Integrating Memory-Based Perturbation Operators into a Tabu Search Algorithm for Real-World Production Scheduling Problems

Manuel Schlenkrich[1,2], Michael Bögl[2], Anna Gattinger[2], Ionela Knospe[2] and Sophie N. Parragh[1]
[1]*Johannes Kepler University Linz, Altenbergerstraße 69, 4040 Linz, Austria*
[2]*RISC Software GmbH, Softwarepark 32a, 4232 Hagenberg im Mühlkreis, Austria*

Keywords: Real-World Production Scheduling, Tabu Search, Perturbation, Destroy and Repair Operator, Elite Solutions.

Abstract: Production scheduling problems, arising in real-world use cases, are characterized by a very large number of operations and complex constraints. In order to handle such problems, practical solution approaches need to be generic enough, to capture all relevant restrictions, while being able to calculate good solutions in a short amount of time. Metaheuristic methods, especially combinations of trajectory and population-based approaches, are promising techniques to meet this criterion. In this work, we develop a framework for deriving and integrating memory-based perturbation operators into a highly flexible Tabu Search algorithm for scheduling problems, in order to enhance its overall performance. The perturbation operators are inspired by evolutionary algorithms and collect valuable solution information during the Tabu Search procedure via an elite solution pool. This information is used in a destroy-and-repair step integrated into the Tabu Search procedure, aiming to preserve promising solution structures. We investigate several parameters and perform computational experiments on job-shop benchmark instances from literature, as well as on a real-world industry use case. Integrating the developed memory-based perturbation operators into the Tabu Search algorithm leads to significant performance improvements on the real-world problem. The benchmark evaluations demonstrate the robustness of the approach, when dealing with sensitive parameters.

1 INTRODUCTION

The rise of Industry 4.0 across companies in various sectors, accompanied by the introduction of digital twins and automated decision making, has created a number of challenging planning problems and increased the need for practical solution approaches. Among those planning problems, scheduling of complex production systems, as they occur in semiconductor manufacturing, textile processing or the chemical industry, is one of the most demanding ones.

In the literature several different approaches to tackle production scheduling problems are available, such as the use of mixed integer linear programming, constraint programming, machine learning techniques or metaheuristic methods, to name the most popular choices. Even though a broad range of solution methods exists, the availability of approaches for problems of practical size and complexity is limited (Schlenkrich and Parragh, 2023). While some methods are rather restricted in the size of the problems, that they can solve in reasonable time, metaheuristic solution methods are often able to capture complicated constraints arising in real-world settings and can provide good solutions in a short amount of time.

In their work on real-life scheduling problems with rich constraints and dynamic properties, Bögl et al. (2021) give a profound overview of the most important aspects, that need to be considered when developing scheduling tools for real industry cases. They present a Tabu Search (TS) framework, making use of an activity list-based representation of the schedule, following the approach of Moumene and Ferland (2009). The TS framework is embedded in the production planning and execution software environment of our partner company and is currently used as the scheduling tool. The algorithm allows to handle very large problem instances up to tens of thousands of operations on a large variety of machines. However, with increasing problem size, also the required computation time to reach a solution of acceptable quality rises. Thus, enhancing the existing solution method, to counteract the growing computational burden, is of utmost importance and therefore the main goal of this work.

Schlenkrich, M., Bögl, M., Gattinger, A., Knospe, I. and Parragh, S.
Integrating Memory-Based Perturbation Operators into a Tabu Search Algorithm for Real-World Production Scheduling Problems.
DOI: 10.5220/0012271900003639
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 13th International Conference on Operations Research and Enterprise Systems (ICORES 2024)*, pages 213-220
ISBN: 978-989-758-681-1; ISSN: 2184-4372
Proceedings Copyright © 2024 by SCITEPRESS – Science and Technology Publications, Lda.

For this purpose, we extend the TS of Bögl et al. (2021), by deriving and integrating memory-based perturbation (MBP) operators to foster diversification within the trajectory-based metaheuristic method. We introduce a framework for deducing perturbation operators that modify solution candidates on the basis of a set of elite solutions. Promising solution patterns, that are shared among the elite solutions, are conserved, while the remaining part of the solution candidate is largely modified, in order to diversify the investigated solutions in the search space.

The main inspiration for our approach is coming from other available works, that investigate diversification strategies in different metaheuristic procedures, such as Adaptive Large Neighborhood Search (ALNS) (Ropke and Pisinger, 2006). It is an extension of Large Neighborhood Search (LNS) (Shaw, 1997). LNS and ALNS respectively, use large modifications that can rearrange a solution candidate by up to 30% or 40% to explore various regions of the search space. A similar core idea is used in Iterated Local Search (ILS), see e.g., Braune et al. (2007) for an application in the scheduling context. ILS is a trajectory-based metaheuristic method, that alternates intensification and diversification phases, via small and large step modifications respectively. The large modifications are reached by performing perturbation steps, that oftentimes rely on random destroy and repair operators.

While both approaches, ALNS and ILS, perform intensification and diversification steps iteratively, there also exist procedures that aim to foster these principles in two consecutive phases. In the work of Lunardi et al. (2021), four metaheurstics are tailored to a complex scheduling problem. The authors propose to combine different methods in order to create more powerful approaches and use the solution of one method as the starting point for a second one.

In our work we make use of the destroy and repair concept similar to ALNS and ILS. In contrast to ALNS however, we do not perform several large modifications of the solution candidate in a row, but use it as a one-time perturbation operator, if the Tabu Search procedure seems to be trapped in a local minimum. In contrast to ILS, we do not randomly destroy parts of the solution, but aim at preserving promising parts of the solution candidate, by collecting structural information on the best solutions found so far during the TS.

In the following, we first present the investigated scheduling problem in Section 2. Then, in Section 3 we describe our solution approach and Section 4 summarizes the numerical experiments. Finally, in Section 5 we conclude our work and provide future research directions.

2 PROBLEM DESCRIPTION

Within this section we describe the investigated production scheduling problem, which has also been addressed by Bögl et al. (2021). Since we propose an extension to their approach, we tackle the same underlying problem.

The task is to schedule a set of n jobs on a set of k machines, such that a given objective function is optimized. Each job j_i consists of $\hat{n}_i$ operations $\{o_{i1}, \ldots, o_{i\hat{n}_i}\}$. While machines have different attributes concerning their availability to process one or more operations at each given time, also operation-specific restrictions need to be considered.

For each operation to be scheduled, the following data is available: An operation cannot be scheduled before its associated earliest start time. It can finish later than its latest finish time, but this might influence the objective function value, in the case that activity tardiness is considered. Each operation must be assigned to a machine out of the set of compatible machines. For each of the compatible machines the consumed capacity, production duration and production costs are defined. After production the teardown duration for each activity needs to be respected. Setup times for activities are both machine- and sequence-dependent and need to be considered. For each production setup there is a machine dependent setup cost. Any operation has a set of predecessors, which must be finished before the activity itself can start, and a set of successors, which can only start after the activity itself is finished. These sets are not restricted to a single predecessor and successor, but can contain multiple ones. Also, these sets may contain operations that belong to other jobs than the activity itself. For activities that have a direct precedence relation, an overlap can be defined. Waiting times have to be considered after an operation is finished, however without occupying the assigned machines. Also, transportation times arise, when operations are transferred to another machine. Transportation times are machine dependent and are provided in a transportation time matrix. Additional machines might be needed in order to perform an operation. In this case an additional operation is created that needs to be scheduled simultaneously. In case that a partial schedule is already under execution, current start and end times of execution can be specified. Finally, several objective functions can be defined. Popular choices are minimizing the makespan, total weighted tardiness on activity level (TWT) or total weighted order tardiness (TWOT).

3 MEMORY-BASED PERTURBATION FRAMEWORK

TS is a trajectory-based metaheuristic method, iteratively evaluating solution candidates in order to find near-optimal solutions for optimization problems, such as production scheduling problems. Neighborhood operations, which are small modifications of the current solution, are applied to generate new candidates. The term ”tabu” in the name of the method refers to a list of forbidden neighborhood moves, the so called tabu list, which helps the algorithm avoid getting trapped in local optima. Tabu Search's strength lies in its ability to intensify the search for better solutions, rather than diversifying the search space. This can be advantageous in promising areas of the solution space, however might delay the search procedure in the case that the algorithm is stuck in an unfavorable area. In order to support diversification of trajectory based metaheuristics, several works propose combinations with population based approaches, see e.g. Lunardi et al. (2021).

We foster this concept of combining trajectory and population-based metaheuristics, but instead of sequentially performing different methods we directly integrate principles of evolutionary algorithms into an operator used within the TS framework. The idea is to diversify solution candidates, as soon as the speed of improvement falls below a certain level, meaning that the TS is not able to make use of its strong intensification mechanism. Perturbation operators are a common tool to diversify solution candidates. They perform a large modification of the current candidate, moving to another area in the solution space, which is possibly more promising. The problem with common perturbation operators is, however, that much of the effort already invested by the TS algorithm is lost, because the structure of the candidate is modified randomly. To counteract this issue, we develop a memory-based perturbation framework, that aims at preserving promising solution structures of the current solution, while only destroying non-promising parts in order to reach diversification. Information about promising structures is extracted from an elite solution pool, which is collected during the TS procedure. The solution pool is then evaluated by means of a configurable evaluation criterion, that detects shared properties or structures among the elite solutions. After that a parameterizable acceptance criterion decides, which of the solution characteristics are shared among enough of the elite solutions, in order to be classified as promising properties, that should be conserved for later iterations within the TS. In the following, the unpromising part of the candidate is removed, while the promising part is preserved, using the destroy-and-repair concept. After rearranging the removed part, the newly generated solution candidate is diversified, without destroying the most promising solution patterns.

Algorithm 1 presents the TS procedure using a first improvement implementation. It includes the mechanism to trigger a perturbation step, which could be either a random perturbation or one of the proposed MBP operators. The remainder of the work will focus on the perturbation step, while keeping the core components and parameters of the TS unchanged.

```
Data: Problem data, Tabu Search parameters,
      Perturbation parameters
Result: Best found solution
initial solution $s \leftarrow$ construction heuristic;
incumbent solution $s' \leftarrow s$;
best solution $s^* \leftarrow s$;
$nonImprovingIterations = 0$;
while termination criterion not met do
    $bestObjective = \infty$;
    $nonImprovingIterations++$;
    while maxNeighborhoodSize not reached
      do
        generate next non tabu neighbor $\hat{s}$ of
          $s'$;
        if $objective(\hat{s}) \leq bestObjective$ then
            $\bar{s} \leftarrow \hat{s}$;
            $bestObjective \leftarrow objective(\bar{s})$;
        end
        if $objective(\hat{s}) \leq objective(s')$ then
            $nonImprovingIterations = 0$;
            break;
        end
    end
    $s' \leftarrow \bar{s}$;
    update tabu list;
    if $objective(s') < objective(s^*)$ then
        $s^* \leftarrow s'$;
    end
    if $nonImprovingIterations \geq$
      $perturbationTriggerLimit$ then
        $s' \leftarrow$ PerturbationOperator($s'$);
        $nonImprovingIterations = 0$;
    end
end
Return $s^*$ as best found solution;
```

Algorithm 1: Tabu Search procedure in the ”first improvement” implementation including perturbation steps.

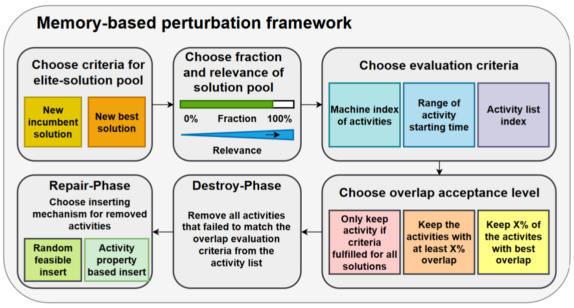
Memory-based perturbation framework
Choose criteria for elite-solution pool
New incumbent solution
New best solution
Choose fraction and relevance of solution pool
0%
Fraction
100%
Relevance
Choose evaluation criteria
Machine index of activities
Range of activity starting time
Activity list index
Choose overlap acceptance level
Only keep activity if criteria fulfilled for all solutions
Keep the activities with at least X% overlap
Keep X% of the activites with best overlap
Destroy-Phase
Remove all activities that failed to match the overlap evaluation criteria from the activity list
Repair-Phase
Choose inserting mechanism for removed activities
Random feasible insert
Activity property based insert

Figure 1: The Memory-Based Perturbation Framework.

There are several possibilities to construct a perturbation operator, that serves the purpose described above. We have developed a framework, which can be used to derive different operators, based on four important components, namely the design of the elite solution pool, choice of a suitable evaluation criterion, definition of an acceptance level and procedure for the destroy-and-repair phase. Figure 1 presents the developed MBP framework, which is further described in the following sections.

Elite Solution Pool. The elite solution pool is a collection of solution candidates obtained during the execution of the TS. Since the goal is to gather information on promising solution structures, the solution candidates selected for the elite solution pool are supposed to be of adequate quality. This could, for example, mean that every newly best found solution is added to the pool. Another option would be to select all newly found incumbent solutions, which are the solution candidates used by the TS to generate neighboring solutions. On top of the initial selection criterion for the elite solution pool, also a time-dependent criterion can be defined, restricting the elite solutions, based on the TS iteration in which they have been added to the set. This could either mean assigning higher relevance to more recent candidates or deleting earlier and therefore more outdated candidates.

Evalution Criteria. Once the relevant elite solution pool is defined, a criterion for evaluating the selected solution candidates need to be specified. This criterion determines the solution properties, that should be compared among the elite solutions in order to deduce promising patterns among them, meaning that several solutions of good quality share the same characteristic. There are numerous possibilities to define such an evaluation criterion, e.g., to compare the machine index of activities, the activity list index or the overlap of ranges around activity starting times. Through the highly modular approach, this criterion can be exchanged effortless, leading to different MBP operators.

Overlap Acceptance Level. After evaluating the solutions in the elite solution pool according to the chosen evaluation criteria, an overlap acceptance level needs to be specified. This acceptance level determines, if the chosen properties are shared among enough elite solutions, to be defined as promising solution patterns. These acceptance levels can again be defined and replaced in a highly flexible manner, leading to different operators. A very strict acceptance level would be to categorize solution patterns as promising, only if they are shared among all solutions in the elite solution pool. This criterion could be moderated by only demanding an overlap on a fraction of the elite solutions. An approach to keep the size of the preserved part of the solution stable, is to always categorize a fixed fraction of the solution as promising, namely the one with the highest overlap among the elite solutions. We will later refer to this approach by using a so called ”destroy percentile”, meaning that the percentile of the solution candidate with the lowest overlap will be destroyed, while the rest is preserved.

Destroy-and-Repair Phase. In the last phase of the MBP, the gathered information about high quality solutions and their shared promising structures is used to preserve those structures in the current solution, while diversifying the remaining parts. The so called destroy phase removes all activities, that do not meet the selected overlap acceptance criteria from the current solution. Those removed activities are then inserted into the perturbated candidate, using a configurable insertion mechanism, that could either be based on activity properties or inserts randomly. The result of the repair phase, is a new diversified solution candidate, that still contains the most promising patterns, identified in the elite solution pool.

4 NUMERICAL STUDY

In order to evaluate the impact of the proposed approach on the performance of the TS algorithm, which in its base version is currently used in the planning software of the company, we derive two types of MBP operators. The first one uses the activity list index evaluation criterion, meaning that elite solutions are compared concerning the positions of each activity in the encoded representation of the solution, namely the activity list. The second operator uses the machine index evaluation criterion, meaning that shared properties are evaluated by means of the position of each activity on the machines. Both operators store the new best found solutions within the elite solution pool and use the destroy percentile acceptance criterion.

We first test the TS algorithm on job-shop scheduling benchmark instances from the literature, where optimal solutions are known. Note, that these problems represent a simplified version of the real-world scheduling problem that is described in Section 2. See e.g., Adams et al. (1988) for more information about the job-shop scheduling problem. It is not our goal to compete with solution approaches and solvers, that are tailored to the job-shop scheduling problem, such as the CP Optimizer (Laborie et al., 2018). For our experiments on the benchmark instances, we performed optimization runs with a CPU time limit of 15 minutes per instance, which allows at most 10,000 to 50,000 iterations per instance, due to the overhead of the software framework. The aim of the experiments on the benchmark instances is to better understand the sensitivity of the approach towards parameters, such as the number of non-improving iterations before a perturbation is triggered or the size of the destroy percentile. In order to put the obtained results into perspective, we compare them to the recent approach of Yuan et al. (2023), who use deep reinforcement learning to learn effective policies for job-shop scheduling problems. We have chosen a learning-based approach as a comparison, because in the face of the complexity of our real-world scheduling problem, this technique would be a reasonable alternative to the currently used algorithm. In contrast, integrating a constraint programming solver into the software environment of the company was not a feasible option.

In a second step, we apply the TS algorithm to a real-world industry use case, which was provided by the partner company, in order to assess its capability to improve solution quality under realistic conditions. For all following experiments we chose an equal base setting for the TS, such as employed construction heuristics and available neighborhood operators. The length of the tabu list is 100 and the maximum neighborhood size for the operators is set to 300, to name a few of the most important parameters. All experiments were carried out on an Intel(R) Xeon(R) Gold 5315Y CPU @3.20GHz with 8GB RAM.

4.1 Job-Shop Benchmark Results

The job-shop benchmark instances consist of two separate test sets, with different objective functions. For instances of the first set, the makespan should be minimized, while for instances of the second test set, the total weighted tardiness (TWT) is to be minimized. The makespan test set consists of 153 instances and consists of five subsets, taken from different works in the literature, namely *abz* from Adams et al. (1988), *la* from Lawrence (1984), *orb* from Applegate and Cook (1991), *swv* from Storer et al. (1992) and *ta* from Taillard (1993). The number of jobs ranges from 10 to 100, while the number of machines ranges from 5 to 20, making the largest instance have 2000 operations. The TWT test set consists of 63 instances and is divided into four subsets, namely *abz*, *la*, *mt* and *orb*, which are all taken from Singer and Pinedo (1998). All instances consist of 10 jobs on 10 machines, resulting in 100 operations each.

As a base version we tested the TS algorithm without any perturbation of the solution. In a next step, we implemented a random perturbation operator with a fixed destroy percentile. This random operator removes a randomly chosen part of the solution candidate and inserts it back randomly. For our experiments we have chosen a destroy percentile of 30%, which turned out to be a good choice for random perturbation in earlier tests. A very sensitive parameter is the number of non-improving steps, after which the perturbation operator is called. For our numerical study we choose to investigate the settings 10 and 15 for this parameter. We have chosen these values, as we have seen that choosing a smaller value than 10 results in a search procedure, in which the solution is destroyed too often, and on the other hand, for values larger than 15 the solutions are very similar to the base version, since the perturbation is rarely called. It must be noted that the best setting for this parameter very much depends on the problem instances and that intensive parameter tuning is not always possible in practice. Perturbation operators that perform well, even for suboptimal choices of this parameter are therefore favorable.

In a next step we performed experiments using the TS with implementations of the MBP operators. The first derived operator uses the activity list index evaluation criterion. We performed test runs with parameter NonImproving set to 10 and 15 in order to compare the behavior to the random perturbation setting. Since we have observed similar results for the MBP operator in both settings, we decided to use the slightly better performing parameter setting of 15 for the final experiment, which was running the TS with the MBP operator, that uses the machine index evaluation criterion. For each of the derived perturbation operators we investigate three settings for the destroy percentile, namely 10%, 30% and 50%.

Table 1 presents the results for the makespan test instances, displaying the percentage deviation from the optimal makespan, averaged over the respective instance group. For the makespan instances we additionally report the results obtained by Yuan et al. (2023). Table 2 shows the results for the TWT instances, reporting the absolute deviation of the optimal tardiness in minutes, again averaged over the respective instance group.

Integrating the MBP operators into the TS leads to improvements on both instance sets, compared to the base version. The average deviation from the optimal makespan over all 153 test instances was reduced from 12,7% to 10,5%. In contrast, the random perturbation with the NonImproving parameter set to 15 led to nearly no improvement. With this parameter set to 10, the random perturbation even led to a worse makespan than the base version. This indicates that choosing the trigger level for the perturbation too low, in combination with large uncontrolled modifications, can harm the search procedure by jumping to another search area, without exploiting the full potential of the current region. Among the MBP operators, however, we observe rather stable results, meaning that even for a high number of perturbation calls, solution quality stays adequate. This is a practically highly relevant aspect, since those parameters cannot always be tuned in a real-world setting, where time is scarce.

On the second benchmark test set the absolute deviation of the TWT from the optimal value averaged over all 63 instances was reduced from 194 minutes to 182 minutes by applying the MBP operators. Also, for the TWT instances the results show, how sensitive the search procedure is to the NonImproving parameter. For a wrong choice of the parameter, as can be seen in the columns for a value of 10 in the random perturbation case, the results drastically deteriorate. Again, this can be reasoned by the fact, that solution candidates are modified too drastically, before new better solutions can be found by the neighborhood operators. This test set, however, also demonstrates, that for good choices of the NonImproving parameter, even the random perturbation can lead to better results, as it can be seen for the choice of 15.

4.2 Real-World Industry Case

Finally, we have the possibility to evaluate the proposed perturbation operators on a real-world industry case with 905 operations on 67 machines. It includes all the aspects and constraints described in Section 2. Activity deadlines, as well as order deadlines are available in the data, meaning that we can evaluate according to the two objectives, that are relevant for the company, namely total weighted tardiness (on activity level) and total weighted order tardiness (on order level). For each of the two objectives we perform test runs, using the TS without perturbation as the base version, with random perturbation, as well as with MBP operators using the activity list index and the machine index evaluation criteria. For the perturbation operators we again test 10%, 30% and 50% destroy percentiles. We have seen, that for the real-world use case strong diversification after a small number of unsuccessful iterations is even more crucial than for the benchmark instances, due to the smaller overall number of possible iterations. Therefore, we evaluate the values 2 and 10 for the number of non-improving steps before perturbation. We perform 100 iterations per approach and objective, resulting in a runtime of approximately 1 hour for each of the experiments. Tables 3 and 4 present the evaluation results on the real-world industry case for the TWT and TWOT objective, respectively. A more detailed representation of the search procedure and the objective value paths, reached with the best parameter settings for each approach, can be found in Figure 2 for the TWT case and in Figure 3 for the TWOT case. For all perturbation approaches the best evaluated parameter for the destroy percentile was 10%. In the TWT case, the best found setting for the non-improving parameter was 2 for MBP with the activity list index criterion, while it was 10 for random perturbation and MBP with the machine index criterion. For the TWOT objective on the other hand, a value of 2 performed best for the MBP with the machine index criterion, while 10 was superior for MBP with the activity list index criterion. For random perturbation both 2 and 10 gave the same result.

First of all, we see that random perturbation is not effective for the real-world use case. Destroying a random part of the solution makes it difficult to recover quickly and can even lead to worse results than using no perturbation at all. For the TWT objective function, TS with MBP using the machine index criterion leads to a solution that is 26% better than the base version. TS with MBP using the activity list index criterion is 29% better. Figure 2 shows that the four approaches perform equally until iteration 60. The base approach without perturbation is from this point stuck in a local minimum. Random perturbation cannot recover from its large modifications, which destroyed a large part of the previously achieved solution quality. Both MBP operators, however, continue to find new best solutions by applying more controlled perturbation steps, preserving the best solution properties.

In the TWOT case, the ranking of the two MBP operators is switched, with the machine index criterion performing best. It leads to an improvement of 40% over the base version, while TS using MBP with activity list index leads to a slightly lower improve-

Table 1: Evaluation results for the MBP operators on the makespan test set: percentage deviation of the optimal makespan averaged over instance group.

Perturbation	None (Base)	Random		MBP - Activity list index						MBP - Machine index			DRL[1]
NonImproving		10	15	10			15			15			
Destroy percentile		30%	30%	10%	30%	50%	10%	30%	50%	10%	30%	50%	
abz	13.4%	17.9%	14.0%	12.5%	11.3%	10.8%	13.0%	11.8%	13.3%	13.1%	12.6%	12.5%	14.7%
la	3.3%	7.5%	3.4%	2.6%	2.1%	2.6%	2.7%	2.1%	2.3%	2.5%	2.5%	2.8%	10.9%
orb	7.1%	11.3%	4.4%	4.4%	3.5%	3.7%	5.8%	3.6%	4.2%	4.7%	6.1%	5.0%	20.9%
swv	19.3%	21.1%	17.8%	16.0%	16.4%	17.4%	16.1	16.2%	17.6%	17.7%	17.4%	16.6%	22.4%
ta	16.2%	18.5%	16.6%	13.5%	14.1%	15.3%	14.0%	13.9%	15.1%	14.1%	14.1%	14.9%	18.3%
Average	12.7%	15.6%	12.5%	10.5%	10.6%	11.5%	10.8%	10.5%	11.4%	11.0%	11.0%	11.4%	16.8%

[1] Yuan et al. (2023)

Destroy percentile = fraction of the solution that is destroyed in the perturbation step, DRL = Deep Reinforcement Learning, MBP = Memory-Based Perturbation, NonImproving = Number of non-improving Tabu Search steps before perturbation is triggered.

Table 2: Evaluation results for the MBP operators on the TWT test set: total difference to the optimal tardiness in minutes averaged over instance group.

Perturbation	None (Base)	Random		MBP - Activity list index						MBP - Machine index		
NonImproving		10	15	10			15			15		
Destroy percentile		30%	30%	10%	30%	50%	10%	30%	50%	10%	30%	50%
abz	66.7	253.8	37.7	58.2	63.8	31.2	66.7	59.3	63.7	63.7	68.8	60.7
la	80.7	173.7	69.0	82.6	80.7	74.4	76.9	74.2	73.4	77.9	78.3	78.0
mt	165.7	197.0	185.3	175.33	162.3	175.3	153.0	175.3	175.3	175.3	168.0	175.3
orb	340.0	576.7	311.9	336.2	333.9	328.3	319.7	332.5	317.7	331.9	327.5	326.7
Average	194.6	355.2	175.7	193.4	191.5	183.9	183.6	188.3	182.1	190.1	187.9	187.6

Destroy percentile = fraction of the solution that is destroyed in the perturbation step, MBP = Memory-Based Perturbation, NonImproving = Number of non-improving Tabu Search steps before perturbation is triggered.

Table 3: Evaluation results in minutes for the real-world use case, displayed for the total weighted tardiness (TWT) after 100 iterations per approach.

NonImproving		2			10	
Destroy percentile	10%	30%	50%	10%	30%	50%
No perturbation (Base)			107,572			
Random perturbation	146,217	146,217	146,217	107,572	107,572	107,572
MBP-Activity list index	**76,548**	146,217	146,217	107,572	107,572	107,572
MBP-Machine index	102,315	82,663	146,217	79,967	107,572	107,572

Table 4: Evaluation results in minutes for the real-world use case, displayed for the total weighted order tardiness (TWOT) after 100 iterations per approach.

NonImproving		2			10	
Destroy percentile	10%	30%	50%	10%	30%	50%
No perturbation (Base)			50,328			
Random perturbation	50,328	50,328	50,328	50,328	50,328	50,328
MBP-Activity list index	45,027	50,328	50,328	31,322	50,328	50,328
MBP-Machine index	**29,898**	48,133	30,900	48,133	39,615	50,328

ment of 37%. As it can be seen in Figure 3, also for this objective all approaches perform equally during the early phases of the search procedure, however diverge in later iterations.

5 CONCLUSIONS

We have proposed a framework to derive perturbation operators that preserve promising structures of solution candidates, based on an elite solution pool, collected during a TS procedure. After presenting a variety of possibilities to configure the parameters for these perturbation operators, we have implemented two variants and performed numerical experiments on job-shop benchmark instances from the literature, as well as on a real-world industry use case. In several test runs, we have compared the performance

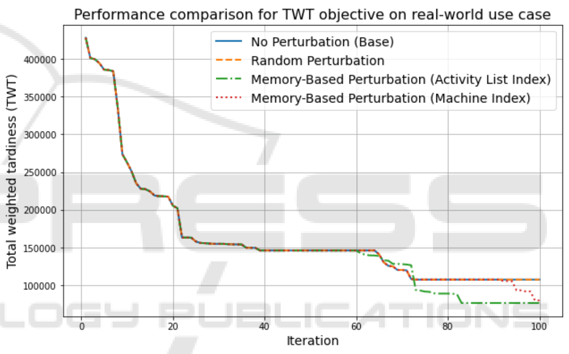

Figure 2: Objective value path over 100 Tabu Search iterations for the real-world industry case with the objective to minimize total weighted tardiness (TWT).

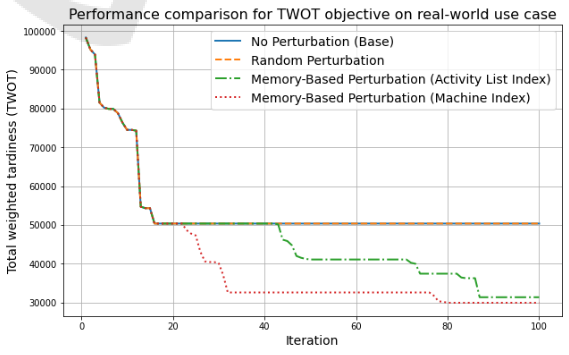

Figure 3: Objective value path over 100 Tabu Search iterations for the real-world industry case with the objective to minimize total weighted order tardiness (TWOT).

of the TS before and after the integration of the derived perturbation operators. We see that the TS including the proposed perturbation steps is competitive with learning-based techniques, such as deep reinforcement learning. Due to their ability to capture the complexity of the problem at hand, those methods would have been an implementable alternative to the currently used TS algorithm. However, our results support the choice for the enhancement of our metaheuristic approach. Moreover, experiments on benchmark instances demonstrate how sensitive random perturbation operators react to crucial parameters, such as the number of non-improving solutions before triggering a perturbation. Concerning stability, the MBP operators have shown a significantly better behavior.

Most notably, the integration of the MBP operator leads to significant improvements on the real-world case study. For both evaluated objective functions, TWT and TWOT, the TS with MBP operator outperformed the base version without perturbation steps, as well as the random perturbation approach. In the TWT case, activity tardiness was reduced by 29%, while in the TWOT case order tardiness was reduced by 40%. This shows, that especially on large instances the mechanism to preserve promising solution structures within the perturbation steps has the potential to increase overall solution performance. Since the presented framework relies on several different parameters that can be analyzed, we plan to further investigate the impact of other factors on the overall solution performance. We will in more detail evaluate the influence of the neighborhood size, which determines how many neighbors are evaluated by the neighborhood operators of the TS. Also other neighbor selection strategies, such as best improvement, will be compared to the current algorithmic setting. Moreover, we plan to derive additional MBP operators from the proposed framework, investigating other evaluation criteria, that could be based, e.g., on activity starting times, or more sophisticated repair-mechanisms.

ACKNOWLEDGEMENTS

The research was supported by the Austrian Research Promotion Agency (FFG) (grant # FO999907709). This support is gratefully acknowledged.

REFERENCES

Adams, J., Balas, E., and Zawack, D. (1988). The Shifting Bottleneck Procedure for Job Shop Scheduling. *Management Science*, 34(3):391–401.

Applegate, D. and Cook, W. (1991). A Computational Study of the Job-Shop Scheduling Problem. *ORSA Journal on Computing*, 3(2):149–156.

Bögl, M., Gattinger, A., Knospe, I., Schlenkrich, M., and Stainko, R. (2021). Real-life scheduling with rich constraints and dynamic properties – an extendable approach. *Proceedings of the 2nd International Conference on Industry 4.0 and Smart Manufacturing (ISM 2020)*, 180:534–544.

Braune, R., Wagner, S., and Affenzeller, M. (2007). Optimization Methods for Large-Scale Production Scheduling Problems. In Moreno Díaz, R., Pichler, F., and Quesada Arencibia, A., editors, *Computer Aided Systems Theory – EUROCAST 2007*, pages 812–819, Berlin, Heidelberg. Springer Berlin Heidelberg.

Laborie, P., Rogerie, J., Shaw, P., and Vilím, P. (2018). IBM ILOG CP optimizer for scheduling. *Constraints*, 23(2):210–250.

Lawrence, S. (1984). Resouce constrained project scheduling : an experimental investigation of heuristic scheduling techniques (supplement). *Graduate School of Industrial Administration, Carnegie-Mellon University*.

Lunardi, W. T., Birgin, E. G., Ronconi, D. P., and Voos, H. (2021). Metaheuristics for the online printing shop scheduling problem. *European Journal of Operational Research*, 293(2):419–441.

Moumene, K. and Ferland, J. A. (2009). Activity list representation for a generalization of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 199(1):46–54.

Ropke, S. and Pisinger, D. (2006). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455 – 472.

Schlenkrich, M. and Parragh, S. N. (2023). Solving large scale industrial production scheduling problems with complex constraints: an overview of the state-of-the-art. *4th International Conference on Industry 4.0 and Smart Manufacturing*, 217:1028–1037.

Shaw, P. (1997). A new local search algorithm providing high quality solutions to vehicle routing problems. *APES Group, Dept of Computer Science, University of Strathclyde, Glasgow, Scotland, UK*, 46.

Singer, M. and Pinedo, M. (1998). A computational study of branch and bound techniques for minimizing the total weighted tardiness in job shops. *IIE Transactions*, 30(2):109–118.

Storer, R. H., Wu, S. D., and Vaccari, R. (1992). New Search Spaces for Sequencing Problems with Application to Job Shop Scheduling. *Management Science*, 38(10):1495–1509.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285.

Yuan, E., Cheng, S., Wang, L., Song, S., and Wu, F. (2023). Solving job shop scheduling problems via deep reinforcement learning. *Applied Soft Computing*, 143:110436.