson 1979). Furthermore, even the minimisation of only tardiness has been shown to be NP-hard by Du and Leung (1990).

PMSPs have been the subject of thorough research in the past, and two surveys by Allahverdi et al. (2008) and Allahverdi (2015) give an overview of the related literature. Sequence-dependent setup times on unrelated machines have been described for many problems that have been studied in the literature. A well-known problem in this area is for example the Unrelated Parallel Machine Scheduling Problem with Sequence-Dependent Setup Times, aiming to minimise the makespan ($R_m | s_{ijk} | C_{\text{max}}$). Among the first to investigate this problem variant were Al-Salem (2004), who propose a Partitioning Heuristic as solution method and Rabadi et al. (2006) who tackle the problem using a novel technique called Meta-Heuristic for Randomised Priority Search. Arnaout et al. (2010) proposed an Ant Colony Optimisation algorithm, which they further improved later (Arnaout et al. 2014). Vallada and Ruiz (2011) propose Genetic Algorithms for the problem and create a set of benchmark instances for their experiments. Avalos-Rosales et al. (2015) propose two new mathematical formulations and a Variable Neighbourhood Descent metaheuristic. They show that their proposed MIP models outperform existing models on the set of benchmark instances provided by Vallada and Ruiz (2011). More recent contributions include Santos et al. (2019) who use Stochastic Local Searches on Vallada’s instances. They find that Stochastic Annealing offers good performance over all evaluated instance sizes. Tran et al. (2016) apply both Logic-based Benders Decomposition and Branch and Check as exact methods for solving the problem. Gedik et al. (2018) propose a Constraint Programming formulation of the problem, leveraging the benefits of interval variables. Fanjul-Peyro et al. (2019) propose a new MIP model for the problem and an algorithm based on mathematical programming. They replace the sub-tour elimination constraints in the MIP model from Avalos-Rosales et al. (2015) by constraints adapted from previous traveling salesperson problem formulations. This results in a more efficient mathematical formulation for the problem which does not compute all of the jobs’ completion times. Other contributions for this problem variant include a Tabu Search approach (Helal et al. 2006) and a Simulated Annealing approach (Ying et al. 2012).

PMSPs that include machine eligibility constraints have been considered several times in the literature. Rambod and Rezaeian (2014) consider a PMSP with sequence-dependent setup times and machine eligibility constraints that focuses on minimising the makespan ($R_m | s_{ijk}, M_j | C_{\text{max}}$). Additionally, they include the likelihood of manufacturing defects in their objective function. Afzalirad and Rezaeian (2017) minimise a bi-criterion objective consisting of mean weighted tardiness ($MWT$) and mean weighted flow time ($MWFT$) in a PMSP with sequence-dependent setup times and machine eligibility. They assume different release times of jobs ($r_j$) and precedence constraints ($prec$) among jobs ($R_m | s_{ijk}, M_j, r_j, prec, res | C_{\text{max}}$).

Afzalirad and Rezaeian (2016) try to minimise the makespan for a similar problem where the execution of jobs requires additional resources ($res$) with limited availability ($R_m | s_{ijk}, M_j, r_j, prec, res | C_{\text{max}}$). Bektur and Saraç (2019) consider a problem variant similar to the one we investigate in this paper, where they minimise the total weighted tardiness. Additionally, they require the availability of a single server ($S_1$) to perform the setups between jobs ($R_m | s_{ijk}, M_j, S_1 | \sum_j(w_j \cdot T_j)$). Chen (2006) considers a problem with machine eligibility, where fixed setup times are only required if two consecutive jobs produce different product families. Their objective is to minimise the maximum tardiness of all jobs. Afzalirad and Shafipour (2018) try to minimise the makespan in a PMSP with machine eligibility and resource restrictions and assume that setup times are included in the processing times.

The problem statements most closely resembling the problem considered in this paper are studied by Caniyilmaz et al. (2015), Adan et al. (2018) and Perez-Gonzalez et al. (2019). All three papers use sequence-dependent setup times, due dates and machine eligibility constraints. However, each of these papers focuses on the minimisation of a slightly different objective function.

Caniyilmaz et al. (2015) try to minimise the sum of makespan and cumulative tardiness ($C_{\text{max}} + \sum_j T_j$). They implement an Artificial Bee Colony algorithm and compare its performance against a Genetic Algorithm on a real-life instance originating from a quilting work centre. This most closely resembles our objective of minimising tardiness as primary target and makespan as secondary target. Adan et al. (2018) try to minimise a three-part objective function, con-
sisting of a weighted sum of total tardiness, setup times and processing times (\(\alpha \cdot \sum_j T_j + \beta \cdot \sum_{jk} p_{jk} + \gamma \cdot \sum_{ijk} s_{ijk}\)), where \(\alpha\), \(\beta\) and \(\gamma\) are weights. This objective function coincides with our objective function when there is only a single machine available and the weights are chosen appropriately. They implement a Genetic Algorithm very similar to the one described by Vallada and Ruiz (2011) and apply it to three real-life datasets. Perez-Gonzalez et al. (2019) also consider a problem that is similar to ours, but they only take the tardiness of jobs into consideration and disregard the makespan. Furthermore, they propose a MIP model for their problem which is based on the mathematical formulation from Vallada and Ruiz (2011), along with five different constructive heuristics and an immune-based metaheuristic. They are the first to create a sizeable dataset that is available for other researchers.

In summary, we can see that a large variety of PMSPs have been studied in the past. However, to the best of our knowledge, efficient metaheuristic and exact solution methods for the particular problem variant that considers a lexicographically ordered minimisation of total tardiness and makespan under machine eligibility constraints and sequence-dependent setup times remains to be investigated.

### 3 Mixed-integer programming

In their work, Perez-Gonzalez et al. (2019) propose a mathematical formulation for a similar PMSP that is based on the model described by Vallada and Ruiz (2011). As their objective function does not consider the makespan, their model does not include corresponding constraints. We therefore extend their proposed model by constraint sets (5) and (8), to calculate the makespan (which includes the clearing times).

The variables used in the formulation are described in Table 1. The set \(J_0\) includes a dummy job (0), that represents the start and end points of each machine schedule. The predecessor of the first job assigned to each machine is set to be the dummy job. Similarly, the successor of the last job assigned to each machine is set to be the dummy job. \(X_{i,j,m}\) are binary decision variables which are set to 1 if and only if job \(j\) is scheduled directly after job \(i\) on machine \(m\) and variables \(T_j\) represent the completion time of job \(j\) on machine \(m\) and \(C_{j,m}\) denotes the completion time of job \(j\) on machine \(m\) and \(C_{max}\) is set to the total makespan which includes the clearing times.

The resulting model \(M1\) can be stated as follows:

\[
\begin{align*}
\text{minimise} \quad & L\epsilon x(\Sigma_{j \in J}(T_j), C_{max}), \\
\text{subject to} \quad & \ T_j \geq C_{j,m} - d_{jm}, \forall m \in M, j \in J, \quad (1) \\
\ & \sum_{m \in M} \Sigma_{i,k,j \neq j} X_{i,j,m} = 1, \forall i \in J, \quad (2) \\
\ & \sum_{m \in M} \Sigma_{j \neq j} X_{i,j,m} \leq 1, \forall i \in J, \quad (3) \\
\ & \sum_{j \in J} X_{0,j,m} \leq 1, \forall m \in M, \quad (4) \\
\ & \sum_{m \in M} \Sigma_{i \neq j} X_{i,j,m} = \sum_{m \in M} \Sigma_{i \neq j} X_{j,i,m}, \quad (5) \\
\end{align*}
\]

Constraint set (1) binds the tardiness of each job. Constraint set (2) ensures that every job has exactly one predecessor and is scheduled on one machine, while constraint set (3) restricts every job to have only one successor. They are not instantiated for the dummy job, because it is shared over all machines and thus can have multiple predecessors (successors) in the solution. Constraint set (4) restricts every machine to schedule at most one job at position one. Constraint set (5) forces each job (except for the dummy job) to have a single predecessor and successor on the machine where it is scheduled. Constraint set (6) checks that if a job \(j\) is scheduled after another job \(i\) on the same machine, then \(i\) has at least one predecessor as well. Constraint set (7) calculates the completion time \(C_{j,m}\) for every job \(j\) on machine \(m\). Note that \(V \cdot (1 - X_{i,j,m})\) evaluates to 0, if job \(i\) is the predecessor of job \(j\) on some machine. Otherwise, it evaluates to \(V\) and thereby fulfils the inequality. Given that every job has a processing time greater than zero on every machine, constraint set (7) also enforces sub-tour elimination regarding the predecessor relations. Constraint set (8) calculates the makespan which includes the clearing time after the final job. Constraint set (9) forces the completion time of the dummy job to be 0 on every machine. Constraint sets (10) to (12) restrict the domains of the decision variables.

Note that constraint set (2) does not ensure that jobs are scheduled on one of their eligible machines. Instead, \(M1\) sets the processing times of each job on its ineligible machines to \(V\) (an upper bound to the makespan) so that the minimisation will implicitly avoid assignment of jobs on ineligible machines. This way of implicitly modelling the machine eligibility constraints was used in the parallel machine scheduling model proposed by Perez-Gonzalez et al. (2019).

We additionally investigate in this paper an alternative model \(M2\) that simply models machine eligibility explicitly by constraint sets (13) and (14) that replace constraint set (2).

\[
\begin{align*}
\sum_{m \in E_j} \Sigma_{i \in J_0,i \neq j} X_{i,j,m} = 1, \forall j \in J, \quad (13) \\
\sum_{m \in M \setminus E_j} \Sigma_{i \in J_0,i \neq j} X_{i,j,m} = 0, \forall j \in J, \quad (14)
\end{align*}
\]

Constraint set (13) ensures that every job is scheduled on exactly one of its eligible machines, while constraint
set (14) prohibits jobs from being scheduled on any other machine.

Avalos-Rosales et al. (2015) propose new mathematical formulations for the problem described by Vallada and Ruiz (2011). Most notably, they include a new set of binary decision variables \( Y_{j,m} \) to describe whether or not job \( j \) is scheduled on machine \( m \) and they additionally introduce the notion of *machine spans*. Further, they replace the decision variables for the completion time \( C_j \) of job \( j \) on machine \( m \) by the variables \( C_j \).

In addition to the above-mentioned models M1 and M2, we extend the model proposed by Avalos-Rosales et al. (2015) to our problem statement to derive alternative mathematical formulations. These extensions include machine eligibility, due dates and the incorporation of final clearing times.

The resulting model M3 can be stated as:

\[
\text{minimise } L.e.x.(\Sigma_{j \in J}(T_j), C_{\text{max}}), \text{ subject to }
\]

\[
\begin{align*}
\Sigma_{m \in M}(Y_{j,m}) &= 1, \forall j \in J \quad (15) \\
\Sigma_{i \in J_0, i \neq j}(X_{i,j,m}) &= Y_{j,m}, \forall j \in J, m \in M \quad (16) \\
\Sigma_{j \in J_0, i \neq j}(X_{i,j,m}) &= Y_{m,i}, \forall i \in J, m \in M \quad (17) \\
C_j &\geq C_i + s_{i,j,m} + p_{j,m} + V \cdot (X_{i,j,m} - 1), \forall i \in J_0, j \in J, m \in M \\
\Sigma_{j \in J}(X_{0,j,m}) &\leq 1, \forall m \in M \quad (18) \\
\Sigma_{i \in J_0, j \neq j}(s_{i,j,m} \cdot X_{i,j,m}) + \Sigma_{i \in J}(p_{i,m} \cdot Y_{i,m} + s_{i,0,m} \cdot X_{i,0,m}) &\leq C_{\text{max}}, \forall m \in M \quad (19) \\
T_j &\geq C_j - d_j, \forall j \in J \quad (20) \\
T_j &\geq 0, \forall j \in J \quad (21) \\
C_0 &= 0 \quad (22) \\
X_{i,j,m} &\in \{0, 1\}, \forall i, j \in J, m \in M \quad (23) \\
Y_{i,m} &\in \{0, 1\}, \forall j \in J, m \in M \quad (24)
\end{align*}
\]

Constraint set (15) ensures that every job is scheduled on exactly one machine. Constraint sets (16)(17) ensure that every job has predecessors and successors on a machine if and only if the job is scheduled on this machine. Constraint set (18) connects the completion time for each job to its predecessors. Constraint set (19) ensures that there is at most one first job on each machine. Constraint set (20) binds the machine span for each machine by summing up the setup times between its scheduled jobs and their processing times. Constraint sets (21) and (22) involve the tardiness of each job and force it to be nonnegative. Constraint (23) sets the dummy job’s completion time to 0. Constraint (24) and (25) enforce that variables \( X_{i,j,m} \) and \( Y_{i,m} \) have a binary domain.

The model M3 incorporates machine eligibility via penalisation of the corresponding processing times similar as it is done in M1, and therefore also sets the processing time of each job on its ineligible machines to \( V \). We investigate in this paper an alternative model M4 that is based on M3 but explicitly models the machine eligibility constraints using hard constraints. M4 therefore replaces constraint set (15) with constraint sets (26) and (27).

\[
\begin{align*}
\Sigma_{m \in E_j}(Y_{j,m}) &= 1, \forall j \in J \quad (26) \\
\Sigma_{m \in M \setminus E_j}(Y_{j,m}) &= 0, \forall j \in J \\
\end{align*}
\]

Constraint set (26) ensures that every job is scheduled on exactly one of its eligible machines, while constraint set (27) prohibits jobs from being scheduled on any ineligible machines.

Helal et al. (2006) use a different formulation for constraint set (18) which aggregates the machines via sums instead of instantiating the constraints for every machine. Replacing constraint set (18) in models M3 and M4 by constraint set (28) results in models M5 and M6.

**Table 1** Variables used in MIP models M1 and M2

| Parameter  | Additional Information | Description |
|------------|------------------------|-------------|
| \( J \)    | —                      | Set of Jobs |
| \( J_0 \)  | —                      | Set of Jobs, including Dummy Job 0 |
| \( M \)    | —                      | Set of Machines |
| \( E_j \)  | \( j \in J, E_j \leq M \) | Eligible Machines of job \( j \) |
| \( d_j \)  | \( j \in J \)         | Due Date of job \( j \) |
| \( p_{j,m} \) | \( j \in J, m \in M \) | Processing Time of job \( j \) on machine \( m \) |
| \( s_{i,j,m} \) | \( i, j \in J_0, m \in M \) | Setup Time between jobs \( i \) and \( j \) on machine \( m \) |

**Variable**

| Additional Information | Description |
|------------------------|-------------|
| \( X_{i,j,m} \) | Job \( i \) is the predecessor of job \( j \) on machine \( m \) |
| \( C_{j,m} \) | Completion time of job \( j \) on machine \( m \) |
| \( T_j \) | Tardiness of job \( j \) |
| \( C_{\text{max}} \) | Makespan |
| \( V \) | Large Constant, e.g. Upper Bound for Makespan |
\[ C_j \geq C_i + \sum_{m \in M} (X_{i,j,m} \cdot (s_{i,j,m} + p_{j,m})) + V \cdot (\sum_{m \in M} (X_{i,j,m}) - 1), \] \hspace{1cm} (28)
\[ \forall i \in J_0, j \in J \]

We additionally experimented with an alternative variant for constraint set (5), which removes the set of machines from the summation and instead includes it in the \( \forall \) quantifier. We therefore introduce constraint set (29):

\[ \sum_{i \in J_0, i \neq j} X_{i,j,m} = \sum_{i \in J_0, i \neq j} X_{i,j,m}, \]
\[ \forall j \in J, m \in M \] \hspace{1cm} (29)

Replacing constraint set (5) in models M1 and M2 by constraint set (29) results in models M7 and M8.

The implementation of models M1--M8 and their effectiveness will be discussed in Sect. 6.1.

4 Metaheuristic approaches

In addition to the MIP formulations, we propose several Simulated Annealing variants to quickly find high-quality solutions for large instances as they appear in the real world. We first describe how initial solutions for local search can be generated in Sect. 4.1. Afterwards, we propose neighbourhood operators for the PMSP in Sect. 4.2, and finally, we describe three Simulated Annealing variants in Sect. 4.3.

4.1 Constructing initial solutions

One way to create an initial solution for local search is to randomly assign jobs to machines. We can do this by selecting one of the eligible machines for each job randomly and then scheduling all jobs in random order on the selected machines.

An alternative to using a random construction of initial solutions is to greedily build an initial schedule. In our case, we propose a constructive greedy heuristic which aims to minimise both tardiness and makespan as follows: first, we sort the set of jobs in ascending order by the due dates. Afterwards, we process the ordered jobs and schedule one job after the other on one of its eligible machines. To decide which machine should be selected for a job, the greedy heuristic compares the total machine spans that would be caused by each of the feasible machine assignment and finally selects the assignment that leads to the lowest machine span (ties are broken randomly). If multiple jobs exist that have exactly the same due date, we compare possible machine assignments for all of these jobs in a single step instead of processing them in random order. In such a case we then also select the job to machine assignment that leads to the lowest increase in machine span. The detailed pseudo-code for this heuristic can be seen in Algorithm 1.

Algorithm 1 Constructive Heuristic (CH)

1: function CONSTRUCTSOLUTION(Jobs, Machines)
2:   for all \( m \in \text{Machines} \) do
3:     \( \triangleright \) initialise empty machine schedules
4:   set machine span to 0
5:   \( t_m \leftarrow 0 \)
6:   \( \triangleright \) set last scheduled job id to 0 (no job) at first
7:   \( l_m \leftarrow 0 \)
8: end for
9: \( G \leftarrow \text{sort and group Jobs by due dates} \)
10: for all \( g \in G \) do
11:   while \( |g| > 0 \) do
12:     \( \triangleright \) find job/machine causing lowest machine span
13:     \( j, m \leftarrow \text{argmin}_{e \in g, n \in M} (t_n + s_{n,im} + p_{im}) \)
14:     \( t_m \leftarrow t_m + s_{n,im} + p_{im} \)
15:     \( l_m \leftarrow j \)
16:     \( \triangleright \) schedule job \( j \) on machine \( m \)
17:     \( \text{Schedule}_m \cdot \text{APPEND}(j) \)
18:     \( g \leftarrow g \setminus \{j\} \)
19:   end while
20: end for
21: end function

4.2 Search neighbourhoods

In this section, we introduce the neighbourhood relations that we use in our search method. We begin with the atomic neighbourhoods Shift and Swap, and then we describe the more complex block moves, called BlockShift and BlockSwap. Finally, we discuss the general notion of guidance used to bias the random selection toward promising moves.

4.2.1 Shift neighbourhood

A Shift move is configured to shift a given job \( j \) onto machine \( m \) at position \( p \). In other words, the job \( j \) is first removed from its original location in the current solution. Any successor on the associated machine is then shifted by one position towards the front of the schedule. Finally, job \( j \) is re-inserted into the solution at its target position \( p \) in the schedule of machine \( m \). Any job that is present on the target schedule at a later or equal position is shifted towards the end of the schedule.

We call a shift move an intra-machine shift move if job \( j \) is already scheduled on machine \( m \) in the current solution. Otherwise, if \( j \) is currently assigned to a machine different to \( m \), it is called an inter-machine shift move. An example of an inter-machine shift move is visualised in Fig. 1. The figure shows a schedule for two machines, before and after job 3 is moved from machine 1 to machine 2. Note that the length of the arrows between two consecutive jobs indicates the length of the required setup time.

Shift moves are feasibility-preserving as long as the target machine \( m \) is eligible for the selected job \( j \).
4.2.2 Swap neighbourhood

A Swap move swaps the position of two distinct jobs $j_1$ and $j_2$. If both jobs are scheduled on the same machine, we refer to such a move as an intra-machine swap. Otherwise, if jobs are scheduled on different machines, we call it an inter-machine swap. An example of an inter-machine swap move is visualised in Fig. 2. The figure shows a schedule for two machines, before and after job 3 is swapped with job 11 between machine 1 and machine 2.

Note that in the case of inter-machine swaps it is likely that the processing times and associated setup times for both jobs will change. Furthermore, completion times of all jobs that are scheduled on the affected machines after the swapped jobs need to be updated. When performing intra-machine swaps, the processing times of the swapped jobs do not change. However, the completion times of other jobs on the same schedule still need to be updated.

To preserve feasibility for swap moves, it has to be ensured that the first job’s machine has to be eligible for the second job and vice versa.

4.2.3 Block moves

We introduce the notion of block moves, as a variant of the basic shift and swap neighbourhoods: A block is defined as a set of jobs that are scheduled consecutively on a single machine. Therefore, a block move operates on a set of jobs instead of a single job. This concept can be applied to both Swap and Shift moves, leading to two new neighbourhoods that we call BlockSwap and BlockShift, respectively. BlockShift moves are similar to regular shift moves, but include an additional parameter $l$ determining the length of the block. In turn, every BlockSwap move uses two additional attributes $l_1$ and $l_2$ representing the length of the blocks.

Block moves are motivated by our real-life application, where usually several jobs process the same material type and thus are in the best case scheduled consecutively to avoid unnecessary setup times. Therefore, moving blocks of jobs at once can be beneficial to preserve low setup times when searching for neighbourhood solutions. To the best of our knowledge, using such block moves to approach parallel machine scheduling problems has not been considered in the literature before.

To be feasibility-preserving, the target machine of a BlockShift move has to be included in the intersection of all eligible machine sets of the affected blocks. For intra-machine BlockSwap moves, the blocks may not overlap and the second block’s machine has to be contained in the intersection of the eligible machine sets of all jobs in the first block and vice versa.

4.2.4 Guidance in random move generation

As customary for many practical scheduling problems, the size of the search neighbourhood becomes tremendously large for real-world instances. Therefore, it might be infeasible to explore all possible neighbourhood moves in reasonable time and the probability of randomly guessing an improving move usually is very low. Thus, it can be beneficial to introduce problem-specific strategies that guide exploration towards promising areas in the search neighbourhood. The general idea is to preferably select moves affecting jobs and machines that contribute to the cost of the solution.

Santos et al. (2019) propose a guidance strategy for their Simulated Annealing approach that focuses on reducing the makespan. They point out that a move can only improve the makespan if it involves the machine with the longest machine span. They generate moves such that at least one of the involved machines is fixed to be this machine, in addition to randomly generated moves.
Similarly, in our problem the tardiness of a solution can only be improved if a tardy job is either rescheduled at an earlier time, or its predecessors are shifted to other machines or later positions. Therefore, we propose a move selection strategy that is biased towards moves that shift tardy jobs to earlier positions in the schedule as follows: Whenever a tardy job exists in the schedule, either the job itself or one of its predecessors is selected randomly. Additionally, in case of an intra-machine (block) shift move we restrict the tardy job exists, any random job is chosen. Furthermore, in the case of an intra-machine (block) shift move we restrict the tardy job exists in the schedule, either the job itself or one to earlier positions in the schedule as follows: Whenever a tardy job exists, any random job is chosen. Additionally, in case of an intra-machine (block) shift move we restrict the tardy job exists in the schedule, either the job itself or one to earlier positions in the schedule as follows: Whenever a tardy job exists, any random job is chosen. Additionally, in case of an intra-machine (block) shift move we restrict the tardy job exists in the schedule, either the job itself or one to earlier positions in the schedule as follows: Whenever a tardy job exists, any random job is chosen. Additionally, in case of an intra-machine (block) shift move we restrict the
detailed procedure to select a job that can improve tardiness

To generate the moves we include, in addition to a completely random move generation, both the makespan guidance and the tardiness guidance strategies. One of the mentioned generation strategies is chosen during a search iteration based on random probabilities that are configured by parameters.

4.3 Simulated annealing

Simulated Annealing is a metaheuristic procedure which is inspired by the cooling processes appearing in metallurgy and has first been proposed by Kirkpatrick et al. (1983). The main idea is to generate random neighbourhood moves and determine the probability of move acceptance based on the change in solution quality caused by the move. Moves that lead to an improved objective function or no change in solution cost are accepted in any case. To determine whether or not a move that weakens the solution quality should be accepted, the notion of temperature is used. Simply put, the higher the temperature, the higher is the probability to accept also worsening moves. As the search goes on, the temperature lowers its value according to some cooling scheme and eventually reaches values close to zero. Towards the end of search

Simulated Annealing therefore evolves into a Hill-Climber as only improving moves are accepted.

Pseudo-code for the basic procedure of Simulated Annealing can be seen in Algorithm 3, where $T_{\text{max}}$, $N_s$, and $\alpha$ are the initial temperature, the number of samples per temperature, and the cooling rate, respectively.

Algorithm 3 Simulated Annealing

```
1: function SA(Solution)
2:     $c \leftarrow \text{Solution}$
3:     $b \leftarrow c$
4:     $t_c \leftarrow T_0$
5:     while ¬timeout do
6:         for $i \leftarrow 0$ until $N_s$ do
7:             $x \leftarrow \text{GENERATE\_NEIGHBOUR}(c)$
8:                 if ACCEPT($x$, $t_c$) then
9:                     $c \leftarrow x$
10:                if COST($x$) < COST($b$) then
11:                    $b \leftarrow x$
12:            end if
13:         end if
14:     end for
15:     $t_c \leftarrow t_c \cdot \alpha$
16: end while
17: return $b$
18: end function
```

In the remainder of this section, we propose three different variants of Simulated Annealing with different cooling schemes.

4.3.1 Reheating simulated annealing (SA-R)

This uses a geometric cooling scheme, i.e. $t_{i+1} := t_i \cdot \alpha$, where the cooling rate $\alpha$ is a predefined constant. At each temperature, a number of moves $N_s$ are generated before the next cooling step is applied. To determine the neighbourhood from which to sample the next move, we use a set of hierarchical probabilities ($p_1$, $p_S$, and $p_B$). When the next move is determined to be a block move, we determine its size by uniformly sampling from the set $\{2, ..., B_{\text{max}}\}$. Further options for move generation are whether or not to enable guidance towards minimising tardiness or makespan. This is again handled by corresponding probabilities, $p_T$ and $p_M$.

If the generated move improves the solution, it is accepted immediately. The probability of accepting a worsening move is calculated via Eq. (30) where $\delta$ denotes the cumulative weighted delta cost introduced by the move and $t_c$ is the current temperature.

$$p := e^{-\frac{\delta}{t_c}}$$  \hfill (30)

In order to compute $\delta$, we define the weighted cost of a solution $S$ as $C(S)$ Eq. (31), where $C(S)$ is the weighted sum of total tardiness and the makespan.
\[ C(S) := W \cdot \Sigma_j T_j(S) + C_{\text{max}}(S) \] (31)

\( W \) should be a constant that is sufficiently large to ensure the lexicographic order of the objectives (in our experiments we set \( W := 10^5 \)). Given a current solution \( S \) and a solution \( S' \), we then set \( \delta := C(S') - C(S) \).

Since improving moves are accepted unconditionally, only positive values for \( \delta \) can occur in this formula. It should be noted that higher values for \( \delta \) lead to lower values in the exponent and thus lower acceptance probabilities.

When the algorithm reaches its minimum temperature \( T_{\text{min}} \), a reheat occurs and its temperature is set to the initial temperature \( T_{\text{max}} \). Execution stops when a time limit is reached.

The parameters for this variant of Simulated Annealing are the following:

- \( T_{\text{max}} \): Initial Temperature
- \( T_{\text{min}} \): Minimum Temperature
- \( N_s \): Samples per Temperature
- \( \alpha \): Cooling Rate
- \( p_I \): Probability of generating inter-machine moves (as opposed to intra-machine moves)
- \( p_S \): Probability of generating shift moves (as opposed to swap moves)
- \( p_B \): Probability of generating block moves (as opposed to single-job moves)
- \( p_T \): Probability of applying tardiness guidance in the move generation
- \( p_M \): Probability of applying makespan guidance in the move generation
- \( B_{\text{max}} \): Maximum size of blocks

### 4.3.2 Simulated annealing with dynamic cooling (SA-C)

This variant tries to continually adapt its cooling rate in such a way that the minimum temperature is reached at the end of the algorithm’s time limit. Therefore, an estimate of how many iterations can still be done within the time limit is calculated after each iteration. Before every cooling step, the current cooling rate \( \alpha_t \) is computed according to Eq. (32), where \( x \) is the number of cooling steps left, \( T_{\text{min}} \) and \( t_c \) are the minimum and current temperature, respectively. Pseudo-code for this dynamic cooling procedure can be seen in Algorithm 4.

\[ \alpha_t := \sqrt{\frac{T_{\text{min}}}{t_c}} \] (32)

In order to minimise the time spent in high temperatures, we further apply a cut-off mechanic. This counts the number of accepted moves at every temperature and forces an immediate cooling step if a specified threshold \( N_a \) is exceeded.

#### Algorithm 4 Calculating the Cooling Rate

```plaintext
1: function COOLOFF(t_c, iterationsDone, elapsedTime, timeLeft)
2: \( t_c \leftarrow \frac{\text{iterationsDone}}{\text{elapsedTime}} \)
3: \( l \leftarrow t_c \cdot \text{timeLeft} \)
4: \( q \leftarrow \frac{T_{\text{min}}}{t_c} \)
5: \( \alpha \leftarrow \sqrt{\frac{l}{q}} \)
6: return \( t_c \cdot \alpha \)
7: end function
```

This threshold is often specified as ratio \( \rho = \frac{N_a}{N_s} \) and a typical value is 0.05. Overall, SA-C requires the same set of parameters as \( \text{SA-R} \), except for \( \alpha \), which is replaced by \( \rho \).

### 4.3.3 Simulated annealing with iteration budget (SA-I)

This variant uses a constant cooling rate to determine the temperature for each iteration. In addition to a time limit, it uses a fixed iteration budget \( \mathcal{I} \) to limit its run time. The idea is to choose the value for \( \mathcal{I} \) to estimate the possible number of iterations within the given time limit, based on the execution speed of sample iterations on the benchmark machine (in our experiments, we set \( \mathcal{I} \) to 185 \( \cdot \) \( L \), where \( L \) is the time limit in milliseconds). The provided iteration budget is split evenly over the temperatures, such that the minimum temperature is reached when its iteration budget is exhausted. As such, the number of samples per temperature is defined as the value of a function depending on the maximum and minimum temperature, which can be seen in Eq. (33).

\[ N_s := \frac{\mathcal{I}}{\log_{\alpha}(\frac{T_{\text{min}}}{T_{\text{max}}})} \] (33)

We apply the same cut-off mechanism as in SA-C. Since the cooling rate is adjusted differently than for SA-C, this may cause the temperature to fall below the actual minimum temperature. Due to \( N_s \) being calculated directly by \( \text{SA-I} \), it does not have to be provided as a parameter. However, the cooling rate \( \alpha \) has to be stated as a parameter for SA-I.

### 5 Instances

To evaluate the proposed approaches, we perform a large number of experiments with a set of randomly generated instances, a set of real-life instances, and instances on a related PMSP from the literature. In Sect. 5.1, we provide information on our instance generator, then we describe the set of randomly generated instances in Sect. 5.2. Later in Sect. 5.3, we present the real-life instances that have been provided to us by our industrial partner. Additional information on the problem instances from the literature are given in Sect. 5.4.
5.1 Instance generator

To describe the generation of instances we use the notion of materials, as our real-life data determines the setup times using materials instead of jobs. As every job processes a single material, converting a given material-based setup time matrix into a job-based setup time matrix is a simple preprocessing step. The main reason for this kind of specification of the setup times is to reduce space requirements of the instance files. A similar instance format has also been used previously in the literature (e.g. Caniyilmaz et al. 2015).

To create a large pool of instances we implement a random instance generator which is configured by parameters that specify the desired instance size (i.e. the number of machines, materials and jobs) as well as a random seed. This instance generator is based on a similar one proposed by Vallada and Ruiz (2011), but we extend it to include also the generation of machine eligibility constraints and due dates. The detailed pseudo-code is provided in Algorithm 5 (where parameters NoMach, NoMat, NoJbs are the number of machines, the number of materials, and the number of jobs).

The processing time of each job on each machine and the setup time between every pair of materials are drawn from uniform distributions [1, 100] and [1, 125], respectively. Setup times between two jobs sharing the same material are set to zero.

Exactly one material is assigned to each job as follows: At first, one job after the other gets matched to an unused material until every material has been assigned exactly once. Afterwards, a randomly selected material is assigned to any job that has not been matched to a material yet. Thus, no two jobs will process the same material if the number of jobs is lower or equal to the number of materials.

For every job, the corresponding set of eligible machines is determined in the following way: First, the eligible machine count $m$ is sampled from a uniform distribution $[1, M]$, where $M$ is the total number of machines. Then, the set of available machines is sampled $m$ times without replacement to determine the eligible machines for the job.

We further use three different procedures to assign due dates randomly to create different sets of benchmark instances.

In the first two procedures, the due dates are determined by constructing a reference solution for the problem and afterwards setting the scheduled completion time of each job as the corresponding due date. Thus, it is ensured by construction that a feasible schedule exists for every generated problem instance even though the generated reference solution may not be optimal with respect to makespan. The construction of such a reference solution consists of the following steps (see Algorithm 5, lines 35–46): First, we determine a random order of jobs in which we will schedule them one after the other. Then, for every job we randomly select one of its eligible machines according to an independently specified selection strategy. The job is then appended to the selected machine’s schedule and the job’s due date is set to its completion time based on the current schedule.

Our first due date generation procedure (S-style) constructs a solution as described above with the use of a random machine selection strategy.

The second due date generation procedure (T-style) also constructs a reference solution but aims to obtain tighter due dates through the use of an alternative greedy machine selection heuristic (see Algorithm 6). This alternative strategy greedily selects the machine that causes the lowest setup time when the corresponding job is scheduled.

The third due date generation procedure (P-style) does not rely on constructing a reference solution but assigns random due dates by sampling the values from a uniform distribution $[\tilde{C}_{\text{max}} \cdot (1 - \tau - R/2), \tilde{C}_{\text{max}} \cdot (1 - \tau + R/2)]$. The variables $\tau$ and $R$ in this case are parameters determining the tightness and variance of the generated due dates. For the calculation of the approximate makespan $\tilde{C}_{\text{max}}$, we use the formula suggested by Perez-Gonzalez et al. (2019) (see Eq. (34)). Similar approaches to randomly sample due dates have been used by Potts and Wassenhove (1982), Chen (2006) and Lee et al. (2013).

\[
\tilde{C}_{\text{max}} := \max_{j \in J} \left( \frac{\sum_{m \in E_j} (p_{j,m})}{|E_j|} + \frac{\sum_{i \in J \setminus j} (s_{i,j,m})}{n, \frac{n}{m}} \right)
\]  

5.2 Generated instances

Using the previously described instance generator, we generated 560 instances that can be separated into six different categories. The instances of each category have been generated with a differently configured random instance generator (see Table 2).

When every job in an instance is assigned to a different material (i.e. no pair of jobs use the same material), then the instance is classified as using Unique Materials. In this case, between any pair of jobs, the setup time is greater than zero. This is the case commonly found in the literature.

Instances with less materials than jobs are classified as using Shared Materials, because at least one material is shared between multiple jobs. As mentioned earlier, in such a case the setup time between pairs of jobs is zero, because no change in tooling is required. This reflects the structure observed in our real-life instances.

For the due date generation procedure that does not use a reference solution, we set the parameters $\tau = 0.25$ and $R = 0.5$, to generate instances that are unlikely to have zero-cost solutions with respect to tardiness.
Table 2  Number of generated instances per category

|                          | S-style due dates | T-style due dates | P-style due dates |
|--------------------------|-------------------|-------------------|-------------------|
| Unique materials         | 110               | 110               | 60                |
| Shared materials         | 110               | 110               | 60                |

Table 3  Ranges for specifics of generated instances

| Variable                | Distribution range | Remarks                           |
|-------------------------|--------------------|-----------------------------------|
| Number of machines      | [1, 30]            | Max. $\frac{1}{2} \cdot |Jobs|$ |
| Number of jobs          | [20, 1000]         | Step size: 20                     |
| Number of materials     | [1, |Jobs|)          | Only with shared materials        |

We sample the input parameters for our instance generator (i.e. the number of jobs, machines and materials) from the uniform distributions described in Table 3. For instances with shared materials, both the number of jobs and the number of materials are sampled separately from a uniform distribution. For instances with unique materials, the number of materials is set to be equal to the number of jobs. The bounds for the value ranges have been chosen to generate instances with realistic size.

As our metaheuristic approaches rely on a number of parameters, we use the automated parameter configuration tool SMAC to find efficient parameter configurations. To avoid over-fitting of the tuned parameters on the instances we use in our experiments, we split the 560 randomly generated instances into a training set for parameter tuning and a validation set that is used in our final experimental evaluation. To create our training set we uniformly sampled 90 S-style instances with shared materials and 90 S-style instances with unique materials. Similarly, we further sampled two sets of 90 instances from the T-style ones. Finally, we also sampled 40 P-style instances with shared materials and another 40 P-style ones with unique materials. Our training set therefore consists of 440 instances in total, while the validation set consists of the remaining 120 instances. Further details on the parameter tuning are given in Sect. 6.2.1.

The generated instances are publicly available at https://doi.org/10.5281/zenodo.4118241, and the instance generator is available on request.

5.3 Real-life instances

For testing purposes, our industrial partners provided us with three real-life instances (A, B and C) that represent planning scenarios from industrial production sites. The characteristics of these instances can be seen in Table 4.

We further created four additional instances by scaling up instance C (up to 16 times the original size). Additionally, we include a new instance, called C-assigned, that uses the same set of jobs and machines as instance C, but predetermines a machine assignment for each job. Similar to instance C, we scaled up instance C-assigned to create another four instances. The set of all real-life based instances used in our benchmark experiments can be seen in the first column of Table 13.

5.4 Instances from the literature

We evaluate the performance of our metaheuristics on instances for the PMSP variant that has been described by Perez-Gonzalez et al. (2019). The only difference to the problem we investigate is that their objective function only considers the minimisation of total tardiness and does not include the makespan. Since the makespan is the secondary objective in our problem and thus incomparably less important than total tardiness, we can directly use our metaheuristics to approach the problem from Perez-Gonzalez et al. (2019) by simply ignoring the makespan.

6 Computational results

To evaluate our approaches we performed a large number of experiments based on the sets of instances described in Sect. 5.1.

1 The instances we use in our experiments have been kindly provided to us by the authors.
Algorithm 5 Instance Generator

1: function GENERATEINSTANCE(NoMach, NoMat, NoJbs)  
2: Mach ← list from 1 to NoMach  
3: Mat ← list from 1 to NoMat  
4: Jbs ← list of jobs from 1 to NoJbs  
5:  
6: for i ← 1 until NoJbs do  
7: n ← UNIFORMRANDOM(1, NoMach + 1)  
8: E ← select n random elements from Mach  
9: p ← empty dictionary  
10: for all m ∈ E do  
11:   processingTime ← UNIFORMRANDOM(1, 100)  
12:   p.Add(m, processingTime)  
13: end for  
14: t ← select element from Mat  
15: Jbs[i].EligibleMachines ← E  
16: Jbs[i].processingTimes ← p  
17: Jbs[i].material ← t  
18: end for  
19:  
20: > Initialise empty setup time matrix  
21: ST ← (NoMat + 1) × (NoMat + 1) × (NoMach) matrix  
22: for all pred in Mat ∪ {0} do  
23:   for all succ in Mat ∪ {0} do  
24:     for all m ∈ Mach do  
25:       if pred = succ then  
26:         st = 0  
27:     else  
28:       st ← UNIFORMRANDOM(1, 125)  
29:     end if  
30:     ST[pred, succ, m] = st  
31:   end for  
32: end for  
33: end for  
34:  
35: S ← empty dictionary  
36: for all m ∈ Mach do  
37:   S.Add(m, empty schedule)  
38: end for  
39:  
40: > Determine the Due Dates  
41: for all j ∈ SHUFFLE(Jobs) do  
42:   m ← SELECTMACHINE(j)  
43:   S[m].append(j)  
44:   c ← completion time of j on S[m]  
45:   j.dueDate = c  
46: end for  
47:  
48: return NEWINSTANCE(Mach, Mat, Jbs, ST)  
49: end function

Algorithm 6 Greedy Machine Selection

1: function SELECTGREEDYMACHINE(j)  
2:  
3: m ← null  
4: for all e ∈ j.eligibleMachines do  
5:   i ← ID of last job on e or 0 if the schedule is empty  
6:   > ST as defined in Algorithm 5  
7:   t ← ST[i, j, m]  
8:   if (m = null) ∨ (t < s) then  
9:     s ← t  
10:   m ← e  
11: end if  
12: end for  
13: return m  
14: end function

6.1 Comparison of mixed-integer programming formulations

We implemented the MIP models described in Sect. 3 with Gurobi 8.1.1. Experiments with the MIP models were performed on a computer with an AMD Ryzen 2700X Eight-Core CPU and 16 GB RAM.

All eight models were evaluated on the 25 smallest generated instances from the validation set under a time limit of 1800 s per instance. Table 5 summarises the results (tardiness/makespan) for each instance.

It can be seen that the models that account for the machine eligibility requirements by a set of hard constraints (M2, M4, M6, and M8) are able to provide solutions for more instances than their counterparts (M1, M3, M5, M7) that set ineligible machines processing times to a makespan upper bound. In fact, models M2 and M6 are able to provide solutions for all 25 randomly generated instances, while M4 and M8 can provide solutions for all instances but one. Interestingly, the solutions found by M1 exhibit often higher quality than the corresponding solutions found by M2, the same is true for M7 and M8 which are similar to M1 and M2, but differ only in the formulation of constraint set (29). If we compare the results produced by M1 and M7, we see that the solution quality is only slightly different for most instances. However, it seems that M1 reaches overall best results more often than M7. Similarly, we can see that M2 produces slightly more feasible results than its counterpart M8, although M8 produces a better solution quality for some instances and can reach overall best results for four instances. If we compare results produced by models M3 and M5 with their counterparts M4 and M6 we can clearly see that M4 and M6 produce better results for the majority of the instances. The novel model adaptation that we described in Sect. 3 (M6) is able to find the best solutions for 14 of the randomly generated instances. Nine of these instances are P-style, four are S-style and one is a T-style instance. M4 provides the best solution two times for P-style instances, three times for S-style ones and four times for T-style ones. Further, M4 is able to prove optimality for two solutions, while M6 is able to find one optimal solution. Overall, M4 and M6 provide the best solutions for 22 randomly generated instances.

Tables 6 and 7 show initial and final dual bounds obtained by the MIP solver for each instance. We can see M4 and M6 produce the best bounds for most instances, indicating that these two formulations often lead to a tighter linear relax-
| Instance | M1          | M2          | M3          | M4          | M5          | M6          | M7          | M8          |
|----------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| A        | 298,024/1,091,223 | 76,593/1,091,223* | – | 76,593/1,091,223* | 76,593/1,091,223* | 1,057,056/1,355,456 | 76,593/1,091,223* |
| B        | 68,907,330/3,128,548 | 14,414,806/3,128,548 | – | 15,328,158/3,128,548 | – | 15162322/3,128,548 | – | 14,820,358/3,128,548 |
| C-x16    | – | – | – | – | – | – | – | – |
| C-x2     | – | – | – | – | – | – | – | – |
| C-x4     | – | – | – | – | – | – | – | – |
| C-x8     | – | – | – | – | – | – | – | – |
| C        | – | 13,309,200/6,796,800 | – | 12,884,400/2,671,200 | – | 12,884,400/2,581,200 | – | – |
| C-assigned-x16 | – | – | – | – | – | – | – | – |
| C-assigned-x2 | – | – | – | 40,485,600/10,512,000 | – | 47,937,600/10,490,400 | – | 113,202,000/10,468,800 |
| C-assigned-x4 | – | – | – | – | – | – | – | – |
| C-assigned-x8 | – | – | – | – | – | – | – | – |
| C-assigned | – | – | – | 12,884,400/5,094,000* | – | 12,884,400/5,094,000* | – | – |
| P_13-80-80_1 | 5388/601 | 10,015/879 | – | 2134/520 | 4234/673 | 1327/471 | 5207/718 | 7935/952 |
| P_15-60-60_1 | 1096/379 | 1349/341 | 1480/406 | 964/427 | 1323/430 | 932/371 | 962/385 | 1192/364 |
| P_15-63-80_1 | 4156/568 | 8901/918 | – | 1900/496 | 1369/496 | 2960/495 | 1560/456 | 7053/738 |
| P_16-100-100_1 | 12,208/838 | 12,412/997 | – | 3736/545 | 5359/597 | 2917/523 | 23,087/1745 | 11,750/925 |
| P_16-180-180_1 | – | 60,749/1939 | – | 78,940/2555 | – | 4817/1638 | – | 42,226/1669 |
| P_17-100-100_1 | 11,384/729 | 11,783/726 | – | 5122/475 | 7097/588 | 3649/532 | 8290/655 | 9329/624 |
| P_18-80-80_2 | 4393/457 | 5815/669 | 1787/372 | 2459/427 | 2323/368 | 1649/401 | 694/684 | 6963/725 |
| P_20-180-180_1 | – | 49,713/2046 | – | 64,061/2159 | – | 34,619/1457 | – | – |
| P_22-140-140_1 | – | 24,091/1226 | – | 11,871/683 | – | 11,655/677 | – | 13,689/783 |
| P_29-140-140_1 | – | 14,375/657 | – | 9947/664 | – | 7130/464 | – | 12,079/705 |
| P_3-17-20_1 | 927/520 | 1029/576 | 957/611 | 927/520 | 927/520 | 1029/576 | 928/529 | 928/529 |
| P_7-19-40_1 | 1955/514 | 2204/497 | 1976/453 | 2173/472 | 2419/529 | 2237/503 | 2008/465 | 2174/541 |
| P_9-180-180_1 | – | 13,886/3339 | – | 80,059/2784 | – | 78,859/2645 | – | 94,774/2721 |
| Instance           | M1             | M2             | M3              | M4              | M5              | M6              | M7              | M8              |
|--------------------|----------------|----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| S_1-3-100_1        | 35,660/7664    | 35,660/7664    | 34,442/7589     | 34,442/7589     | 56,919/8097     | 56,919/8097     | 42,255/7708     | 42,255/7708     |
| S_10-120-180_1     | –              | 50,869/2467    | –               | 62,029/2998     | –               | 33,756/2427     | –               | 34,396/2261     |
| S_15-80-80_1       | 982/688        | 6108/999       | –               | 94/859          | 71/721          | 59/734          | 495/702         | 4176 / 821      |
| S_15-80-80_3       | 1084/944       | 1689/871       | 110/649         | 0/368           | 2/961           | 0/322           | 5/734           | 1612/752        |
| S_22-149-160_1     | –              | 47,583/2483    | –               | 6467 / 1072     | –               | 14,552/1383     | –               | 16,292/1328     |
| S_4-16-20_1        | 0/409*         | 0/409*         | 0/409*          | 0/409*          | 0/409*          | 0/409*          | 0 / 409*        |
| T_10-24-40_1       | 0/373*         | 0/373*         | 32/512          | 0/373*          | 0/373*          | 47/456          | 0 / 373*        |
| T_15-77-80_1       | 659/700        | 5970/813       | –               | 14/644          | 617/751         | 38/614          | 1389/570        | 6268/813        |
| T_18-56-100_1      | 28,538/1536    | 11,365/1123    | –               | 1455 / 524      | 1870/662        | 1734/648        | –               | 6706/894        |
| T_20-76-100_1      | –              | 9036/647       | –               | 877/580         | 1283/573        | 1106/526        | –               | 5667 / 792      |
| T_28-34-100_1      | –              | 2545/468       | –               | 165/382         | –               | 23/478          | 2441/490        | 3842/445        |
| T_3-12-200_1       | 192,462/7190   | 300,069/8044   | –               | –               | –               | 192,624/6408    | –               | 212,963 /7316   |

Bold entries mark the best solution per instance. Asterisks mark proven optimal solutions. Dashes mark time-outs.
Table 6  Comparison of initial dual bounds found by MIP formulations. Bold entries mark the best bounds per instance

| Instance     | M1         | M2             | M3         | M4            | M5         | M6         | M7         | M8            |
|--------------|------------|----------------|------------|---------------|------------|------------|------------|---------------|
| A            | 1,088,864.0| 1,088,864.0    | 1,088,863.0| 1,088,863.0   | 1,088,864.0| 1,088,863.0| 1,088,864.0| 1,088,864.0   |
| B            | 0.0        | 0.0            | 2,294,640.0| 12,700,800.0  | 12,700,800.0| 2,294,640.0| 1,088,864.0| 1,088,864.0   |
| C            | 0.0        | 0.0            | 25,401,600.0| 25,401,600.0  | 25,401,600.0| 25,401,600.0| 25,401,600.0| 25,401,600.0  |
| C-x2         | 0.0        | 0.0            | 0.0        | 0.0           | 0.0        | 0.0        | 0.0        | 0.0           |
| C-x4         | 0.0        | 0.0            | 0.0        | 0.0           | 0.0        | 0.0        | 0.0        | 0.0           |
| C-x8         | 0.0        | 0.0            | 0.0        | 0.0           | 0.0        | 0.0        | 0.0        | 0.0           |
| C-x16        | 0.0        | 0.0            | 0.0        | 0.0           | 0.0        | 0.0        | 0.0        | 0.0           |
| C-assigned   | 0.0        | 0.0            | 4,986,939.0| 4,986,939.0   | 12,700,800.0| 0.0        | 0.0        | 0.0           |
| C-assigned-x2| 0.0        | 0.0            | 25,401,600.0| 25,401,600.0  | 25,401,600.0| 25,401,600.0| 25,401,600.0| 25,401,600.0  |
| C-assigned-x4| 0.0        | 0.0            | 50,803,200.0| 50,803,200.0  | 0.0        | 0.0        | 0.0        | 0.0           |
| C-assigned-x8| 0.0        | 0.0            | 101,606,400.0| 101,606,400.0| 0.0        | 0.0        | 0.0        | 0.0           |
| C-assigned-x16| 0.0       | 0.0             | 0.0        | 0.0           | 0.0        | 0.0        | 0.0        | 0.0           |
| P_3-17-20_1  | 0.0        | 0.0            | 0.0        | 0.0           | 0.0        | 0.0        | 0.0        | 0.0           |
| P_7-19-40_1  | 0.0        | 0.0            | 0.0        | 0.0           | 0.0        | 0.0        | 0.0        | 0.0           |
| P_9-180-180_1| 0.0        | 0.0            | 0.0        | 0.0           | 0.0        | 0.0        | 0.0        | 0.0           |
| P_13-80-80_1 | 0.0        | 0.0            | 0.0        | 0.0           | 0.0        | 0.0        | 0.0        | 0.0           |
| P_15-60-60_1 | 0.0        | 0.0            | 0.0        | 0.0           | 0.0        | 0.0        | 0.0        | 0.0           |
| P_15-63-80_1 | 0.0        | 0.0            | 0.0        | 0.0           | 0.0        | 0.0        | 0.0        | 0.0           |
| P_16-100-100_1| 0.0      | 0.0             | 0.0        | 0.0           | 0.0        | 0.0        | 0.0        | 0.0           |
| P_16-180-180_1| 0.0     | 0.0             | 0.0        | 0.0           | 0.0        | 0.0        | 0.0        | 0.0           |
| P_17-100-100_1| 0.0     | 0.0             | 0.0        | 0.0           | 0.0        | 0.0        | 0.0        | 0.0           |
| P_18-80-80_2 | 0.0        | 0.0            | 0.0        | 0.0           | 0.0        | 0.0        | 0.0        | 0.0           |
| P_20-180-180_1| 0.0    | 0.0             | 0.0        | 0.0           | 0.0        | 0.0        | 0.0        | 0.0           |
| P_22-140-140_1| 0.0    | 0.0             | 0.0        | 0.0           | 0.0        | 0.0        | 0.0        | 0.0           |
| P_29-140-140_1| 0.0    | 0.0             | 0.0        | 0.0           | 0.0        | 0.0        | 0.0        | 0.0           |
| S_1-3-100_1  | 0.0        | 0.0            | 0.0        | 0.0           | 0.0        | 0.0        | 0.0        | 0.0           |
| S_4-16-20_1  | 340.0      | 340.0          | 340.0      | 340.0         | 340.0      | 340.0      | 340.0      | 340.0         |
| S_10-120-180_1| 0.0     | 0.0             | 0.0        | 0.0           | 0.0        | 0.0        | 0.0        | 0.0           |
| S_15-80-80_2 | 0.0        | 0.0            | 0.0        | 0.0           | 0.0        | 0.0        | 0.0        | 0.0           |
| S_15-80-80_3 | 0.0        | 0.0            | 0.0        | 0.0           | 183.0099   | 0.0        | 182.9401   | 0.0           |
| S_22-149-160_1| 0.0   | 0.0             | 0.0        | 0.0           | 0.0        | 0.0        | 0.0        | 0.0           |
| T_3-12-200_1 | 0.0        | 0.0            | 0.0        | 0.0           | 0.0        | 0.0        | 0.0        | 0.0           |
| T_10-24-40_1 | 229.0454   | 225.803        | 0.0        | 222.2626      | 229.0454   | 0.0        | 229.0454   | 229.0454      |
| T_15-77-80_1 | 0.0        | 0.0            | 0.0        | 0.0           | 0.0        | 0.0        | 0.0        | 0.0           |
| T_18-56-100_1| 0.0        | 0.0            | 0.0        | 0.0           | 0.0        | 0.0        | 0.0        | 0.0           |
| T_20-76-100_1| 0.0        | 0.0            | 0.0        | 0.0           | 0.0        | 0.0        | 0.0        | 0.0           |
| T_28-34-100_1| 0.0        | 0.0            | 0.0        | 0.0           | 0.0        | 0.0        | 0.0        | 0.0           |

Dashes mark no bounds being found

ation. However, other formulations are able to provide better bounds in some cases.

To calculate the relative performance, we use the Relative Percentage Deviation (RPD) for every instance \( I \) and solution \( S \) as defined in Eq. (35). RPD values have also been used as a performance measure in publications on related problems (e.g. Vallada and Ruiz 2011).

\[
\text{RPD}_{I,S} := \frac{\text{cost}_{I,S} - \text{best}_I}{\text{best}_I} \tag{35}
\]

Figure 3 visualises the RPD values of each model in the form of box plots. In Fig. 3a, results for all models except for M3 are presented. Figure 3b shows RPD values of models with explicit machine eligibility constraints and Fig. 3c shows RPD values for models M4, M5 and M6. Model M3 is excluded from the comparisons, as it was unable to solve
Table 7  Comparison of best dual bounds found by MIP formulations within the time limit

| Instance | M1    | M2          | M3          | M4          | M5          | M6          | M7          | M8          |
|----------|-------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| A        | 132,156.0 | 1,091,223.0 | -           | 1,091,223.0 | 1,091,223.0 | 1091,223.0  | 108,400.0   | 1,091,223.0 |
| B        | 139,228.0 | 12,720.0    | -           | 12,720.0    | 0.0         | 12,720.0    | 11,878,016.0| 12720.0     |
| C        | -     | 12,700,800.0 | -           | 2,294,640.0 | 12,700,800.0| 2,309,760.0| -           | 12,700,800.0|
| C-x2     | -     | -           | 25,401,600.0| 25,401,600.0| 25,401,600.0| -           | -           | -           |
| C-x4     | -     | -           | -           | -           | -           | -           | -           | -           |
| C-x8     | -     | -           | -           | -           | -           | -           | -           | -           |
| C-x16    | -     | -           | -           | -           | -           | -           | -           | -           |
| C-assigned | -   | -           | -           | 5,094,000.0 | 12,700,800.0| 5,094,000.0| -           | -           |
| C-assigned-x2 | - | -         | 25,790,400.0| 25,401,600.0| 257904000.0 | -           | 9,626,400.0| -           |
| C-assigned-x4 | - | -         | -           | 50,803,200.0| 50,803,200.0| -           | -           | -           |
| C-assigned-x8 | - | -         | -           | 101,606,400.0| 101,606,400.0| -           | -           | -           |
| C-assigned-x16 | - | -        | -           | -           | -           | -           | -           | -           |
| P_3-17-20_1 | 148.0 | 128.0      | 142.0       | 148.0       | 125.0       | 140.0       | 173.0       | 166.0       |
| P_7-19-40_1 | 154.0 | 172.0      | 148.0       | 146.0       | 127.0       | 128.0       | 182.0       | 189.0       |
| P_9-180-180_1 | 31.0 | 47.0       | 0.0         | 13.0       | 16.0        | 13.0        | 43.0        | 62.0        |
| P_15-60-60_1    | 179.0 | 213.0      | 111.0       | 124.0       | 121.0       | 153.0       | 175.0       | 172.0       |
| P_15-63-80_1    | 19.0  | 24.0       | 0.0         | 9.0         | 10.0        | 10.0        | 14.0        | 22.0        |
| P_16-100-100_1  | 0.0   | 11.0       | 0.0         | 0.0         | 0.0         | 0.0         | 0.0         | 26.0        |
| P_16-180-180_1  | 0.0   | 0.0        | 0.0         | 0.0         | 0.0         | 0.0         | 0.0         | 0.0         |
| P_17-100-100_1  | 0.0   | 81.0       | 0.0         | 66.0        | 66.0        | 66.0        | 0.0         | 146.0       |
| P_18-80-80_2    | 56.0  | 70.0       | 0.0         | 0.0         | 0.0         | 64.0        | 0.0         | 71.0        |
| P_20-180-180_1  | -    | 0.0        | 0.0         | 0.0         | 0.0         | 0.0         | 0.0         | 0.0         |
| P_22-140-140_1  | 0.0   | 0.0        | 0.0         | 0.0         | 0.0         | 0.0         | 0.0         | 0.0         |
| P_29-140-140_1  | 0.0   | 0.0        | 0.0         | 0.0         | 0.0         | 0.0         | 0.0         | 0.0         |
| S_1-3-100_1    | 0.0   | 0.0        | 0.0         | 0.0         | 0.0         | 0.0         | 0.0         | 0.0         |
| S_4-16-20_1    | 409.0 | 409.0      | 409.0       | 409.0       | 409.0       | 409.0       | 409.0       | 409.0       |
| S_10-120-180_1 | 0.0   | 0.0        | 0.0         | 0.0         | 0.0         | 0.0         | 0.0         | 0.0         |
| S_15-80-80_2   | 0.0   | 0.0        | 0.0         | 0.0         | 0.0         | 0.0         | 0.0         | 0.0         |
| S_15-80-80_3   | 0.0   | 0.0        | 0.0         | 192.0       | 0.0         | 193.0       | 0.0         | 0.0         |
| S_22-149-160_1 | 0.0   | 0.0        | 0.0         | 0.0         | 0.0         | 0.0         | 0.0         | 0.0         |
| T_3-12-200_1   | 373.0 | 373.0      | 373.0       | 373.0       | 373.0       | 0.0         | 373.0       | 373.0       |
| T_10-24-40_1   | 373.0 | 373.0      | 373.0       | 373.0       | 373.0       | 0.0         | 373.0       | 373.0       |
| T_15-77-80_1   | 0.0   | 0.0        | 0.0         | 0.0         | 0.0         | 0.0         | 0.0         | 0.0         |
| T_18-56-100_1  | 0.0   | 0.0        | 0.0         | 0.0         | 0.0         | 0.0         | 0.0         | 0.0         |
| T_20-76-100_1  | 0.0   | 0.0        | 0.0         | 0.0         | 0.0         | 0.0         | 0.0         | 0.0         |
| T_28-34-100_1  | 0.0   | 0.0        | 0.0         | 0.0         | 0.0         | 0.0         | 0.0         | 0.0         |

Bold entries mark the best bounds per instance. Dashes mark no bounds being found.

most instances. Only those instances that could be solved by all compared models within the time limit are included in the plots to avoid missing values.

Overall, we conclude that models M4 and M6 provide the best results for the majority of instances in our experiments. The best model depends on the instance characteristics: For T-style instances, M4 shows the best performance, while M6 produces the best results for P-style instances.

### 6.2 Comparison of metaheuristic approaches

Both parameter tuning and benchmark experiments for the metaheuristics have been executed on a computer with an Intel Xeon E5-2650 v4 12-Core processor that has 24 logical cores and 252 gigabytes of RAM.
6.2.1 Parameter tuning

As our methods include various parameters which have an impact on the final results, we performed automated parameter tuning using Sequential Model-based Algorithm Configuration (SMAC) as proposed by Hutter et al. (2011).

For every Simulated Annealing variant, we started 24 parallel SMAC runs with a wallclock time budget of 18 h per run. The initial configurations used for the starting point of SMAC are based on intuition about plausible parameter values. Table 8 shows the value ranges for all parameters as well as their initial values. The tuned parameters proposed by SMAC are listed in Table 9.

To investigate the robustness of the SA variants against different configurations, we conducted additional experiments for the so-called incumbent configurations (i.e. the best found configurations) of each parallel SMAC run. We let every SA variant run 20 times with every incumbent configuration on the validation set. Box plots for the resulting RPD values can be seen in Fig. 4. It can be seen that SA-I and SA-R (on the left and right thirds of the plot, respectively) are much more robust against changes in the configuration than SA-C. Figure 5 shows the RPD values for all incumbents, grouped per SA variant for a clearer visual representation. Inspection of the plot scales shows that SA-C and SA-R are the least and most robust of the compared SA variants, respectively. Figure 5a, c shows that most configurations result in a very similar performance for both SA-I and SA-R. However, some configurations lead to significantly better results than the others. This suggests that both SA-I and SA-R are largely robust against changes in the configuration, but still have the potential for some optimisation.
6.2.2 Results on randomly generated instances

In this section, we present the results produced by the proposed Constructive Heuristic (Algorithm 1) and Simulated Annealing variants. For each of the investigated Simulated Annealing variants, we performed experiments with a randomly generated initial solution (R) and initial solutions produced by our Constructive Heuristic (CH). All experiments were performed within a time limit of 60 s per run regardless of the instance size. For SA-I, we set an iteration budget of $11,100,000$ iterations, as this corresponds to the average number of iterations that the other Simulated Annealing variants performed within the 60 seconds time limit.

Tables 10, 11 and 12 show the detailed experimental results for all SA variants. Fig. 6 further visualises an overview of the relative algorithm performances on the entire validation set as box plots. As expected, all Simulated Annealing variants are able to significantly improve the quality of their initial solution (Algorithm 1). Further, we can see in Fig. 6b that SA starting from a good initial solution produces significantly lower mean RPDs than SA starting from a random initial solution. Figure 6c shows box plots over the median RPD values per instance for each algorithm. The fact that the median RPD values are not notably different from the mean RPD values for SA (CH) indicates its robustness.

Comparing SA variants with random initial solution, we conclude that SA-R shows a slightly better performance compared to the other two SA variants. We further observe that all SA variants produce similar results when starting from a greedily constructed initial solution.

6.2.3 Results on real-life instances

To evaluate our approaches on the real-life instances, we use the same computational environment and time limits as in Sects. 6.1 and 6.2. Table 13 shows the best solution costs...
obtained by MIP (across all models) and the median absolute solution costs provided by each SA variant. All methods obtained the same solution costs for instances A and C-assigned, which also could be solved to optimality by MIP. For the rest of the instances, SA produced better results than MIP in our experiments. All SA variants achieved the same result in all 20 runs for four instances (A, B, C-restricted and C-restricted x2). Only for the larger instances we observe varying solution qualities. SA-R exhibits a slightly better performance on the larger instances (C-x8, C-x16 and C-assigned-x8), while SA-I performs best on smaller instances (C-x2 and C-x4). The box plots in Fig. 7 show that all SA variants perform very similarly, with RPDs close to zero.

6.2.4 Influence of the features

To investigate the influence of block moves and guidance strategies on the performance of the metaheuristics, we performed additional experiments with different configurations on the validation set. We created further configurations for each SA variant by changing the value for a single feature in the configuration while leaving everything else fixed. The derived configurations are the following:

- **Standard** The configuration proposed by SMAC
- **X% Blocks** The probability for generating block moves $p_B$ is set to $0\%$, $10\%$, $20\%$, and $30\%$, corresponding to $p_B = 0.0$, $0.1$, $0.2$, and $0.3$, respectively.
- **G-T** The probability for applying guidance towards minimising makespan $p_M$ is set to zero
- **G-M** The probability for applying guidance towards minimising tardiness $p_T$ is set to zero
- **G-None** Both guidance probabilities ($p_M$ and $p_T$) are set to zero

The RPD values for all three SA variants over the validation instances can be seen in Fig. 8. For SA-I, any modification of the standard configuration leads to performance degradation as can be seen in Fig. 8a. For SA-C and SA-R, the results are not as clear (see Fig. 8b, c). In the case of SA-C, block moves apparently have the most negative impact, as increases in block move probabilities lead to increases in RPD values. SA-R loses some performance when the guidance is reduced in any way but does not perform significantly different with higher probabilities for block moves.

Overall, there is no configuration for any of the SA variants that outperforms the standard configuration. Thus, we conclude that a limited amount of guidance towards minimisation of both makespan as well as tardiness is favourable.
The lowest values are highlighted in bold.
Table 11  Median results for 
*T-style* instances in the 
validation set

| Instance     | SA-I (CH) | SA-I (R) | SA-C (CH) | SA-C (R) | SA-R (CH) | SA-R (R) |
|--------------|-----------|----------|-----------|----------|-----------|----------|
| 1-73-620_1   | 0/35794   | 0/37,602 | 0/37,505  |          |           |          |
| 10-230-700_1 | 1/3246    | 2/3137   | 36/3444   |          |           |          |
| 10-24-40_1   | 18/319    | 68/319   | 18/319    |          |           |          |
| 10-389-940_1 | 45/4996   | 234/4929 | 234/4929  |          |           |          |
| 11-400-400_1 | 1/3246    | 36/3813  | 55/3126   | 167/3100 | 2/3137    | 68/3444  |
| 12-351-1000_1| 18/319    | 351/341  | 63/341    | 68/319   | 18/319    |          |
| 13-940-940_1 | 45/4996   | 156/4706 | 156/4706  |          |           |          |
| 14-520-520_1 | 18/319    | 167/3100 | 167/3100  |          |           |          |
| 14-680-680_1 | 18/319    | 222/3120 | 222/3120  |          |           |          |
| 15-400-400_1 | 18/319    | 68/319   | 68/319    |          |           |          |
| 15-740-740_2 | 0/2342    | 364/2755 | 0/2209    | 18/319   | 0/2342    | 90/2989  |
| 15-77-80_1   | 0/317     | 0/337    | 0/335     | 0/334    | 0/317     | 90/2989  |
| 17-520-520_1 | 18/319    | 18/319   | 18/319    |          |           |          |
| 18-56-100_1  | 64/318    | 45/318   | 45/318    | 10/316   | 64/318    | 45/318   |
| 18-820-820_2 | 45/2557   | 222/3120 | 222/3120  |          |           |          |
| 19-900-900_1 | 18/319    | 18/319   | 18/319    |          |           |          |
| 20-76-100_1  | 0/1081    | 31/1597  | 31/1597   |          |           |          |
| 22-471-480_1 | 130/864   | 222/3120 | 222/3120  |          |           |          |
| 22-760-760_1 | 18/319    | 18/319   | 18/319    |          |           |          |
| 24-280-280_1 | 4/508     | 11/558   | 11/558    | 0/512    | 4/508     | 11/558   |
| 24-400-400_1 | 34/663    | 34/663   | 34/663    | 0/512    | 34/663    | 34/663   |
| 25-352-680_1 | 5/1061    | 189/1463 | 189/1463  |          |           |          |
| 26-220-240_1 | 0/388     | 0/452    | 0/452     | 0/400    | 0/388     | 0/452    |
| 26-334-480_1 | 49/720    | 45/809   | 45/809    | 52/1030  | 49/720    | 45/809   |
| 26-540-540_1 | 10/787    | 26/864   | 26/864    | 10/837   | 10/787    | 26/864   |
| 27-360-360_1 | 22/544    | 65/537   | 65/537    | 0/512    | 22/544    | 65/537   |
| 27-540-540_1 | 35/784    | 35/784   | 35/784    | 0/512    | 35/784    | 35/784   |
| 28-189-240_1 | 0/384     | 0/419    | 0/419     | 0/379    | 0/384     | 0/419    |
| 28-34-100_1  | 0/231     | 0/231    | 0/231     | 0/231    | 0/231     | 0/231    |
| 28-371-520_1 | 0/714     | 110/858  | 110/858   | 0/741    | 110/858   | 110/858  |
| 28-620-620_1 | 2/980     | 72/1266  | 72/1266   | 135/788  | 2/980     | 72/1266  |
| 29-137-340_1 | 21/454    | 52/521   | 52/521    | 54/423   | 21/454    | 52/521   |
| 3-12-200_1   | 2/3133    | 107/3138 | 107/3138  | 39/341   | 2/3133    | 107/3138 |
| 30-580-580_1 | 16/929    | 175/838  | 175/838   | 16/929   | 16/929    | 175/838  |
| 6-540-540_1  | 19/4516   | 54/423   | 54/423    | 54/423   | 19/4516   | 54/423   |
| 6-860-860_1  | 92/9345   | 45/7966  | 45/7966   | 78/7911  | 92/9345   | 45/7966  |
| 8-3-260_1    | 178/1229  | 221/1182 | 221/1182  | 220/1231 | 178/1229  | 221/1182 |

The lowest values are highlighted in bold.
| Instance        | Instance | SA-I (CH) | SA-I (R) | SA-C (CH) | SA-C (R) | SA-R (CH) | SA-R (R) |
|-----------------|----------|-----------|----------|-----------|-----------|-----------|-----------|
| 1-364-580_1     | 0/4138  | 0/4148t   | 0/3282   | 0/32504   | 0/33940   | 0/34177   |
| 1-829-860_1     | 3314/62,953 | 15,353/62,253 | 5432/51146 | 8441/50725 | 4194/60,661 | 7716/57,869 |
| 10-376-680_1    | 418/4076 | 473/3755  | 527/3270  | 582/3154  | 438/3590  | 667/3550  |
| 13-760-760_1    | 710/3583 | 934/3467  | 749/2806  | 701/2730  | 698/3157  | 776/2950  |
| 13-80-80_1      | 665/359  | 618/377   | 436/359   | 414/334   | 357/341   | 423/370   |
| 15-60-60_1      | 548/369  | 536/386   | 600/369   | 515/424   | 776/2950  | 540/391   |
| 15-640-640_1    | 340/2607 | 281/2536  | 298/2034  | 350/1960  | 324/2598  | 324/2598  |
| 16-100-100_1    | 350/335  | 492/359   | 314/334   | 599/326   | 321/336   | 354/327   |
| 20-180-180_1    | 418/4076 | 473/3755  | 527/3270  | 582/3154  | 438/3590  | 667/3550  |
| 20-340-340_1    | 1346/353 | 1213/350  | 1342/355  | 1110/330  | 1287/342  | 901/322   |
| 20-340-340_1    | 851/306  | 931/327   | 803/307   | 756/307   | 805/307   | 752/312   |
| 21-62-740_1     | 565/1656 | 551/1670  | 552/1337  | 546/1307  | 659/1783  | 544/1411  |
| 22-140-140_1    | 597/577  | 566/572   | 564/571   | 552/577   | 623/540   | 605/540   |
| 26-223-260_1    | 444/1929 | 641/1977  | 525/1591  | 458/1627  | 562/2190  | 543/785   |
| 27-268-860_1    | 456/317  | 681/324   | 474/333   | 431/318   | 457/315   | 375/341   |
| 29-140-140_1    | 680/685  | 830/686   | 621/794   | 769/791   | 709/799   | 664/796   |
| 30-13-560_1     | 113/1786 | 146/1925  | 104/1498  | 148/1462  | 134/1862  | 224/1434  |
| 30-148-220_1    | 808/576  | 662/615   | 823/598   | 749/605   | 565/719   | 884/709   |
| 30-332-340_1    | 512/1590 | 616/1621  | 443/1277  | 690/1258  | 430/1682  | 507/1363  |
| 30-52-420_1     | 665/307  | 546/305   | 630/307   | 742/310   | 814/307   | 634/287   |
| 30-13-560_1     | 658/685  | 830/686   | 621/794   | 769/791   | 709/799   | 664/796   |
| 30-148-220_1    | 710/3583 | 934/3467  | 749/2806  | 701/2730  | 698/3157  | 776/2950  |
| 3-17-20_1       | 430/932  | 487/781   | 485/776   | 436/768   | 503/759   | 543/785   |
| 4-620-620_1     | 444/1929 | 641/1977  | 525/1591  | 458/1627  | 562/2190  | 543/785   |
| 5-83-360_1      | 456/317  | 681/324   | 474/333   | 431/318   | 457/315   | 375/341   |
| 5-83-360_1      | 294/792  | 283/899   | 205/817   | 246/908   | 335/868   | 349/863   |
| 6-680-680_2     | 282/2214 | 705/2311  | 290/1743  | 910/1714  | 347/2056  | 553/2181  |
| 7-19-40_1       | 673/411  | 599/428   | 503/482   | 803/486   | 591/491   | 755/469   |
| 7-249-980_1     | 113/1786 | 146/1925  | 104/1498  | 148/1462  | 134/1862  | 224/1434  |
| 7-431-480_1     | 1346/353 | 1213/350  | 1342/355  | 1110/330  | 1287/342  | 901/322   |
| 9-180-180_1     | 1346/353 | 1213/350  | 1342/355  | 1110/330  | 1287/342  | 901/322   |
| 11-500-500_1    | 365/1656 | 551/1670  | 552/1337  | 546/1307  | 659/1783  | 544/1411  |
| The lowest values are highlighted in bold |
Fig. 6 RPD values for SA variants, with constructive heuristic (CH) and random initial solution (R)

over a purely unguided search. The block moves in their current implementation, however, do not provide a significant benefit and, if selected to often, are detrimental to the performance of SA-I and SA-C.

To further analyse the influence of block moves on the instance structure, we can look on only the results for the set of instances which use *unique materials* without including the results for *shared materials* instances and vice versa.

Figures 9, 10, and 11 show the results produced by different block move configurations using SA-I, SA-C, and SA-R separately for *unique material* instances and *shared material* instances.

For SA-I we can see that using no block moves still produces the best results for both *unique material* and *shared material* instances. However, for SA-C and SA-R the results show that for the *unique material* instances the 10% Blocks configuration produced the best results, whereas for *shared material* instances again the 0% Blocks configuration performs best.

### 6.2.5 Comparison to results from the literature

To assess the quality of our methods, we compare them to the state of the art approach that was proposed recently for the problem provided by Perez-Gonzalez et al. (2019).

Since the optimal solution for the majority of these instances has an objective function value of zero, we use the Relative Deviation Index (RDI, Eq. (36)) as performance measure instead of the RPD. Similar performance measures have been used in previous publications (e.g. Perez-Gonzalez et al. 2019).
We ran each of our SA variants 20 times on every instance with a time limit that is based on the run time formula used by Perez-Gonzalez et al. (2019). The parameters for each of our metaheuristics are set to the ones determined by SMAC. The results for the Clonal Selection Algorithms (CSA_t and CSA_f) described by Perez-Gonzalez et al. (2019) have been kindly provided to us by the authors. Publicly available CPU benchmarks\(^2\) show that the CPU used in our experiments is roughly 1.5 times faster than the Intel i7 7700 processor that was used by Perez-Gonzalez et al. (2019) with respect to single-thread performance. Therefore, we used a time limit of \(M \cdot N \cdot 20\) milliseconds per instance, where \(M\) is the number of machines and \(N\) is the number of jobs (Perez-Gonzalez et al. (2019) used a time limit of \(M \cdot N \cdot 30\) in their experiments).

For the majority of the instances (2469 small, 5058 medium and 5909 big), all runs of the Simulated Annealing variants and all runs of the Clonal Selection Algorithms produced solutions with the exact same solution cost. Figure 13 shows box plots for all evaluated algorithms on the remaining 1371 small, 702 medium and 91 big instances, respectively. In the aggregated results it can be seen that all SA variants outperform the CSA algorithms regarding solution quality for most of the instances. This observation is further supported by Mann–Whitney–Wilcoxon tests using a confidence level of 0.95 which show that the proposed SA variants produce significantly improved results than both CSA_t and CSA_f in our experiments.

\(\text{RDI}_{I,S} := \frac{\text{cost}_{I,S} - \text{best}_{I}}{\text{worst}_{I} - \text{best}_{I}}\)  

(36)

\(^2\) https://www.cpubenchmark.net/compare/Intel-i7-7700-vs-Intel-Xeon-E5-2650-v4/2905vs2797.
When looking at the best results per instance for all SA and both CSA variants, respectively, we observe that in 3291 small, 5548 medium and 5979 big instances, the best SA result has the same cost as the best CSA result. For 506 small, 208 medium and 21 big instances, the best SA result is better than the best CSA result. On the other hand, for 43 small and 4 medium instances, the best CSA result is better than the best SA result. Figure 12 shows the number of instances where CSA outperforms SA, grouped by the number of jobs per instance. We observe that the number of instances where CSA outperforms SA declines with instance size, which indicates that SA scales better with increasing instance size.
Detailed results of all our experiments are publicly available for download at https://doi.org/10.5281/zenodo.4284100.

7 Conclusions

In this paper, we have investigated several solution approaches for a novel variant of the Unrelated Parallel Machine Scheduling Problem. To approach the problem, we have proposed several Simulated Annealing-based metaheuristics that utilise different neighbourhood operators and have investigated variations of a mathematical formulation.

Based on a set of randomly generated and real-life instances, we performed a thorough evaluation of all investigated methods. Furthermore, we evaluated the performance
Fig. 13 Comparison of metaheuristics on literature instances

of the proposed metaheuristics on a related problem from the literature, and compared the results to the state-of-the-art on a set of existing benchmark instances. The experimental results show that Simulated Annealing is able to provide high-quality solutions within short run times and outperforms other approaches for the large majority of instances. Using our approach, we were further able to provide previously unknown upper bounds for a large number of benchmark instances from the literature.

For future work, it may be interesting to select Block Moves based on solution-specific information. For instance, a more sophisticated heuristic for the selection of job blocks could be employed by defining a distance measure between jobs and building blocks based on this measure. Additionally, it would be interesting to investigate new neighbourhood operators and hybrid algorithms for this problem.

Acknowledgements The financial support by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development and the Christian Doppler Research Association is gratefully acknowledged. Further, we would like to thank Paz Perez-Gonzalez for providing us with their problem instances and results for comparison.

Funding Open access funding provided by TU Wien (TUW).

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article’s Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article’s Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

Adan, J., Adan, I. J. B. F., Akcay, A., Van den Dobbelsteen, R., & Stokkermans, J. (2018). A hybrid genetic algorithm for paral-
hel machine scheduling at semiconductor back-end production. In ICAPS (pp. 298–302). AAAI Press.
Afzalirad, M., & Rezaeian, J. (2016). Resource-constrained unrelated parallel machine scheduling problem with sequence dependent setup times, precedence constraints and machine eligibility restrictions. Computers & Industrial Engineering, 98, 40–52.
Afzalirad, M., & Rezaeian, J. (2017). A realistic variant of bi-objective unrelated parallel machine scheduling problem with machine eligibility restrictions. Journal of Intelligent Manufacturing, 29(2), 423–437.
Allahverdi, A. (2015). The third comprehensive survey on scheduling problems with setup times/costs. European Journal of Operational Research, 246(2), 345–378.
Allahverdi, A., Ng, C. T., Cheng, T. C. E., & Kovalyov, M. Y. (2008). A comprehensive survey of scheduling problems with setup times. European Journal of Operational Research, 187(3), 985–1032.
Al-Salem, A. (2004). Scheduling to minimize makespan on unrelated parallel machines with sequence dependent setup times. Engineering Journal of the University of Qatar, 17(1), 177–187.
Arnaout, J.-P., Musa, R., & Rabadi, G. (2014). A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines—Part II: Enhancements and experimentalizations. Journal of Intelligent Manufacturing, 25(1), 43–53.
Arnaout, J.-P., Rabadi, G., & Musa, R. (2010). A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times. Journal of Intelligent Manufacturing, 21(6), 693–701.
Avalos-Rosales, O., Angel-Bello, F., & Alvarez, A. (2015). Efficient metaheuristic algorithm and re-formulations for the unrelated parallel machine scheduling problem with sequence and machine-dependent setup times. The International Journal of Advanced Manufacturing Technology, 76(9–12), 1705–1718.
Bektur, G., & Sarac, T. (2019). A mathematical model and heuristic algorithms for an unrelated parallel machine scheduling problem with sequence-dependent setup times, machine eligibility restrictions and a common server. Computers & OR, 103, 46–63.
Caniyilmaz, E., Benli, B., & Ikay, M. S. (2015). An artificial bee colony algorithm approach for unrelated parallel machine scheduling with processing set restrictions, job sequence-dependent setup times, and due date. The International Journal of Advanced Manufacturing Technology, 77(9), 2105–2115.
Chen, J.-F. (2006). Minimization of maximum tardiness on unrelated parallel machines with process restrictions and setups. The International Journal of Advanced Manufacturing Technology, 29(5), 557–563.
Du, J., & Leung, J.Y.-T. (1990). Minimizing total tardiness on one machine is NP-hard. Math. Oper. Res., 15(3), 483–495.
Fanjul-Peyro, L., Ruiz, R., & Perea, F. (2019). Reformulations and an exact algorithm for unrelated parallel machine scheduling problems with setup times. Computers & OR, 101, 173–182.
Garey, M. R., & Johnson, D. S. (1979). Computers and intractability: A guide to the theory of NP-completeness. San Francisco: W. H. Freeman.
Gedik, R., Kalathia, D., Egilmez, G., & Kirac, E. (2018). A constraint programming approach for solving unrelated parallel machine scheduling problem. Computers & Industrial Engineering, 121, 139–149.
Graham, R. L., Lawler, Lenstra, E. L. J. K., & Rinnooy Kan, A. H. G. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. In Annals of discrete mathematics (Vol. 5, pp. 287–326). Elsevier.
Helal, M., Rabadi, G., & Al-Salem, A. (2006). A tabu search algorithm to minimize the makespan for the unrelated parallel machines scheduling problem with setup times. International Journal of Operations Research, 3(3), 182–192.
Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In LION, volume 6683 of Lecture Notes in Computer Science (pp. 507–523). Springer.
Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. Science, 220(4598), 671–680.
Lee, J.-H., Jae-Min, Yu., & Lee, D.-H. (2013). A tabu search algorithm for unrelated parallel machine scheduling with sequence-and machine-dependent setups: minimizing total tardiness. The International Journal of Advanced Manufacturing Technology, 69(9–12), 2081–2089.
Perez-Gonzalez, P., Fernandez-Viagas, V., Garcia, M. Z., & Framinan, J. M. (2019). Constructive heuristics for the unrelated parallel machines scheduling problem with machine eligibility and setup times. Computers & Industrial Engineering, 131, 131–145.
Potts, C. N., & Van Wassenhove, L. N. (1982). A decomposition algorithm for the single machine total tardiness problem. Operations Research Letters, 1(5), 177–181.
Rabadi, G., Moraga, R. J., & Al-Salem, A. (2006). Heuristics for the unrelated parallel machine scheduling problem with setup times. Journal of Intelligent Manufacturing, 17(1), 85–97.
Rambod, M., & Rezaeian, J. (2014). Robust meta-heuristics implementation for unrelated parallel machines scheduling problem with rework processes and machine eligibility restrictions. Computers & Industrial Engineering, 77, 15–28.
Santos, H. G., Toffolo, T. A. M., Silva, C. L. T. F., & Berghe, G. V. (2019). Analysis of stochastic local search methods for the unrelated parallel machine scheduling problem.ITOR, 26(2), 707–724.
Tran, T. T., Araujo, A., & Beck, J. C. (2016). Decomposition methods for the parallel machine scheduling problem with setups. INFORMS Journal on Computing, 28(1), 83–95.
Vallada, E., & Ruiz, R. (2011). A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. European Journal of Operational Research, 211(3), 612–622.
Ying, K.-C., Lee, Z.-J., & Lin, S.-W. (2012). Makespan minimization for scheduling unrelated parallel machines with setup times. Journal of Intelligent Manufacturing, 23(5), 1795–1803.

Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.