AN EFFECTIVE CONSTRUCTIVE HEURISTIC FOR THE NO-WAIT FLOWSHOP WITH SEQUENCE-DEPENDENT SETUP TIMES PROBLEM

Daniella Castro Araujo (USP)  
daniella.araujo@usp.br  
Marcelo Seido Nagano (USP)  
drnagano@usp.br

This paper presents a new heuristic named GAPH based on a structural property for minimizing makespan in no-wait flowshop with sequence-dependent setup times. Experimental results demonstrate the superiority of the proposed approach over three of the best-known methods in the literature. Experimental and statistical analyses show that the new heuristic provides better solutions regarding the solution quality and computational effort.

Palavras-chaves: Scheduling, Heuristic, No-wait flowshop, Sequence-dependent setup, Makespan.
1. Introduction

The first systematic approach to scheduling problems was undertaken in the mid-1950s. Since then, thousands of papers on different scheduling problems have appeared in the literature. The majority of these papers assumed that the setup time is negligible or part of the job processing time. While this assumption simplifies the analysis and reflects certain applications, it adversely affects the solution quality of many applications of scheduling that require an explicit treatment of setup times (ALLAHVERDI et al., 2008).

There are two types of setup time: sequence-independent and sequence-dependent. If setup time depends solely on the task to be processed, regardless of its preceding task, it is called sequence-independent. On the other hand, in the sequence-dependent type, setup time depends on both the task and its preceding task (ALLAHVERDI and SOROUSH, 2008).

In today's scheduling problems in both manufacturing and service environments it is of significance to efficiently utilize various resources. Treating setup times separately from processing times allows operations to be performed simultaneously and hence improves resource utilization. This is, in particular, important in modern production management systems such as just-in-time (JIT), optimized production technology (OPT), group technology (GT), cellular manufacturing (CM), and time-based competition (ALLAHVERDI et al., 2008).

Another important area in scheduling arises in no-wait flowshop problems, where jobs have to be processed without interruption between consecutive machines. There are several industries where the no-wait flowshop problem applies including the metal, plastic, and chemical industries. For instance, in the case of steel production, the heated metal must continuously go through a sequence of operations before it is cooled in order to prevent defects in the composition of the material (ALDOWAISAN, 2001). As noted by Hall and Sriskandarajah (1996), the first of two main reasons for the occurrence of a no-wait or blocking production environment lies in the production technology itself. In some processes, for example, the temperature or other characteristics (such as viscosity) of the material require that each operation follow the previous one immediately. According to Bianco et al. (1999), flowshop no-wait scheduling problems are also motivated by concepts such as JIT and zero inventory in modern manufacturing systems.
Figure 1 – No-wait flowshop with $m$ machines and $n$ jobs

The main feature of the no-wait flowshop is that the task operation $i + 1$ must be processed soon after the end of operation $i$, where $1 \leq i \leq m - 1$. Thus, there can be no waiting time in processing a job from one machine to the next. An example of the no-wait flowshop with sequence-dependent setup times problem is shown in Figure 1.

Figure 1 shows the scheduling problem of $n$ jobs on $m$ machines, considering that all tasks are available at the same time on the factory floor. A solution can be represented by $\sigma = (j_1, j_2, j_3, ..., j_n)$, where $j_i$ is the task that appears in the $i$-th position of the sequence.

The no-wait flowshop scheduling problem consists of a set $J = \{j_1, j_2, j_3, ..., j_n\}$ of $n$ jobs which are to be processed on a set $M = \{m_1, m_2, m_3, ..., m_m\}$ of $m$ dedicated machines, each one being able to process only one job at a time. Job $j_i$ consists of $m$ operations $op_{k_1}, op_{k_{i+1}}, ..., op_{k_m}$, to be executed in this order, where operation $op_{k_i}$ must be executed on machine $k_i$ with $p_{k_i}$ processing time. Furthermore, operation $op_{k_{i+1}}$ must start immediately after operation $op_{k_i}$ is completed.

Moreover, to execute operation $op_{k_{i+1}}$, the machine $k$ requires a sequence-dependent setup $s_{i_{j+1}}$, if operation $op_{k_i}$ is processed immediately before $op_{k_{i+1}}$.

Figure 2 shows an example of the scheduling problem. In particular, an instance with 2 machines and 3 jobs is represented. For that instance, a feasible solution and an optimal solution are represented.
This paper addresses the \( m \)-machine no-wait flowshop problem to minimize makespan where setup times are separated and sequence-dependent (\( Fm/ST_{sd}, no \text{-} wait / C_{\text{max}} \)). As this problem is proved to be \( NP \)-hard (BIANCO et al., 1999), we propose a heuristic solution algorithm and compare it to Bianco et al. (1999) algorithms, BAH and BIH, and to Brown et al. (2004) algorithm, TRIPS, adapted to this problem.

This paper is organized as follows. In Section 2, a review of the no-wait flowshop with sequence-dependent setup times problem is provided. In Sections 3 and 4, we describe the set of constructive heuristics available for the problem and the new heuristic proposed, respectively. In Section 5, we test the new heuristic effectiveness. Finally, conclusions and final considerations are given in Section 6.

2. Literature review

Gupta (1986) formulated the \( Fm/ST_{sd} / C_{\text{max}} \) problem as an ATSP (Asymmetric Travelling Salesman Problem) and showed that the flowshop scheduling problems with sequence-dependent setup times are strongly \( NP \)-hard for the zero, limited, or infinite intermediate storage space cases.

Stafford and Tseng (1990) developed three new MILP (Mixed-Integer Linear Programming) formulations for regular flowshop with setup times problems and for no-wait flowshop problems with sequence-independent/dependent and setup times to minimize the mean flowtime or the makespan.

Bianco et al. (1999) considered the \( Fm/ST_{sd}, no \text{-} wait, r_j / C_{\text{max}} \) problem and showed that it is equivalent to the ATSP-RT (Asymmetric Travelling Salesman Problem with Ready Times), which is strongly \( NP \)-hard. They also presented two lower bounds, named LB1 and LB2, in which they dualized the ready time constraints and modified the ATSP cost matrix, respectively. Besides, they proposed two heuristic algorithms, BAH – Best Adding Heuristic, which a feasible sequence of jobs is obtained by adding jobs at the end of a partial sequence, and BIH – Best Insertion Heuristic, in which a feasible sequence is obtained by inserting jobs in a partial sequence. Computational results showed that the heuristics have acceptable performances in almost all of the test problems, and BIH always gives better solutions, although its computational time is high. Lower bounds behavior is strictly related to the range in which the ready times are assigned and to the size of the problem instances. In particular, in congested cases, LB1 generally gives greater values than the values of LB2; vice-versa, in non-congested cases.

Allahverdi and Aldowaisan (2001) considered the \( F2/ST_{sd}, no \text{-} wait / \sum C_j \) problem. They showed that certain sequences are optimal if certain conditions hold. Moreover, they developed a dominance relation and presented five heuristic algorithms with the computational complexities of \( O(n^2) \) and \( O(n^3) \). The heuristics consist of two phases; in the first phase a starting sequence is developed, and in the second a repeated insertion technique is applied to get a solution. They also developed a branch-and-bound algorithm, in order to test the effectiveness of the heuristics. Computational experiments demonstrated that the concept of repeated insertion application is quite useful for any starting sequence, and that the solutions for all the starting sequences converge to flowshop with sequence-dependent setup times and non-zero transfer times and mixed-storage policies.

Stafford and Tseng (2002) proposed two MILP models, referred as WST and SGST. Each model may be used to solve the regular, no-wait, sequence-dependent setup times, and no-
wait with sequence-dependent setup times flowshop problems. The optimal job sequence, based on minimizing makespan, is achieved by a positional assignment approach in the WST model, and by pairs of disjunctive (either–or) constraints in the SGST model. A 12-cell experimental design was used to compare the solution times of the two models for each of the four flowshop problems. The WST model used significantly less computer time than the SGST model for all problem sizes of problems. Both models were competitive when \( n \leq 7 \), but the WST model proved significantly better for \( n \geq 8 \).

França et al. (2006) considered the same problem as Bianco et al. (1999) and solved it by an evolutionary approach. They presented a hybrid genetic algorithm that addresses a new hierarchically organized complete ternary tree to represent the population that put together with a recombination plan resembles a parallel processing scheme for solving combinatorial optimization problems. They also proposed a novel recursive local search scheme, named RAI (Recursive Arc Insertion), which effectiveness was crucial, given that it was responsible for about 90% of the total processing time of the algorithm. In the computational experiments, they showed that the algorithm proposed obtains better solutions than BIH, with lower computational effort.

Table 1 summarizes the researches involving the no-wait flowshop with sequence-dependent setup times problem.

<table>
<thead>
<tr>
<th>Reference</th>
<th># of stages</th>
<th>Criterion</th>
<th>Comments</th>
<th>Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gupta (1986)</td>
<td>( m )</td>
<td>( C_{\text{max}} )</td>
<td>Also considered the limited and infinite buffer problem</td>
<td>Proved that the problem is ( NP )-complete/ Mathematical Formulation (TSP)</td>
</tr>
<tr>
<td>Stafford e Tseng (1990)</td>
<td>( m )</td>
<td>( C_{\text{max}} ), ( F )</td>
<td>Formulations for: flowshops with setup times and no-wait flowshops with sequence-independent/dependent setup times problems</td>
<td>3 MILP Formulations</td>
</tr>
<tr>
<td>Bianco et al. (1999)</td>
<td>( m )</td>
<td>( C_{\text{max}} )</td>
<td>Ready Times</td>
<td>2 Lower Bounds/ Mathematical Formulation (ATSP-RT)/ 2 Heuristics</td>
</tr>
<tr>
<td>Allahverdi e Aldowaisan (2001)</td>
<td>2</td>
<td>( \Sigma C_j )</td>
<td>-</td>
<td>Dominance Relation/ Mathematical Formulation/ 5 Heuristics/ Branch-and-Bound</td>
</tr>
<tr>
<td>Sittaford e Tseng (2002)</td>
<td>( m )</td>
<td>( C_{\text{max}} )</td>
<td>Formulations for: regular, no-wait, sequence-dependent setup times and no-wait with sequence-dependent setup times flowshop problems</td>
<td>2 MILP Formulations</td>
</tr>
<tr>
<td>França et al. (2006)</td>
<td>( m )</td>
<td>( C_{\text{max}} )</td>
<td>Ready Times</td>
<td>Hybrid Genetic Algorithm/ Local Search Scheme</td>
</tr>
</tbody>
</table>

Table 1 – Summary of the researches on no-wait flowshop with sequence-dependent setup times

3. **Existing constructive heuristics for the problem**

In this section, we review the main contributions to the problem regarding constructive methods. More specifically, we explain in detail the constructive heuristics BAH and BIH, from Bianco et al. (1999), and TRIPS, from Brown et al. (2004).

3.1. BAH
BAH algorithm finds a feasible sequence in \( n \) iterations. At each iteration, given a partial sequence of the scheduled jobs computed in the previous iteration, the algorithm examines a set of candidates of the unscheduled jobs, and appends a candidate job to a partial sequence minimizing the time when the shop is ready to process an unscheduled job.

The pseudo-code of the heuristic is as follows:

Given a set \( J = \{j_1, j_2, j_3, \ldots, j_n\} \) of \( n \) jobs, let \( \sigma \) be the set of programmed jobs and \( U \) be the set of non-programmed jobs.

**Step 1:** \( U \leftarrow J ; \sigma \leftarrow \emptyset \);

**Step 2:** While \( U \neq \emptyset \), do:

**Step 2.1:** Choose the job \( j_i \in U \) to be added at the end of the sequence \( \sigma \), such that the makespan is minimum;

**Step 2.2:** Add job \( j_i \) to the end of the sequence \( \sigma \);

**Step 2.3:** \( U \leftarrow U - j_i \).

### 3.2. BIH

The BIH algorithm also finds a sequence of \( n \) jobs on \( n \) iterations. But in this algorithm, at each iteration it considers a sequence of a subset of jobs, and finds the best sequence obtained inserting an unscheduled job in any position of the given sequence.

A more detailed description of the heuristic is as follows:

Given a set \( J = \{j_1, j_2, j_3, \ldots, j_n\} \) of \( n \) jobs, let \( \sigma \) be the set of programmed jobs, \( U \) be the set of non-programmed jobs and \( h \) the relative insertion position.

**Step 1:** \( U \leftarrow J ; \sigma \leftarrow \emptyset \);

**Step 2:** While \( U \neq \emptyset \), do:

**Step 2.1:** Choose the job \( j_i \in U \) which can be inserted in the sequence \( \sigma \), such that the makespan is minimum. Let \( h \) be the relative insertion position;

**Step 2.2:** Insert job \( j_i \) at position \( h \) in the sequence \( \sigma \);

**Step 2.3:** \( U \leftarrow U - j_i \).

### 3.3. TRIPS

TRIPS heuristic was developed for the no-wait flowshop with sequence-independent setup times, for minimizing total flowtime \( (F_m / ST_{\text{u}}, \text{no-wait} / \sum C_j) \) or makespan \( (F_m / ST_{\text{u}}, \text{no-wait} / C_{\text{max}}) \). In this paper, because there are only BIH and BAH constructive heuristics for the \( F_m / ST_{\text{u}}, \text{no-wait} / C_{\text{max}} \) problem, we will adapt it to this problem.

TRIPS examines all possible three-job combinations from the set of unscheduled jobs \( U \) and chooses the sequence \((j_w, j_x, j_y)\) that minimizes the three-job objective. Then, assigns job \( j_w \) to the last empty position in the sequence \( \sigma \) and removes \( x_w \) from \( U \). The heuristic repeats the process, assigning one more job to \( \sigma \) for each set of triplets examined until only three jobs
are left. Then, it selects the optimal sequence for these jobs and places them in the final positions of heuristic sequence $\sigma$.

The pseudo-code of the heuristic is as follows:

Given a set $J = \{j_1, j_2, j_3, ..., j_n\}$ of $n$ jobs, let $\sigma$ be the set of programmed jobs and $U$ be the set of non-programmed jobs.

**Step 1:** $U \leftarrow J; \sigma \leftarrow \emptyset; h \leftarrow 0$.

**Step 2:** While $h < n-2$, do:

**Step 2.1:** Given that the first $h$ jobs are assigned in sequence $\sigma$, compare all ordered triplets of jobs from $U$;

**Step 2.2:** Choose the triplet $\{j_w, j_x, j_y\}$ such that the performance measure (either makespan or flowtime) is minimized for jobs $\{j_w, j_x, j_y\}$ in positions $h+1$, $h+2$, $h+3$, respectively, of sequence $\sigma$.

**Step 2.3:** Place $j_w$ in position $h+1$ of $\sigma$;

**Step 2.4:** $h \leftarrow h+1; \ U \leftarrow U - j_w$.

**Step 3:**

Assign $j_x$ and $j_y$ to the last two positions, respectively, of $\sigma$.

4. A structural property for the new heuristic

Given a sequence $\sigma$ of $J$, $j_i$ is the job of $J$ occupying position $i$ in $\sigma$. The time break between the beginning of job $j_{i+1}$ and the beginning of job $j_i$ at machine $k$ is $\Delta t^k_{i,i+1}$, calculated as follows:

$$\Delta t^k_{i,i+1} = \max_{1 \leq l \leq m} \left[ s^k_{i,i+1} + \sum_{h=1}^{k} \left( p_{b[i]} - p_{b[i+1]} \right) + p_{k[i+1]} \right]$$

$$\Delta t^k_{i,i+1} = \Delta t^l_{i,i+1} + \sum_{h=1}^{k-1} \left( p_{b[i+1]} - p_{b[i]} \right)$$

Defining $GAP^k_{i,i+1}$ as the time break between the end of job $j_i$ and the beginning of job $j_{i+1}$ at machine $k$, it can be calculated as follows:

$$GAP^k_{i,i+1} = \Delta t^k_{i,i+1} - p_{k[i]}$$

The $GAP$ of the first job in the sequence on machine $k$ is defined by expression (4):

$$GAP^k_{0,1} = \sum_{h=1}^{k-1} p_{a[i]}$$

Figure 3 shows the time break between the end of job $j_2$ and the beginning of job $j_3$ on machine 2 ($GAP^2_{2,3}$).
4.1. The new heuristic

The new heuristic proposed in this paper will be called GAPH – Gap Heuristic. The pseudo-code of the algorithm is given next:

Given a set \( J = \{j_1, j_2, j_3, \ldots, j_n\} \) of \( n \) jobs, let \( U \) be the set of non-programmed jobs and \( \sigma_x = \{j_{x1}, j_{x2}, j_{x3}, \ldots, j_{xn}\} \) be the sequence of \( n \) jobs scheduled, where \( x = \{1,2,3,4\} \). Calculate the \( GAP^k_{ij} \) of each job \( i = 1, \ldots, n \) to each job \( j = 1, \ldots, n \) at all \( m \) machines.

**Step 1:** \( U \leftarrow J; \ \sigma_1 \leftarrow \emptyset; \)

**Step 2:** While \( U \neq \emptyset \), do:

- **Step 2.1:** Calculate the total cost\(^*\) on the last machine for all possible insertions of each job \( j_i \in U \) in the sequence \( \sigma_1 \). Let \( h \) be the relative insertion position;

- **Step 2.2:** Choose the job \( j_i \) that gives the lower total cost at position \( h \);

- **Step 2.3:** Insert job \( j_i \) at position \( h \) of the sequence \( \sigma_1 \);

- **Step 2.4:** \( U \leftarrow U - j_i \);

**Step 3:** \( U \leftarrow J; \ \sigma_2 \leftarrow \emptyset \)

**Step 4:** While \( U \neq \emptyset \), do:

- **Step 4.1:** Calculate the total \( GAP^{**} \) for all possible insertions of each job \( j_i \in U \) in the sequence \( \sigma_2 \). Let \( h \) be the relative insertion position;

- **Step 4.2:** Choose the job \( j_i \) that gives the lower \( GAP^{**} \) at position \( h \);

- **Step 4.3:** Insert job \( j_i \) at position \( h \) of the sequence \( \sigma_2 \);

- **Step 4.4:** \( U \leftarrow U - j_i \);

**Step 5:** \( U \leftarrow J; \ \sigma_3 \leftarrow \emptyset \)

**Step 6:** While \( U \neq \emptyset \), do:

- **Step 6.1:** Calculate the sum of the \( GAPs \) on the last machine for all possible insertions of each job \( j_i \in U \) in the sequence \( \sigma_3 \). Let \( h \) be the relative insertion position;

- **Step 6.2:** Choose the job \( j_i \) that gives the lower sum of the \( GAPs \) on the last machine at position \( h \);

- **Step 6.3:** Insert job \( j_i \) at position \( h \) of the sequence \( \sigma_3 \);
Step 6.4: $U \leftarrow U - j_i$;

Step 7: $U \leftarrow \emptyset$; $\sigma_2 \leftarrow \emptyset$;

Step 8: While $U \neq \emptyset$, do:

Step 8.1: Calculate, for all possible insertions of each job $j_i \in U$ in the sequence $\sigma_4$, the sum of the GAPs with the processing time of the job on the last machine. Let $h$ be the relative insertion position;

Step 8.2: Choose the job $j_i$ that gives the lower sum of the GAPs with the processing time of the job on the last machine at position $h$;

Step 8.3: Insert job $j_i$ at position $h$ of the sequence $\sigma_4$.

Step 8.4: $U \leftarrow U - j_i$;

Step 9: Choose, among the sequences $\{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$ the one with the lower makespan.

*The total cost on a $k$ machine is defined as the scheduling total time on this machine. Thus, the total cost encompasses the sum of the GAPs on machine $k$ with the scheduled operations processing times on that machine. Note that the total cost on the last machine is equivalent to the makespan (see Figure 4).

**The total GAP is the sum of all GAPs in all machines.

In Figure 5, the total GAP is: $GAP^1 \{1\} [2] + GAP^2 \{2\} [3] + GAP^3 \{0\} [1] + GAP^4 \{1\} [2] + GAP^5 \{2\} [3]$.


The sum of the GAPs with the processing time of the job to be inserted on the last machine is: $GAP^2 \{0\} [1] + GAP^2 \{1\} [2] + GAP^2 \{2\} [3] + p_{23}$.
Figure 6 shows how the method works for each pair of steps (1-2; 3-4; 5-6; 7-8). In the example, jobs $j_1$ and $j_2$ were already scheduled ($\sigma = (j_1, j_2)$) and job $j_3$ is scheduled on each possible position of the sequence ($h$).

For example, if the method were on the second step, the sequence chosen would be the third one, that gives the lower total cost on the last machine.

5. Computational results

We carried out an extensive computational experiment in order to test GAPH, as well as BIH, BAH (BIANCO et al., 1999) and TRIPS (BROWN et al., 2004) heuristics. In addition, we also included BIH’, an adaptation made for BIH. BIH’ algorithm consists of steps 1 and 2 of GAPH, giving the same solutions as BIH, with much lower computational effort.

The heuristics were tested in the well-known testbed of Taillard (1993). This testbed contains twelve sets for a given combination of jobs and machines, i.e., $n \{20, 50, 100, 200, 500\}$ and $m \{5, 10, 20\}$. We performed four experiments, one for each of the four different sequence-dependent Taillard-based instance sets from Ruiz et al. (2005). The tests contain four different processing times to sequence-dependent setup times ratios. For example, the instance set SSD-10 is composed of 120 instances where the processing times are those of Taillard’s benchmark and where the sequence-dependent setup times are 10% of the processing times. In the instance set SSD-50, the setup times are 50% of the processing times and the instance sets SSD-100 and SSD-125 have setup times that are 100% and 125% of the processing times respectively. So for example, if the processing times in Taillard’s instances are generated from a uniform distribution in the range $[1; 99]$, in the SSD-10 instance set the setup times are uniformly distributed in the range $[1; 9]$, $[1; 49]$, $[1; 99]$ and $[1; 124]$ for the instance sets SSD-50, SSD-100 and SSD-125 respectively. Thus, we have four problem sets and a total of 480 different instances. The 500 job instance was rather large and we chose to solve only the first 110 instances (up to 200 jobs and 20 machines).
The instances in the testbed have been solved by the selected heuristics (coded in Python) in a computer with a Pentium IV 3.00GHz processor and 512MB RAM.

Table 2 summarizes the result obtained for the different heuristics in terms of the success percentage, the average relative percentage deviation (ARPD), and the average CPU time.

The success rate is defined by the ratio between the number of problems for which a particular method was the best solution and the total number of problems solved. Therefore, when two methods get the best solution for the same problem, their percentages of success are both improved.

The ARPD quantifies the average deviation that method \( h \) obtains compared to the minor makespan obtained for each problem, computed as follows:

\[
ARPD(\%) = \frac{D_h - D^*}{D^*} \times 100
\]

where:

- \( D_h \) is the makespan computed by method \( h \);
- \( D^* \) is the best makespan computed by the methods.

<table>
<thead>
<tr>
<th>( n \times m )</th>
<th>SSD-10</th>
<th>SSD-50</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BAH</td>
<td>BIH</td>
</tr>
<tr>
<td>0,00*</td>
<td>60,00</td>
<td>60,00</td>
</tr>
<tr>
<td>15,54**</td>
<td>1,83</td>
<td>1,83</td>
</tr>
<tr>
<td>0,03***</td>
<td>1,26</td>
<td>0,11</td>
</tr>
<tr>
<td>20x5</td>
<td>0,00</td>
<td>50,00</td>
</tr>
<tr>
<td>20x10</td>
<td>15,87</td>
<td>2,48</td>
</tr>
<tr>
<td>0,06</td>
<td>2,78</td>
<td>0,15</td>
</tr>
<tr>
<td>20x20</td>
<td>13,42</td>
<td>0,42</td>
</tr>
<tr>
<td>0,18</td>
<td>7,79</td>
<td>0,26</td>
</tr>
<tr>
<td>50x5</td>
<td>0,00</td>
<td>56,67</td>
</tr>
<tr>
<td>Média</td>
<td>14,94</td>
<td>1,58</td>
</tr>
<tr>
<td>0,09</td>
<td>3,94</td>
<td>0,17</td>
</tr>
<tr>
<td>50x10</td>
<td>0,00</td>
<td>70,00</td>
</tr>
<tr>
<td>13,42</td>
<td>0,41</td>
<td>0,41</td>
</tr>
<tr>
<td>0,17</td>
<td>55,00</td>
<td>2,87</td>
</tr>
<tr>
<td>0,00</td>
<td>90,00</td>
<td>90,00</td>
</tr>
<tr>
<td>50x20</td>
<td>13,92</td>
<td>0,10</td>
</tr>
<tr>
<td>1,16</td>
<td>272,30</td>
<td>3,84</td>
</tr>
<tr>
<td>Média</td>
<td>13,03</td>
<td>0,20</td>
</tr>
<tr>
<td>0,58</td>
<td>144,12</td>
<td>3,27</td>
</tr>
<tr>
<td>100x5</td>
<td>0,00</td>
<td>70,00</td>
</tr>
<tr>
<td>10,11</td>
<td>0,73</td>
<td>0,73</td>
</tr>
<tr>
<td>0,83</td>
<td>1217,10</td>
<td>40,70</td>
</tr>
<tr>
<td>100x10</td>
<td>0,00</td>
<td>40,00</td>
</tr>
<tr>
<td>11,16</td>
<td>0,85</td>
<td>0,85</td>
</tr>
<tr>
<td>n x m</td>
<td>BAH</td>
<td>BIH</td>
</tr>
<tr>
<td>-------</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>100x20</td>
<td>0.00</td>
<td>60.00</td>
</tr>
<tr>
<td>Média</td>
<td>10.89</td>
<td>0.61</td>
</tr>
<tr>
<td>Média</td>
<td>2.70</td>
<td>2569.57</td>
</tr>
<tr>
<td>Média</td>
<td>0.00</td>
<td>50.00</td>
</tr>
<tr>
<td>200x10</td>
<td>8.11</td>
<td>0.33</td>
</tr>
<tr>
<td>Média</td>
<td>6.88</td>
<td>42076.88</td>
</tr>
<tr>
<td>Média</td>
<td>0.00</td>
<td>70.00</td>
</tr>
<tr>
<td>Média</td>
<td>21.63</td>
<td>82021.57</td>
</tr>
<tr>
<td>Média</td>
<td>0.00</td>
<td>58.89</td>
</tr>
<tr>
<td>Média</td>
<td>9.23</td>
<td>0.35</td>
</tr>
<tr>
<td>Média</td>
<td>10.40</td>
<td>42222.67</td>
</tr>
</tbody>
</table>

* Success Rate (%)  ** ARPD (%)  *** Average CPU time (second)

Table 2 – Comparison of results in Taillard’s testbed SSD-10 and SSD-50
As we can see from the results in Tables 2 and 3, the proposed heuristic obtains better results than the rest of the constructive heuristics. Over all configurations, the maximal ARPD from the best solution found was 0.14% for GAPH (when TRIPS found the best solution) and 2.48% for BIH and BIH’. The maximal ARPD for BAH was 15.87% and 9.27% for TRIPS. All success rates for BAH were zero, and TRIPS only got the best solution once. So, we can conclude that BAH and TRIPS are not competitive with the other heuristics tested.

One interesting characteristics from the experimental analysis is that the methods seem to be unaffected by distribution of processing or setup times, i.e., there are no better methods depending on the specific distribution of processing or setup times.

Comparing GAPH with BIH and BIH’, we observe that GAPH always gets equal or better results than BIH and BIH’. With respect to the CPU time, BIH and TRIPS require much more computational effort than GAPH. As it can be observed in Table 2, for the biggest problem analyzed (200x20), the average CPU time of BIH was nearly 8200s, while TRIPS required nearly 3660s and GAPH required nearly 1850s. The adaptation made in BIH changed substantially its average CPU time. From 8300s for the worst case, BIH’ required only 630s, nearly 132 times less. BIH’ is faster than GAPH about 3 times.

Comparing GAPH with TRIPS, in respect to the CPU time, it can be observed that GAPH is much faster.

We can conclude that GAPH is a very effective and efficient heuristic. GAPH always gets the best solution, and is much more efficient than TRIPS and BIH. Futhermore, we can also say that BIH’ is a good heuristic because, although GAPH always gets equal or better results than BIH’.

6. Conclusions

In this paper, we dealt with the problem of scheduling a no-wait flowshop with sequence-dependent setup times with a makespan objective by means of constructive heuristics. We presented a new heuristic, named GAPH, and created BIH’, a heuristic that always gets the same solution as the best existing heuristic for the problem (BIH), but with much lower CPU times. An extensive computational experiment has been carried out and it showed that GAPH gets better results than the other constructive heuristics tested, while its CPU time is much
lower than BIH. Henceforth, it can be concluded that the proposed heuristic obtains much higher solution quality comparing to the existing constructive heuristics for the problem, in acceptable computational times.

References


