Scheduling of hybrid types of machines with two-machine flowshop as the first type and a single machine as the second type

Ming-Chih Hsiao¹, Ling-Huey Su²

¹, ² Department of Industrial and Systems Engineering, Chung Yuan Christian University, Taiwan.
E-mail: t6817361@cycu.org.tw

Abstract. This research addresses the problem of scheduling hybrid machine types, in which one type is a two-machine flowshop and another type is a single machine. A job is either processed on the two-machine flowshop or on the single machine. The objective is to determine a production schedule for all jobs so as to minimize the makespan. The problem is NP-hard since the two parallel machines problem was proved to be NP-hard. Simulated annealing algorithms are developed to solve the problem optimally. A mixed integer programming (MIP) is developed and used to evaluate the performance for two SAs. Computational experiments demonstrate the efficiency of the simulated annealing algorithms, the quality of the simulated annealing algorithms will also be reported.

1. Introduction
Most studies on the hybrid machine scheduling consider all n jobs to be processed on a hybrid system that consists of a series of production stages or workshops and at least of which must have several facilities [1]. We consider the different machine environments in which two production machine types are considered and each job instead of being processed on two machine types. It is processed either on the two-machine flowshop or on the single machine. Gupta and Tunc [2] considered a n jobs, two-stage hybrid flowshop scheduling problem with only one machine at stage 1 and m identical machines at stage 2, they developed approximate algorithms to find a minimum makespan. Wang, Liu and Chu [3] investigated a two-stage no-wait hybrid flowshop scheduling problem in which the first stage contains a single machine and the second stage contains several identical parallel machines, in the paper, they proposed a branch-and-bound algorithm to find minimum makespan. Rabiee, Rad, Mazinani and Shafaei [4] proposed a two-stage no-wait hybrid flowshop scheduling problem considering unrelated parallel machines, sequence-dependent setup times, and they used a hybrid algorithm which is combined with simulated annealing, variable neighborhood search and genetic
algorithm to search minimum makespan. The hybrid flowshop problems are extended by considering various process conditions. Luo, Huang, Zhang, Dai and Chen [5] considered a two-stage hybrid batching flowshop scheduling with blocking and machine availability constraints. Oğuz, Ercan, Cheng and Fung [6] addressed a heuristic algorithm for a two-stage flowshop scheduling problem with multiprocessor tasks by considering a multi-stage setting for the processor environment to minimize the makespan. Jungwattanakit, Reodecha, Chaovalitwongse and Werner [7] considered a hybrid flowshop with unrelated parallel machines in one of stages. The objective is to minimize a convex sum of makespan and the number of tardy jobs. Rossi, Pandolfi and Lanzetta [8] analysed a multiple-stage hybrid flowshop with sequence-independent uniform set-up time, parallel batching machines with compatible parallel batch families. They used a critical ratio setup rule and a sliding time window technique to develop two heuristics to reduce the tardy jobs and the makespan.

2. Problem description
Using the standard three-fielded notation, the problem is denoted as $F_2,1||C_{\text{max}}$, where $F_2$ indicates two-machine flowshop and 1 means a single machine. The jobs can be processed on either the flowshop or the single machine. There are $n$ jobs to be processed either on the two-machine flowshop or on the single machine. The objective is to minimize the makespan.

Assumptions are made in this paper:
- Each job is independent. It is always processed without defect and its machine setup time can be negligible.
- Each job has neither preemption nor priority values for processing in each machine type.
- At any given time, a machine can process only one job. All machines are capable to process all jobs.
- The transportation time of jobs between the machines can be negligible on flowshop machines environment.
- The processing of jobs cannot be interrupted, i.e. once a machine starts processing a job, the job has to be completed without interruptions.
- During scheduling period, there is no maintenance or breakdowns in the machines.

3. Simulated annealing algorithm
The simulated annealing (SA) algorithm, introduced by Kirkpatrick [9], is a local search procedure capable of escaping from local optimum to solve combinatorial optimization problems. To start the procedure, SA draws an initial solution to generate neighborhood solution. If the neighborhood solution is better than the incumbent solution, the former is automatically accepted and replaces the latter; otherwise, the incumbent solution is used. The whole process is repeated until no significant improvement in the neighborhood solution is found or the pre-specified conditions are met. SA uses this repetitive improvement approach, but in particular it enables a search algorithm to escape from a local optimum.

3.1 Initial job sequence
To acquire a good initial solution for the proposed SA, the following steps were used: (1) generate a job sequence randomly; (2) sequence the jobs by Johnson’s rule; (3) assign the jobs to either the flowshop or the single machine, which offers the lowest completion time; (4) and let the solution ($C_{\text{max}}$) as the initial solution.
3.2 Decoding method

The solutions of assignment problems are represented by permutation schemes. Two machine types are available to perform the jobs. These jobs are then assigned to the machines and sequence at each machine type in order to minimize the overall makespan. In here, two decoding methods for the proposed SA are recommended. The first decoding method was overall-permutation decoding (OPD). Since the solution is represented by the permutation of the job indices \(\{1, 2, \ldots, n\}\) and two machine types, hence, there are \((n + 2 - 1)!\) permutations, which stand for the overall solutions. The permutation representation expresses the positions of jobs and the order sequence in the machines. In the permutation schemes, \((n + 1)\) was utilized as the partition number to separate the assignment for each machine types. Prior to the partition number \((n + 1)\), there are \(k\) jobs \((k \in n)\), where \(k\) jobs are assigned to the flowshop machines while the remaining jobs are assigned to the single machine. For the second decoding method, a list scheduling with Johnson’s rule decoding (LSJRD) was applied. The solution is represented by the permutation of the job indices\(\{1, 2, \ldots, n\}\). First, all jobs in Johnson’s rule were sequenced while the permutation representation displayed the assignment sequence of jobs. The jobs were assigned to the machine types with the lowest completion time.

3.3 Stopping criterion

The stopping criteria are the conditions to complete the algorithm. Theoretically, the SA finishes when the temperature converges to \(T_F\) or the solution does not improve after some iterations. In our research, if the temperature is lower than \(T_F\), then the SA will stop.

4. Computational experiments

The experiments are divided into two parts. The first part gives the mixed integer programming optimal solution, which can be used to compare with the proposed SA algorithms as a basis to prove the effectiveness of the SA algorithms. The second part provides the SA algorithms solution. The processing time of flowshop machine is uniformly distributed [1, 25] and the processing time of single machine is between [\(\max\) (the processing time of flowshop machines), sum of processing time of flowshop machines]. The numbers of job \(n = (6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 30, 40, 50, 60, 70, 80, 90, 100, 150, 200, 300)\). The numbers of job generated five different problem instances, which was run thirty times for every instance. Thus, 130 problems were tested in the experiment. For each instance, the optimum makespan and average computation time were obtained and computed from thirty trials.

5. Computational results

This section describes the computational tests to evaluate the performance of the proposed SAs. For small problems, the same test instances were used to run in CPLEX and SAs. The LSJRD and OPD generated 25 optimal solutions (100%) and 24 optimal solutions (96%) for 25 testing problems, respectively (Table 1). The solution performances for two SAs are likely similar with those of the MIP. In addition, the computation time (seconds) needed by the SA is shorter than MIP.
Table 1. Optimal solutions obtained by MIP, OPD and LSJRD.

| Instance # | MIP   | LSJRD | OPD   |
|------------|-------|-------|-------|
|            | Optima| Time(sec) | Solution | Time(sec) | Solution | Time(sec) |
| 6.1        | 28*   | 2.26   | 28*     | 0.011     | 28*      | 0.0106    |
| 6.2        | 43*   | 4.11   | 43*     | 0.0081    | 43*      | 0.0053    |
| 6.3        | 47*   | 3.61   | 47*     | 0.011     | 47*      | 0.0097    |
| 6.4        | 60*   | 4.77   | 60*     | 0.0093    | 60*      | 0.0053    |
| 6.5        | 46*   | 2.06   | 46*     | 0.0119    | 46*      | 0.0107    |
| 7.1        | 45*   | 18.164 | 45*     | 0.0129    | 45*      | 0.0087    |
| 7.2        | 70*   | 15.256 | 70*     | 0.0097    | 70*      | 0.0107    |
| 7.3        | 54*   | 29.166 | 54*     | 0.011     | 55       | 0.0111    |
| 7.4        | 49*   | 20.46  | 49*     | 0.0117    | 49*      | 0.0106    |
| 7.5        | 53*   | 9.97   | 53*     | 0.0113    | 53*      | 0.0087    |
| 8.1        | 64*   | 228.164| 64*     | 0.0232    | 64*      | 0.0082    |
| 8.2        | 50*   | 54.584 | 50*     | 0.0223    | 50*      | 0.0097    |
| 8.3        | 49*   | 51.484 | 49*     | 0.0224    | 49*      | 0.0106    |
| 8.4        | 54*   | 17.392 | 54*     | 0.0221    | 54*      | 0.0087    |
| 8.5        | 53*   | 116.102| 53*     | 0.0231    | 53*      | 0.0092    |
| 9.1        | 64*   | 424.946| 64*     | 0.0171    | 64*      | 0.0121    |
| 9.2        | 50*   | 556.012| 50*     | 0.0218    | 50*      | 0.0112    |
| 9.3        | 61*   | 798.236| 61*     | 0.0213    | 61*      | 0.0082    |
| 9.4        | 64*   | 1792.734| 64* | 0.0168    | 64*      | 0.0106    |
| 9.5        | 55*   | 625.054| 55*     | 0.0206    | 55*      | 0.0092    |
| 10.1       | 88*   | 7200.24| 88*     | 0.0145    | 88*      | 0.0092    |
| 10.2       | 57*   | 4210.084| 57* | 0.0167    | 57*      | 0.0087    |
| 10.3       | 61*   | 6376.158| 61* | 0.0161    | 61*      | 0.0073    |
| 10.4       | 62*   | 4607.54 | 62*    | 0.0197    | 62*      | 0.0131    |
| 10.5       | 59*   | 5791.472| 59* | 0.017     | 59*      | 0.0092    |

To compare two decoding methods for large problems, the decoding methods were run for 130 instances and the best makespan and average computational time were obtained using two decoding methods (OPD and LSJRD). The results were compared for large problems by counting the number of success run for each method. Evidently, the method with larger success run numbers produced better results than the others. Table 2 shows that the comparisons of the two solutions secured by OPD and LSJRD. When the size of the job is small, the OPD and LSJTD have similar performances, however, as the numbers of jobs increase, the performances of OPD decrease. This may be attributed to the obtained number of better solutions from the OPD, which is less than the LSJRD.
Table 2. The solution comparisons for OPD and LSJRD.

|    | 6   | 7   | 8   | 9   | 10  | 11  | 12  | 13  | 14  | 15  | 16  | 17  | 18  | 19  | 20  | 30  | 40  | 50  | 60  | 70  | 80  | 90  | 100 | 150 | 200 | 300 |
|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| LSJRD | Better | 0 | 1 | 0 | 0 | 0 | 0 | 2 | 1 | 2 | 0 | 4 | 2 | 5 | 4 | 5 | 5 | 5 | 5 | 3 | 4 | 3 | 5 | 1 | 2 |
|   | Equal | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
|   | Worse | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| OPD | Better | 4 | 3 | 3 | 2 | 2 | 3 | 3 | 1 | 4 | 2 | 4 | 3 | 3 | 1 | 1 | 1 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|   | Equal | 1 | 1 | 1 | 2 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|   | Worse | 0 | 1 | 1 | 1 | 2 | 1 | 2 | 1 | 2 | 4 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 |

*1: The result of LSJRD is better than OPD. 2: The result of LSJRD is equal to OPD. 3: The result of LSJRD is worse than OPD.

Furthermore, we applied the average of % improvement from the SAs as compared to the initial solutions. As shown in Table 3, the obtained average % improvement for LSJRD demonstrated better results than the OPD. The computation time in seconds needed by the OPD and LSJRD are shown in Table 4.

Table 3. The average of % improvement from OPD and LSJRD.

|    | 6   | 7   | 8   | 9   | 10  | 11  | 12  | 13  | 14  | 15  | 16  | 17  | 18  | 19  | 20  | 30  | 40  | 50  | 60  | 70  | 80  | 90  | 100 | 150 | 200 | 300 |
|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| LSJRD | 5.39 | 13.33 | 12.42 | 9.69 | 8.30 | 9.74 | 7.96 | 14.23 | 10.70 |
| OPD   | 5.39 | 12.97 | 12.42 | 9.69 | 8.30 | 9.74 | 7.96 | 13.82 | 10.16 |
| LSJRD | 13.55 | 8.80 | 10.46 | 11.77 | 9.37 | 8.97 | 8.15 | 7.44  |
| OPD   | 12.82 | 8.80 | 9.60 | 11.11 | 9.54 | 9.06 | 8.20 | 7.79  |
| LSJRD | 7.68 | 6.79 | 5.28 | 6.29 | 6.09 | 6.24 | 4.59 | 4.72  |
| OPD   | 7.16 | 6.35 | 5.65 | 6.06 | 5.87 | 5.77 | 4.38 | 4.57  |

Table 4. The average computation time for OPD and LSJRD.

|    | 6   | 7   | 8   | 9   | 10  | 11  | 12  | 13  | 14  | 15  | 16  | 17  | 18  | 19  | 20  | 30  | 40  | 50  | 60  | 70  | 80  | 90  | 100 | 150 | 200 | 300 |
|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| LSJRD | 0.0103 | 0.0113 | 0.012 | 0.0142 | 0.0168 | 0.0195 | 0.0226 | 0.0242 | 0.0294 |
| OPD   | 0.0083 | 0.01  | 0.0093 | 0.0103 | 0.0095 | 0.0116 | 0.0115 | 0.0125 | 0.0142 |
| LSJRD | 0.0319 | 0.0353 | 0.04 | 0.043 | 0.0472 | 0.0508 | 0.1117 | 0.2003 | 0.3201 |
| OPD   | 0.0146 | 0.015 | 0.0164 | 0.0165 | 0.0163 | 0.0171 | 0.027 | 0.0397 | 0.049 |
| LSJRD | 0.4785 | 0.6877 | 0.9518 | 1.2754 | 1.7412 | 5.383 | 11.876 | 36.617 |
| OPD   | 0.0641 | 0.0777 | 0.0931 | 0.1109 | 0.132 | 0.2497 | 0.4224 | 0.825 |

6. Conclusions
In this paper, we addressed the hybrid machine types scheduling problem, in which one type is a two-machine flowshop and the other type is the single machine. The problem was inspired by the IC final testing operation in IC backend factory. Two SAs were proposed to effectively respond on the large problems by applying the optimal solution for MIP model. A remarkable initial solution was achieved by adopting the efficient initial solution for the proposed SAs. Besides, the two decoding methods were applied to generate the total completion time. From the computational experiments, the optimal solution of small size problems for LSJRD and OPD were 100% and 96%, respectively. The average computation times for the two SAs were extremely shorter to that of the MIP. For large size problems, LSJRS demonstrated better than OPD, which was based on the obtained number of better solutions. Furthermore, the former exhibited better average of % improvement than the latter. It was found that the average computation times of LSJRD started to increase (i.e over 1 second) after 90 jobs. Based on the actual factory environment, 36.6 seconds are needed for 300 jobs, which offer an equitable and acceptable time. This paper shows a new solution on scheduling hybrid machine types, wherein one type is a two-machine flowshop and the other is the single machine, for NP-hard scheduling issue. The proposed SAs proved its capacity in obtaining the optimal solutions efficiently. In the future, we will extend the application of the proposed SAs to hybrid types of machines types with buffer constrains. Also, our research will continue designing meta-heuristics for the problem by taking advantage of good initial solutions generated by the presented heuristics.

7. References
[1] Gupta J N 1988 Two-stage, hybrid flowshop scheduling problem *Journal of the Operational Research Society* 359-64
[2] Gupta J N and Tunc E A 1991 Schedules for a two-stage hybrid flowshop with parallel machines at the second stage *International Journal of Production Research* 29 1489-502
[3] Wang S, Liu M and Chu C 2015 A branch-and-bound algorithm for two-stage no-wait hybrid flow-shop scheduling *International Journal of Production Research* 53 1143-67
[4] Rabiee M, Rad R S, Mazinani M and Shafaei R 2014 An intelligent hybrid meta-heuristic for solving a case of no-wait two-stage flexible flowshop scheduling problem with unrelated parallel machines *International Journal of Advanced Manufacturing Technology* 71 1229-45
[5] Luo H, Huang G Q, Zhang Y F, Dai Q Y and Chen X 2009 Two-stage hybrid batching flowshop scheduling with blocking and machine availability constraints using genetic algorithm *Robotics and Computer-Integrated Manufacturing* 25 962-71
[6] Oğuz C, Ercan M F, Cheng T E and Fung Y-F 2003 Heuristic algorithms for multiprocessor task scheduling in a two-stage hybrid flow-shop *European Journal of Operational Research* 149 390-403
[7] Jungwattanakit J, Reodecha M, Chaovalitwongse P and Werner F 2009 A comparison of scheduling algorithms for flexible flowshop problems with unrelated parallel machines, setup times, and dual criteria *Computers & Operations Research* 36 358-78
[8] Rossi A, Pandolfi A and Lanzetta M 2014 Dynamic set-up rules for hybrid flowshop scheduling with parallel batching machines *International Journal of Production Research* 52 3842-57
[9] Kirkpatrick S, Gelatt C D and Vecchi M P 1983 Optimization by simulated annealing *science* 220 671-80