Scheduling a variant of flowshop problems to minimize total tardiness

B. Naderi¹, A. Arshadi²

Department of Industrial Engineering, Faculty of Engineering, University of Kharazmi, Karaj, Iran

Corresponding author: B. Naderi

ABSTRACT: This paper studies no-wait flowshop scheduling problems to minimize total tardiness. This paper mathematically formulates the problem as an effective mixed integer linear programming model. Moreover, this paper proposes solution methods based on variable neighborhood search and simulated annealing algorithms. Besides, some well-known heuristics of classical flow shops are adapted to the case of no-wait flow shops. To evaluate the algorithms, two experiments are established. First, by a set of small-sized instances, we evaluate the performance of the algorithms against the optimal solutions obtained by the model. Next, by a set of large-sized instances, we compare the algorithms. The results show that the proposed model and solution methods are effective.

Keywords: No-wait flowshop scheduling, Variable neighborhood search algorithm, Simulated annealing, Mixed integer linear programming model

INTRODUCTION

Production scheduling is centered on allocating available finite capacity resources to operate a set of jobs in such a way that technological constraints are all met and one/several objective(s) is/are optimized. Among different production scheduling problems, no-wait flowshop scheduling (NFS) can be defined by a set of n jobs that need to be processed by a set of m machines starting on machine 1, then machine 2 up until machine m. That is, jobs visit machines by the same order. Therefore, in NFS, there is one decision to make, the determination of job sequence on each machine (Pinedo, 2008). It is obvious that the solution of NFS necessarily is permutation flowshop-like. In other words, the job sequence on all machines is the same. Therefore, in NFS, the job sequence only on machine 1 needs to be obtained. The following assumptions are usually characterized to NFS. All the jobs are independent and available for their process at time zero. All machines are continuously available. Each machine can at most process one job at a time. Each job can be processed by at most one machine at a time. The process of a job on a machine cannot be interrupted. There are infinite buffers between all machines. Transportation and setup times are negligible. The objective is to minimize total tardiness. According to three folds notation of Graham et al. (1979), the mentioned problem of NFS can be classified as F/n/wrt/TT.

NFS may occur in many industries. Typical application of no-wait scheduling problems arises in hot metal rolling industries, where the heated metal has to undergo a set of operations at continuously high temperatures before it is cooled in order to prevent defects. Similarly, in the plastic molding and silverware production industries, a set of operations must be performed to immediately follow one another to prevent degradation. Other examples include chemical and pharmaceutical industries, food processing industries, concrete ware production and advanced manufacturing environments (Li et al., 2008; Grabowski and Pempera, 2000; Raaymakers and Hoogeveen, 2000). For a detailed explanation of the applications of no-wait scheduling problems, papers by Goyal and Sriskandarajah (1998) and Hall and Sriskandarajah (1996) could be further references.

Earliest studies on NFS could be papers by Wismer(1972), Liesegan and Ruger(1972)and Gupta (1976). They deal with presenting exact methods for the special case of two-machine SONFS. Röck (1984) shows that the NFS is an NP-hard problem. Therefore, to solve the problem, stochastic algorithms including heuristics and metaheuristics are of interest as well. Some heuristics are proposed for NFS to minimize makespan by Bonney and Gundry (2010), Li et al. (2008) and Rajendran and Chaudhuri (1990), King and Spachis (1980). With regard to metaheuristics for NFS, some different genetic algorithms are applied by Chen and Neppalli (1996), Aldowaisan and Allahverdi (2004).
and Schuster and Framinan (2003). Among the other metaheuristics, a particle swarm optimization by Pan et al. (2008), simulated annealing by Fink and Voß (2003), ant colony optimization by Shyu et al. (2004) and tabu search by Grabowski and Pempera (2005) are also developed.

First, the paper mathematically formulates the problem as an effective mixed integer linear programming model. Then, the models are solved using the fuzzy multiple criteria decision making approach (Carlsson and Fuller, 2003). Furthermore, the paper proposes high performing solution methods based on variable neighborhood search and simulated annealing as well as adaptations of some well-known heuristics in the literature. To evaluate the performance of the proposed algorithm, we utilize two sets of instances. First, by a set of small-sized instances, we evaluate the performance of the algorithms against the optimal solutions obtained by the model. Next, by a set of large-sized instances, we compare the algorithms.

The rest of the paper is organized as follows. Section 2 formulates the problem under consideration. Section 4 presents solution methods. Section 5 describes the experimental design to evaluate the proposed algorithms. Finally, Section 6 gives some interesting conclusions and future studies.

The mathematical model for no-wait flowshop scheduling problems

By mathematical models, all the characteristics of a scheduling problem could be explicitly described (Pan, 2007). Moreover, mathematical models are used in many solution methods such as branch and bound, dynamic programming and branch and price. As a result, presentation of effective models is of interest (Matta, 2009; Naderi and Ruiz, 2010; Pan, 2007). This paper develops one mixed integer linear programming model for NFS problems. In NFS, solving the problem is equivalent to sequencing jobs on the first machine. Therefore, the model must only determine the job sequence. The following parameters and indices are used in both models.

- $n$: Number of jobs
- $m$: Number of machines
- $j, k$: Indices for jobs/positions where $j, k = \{1, 2, \ldots, n\}$
- $i$: Indices for machines where $i = \{1, 2, \ldots, m\}$
- $p_{j,i}$: Processing time of job $j$ on machine $i$
- $d_j$: Due date of job $j$
- $M$: A large positive number

The following binary variables are defined.

- $X_{j,k}$: Binary variable that takes value 1 if job $j$ is processed immediately after job $k$; and 0 otherwise, where $j \neq k$.
- $C_{k,i}$: Continuous variable for the completion time of the job in position $k$ on machine $i$.
- $T_k$: Continuous variable for the tardiness of the job in position $k$.

The model formulates the problem as follows.

Minimize $Z = \sum_{k=1}^{n} T_k$ (1)

Subject to:

\[
\sum_{k=1}^{n} X_{j,k} = 1 \quad \forall j
\]

\[
\sum_{j=1}^{n} X_{k,j} \leq 1 \quad \forall k \in \{1, 2, \ldots, n\}
\]

\[
\sum_{j=1}^{n} X_{0,j} = 1
\]

\[
X_{j,k} + X_{j,k} \leq 1 \quad \forall k \in \{1, 2, \ldots, n-1\}, j > k
\]

\[
C_{j,1} \geq p_{j,1}
\]

\[
C_{j,i} = C_{j,i-1} + p_{j,i}
\]

\[
C_{j,i} \geq C_{k,i} + p_{j,i} + (X_{k,j} - 1) \times M
\]

\[
T_j \geq C_{j,m} - d_j
\]

\[
C_{j,i} \geq 0 \quad \forall j, i
\]

\[
T_j \geq 0 \quad \forall j
\]

\[
M \geq 0
\]
In (1), total tardiness is calculated. Constraint set (2) states that every job is scheduled once. Constraint set (3) ensures that every job must have at most one succeeding job, while Constraint set (4) enforces that dummy job 0 must have exactly one successor. Constraint set (5) certifies that a job cannot be at the same time both predecessor and successor of another job. Constraint set (6) ensures that completion time of each job on machine 1 is greater than its corresponding processing time. Constraint set (7) ensures that $0_{i-1}$ immediately starts after completion of $0_i$. Constraint set (8) assures that if $0_i$ cannot begin before $0_{k, i}$ completes. Constraint set (9) is considered to calculate the makespan. Ultimately, Constraint sets (10) and (11) define the decision variables.

To formulate a problem with $n$ jobs and $m$ machines, the model needs $n^2$ binary variables, $nm + n$ continuous variables and $\frac{3}{2}(n^2 + n) + nm + 1$ constraints.

### The proposed algorithm

**Variable neighborhood search (VNS)**

Variable neighborhood search is a recently proposed metaheuristic to solve combinatorial optimization problems. There are many studies reporting the successful performance of VNS in different fields (Amiri et al., 2009; Liao and Cheng, 2007). VNS is a local search based algorithm centering on the principle of systematically exploring more than one type of neighborhood search structure (NSS) during the search. Contrary to the other local search based metaheuristics (such as tabu search, simulated annealing and iterated local search) which iterate over one constant type of NSS, VNSs make use of some NSSs until one stopping criterion is met. The motivations behind the concept of VNSs are: 1) A local optimum with respect to one neighborhood structure is not necessarily the local optimum with respect to another neighborhood structure. 2) A global optimum is the local optimum with respect to all neighborhood structures.

By exploring multi-NSS, we expect VNS to enjoy a systematic diversification mechanism. Besides this diversification mechanism, the other reasons to high acceptability of VNSs are conceptual simplicity to understand and implement as well as brilliant adaptability to different problems (Liao and Cheng, 2007). Consequently, we have been thinking of evaluating its performance in our problem. We develop a VNS with two types of NSSs based on insertion neighborhoods.

VNS improves its current solution by the means of different NSSs. The proposed VNS utilizes two advanced types of NSSs based on insertion operators. We do so because insertion operators are regarded as effective operators in the field of scheduling and it is shown that the insertion operator outperforms swap or exchange operators (Osman and Potts, 1989; Ruiz et al., 2005). Through the insertion operator, we can produce a sequence $\theta$ by changing the position(s) of a known number of jobs from current solution $\theta$.

The first NSS could be explained as follows: jobs, one by one, random and without repetition, are removed from current sequence $\theta$. Then each removed job is inserted in all the possible positions among the other $n - 1$ jobs. If any improvement is found, the new sequence is accepted and NSS 1 restarts. The whole procedure repeats so long as no improvement is obtained through reinserting all the jobs.

Since all the possible sequences obtained by relocating only one job have been considered, it can be concluded that there is no room for further improvement by NSS 1. Therefore, the necessity of applying NSS 2 is fulfilled. In the sight of the fact that we need another NSS to escape from the local optimum of NSS 1, NSS 2 must be defined so as to generate farther neighbors than just changing the position of one job. In NSS 2, we make use of an operator whose function is to randomly relocate the position of two randomly selected jobs. For a fixed number of times, we apply this operator to current solution $\theta$, and the best produced solution is accepted even if it is inferior to the current solution. Therefore, the algorithm is obliged to accept one move in NSS 2. Figure 1 shows the general outline of the proposed VNS.

![Procedure:VNS](image)

**Figure 1. General outline of the proposed VNS**
Simulated annealing (SA)

Simulated Annealing was first presented as a search algorithm for combinatorial optimizations problems in (Kirkpatrick et al. 1983; Cerny et al. 1985). SA got its existence from the physical annealing of solid metal. In annealing, a metal is first heated to a high temperature and then cooled down with a very slow rate to the room temperature. Sometimes, if cooling is not slow enough, quenching is done. In SA, solutions are randomly generated from a set of feasible solutions. This process accepts not only those solutions, which improved the objective function, but also those solutions, which do not improve objective function on the basics of transition probability (TP). Transition probability depends on the change in objective function and the annealing temperature. The main features of SA that make this algorithm more sophisticated are perturbation annealing schedule and transition probability. Perturbation generates a new solution, annealing schedule control the initial temperature, final temperature and rate of cooling while TP help heuristic to escape local optima. SA is commonly said to be the oldest among the meta-heuristics and surely one of the first algorithms that had an explicit strategy to avoid local optima.

The fundamental idea is to generate a new job sequence \( s \) by a random rule from the neighborhood of present sequence \( x \). This new sequence is accepted or rejected by another random rule. A parameter \( t \), called the temperature, controls the acceptance rule. The variation between objective values of two candidate solution is computed \( \Delta C = \text{TT}(s) - \text{TT}(x) \). If \( \Delta C \leq 0 \), sequence \( s \) is accepted. Otherwise, sequence \( s \) is accepted with probability equal to \( \exp(-\Delta C/t_k) \). The algorithm proceeds by trying a fixed number of neighborhood moves (max) at temperature \( t_k \), while temperature is gradually decreased. The procedure is repeated until a stopping criterion is met.

Moves resulting in solutions of worse quality (uphill move) than the current solution may be accepted to escape from local optima. SA stared at a high temperature \( (T_0) \), so most of the moves are accepted at first steps of the procedure. The probability of doing such a move is decreased during the search. Figure 2 shows the general outline of SA.

Simulated annealing starts from an initial solution, and a series of moves are made according to a user-defined annealing schedule. In this research, the initial solution is obtained using NEHEDD algorithm (Ruiz and Stützle, 2007; Nawaz et al. 1983). The algorithm checks 100 neighbors at temperature \( t_k \) (max = 100). Moving operator generates a neighbor solution from current candidate solution by making a slight change in it. These operators must work in such way that avoids infeasible solutions. In this research, we take into consideration “single point operator (or SPO)”. In SPO, one randomly selected job is randomly relocated.

As earlier mentioned, to avoid local optima, the solutions with worse objective values are probably accepted depending on the value of temperature. When the procedure proceeds, the temperature is slightly lowered under a certain mechanism which is called cooling schedule. Here, we use exponential cooling schedule, \( t_k = \alpha \cdot t_{k-1} \) (where \( \alpha \in (0, 1) \) is temperature decrease rate), which is often believed to be an excellent cooling recipe (Schneider et al., 1998).

Hybridization of SA and VNS

Simulated annealing (SA) is considered as a fast metaheuristic approach with intensifying operators among the local-search-based metaheuristics. Therefore, we intend to examine the hybridization of the VNS and an SA algorithm. In this case, our hybrid algorithm has both the diversification technique to investigate new and unknown areas in the search space, and the intensification technique to make use of the knowledge obtained at solutions previously visited to aid to find promising solutions. Moreover, the combination of the flexible and effective

---

**Figure 2. Pseudo code of SA**

```
ProcedureSA
  Initialization
  while stopping criterion is not met do
    for i = 1 to maxdo
      Generate a new neighbor from current solution
      Acceptance criterion
      Update the best solution so far found
    endfor
  Temperature reduction
endwhile
```
population-based algorithm to search the optimal solution and the convergent characteristics of SA provides the rationale for developing a hybrid algorithm (which is called VNSA).

In this research, we use the SA algorithm as NSS 1. In other words, the initial solution of the SA algorithm is the current solution of VNS. Then instead of checking relocation of all the jobs into all the possible position, we use SA-like procedure. The temperature is fixed during the search. At each time, the SA is applied, the algorithm checks 100 neighbors. Each new sequence (s) is accepted if it is better than the current sequence (x) or with probability equal to \( \exp(-\Delta C/t) \) where \( \Delta C = T_T(s) - T_T(x) \). After checking 100 neighbors, the NSS 2 of the VNS is applied.

<table>
<thead>
<tr>
<th>No.</th>
<th>n</th>
<th>m</th>
<th>Instance</th>
<th>Obj</th>
<th>Time (S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>423</td>
<td>0.09</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
<td>49</td>
<td>0.05</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
<td></td>
<td>290</td>
<td>0.08</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td></td>
<td>108</td>
<td>0.08</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>1</td>
<td></td>
<td>247</td>
<td>0.05</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>2</td>
<td></td>
<td>275</td>
<td>0.09</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>2</td>
<td>1</td>
<td>272</td>
<td>0.26</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>2</td>
<td></td>
<td>611</td>
<td>0.7</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>1</td>
<td></td>
<td>271</td>
<td>0.69</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>2</td>
<td></td>
<td>218</td>
<td>0.13</td>
</tr>
<tr>
<td>11</td>
<td>4</td>
<td>1</td>
<td></td>
<td>78</td>
<td>0.11</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>2</td>
<td></td>
<td>439</td>
<td>0.39</td>
</tr>
<tr>
<td>13</td>
<td>7</td>
<td>2</td>
<td>1</td>
<td>804</td>
<td>5.43</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>2</td>
<td></td>
<td>651</td>
<td>5.05</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>1</td>
<td></td>
<td>292</td>
<td>3.65</td>
</tr>
<tr>
<td>16</td>
<td>3</td>
<td>2</td>
<td></td>
<td>192</td>
<td>0.83</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
<td>1</td>
<td></td>
<td>646</td>
<td>5.76</td>
</tr>
<tr>
<td>18</td>
<td>4</td>
<td>2</td>
<td></td>
<td>893</td>
<td>6.69</td>
</tr>
<tr>
<td>19</td>
<td>8</td>
<td>2</td>
<td>1</td>
<td>1198</td>
<td>77.03</td>
</tr>
<tr>
<td>20</td>
<td>2</td>
<td>2</td>
<td></td>
<td>534</td>
<td>54.66</td>
</tr>
<tr>
<td>21</td>
<td>3</td>
<td>1</td>
<td></td>
<td>737</td>
<td>51.54</td>
</tr>
<tr>
<td>22</td>
<td>3</td>
<td>2</td>
<td></td>
<td>896</td>
<td>80.75</td>
</tr>
<tr>
<td>23</td>
<td>4</td>
<td>1</td>
<td></td>
<td>1286</td>
<td>87.99</td>
</tr>
<tr>
<td>24</td>
<td>4</td>
<td>2</td>
<td></td>
<td>1848</td>
<td>81.67</td>
</tr>
<tr>
<td>25</td>
<td>9</td>
<td>2</td>
<td>1</td>
<td>973</td>
<td>1163.2</td>
</tr>
<tr>
<td>26</td>
<td>2</td>
<td>2</td>
<td></td>
<td>1033</td>
<td>798.84</td>
</tr>
<tr>
<td>27</td>
<td>3</td>
<td>1</td>
<td></td>
<td>1057</td>
<td>1267.6</td>
</tr>
<tr>
<td>28</td>
<td>3</td>
<td>2</td>
<td></td>
<td>1348</td>
<td>1368</td>
</tr>
<tr>
<td>29</td>
<td>4</td>
<td>1</td>
<td></td>
<td>953</td>
<td>1125.6</td>
</tr>
<tr>
<td>30</td>
<td>4</td>
<td>2</td>
<td></td>
<td>1652</td>
<td>1166</td>
</tr>
</tbody>
</table>

Experimental evaluation

This section evaluates the performance of the proposed model and algorithms. First, the model’s capability to solve the problem under consideration is experimented. In this stage, we use small-sized instances. Through this experiment, general performance of the algorithms is evaluated against the optimal solution of the mathematical model. Next, we establish another experiment including large-sized instances. Through this experiment, we further compare the algorithms.

The algorithms are implemented in Borland C++ and run on a PC with Intel Core 2 Duo and 2 GB of RAM memory. We use Relative Deviation Index (or RDI) as a performance measure to compare the methods, because RPI fulfills some drawbacks of Relative Percentage Deviation (or RPD) in case of the tardiness objectives (for more details see Vallada et al., 2008). When TTS of tested algorithms are obtained for a given instance, the best and worst solutions (which are named \( \text{Min}_{\text{sol}} \) and \( \text{Worst}_{\text{sol}} \), respectively) by any of the algorithms are calculated. RDI is obtained by the given formula below:

\[
\text{RDI} = \frac{\text{Alg}_{\text{sol}} - \text{Min}_{\text{sol}}}{\text{Worst}_{\text{sol}} - \text{Min}_{\text{sol}}} \times 100
\]  

where \( \text{Alg}_{\text{sol}} \) is the TT obtained for a given algorithm and instance. Obviously, RDI takes value between 0 and 100. The stopping criterion is set to as a fixed computational time of \( n \times m \times 0.25 \) seconds. In addition to the proposed VNS, SA and VNSA, the NEH and EDD algorithms are brought into the experiment.
Small-sized instances

This subsection intends to assess the general performance of the MILP model as well as the proposed algorithm. To this end, a set of 36 instances are generated. In these instances, there are two instances for each \( n \) and \( m \) combination.

\[
n = \{5,6,7,8,9,10\}; \quad m = \{2,3,4\}
\]

The processing times are uniformly distributed over the range \((1, 99)\). The due dates are produced by the following formula

\[
d_j = (1 + 3 \times \text{random}) \sum_{i=1}^{m} p_{ji}
\]

where \text{random} is a random number from a uniform distribution over \([0, 1]\). We implement the MILP model in CPLEX 10 and run it with a time limit of 1500 seconds.

Table 1 shows the results of the experiment. The MILP model is capable of solving 30 out of 36 instances, i.e., instances up to size \( n = 9 \) and \( m = 4 \) within 1200 seconds computational time. Instances with less than 6 jobs are solved to optimality in less than 0.01 second. Instances with 7 jobs are solved within 10 seconds.

Now, we evaluate the performance of the algorithms against the optimal solutions of the MILP model. Table 2 shows the results. VNSA optimally solves all 30 instances. The second best is VNS with RPI of 1.9%. Moreover, EDD outperforms NEH.

| n | m | Algorithms |  |  |  |  |  |  |  |
|---|---|-----------|---|---|---|---|---|---|
|   |   | SA        | VNSA | VNS | VNA | NEH | EDD |
| 5 | 2 | 16.67     | 0.00 | 0.00 | 50.00 | 22.09 |
| 6 | 2 | 12.53     | 0.00 | 0.00 | 59.32 | 68.69 |
| 7 | 2 | 14.04     | 0.00 | 0.00 | 100   | 33.99 |

Table 3. PDI of algorithms evaluation on large-sized instances

<table>
<thead>
<tr>
<th>N</th>
<th>m</th>
<th>Algorithms</th>
<th>SA</th>
<th>VNSA</th>
<th>VNS</th>
<th>VNA</th>
<th>NEH</th>
<th>EDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>5</td>
<td>13.43</td>
<td>4.83</td>
<td>10.06</td>
<td>25.65</td>
<td>96.79</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>5</td>
<td>13.36</td>
<td>3.28</td>
<td>11.78</td>
<td>20.65</td>
<td>80.52</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>5</td>
<td>10.66</td>
<td>3.16</td>
<td>9.78</td>
<td>20.65</td>
<td>79.79</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>11.27</td>
<td>4.67</td>
<td>9.42</td>
<td>17.89</td>
<td>92.47</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>10.81</td>
<td>5.08</td>
<td>10.49</td>
<td>15.40</td>
<td>95.24</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For the further analysis, we carry out the ANOVA. The related results show that there is statistically significant difference between the performances of the algorithms with \( p \)-value very close to zero. Figure 3 shows the means plot for the algorithms with LSD intervals. Clearly, the proposed VNSA provides statistically better results than other methods.
Large-sized instances

This subsection tests the performance of 5 algorithms (i.e. VNS, SA, VNSA, EDD and NEH) on an existing benchmark for classical permutation flowshop scheduling problem (Ruiz and Stützle, 2005). The instances of these instances are those of Taillard (1993) with inclusion of due dates. There are 10 instances for each of the 11 following sizes \((n; m)\): \((n = 20, 50, 100; m = 5, 10, 20)\) and \((n = 20; m = 10, 20)\). Therefore, we have 130 instances.

The average PDI values for all the algorithms obtained in different problem sizes are presented in Table 3. The results the superiority of VNSA in solving no-wait flowshops comparing to the other algorithms. PDI of VNSA is 4.46% while the VNS obtains the PDI of 9.84%. The worst performing algorithm is EDD with PRI of 90.95%.

![Figure 3. The means plot and LSD intervals of the algorithms](image)

To analyze the performance of the algorithms in different problem sizes, the average PDI of algorithms in different sizes are plotted in Figure 4. As it is clear, VNSA keeps its robust performance in all sizes. NEH works better in larger sizes.

![Figure 4. The means plot of the algorithms in different problem sizes](image)

CONCLUSION AND FUTURE WORK

In this paper, we studied a case of no-wait flowshops. The optimization criterion was the minimization of the total tardiness. We first formulated the problem by a mathematical model in the form of a mixed integer linear program. Due to the built-in complexity existing in such combinatorial optimization problems, the mathematical
models were not an applicable solution strategy for large-sized problems. In this case, we presented three metaheuristic algorithm for the considered problem. This algorithm was variable neighborhood search and simulated annealing. We also applied the adaptation of two well-known heuristics, NEH and EDD, existing in the literature.

To evaluate the performance of the proposed model and metaheuristics, we conducted two experiments. In the first one, we assessed the efficiency of the model on a set of small-sized instances and general performance of tested algorithms against the optimal solutions obtained by the model. In the second experiment, we compared the performance of the proposed algorithms. The results illustrated that the proposed algorithm are effective for the problem of solving the studied problem. As a future study, it would be interesting to apply other solution strategies including exact and approximation ones for the problem studied here. Multi-objective cases of the problem could be explored as well. As another direction, the performance of the other well-known algorithms such as electromagnetism-like, artificial immune systems, can be examined.

REFERENCES


