Proceedings of the 2015 Winter Simulation Conference
L. Yilmaz, W. K. V. Chan, I. Moon, T. M. K. Roeder, C. Macal, and M. D. Rossetti, eds.

FLEXIBLE JOB-SHOP SCHEDULING WITH OVERLAPPING MACHINE SETS

Tao Zhang
Shufang Xie
Oliver Rose

Universität der Bundeswehr München
Department of Computer Science
Neubiberg, 85577, GERMANY

ABSTRACT

In practice, the complexity of the flexible job-shop scheduling problem is quite large, i.e., it is often impossible to find the optimal solution in a reasonable time. But for small problems the optimal solution can be found in a very short time. In our study, a simulation-based segmentation procedure divides the problem into several small subproblems, and then a branch and bound method is used to solve the subproblems one after another. The solutions of the subproblems make up the solution of the whole problem. A method to determine the size of the subproblem is provided. The heuristic for the branching is developed from the machine overlapping features. The experimental results show that the approach performs better than some decision rules.

1 INTRODUCTION

In the flexible job-shop scheduling problem (FJSP), an operation may be processed on several machines. These machines form a machine set. Comparing to the job-shop scheduling problem, the raised issue is the machine allocation, i.e., assigning each operation to a machine from the machine set. In this study, we will consider a special case in which the machine sets can be overlapping. In other words, one machine can process different types of operations. This adds some new features to the problem. The FJSP is one of the most important combinatorial optimization problems. It has been proved to be a NP-hard problem. Usually FJSP is formularized as a mixed binary integer linear programming problem (MBILP) (Demir and Kürşat İşleyen 2013) in which the machine allocation variables are binary and the starting times of the operations are continuous. The disjunctive constraint relaxation also generates some binary variables. There are some exact methods to solve the MBILP problem, such as branch and bound (B&B) (Brucker, Jurisch and Sievers 1994, Pan and Chen 2005), cutting-plane, branch and cut (Karimi-Nasab and Seyedhoseini 2013), and so on. Most of these methods are based on the simplex algorithm which is not a polynomial time algorithm for linear programming. They are also no polynomial time algorithms for MBILP. Therefore, because the size of the practical FJSP can become very large, these methods are unable to find an optimal solution in a reasonably short time. Thus, in order to shorten the computing time, researchers are trying to find an approximate optimal solution instead of the exact solution. Hence, a variety of heuristic procedures and meta-heuristic procedures have been used. Decision rules are typical heuristic procedures (Holthaus and Rajendran 1997). The meta-heuristics procedures include local search (Murovec and Šuhel 2004), tabu search (Bożejko and Makuchowski 2009), simulated annealing (Satake et al. 1999), genetic algorithms (Qing-dao-er-ji and Wang 2012), ant colony algorithms (Rossi and Dini 2007), particle swarm algorithms (Lin et al. 2010), and so on.

In this study, we introduced a totally different approach. The reason that makes the exact methods slow is the big size of the problem. As a consequence, our general idea is to divide the big problem into several subproblems which are smaller and can be solved using exact methods in a short time. Using the exact methods, the new features raised by the overlapping machine sets are used to generate a heuristic for speeding up the solution procedure. We solve the subproblems one by one. The solutions to the subproblems make up the solution to the whole problem. The paper is structured as follows. In Section 2, the problem is described in detail. How to divide the big problem is illustrated in Section 3. Section 4 introduces the B&B and some heuristics to solve the subproblems. Experiments are carried out in Section 5. In the last section, we conclude our study.

2 PROBLEM DESCRIPTION

In this study, there are several types of jobs. Each type of job contains different operations. The operations must be carried out in a fixed sequence. Each type of operation can be processed on one of several machines and the processing times for the operations are different on different machines. Overlapping machine sets are considered in the problem. Here we use an example to describe the overlapping machine set problem. Let us assume that two operations A and B need to be processed. A can be processed on either machine 1 or machine 2 while B can only be processed on machine 2. If we assign machine 2 to A, operation B has to wait for operation A to finish. If we assign machine 1 to A, B won't wait. The decision influences the performance of the shop heavily. When the number of the operations and the machines increases, this problem becomes more complicated. It is worth focusing on this special case. There are often two types of objectives to an FJSP. One is related to the completion time of the jobs, and another is about the due dates. Our objective is to minimize the average completion time of the jobs. In addition, the FJSP contains two separate tasks: job sequencing and machine allocation. The job sequencing decides which operation should be processed first on a machine when there are several waiting jobs. The job sequencing will be done using the priority rules.

The size of FJSP depends on the operation number and the alternative machine number. For instance, if there are 100 operations and each operation has 3 alternative machines, the search space (number of the feasible solutions) for the machine allocation will reach $3^{100} = 5.15E + 47$. It is impossible to use exhaustive search methods. To use branch and bound, we have to be lucky enough because the worst case is the exhaustive search. As we all know, $10 \cdot 3^{10} << 3^{100}$. Therefore if we can divide the 100 operations into 10 groups each group includes 10 operations, i.e. there are 10 subproblems now. Because the size of the subproblems is really small, even the exhaustive search is capable of solving them. In our study, we will use simulation and a special roll-back technique to divide the operations. The subproblems will be solved by B&B with a particular branching heuristic.

3 PROBLEM SEGMENTATION BASED ON SIMULATION

3.1 Size of Subproblems

The first question addressed for the problem segmentation is to determine the size of the subproblems. As we mentioned before, the size of the subproblem will increase dramatically while the operation number just has a very tiny change. Figure 1 illustrates this tendency. In order to deal with this situation, a measurement called AT+1 (additional computing time caused by increasing one operation) is introduced to our study. In Figure 1, we can see that if we change the operation number from 13 to 14, the value of the AT+1 is about $r \cdot 3.2 \times 10^6$, where r is the computing time for evaluating one feasible solution. If we change the operation number from 20 to 21, the value of the AT+1 is about $r \cdot 6.5 \times 10^9$. It is about 2000 times as much as the first case. Obviously, if we need, for instance, one more hour to solve the subproblem just because we add one more operation, this will be unreasonable. Thus we assign a maximal value to AT+1. Increasing the number of operations one by one, the first appeared number whose AT+1 is bigger than the maximal AT+1 is the maximal operation number in the subproblem.

On the other hand, the subproblem should include as many operations as possible so as to ensure that a better solution to the whole problem is achieved. Therefore, we also assign a minimal value to the AT+1. The first appeared number whose AT+1 is smaller than the minimal AT+1 is the minimal operation number. By using the minimal and maximal AT+1 we obtain a range for the operation numbers in the subproblems.

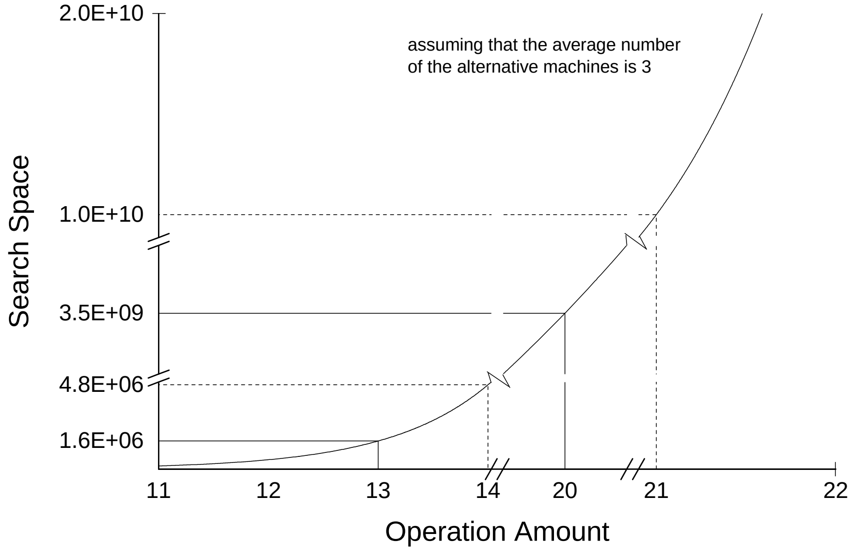

Figure 1: Relationship between search space and operation amount.

According to the application fields, there is always one restriction on the maximal computing time of the whole problem. So the restriction can be used to narrow the range we obtained before. Assuming that each subproblem contains the same number of operations and the average number of the alternative machines for the operations is *m*, the following simultaneous inequalities can be used to determine the range of the subproblems' size. In the equalities n denotes the size of the subproblems, i.e., the operation number in the subproblems. *N* is the total number of the operations. *T* is the maximal computing time of solving the whole problem. $\Delta_{\min}$ and $\Delta_{\max}$ are the minimal and the maximal value of the AT+1. Generally the number in the center of the range is adopted. It is possible that there is no solution for the inequalities. In this case we have to either decrease $\Delta_{\min}$ or increase *T*. Decreasing $\Delta_{\min}$ leads to the reduction of the operation number in the subproblems. So it will lower the quality of the solution for the whole problem.

$$\begin{cases} m^n rN / n < T \\ \Delta_{\min} < (m^{n+1} - m^n) r < \Delta_{\max} \end{cases} \tag{1}$$

3.2 Simulation-based Segmentation

According to the rough estimates for the starting times of the operations, the operations can be simply divided into several groups. The rough starting times are usually calculated without consideration of the waiting times or by using waiting time estimates. However, very often delaying one operation may improve the whole performance. Moreover, with this simple segmentation we will have to compute the initial state for each subproblem. Thus, in our study we use simulation to divide the operations. The segmentation procedure is shown in Figure 2.

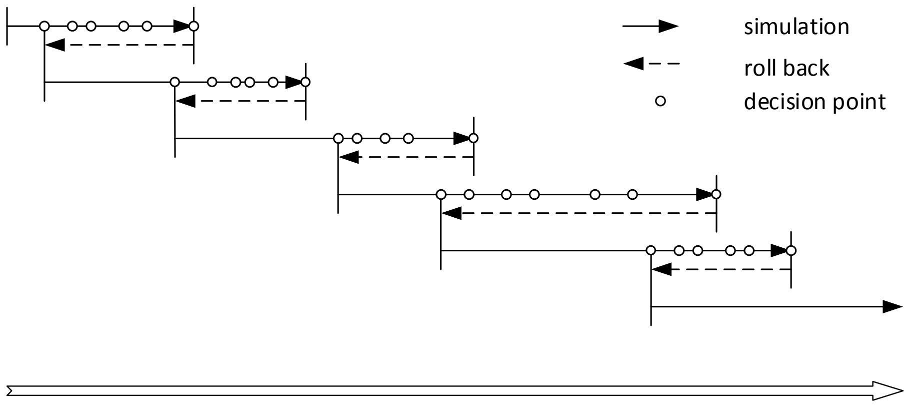

Figure 2: Simulation-based segmentation of the operations.

A list is created for storing the operations we selected. After the simulation starts, once an operation needs to be assigned to a machine from several alternatives, the operation will be put into the list. In the simulation, the decision is made according to an allocation rule. The simulation will pause if the number of operations in the list reaches the target number which has been determined in Section 3.1. The operations in the list make up a subproblem which will then be solved by the B&B algorithm. The solution of the subproblem indicates the machine allocation results for the operations. After that the simulation will roll back to the first decision point in the list and continue from that point. In the simulation the operations in the list will follow the machine allocation decision made by the B&B algorithm. Once a new operation appears and the operation has to make the machine allocation decision, we empty the list and put the new operation into the list. The decision in the simulation is still made by the rule. Meanwhile we save the current state of the simulation model. The state will be treated as the initial state of the new subproblem. When the number of the operations in the list reaches the target number, the simulation pauses and the new subproblem is solved by the B&B algorithm. The simulation rolls back to the first decision point in the list and continues. The remaining operations can be treated in the same manner. The procedure will stop if all operations have been assigned to a machine.

In the simulation, the lowest request (LR) is introduced as the allocation rule. The machine with the lowest request will be selected. For example, if one operation can be processed on machine 1 and 3 other machines, we assign a request value of ¼ to each machine. If machine 1 is also requested by another operation which has 3 alternative machines in total machine 1 is given another request value of 1/3. Finally, we add these values together for each machine and the results indicate the request amount of the machines. If there are several jobs waiting in front of the machines, we add the number of the waiting jobs to the request amount. The machines which have the smallest request amount will be selected.

3.3 Enhanced Segmentation Procedure

From Figure 2 we can easily find that a new decision point may occur before the operations, which have already been assigned to machines, are all finished. This will influence the performance of the whole solution directly. In order to avoid this situation, we improved the procedure. Another list is added to the procedure. To distinguish from the list mentioned above, we call this list the decision execution list (DEL) and call the list above as the decision making list (DML). In the simulation, while a decision made by the B&B is executed, the related operation will be put into the DEL. When a new decision point appears, we will check the DEL. If the number of the operations in the DEL is less than a given number *S*, we will empty the DML and put all operations from the DEL into the DML. Then we clear the DEL. In the subsequent simulation, once a decision needs to be made, no matter if the decision has been made by the B&B or the decision has to be made by the rule, the related operations will all be put into the DML. If the operation number in the DML reaches the target number *n*, the subproblem will be solved. At that time, the simulation will roll back to the first decision point in the DML. We have to mention that this point is not the newly appeared decision point but the point where we have made the decision before by the B&B. Only while a new decision point occurs and the number of operations in the DEL is greater than the given number, we will empty the DML and the DEL, and put the new operation only to the DML. After the subproblem is solved, the simulation will roll back to the first new decision point. So in this way we can make sure that the number of the executed decisions before a new decision point occurs is greater than the given number *S*. The improved procedure after the first roll back is given here. The given number *S* describes the distance between two successive subproblems.

```
After the first roll back
Continue simulation
Set repeat to false
If a decision made by the B&B is executed and the repeat is not true Then
   Put the related operation into the DEL
End If
If the first new decision point appears And the repeat is not true Then
   If size of the DEL > S Then
      Clear DEL
      Put the related operation into the DML
   Else
      Put the operations in the DEL into the DML
      Clear DEL
      Put the related operation into the DML
      Set repeat to true
   End If
End If
If a new decision point appears And it is not the first one Then
   Put the related operation into the DML
End if
If an old decision point appears And the repeat is true Then
   Put the related operation into the DML
End if
If size of the DML = n then
   Pause simulation
   Solve the subproblem represented in the DML by the B&B
   Clear the DML
   If all operations have been assigned machines Then
      Go to Exit Procedure
   End If
   Roll back the simulation to the first decision point in DML
   Go to Continue Simulation
End if
Exit Procedure
```

4 SUBPROBLEM SOLVER USING BRANCHING AND BOUNDING

4.1 Formulism of Subproblems

The objective of the subproblems must coincide with the objective of the whole problem. Thus the objective is set to minimize the average completion time of the selected operations. The difference to the general FJSP is that the initial state of the shop is not empty in the subproblems except the first one and the machines have the overlapping characteristic. Some machines may be processing the operations or their queues may still have waiting operations. In the subproblems we calculate for each machine an available time which is the point of time that all the waiting operations are finished. The operations in the subproblems can start on a machine only after its available time.

The mathematical programming model of the subproblems is as follows. $o(j,i)$ is the *i*-th operation of job j. *O* is the set of the operations in the subproblems. *n* is the size of *O*. $C_{o(j,i)}$ is the completion time of the operation $o(j,i)$. $\tau_{o(j,i)}$ is the starting time of the operation. $p_{o(j,i),m}$ is the processing time of the operation on machine *m*. $M_{o(j,i)}$ is the alternative machine set for the operation. $\kappa_{o(j,i),m}$ is the allocation variable. If the allocation variable is 1, it means the machine *m* is assigned to the operation $o(j,i)$; otherwise, the machine *m* is not assigned to the operation. τ_m is the available time of the machine *m*.

$$
\begin{aligned}
\min: \quad & 1/n \sum_{o(j,i)\in O} C_{o(j,i)} \\
\textit{subject}: \quad & C_{o(j,i)} = \tau_{o(j,i)} + \sum\nolimits_{m\in M_{o(j,i)}} \kappa_{o(j,i),m} p_{o(j,i),m} \\
& \quad \forall o(j,i)\in O \quad \sum\nolimits_{m\in M_{o(j,i)}} \kappa_{o(j,i),m} = 1 \wedge \kappa_{o(j,i),m} \in \{0,1\} \\
& C_{o(j,i)} \le \tau_{o(j,i+1)} \\
& C_{o(j,i)} \le \tau_{o(k,l)} \vee C_{o(k,l)} \le \tau_{o(j,i)} \\
& \quad \forall\, o(j,i), o(k,l) \quad \kappa_{o(j,i),m} = \kappa_{o(k,l),m} \wedge m \in M_{o(j,i)} \cap M_{o(k,l)} \\
& \tau_{o(j,i)} \ge \sum\nolimits_{m\in M_{o(j,i)}} \kappa_{o(j,i),m} \tau_m
\end{aligned}
\tag{2}
$$

The last constraint says that the operations in the subproblems can start on a machine only after the machine becomes available. The second from the last constraint means that one machine can process only one operation at a time. It is a disjunctive constraint. The third from the last constraint is the consequence constraint which means that one operation can start only when its predecessive operation has finished.

4.2 Branching Tree

If the size of the subproblem is quite big, there is no efficient exact algorithm to solve the above model. In our study the size of the subproblems obtained from the simulation is relatively small. So it is possible to use the exact algorithm to solve them. The B&B algorithm is a simple and intuitive exact method to solve the small machine allocation problems. We adopt it in the study. The branching tree is structured as in Figure 3. The tree has several levels and each level represents one operation. Every branch at one level means the selection of one alternative machine for the corresponding operation. One path from the root to any of the leaves is one solution. The operations are in the same sequence as they are collected. The highest level corresponds to the first collected operation.

In addition, while branching one node we use some heuristics to remove some of the node's branches. In other words, we remove some machines from the operation's alternative machine set according to the state of the job shop. The following three heuristic rules were used.

Rule 1 *Comparing with two machines which are only be used by the concerned operation, the machine which can finish the operation earlier will be kept and another machine will be removed from the machine set.*

Comparing with two machines in which one machine (exclusive machine) is used only by the concerned operation and another machines (shared machine) is shared by other operations, we calculate the latest completion time and the earliest completion time for the concerned operation on the shared machine.

Rule 2 *If the completion time on the exclusive machine is later than the latest completion time on the shared machine, we will keep the shared machine and remove the exclusive machine.*

Rule 3 *If the completion time on the exclusive machine is earlier than the earliest completion time on the shared machine, we will keep the exclusive machine and remove the shared machine.*

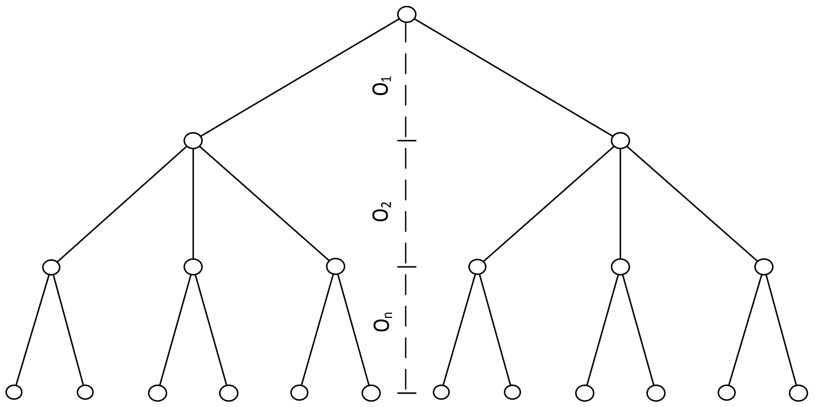

Figure 3: Branching tree of the machine allocation problem.

4.3 Lower Bound and Upper Bound

The key idea of the B&B algorithm is to determine the lower bound and upper bound for the tree node. If the lower bound for some node A is greater than the upper bound for some other node B, then A may be safely discarded from the search. This step is called pruning, and we carry it out by maintaining a global variable (shared among all nodes of the tree) that records the minimum upper bound found so far. Any node whose lower bound is greater than the minimum upper bound can be discarded.

The lower bound for the concerned node is computed in the following way. First, each of the remaining operations is assigned to a machine which can process the operation earliest. Then, if several operations will use the machine, we assume that the machine can start them together at the same time. At last we calculate the average completion time involving both the remaining operations and the assigned operations. It is obvious that the average completion time is the lower bound. For some assigned operations which are assigned to the same machine, FIFO rule is applied. The first assigned operation will be processed first. The remaining operations (if there are successive operations) won't start together. The successors have to be processed after their predecessors.

For the upper bound calculation, each of the remaining operations is assigned to a machine which can process the operation last. If several operations will use the machine, we assume that the machine will sequence them using the longest processing time (LPT) rule and process them one by one in the sequence. The LPT rule will result in the maximal average completion time on the single machine.

5 EXPERIMENTS

A job shop model is created to evaluate the proposed approach. In the model there are 8 machines and 2 types of jobs, type A and B. Type A contains 3 operations while type B contains 4 operations. Each operation has 1 to 4 alternative machines. The operations need different amounts of time if they are processed on the different machines. The machine sets for the operations are heavily overlapping. The shop plans to produce 13 jobs of type A and 15 jobs of type B. So there are in total 99 operations. The average number of the alternative machine for the operations is 3. So the size of the search space is about 1.7E+47. According to the tests, the computing time of dealing with one feasible solution in the search space is 4.2E-6 seconds. The time needed to evaluate all feasible solutions will be 2.3E+34 years. Therefore, we have to divide the problem into several subproblems.

Table 1: Machine sets for the operations.

Job Type A			Job Type B			
O1	O2	O3	O1	O2	O3	O4
M1,M3,M4	M2,M4	M3,M5,M6	M1,M7,M8	M3,M5,M6,M8	M3,M6,M7	M4

We use the inequalities in Section 3.1 to calculate the ranges of the subproblems' sizes. The maximal time we can bear to solve the whole problem is 60 minutes; $\Delta_{\max}$ is 20 minutes; $\Delta_{\min}$ is 20 seconds. We determine that the range of the operation number *n* in the subproblems ranges from 14 to 17. We assume $S = 0.65 \cdot n$. For instance, if we adopt $n = 17$, we have to divide the problem into 6 subproblems. The sum of the search spaces of these subproblems is 7.8E+8 which is far smaller than the search space of the whole problem.

We carry out the experiments for every *n* from 14 to 17 using both the exhaustive search and B&B to solve the subproblem. The allocation rule LR is also used to solve the whole problem. The results are compared in Table 2. Because of the enhanced segmentation procedure, some subproblems may be regrouped and solved many times. So the computing time using the exhaustive search when $n = 17$ is longer than our expected maximal time. The time using the exhaustive search is the worst case of the time using the B&B.

Table 2: Experiment results.

Approach		Computing Time /minutes	Objective /hours
Heuristic Rule	LR	0.14	2.52
Subproblems + Exhaustive search	n=14	4.69	2.08
	n=15	15.01	2.10
	n=16	46.32	2.01
	n=17	123.71	1.92
Subproblems + B&B	n=14	0.32	2.08
	n=15	1.48	2.10
	n=16	6.71	2.01
	n=17	4.80	1.92

From Table 2 we can see that the heuristic rule uses only several seconds. That is because the heuristic rule is just a one-time simulation. However our subproblem approach can always achieve the better objective than the heuristic rule does. And the trend is that the bigger *n* results in the better solution.

6 CONCLUSIONS

The proposed approach is inspired by an obvious fact that $m^N >> nm^{N/n}, m \geq 2, n \geq 2$ while N is big enough. In practice the number of operations N is often greater than 50. So the sum of the computing time of the subproblems is far shorter than the whole problem. The simulation-based segmentation procedure divided the operations into several groups. The procedure can ensure that the operations in the same group are the only operations in the related time period while the final machine allocation decisions are followed. The size of the subproblems we determined can guarantee that we obtain the solution within the maximal computing time, even if the B&B works in the worst case. We obtained the optimal solution to each subproblem. Even though the sum of the sub solutions does not mean the optimal solution to the whole problem, comparing to the decision rule, the approach performs considerably better.

REFERENCES

Bożejko, W., and M. Makuchowski. 2009. "A Fast Hybrid Tabu Search Algorithm for the No-Wait Job Shop Problem." *Computers & Industrial Engineering* 56: 1502-1509.

Brucker, P., B. Jurisch, and B. Sievers. 1994. "A Branch and Bound Algorithm for the Job-Shop Scheduling Problem." *Discrete Applied Mathematics* 49: 107-127.

Demir, Y., and S. Kürşat İşleyen. 2013. "Evaluation of Mathematical Models for Flexible Job-Shop Scheduling Problems." *Applied Mathematical Modelling* 37: 977-988.

Holthaus, O., and C. Rajendran. 1997. "Efficient Dispatching Rules for Scheduling in a Job Shop." *International Journal of Production Economics* 48: 87-105.

Karimi-Nasab, M., and S. M. Seyedhoseini. 2013. "Multi-Level Lot Sizing and Job Shop Scheduling with Compressible Process Times: A Cutting Plane Approach." *European Journal of Operational Research* 231: 598-616.

Lin, T.-L., S.-J. Horng, T.-W. Kao, Y.-H. Chen, R.-S. Run, R.-J. Chen, J.-L. Lai, and I. H. Kuo. 2010. "An Efficient Job-Shop Scheduling Algorithm Based on Particle Swarm Optimization." *Expert Systems with Applications* 37: 2629-2636.

Murovec, B., and P. Šuhel. 2004. "A Repairing Technique for the Local Search of the Job-Shop Problem." *European Journal of Operational Research* 153: 220-238.

Pan, J. C.-H., and J.-S. Chen. 2005. "Mixed Binary Integer Programming Formulations for the Reentrant Job Shop Scheduling Problem." *Computers & Operations Research* 32: 1197-1212.

Qing-dao-er-ji, R., and Y. Wang. 2012. "A New Hybrid Genetic Algorithm for Job Shop Scheduling Problem." *Computers & Operations Research* 39: 2291-2299.

Rossi, A., and G. Dini. 2007. "Flexible Job-Shop Scheduling with Routing Flexibility and Separable Setup Times Using Ant Colony Optimisation Method." *Robotics and Computer-Integrated Manufacturing* 23: 503-516.

Satake, T., K. Morikawa, K. Takahashi, and N. Nakamura. 1999. "Simulated Annealing Approach for Minimizing the Makespan of the General Job-Shop." *International Journal of Production Economics* 60–61: 515-522.

AUTHOR BIOGRAPHIES

TAO ZHANG is a Ph.D. student working on production planning and scheduling at the Department of Computer Science of the Universität der Bundeswehr München, Germany. From 2007 to 2009 he received his Master degree in metallurgical engineering with the subject of production planning and scheduling in iron and steel industry from Chongqing University, China. He is involved in modeling and simulation of complex system and intelligent optimization algorithms. His email address is tao.zhang@unibw.de .

SHUFANG XIE is a research assistant at the Department of Computer Science of the Universität der Bundeswehr München, Germany since 2013. She obtained her Master degree in metallurgical engineering from Chongqing University, China in 2010. Her study is mainly about the project scheduling. Her email address is shufang.xie@unibw.de .

OLIVER ROSE holds the Chair for Modeling and Simulation at the Department of Computer Science of the Universität der Bundeswehr, Germany. He received a M.S. degree in applied mathematics (1992) and a Ph.D. degree in computer science (1997) from Würzburg University, Germany. His research focuses on the operational modeling, analysis and material flow control of complex manufacturing facilities, in particular, semiconductor factories and assembly systems. He is a member of INFORMS Simulation Society, ASIM (German Simulation Society), and GI (German Computer Science Society). In 2012, he served as General Chair of the WSC. Currently, he is member of the board of the ASIM and the ASIM representative at the Board of Directors of the WSC.