The 2-stage assembly flowshop scheduling problem with total completion time: Efficient constructive heuristic and metaheuristic*

Jose M. Framinan[1]†, Paz Perez-Gonzalez[1]

Industrial Management, School of Engineering

University of Seville

Ave. Descubrimientos s/n, E41092 Seville, Spain, {framinan,pazperez}@us.es

July 27, 2017

Abstract

In this paper we address the 2-stage assembly scheduling problem where there are m machines in the first stage to manufacture the components of a product and one assembly station (machine) in the second stage. The objective considered is the minimisation of the total completion time. Since the NP-hard nature of this problem is well-established, most previous research has focused on finding approximate solutions in reasonable computation time. In our paper, we first review and derive a number of problem properties and, based on these ideas, we develop a constructive heuristic that outperforms the existing constructive heuristics for the problem, providing solutions almost in real-time. Finally, for the cases where extremely high-quality solutions are required, a variable local search algorithm is proposed. The computational experience carried out shows that the algorithm outperforms the best existing metaheuristic for the problem. As a summary, the heuristics presented in the paper substantially modify the state-of-the-art of the approximate methods for the 2-stage assembly scheduling problem with total completion time objective.

Keywords: Scheduling, Assembly, Completion Time, Heuristics

*Preprint submitted to Computers & Operations Research. DOI: https://doi.org/10.1016/j.cor.2017.07.012

†Corresponding author. Tel.: +34-954487214.

1 Introduction

Assembly scheduling refers to a branch of scheduling decisions in which parts/components/subsets of products or services must be first manufactured in parallel and later assembled in a final stage. Applications of assembly scheduling in industry and services have been reported in several works: Potts et al. (1995) describe the case of personal computer manufacturing where the different components of the computer are produced in the first stage to be later assembled in a second stage (a packaging station). Lee et al. (1993) describe the case of a fire engine assembly plant. In this case, the body and chassis of fire engines are produced in parallel, and assembled in a second stage. Finally, another application is presented by Al-Anzi and Allahverdi (2006a); Allahverdi and Al-Anzi (2006) and Al-Anzi and Allahverdi (2006b) in the area of distributed database systems.

Different objectives can be considered for the assembly scheduling problem. The first objective addressed in the literature is the minimisation of the makespan or maximum completion time of the set of jobs. This problem has been first tackled by Lee et al. (1993), and its NP-hardness in the strong sense (even when the first stage is composed of 2 machines in parallel) has been established by Potts et al. (1995). A number of efficient heuristics for the problem have been proposed by Sun et al. (2003). Regarding exact solutions, the best approach is the Branch & Bound algorithm proposed by Hariri and Potts (1997), which in many cases is able to schedule up to 8,000 jobs in less than 100 seconds. Another well-studied objective is the minimisation of the sum of completion times of the jobs, which is also the aim of our paper and is discussed in detail below. Other objective considered in the literature is the maximum lateness (Allahverdi and Al-Anzi, 2006; Al-Anzi and Allahverdi, 2006b). Finally, additional constraints such as setup times (Al-Anzi and Allahverdi, 2007), more than one machine in the second stage (Sung and Kim, 2008; Al-Anzi and Allahverdi, 2013), or additional stages for the transportation of components (Koulamas and Kyparisis, 2001; Shoaardebili and Fattahi, 2015) have been also addressed.

As mentioned above, our paper is devoted to the 2-stage assembly scheduling problem with the minimisation of total completion time as objective, which can be denoted as $Am||\sum_j C_j$ according to the notation in Potts et al. (1995). Minimisation of the total completion time is an important scheduling objective since completion time can be viewed as a surrogate for the cycle time of the

jobs, which in turns influences the inventory levels and the lead times that can be quoted by a company (Framinan et al., 2014). Also note that this problem has several connections to other scheduling problems: Perhaps the most clear case is the 2-machine flowshop scheduling problem with total completion time as objective, which can be seen as a particular case of our problem when there is only one component to be manufactured. In turn, this NP-hard problem can be decomposed into consecutive single-machine scheduling problems, for which the Shortest Processing Time (SPT) rule provides the optimal solution, a property used by some of the heuristics for the $Am||\sum_j C_j$ problem. Another connection is with the Customer Order Scheduling problem with total completion time as objective (see e.g. Leung et al., 2005; Framinan and Perez-Gonzalez, 2017), denoted as $PDm||\sum_j C_j$. In this problem, customer orders composed of a number of parts have to be manufactured in dedicated parallel machines. Clearly, $PDm||\sum_j C_j$ and $Am||\sum_j C_j$ problems are equivalent if the processing times of the jobs in the second (assembly) stage are zero, but, as we will show later, there is another less trivial relationship.

The first reference for total completion time minimisation is Tozkapan et al. (2003), where the authors address the problem (weighted minimisation) for the first time. They show that permutation sequences are optimal for this problem, and propose two heuristics, labelled TCK1 and TCK2 in the following. Al-Anzi and Allahverdi (2006a) also address this problem, stating some conditions that the processing times of an instance must fulfil to be optimally solved, and proposing both constructive heuristics and metaheuristics for the problem. Regarding the constructive heuristics, the computational experience carried out by these authors shows that two of them (the aforementioned TCK2 and a new proposal denoted as A1 in the following) are the most efficient heuristics for the problem, being around 8% with respect to the best known solutions while requiring a negligible computational effort. Regarding the metaheuristics proposed, it turns out that a Hybrid Tabu Search (HTS in the following) obtains the best results, being therefore the most efficient metaheuristic for the problem.

Note also that the problem under consideration can be regarded as a special case of the multi-machine assembly scheduling problem, where there are more than one machine in the second (assembly) stage. This problem has been addressed for the total completion time objective first by Sung and Kim (2008) when there are two assembly machines, and later generalised for $m \geq 2$ as-

sembly machines by Al-Anzi and Allahverdi (2012). The most efficient approximate method for the problem is the Artificial Immune Intelligence (AIS) metaheuristic by Al-Anzi and Allahverdi (2013), as these authors conduct an exhaustive computational experience showing that AIS outperforms the rest of existing approximate methods. However, it is worth to note that the inclusion of more than one machine in the second stage may change the structure of solutions of the problem and therefore it remains uncertain whether efficient procedures for the multi-machine case are equally efficient when there is only one assembly machine.

Other related problem is that of the distributed two-stage assembly system, where the jobs have to be assigned to one of f factories each one consisting of a two-stage assembly system like the one treated in our research, and subsequently scheduled to minimise the total completion time. To the best of our knowledge, this problem has been addressed only by Xiong et al. (2014), also considering setup times. These authors propose the so-called ESPT constructive heuristic that could be potentially interesting for our problem and indeed, when there is only one factory and no setups are considered, it is equivalent to one of the already mentioned constructive heuristics by Al-Anzi and Allahverdi (2006a).

Finally, Al-Anzi and Allahverdi (2006a) study a number of theoretical properties for the problem under consideration. The work of these authors represents an important advance on analysing the problem, particularly on identifying distinct sub-cases depending on whether the first stage dominates the second, or vice versa. However, we will show in Section 2 that their results contain some flaws, and we provide a correct formulation. Also, based on the ideas regarding the predominance of one of the stages, we propose a constructive heuristic for the problem which turns out to be much more effective than existing constructive heuristics, i.e. the average error with respect to the best known solution is around five times smaller. Finally, we exploit some of the ideas used in the design of the HTS algorithm to propose a new local search algorithm for the problem which also favourably competes against HTS both in terms of quality of the solutions and in the computational effort required.

The remainder of the paper is as follows: In Section 2 the problem under consideration is formally stated and some properties are presented. Section 3 is devoted to first present the constructive heuristics available for the problem (Section 3.1), and second to discuss a new proposal (Section

3.2). In Section 4 we present a new metaheuristic for the problem, while Section 5 is devoted to the computational experiments. Finally, the main conclusions are discussed in Section 6.

2 Problem Statement and Properties

Formally stated, the problem under consideration consists of scheduling n jobs in a layout composed of two stages: In the first stage there are m machines in parallel, each one capable of processing one of the m components of the jobs. Let us denote by t_{ij} the processing time in machine i of the component of job j in this stage, or equivalently, t_{ij} is the processing time of the i-th component of job j. It is convenient for us to denote the maximum and minimum of t_{ij}, i.e. $t_{max} = \max_{\forall i,j} t_{ij}$, and $t_{min} = \min_{\forall i,j} t_{ij}$. The second stage consists of the assembly of the components, so operations in the second stage cannot start until the m components of the job have been completed. The processing time of job j in this assembly stage is denoted by p_j.

Given a permutation sequence, let us denote job $[j]$ as the job processed in order j-th in the sequence. Furthermore, let $C_{[j]}$ be the completion time of job processed in order $[j]$. Clearly, the following recursive formula holds:

$$C_{[j]} = \max\{C_{[j-1]}; \max_{\forall\, i}\{\sum_{k=1}^{j} t_{i[k]}\}\} + p_{[j]} \tag{1}$$

with $C_{[0]} = 0$.

As mentioned in Section 1, a number of properties for the problem have been studied by Al-Anzi and Allahverdi (2006a). While this work is an important step towards the analysis of the problem, their results contain several flaws which also imply some changes in the tractable subcases. In the next theorems, we provide the correct formulation of the problem properties, point out the differences with the initial statements, and derive new properties.

Theorem 1. *If* $\max_{\forall\, j}\{p_j\} \leq t_{min}$*, then the total completion time of a sequence can be expressed as:*

$$\sum C_j = \sum_{j=1}^{n} \max_{\forall\, i}\{\sum_{k=1}^{j} t_{i[k]}\} + \sum_{j=1}^{n} p_j \tag{2}$$

Proof. From Equation (1) it can be seen that $C_{[1]} = \max\{C_{[0]}; \max_{\forall\, i}\{t_{i[1]}\} + p_{[1]}\} = max_{\forall\, i}\{t_{i[1]}\} + p_{[1]}$. Regarding the second job:

$$C_{[2]} = \max\{C_{[1]}; \max_{\forall\, i}\{t_{i[1]} + t_{i[2]}\}\} + p_{[2]} =$$

$$= \max\{max_{\forall\, i}\{t_{i[1]}\} + p_{[1]}; \max_{\forall\, i}\{t_{i[1]} + t_{i[2]}\}\} + p_{[2]} =$$

$$= \max\{max_{\forall\, i}\{t_{i[1]} + p_{[1]}\}; \max_{\forall\, i}\{t_{i[1]} + t_{i[2]}\}\} + p_{[2]}$$

Since $\max_{\forall\, j}\{p_j\} \leq t_{ik}$ for all i, k, we have:

$$C_{[2]} = \max_{\forall\, i}\{t_{i[1]} + t_{i[2]}\} + p_{[2]}$$

It is then easy to see that, in general, for job in position j, its completion time is given by:

$$C_{[j]} = \max_{\forall\, i}\{\sum_{k=1}^{j} t_{i[k]}\} + p_{[j]}$$

Therefore, the total completion time $\sum C_j$ is:

$$\sum C_j = \sum_{j=1}^{n} C_{[j]} = \sum_{j=1}^{n} \max_{\forall\, i}\{\sum_{k=1}^{j} t_{i[k]}\} + p_{[j]} = \sum_{j=1}^{n} \max_{\forall\, i}\{\sum_{k=1}^{j} t_{i[k]}\} + \sum_{j=1}^{n} p_{[j]} \tag{3}$$

□

Theorem 1 establishes the conditions for the first stage to be dominant in a similar way to Theorem 1 in Al-Anzi and Allahverdi (2006a). However, they incorrectly state that it is sufficient that $\max_{\forall\, j}\{p_j\} \leq \min_{1 \leq k \leq n}\{\max_{1 \leq i \leq m}\{t_{ik}\}\}$. As a consequence, their Algorithm 2 does not provide the optimal solution to the problem as claimed in their paper. Indeed, there is no polynomial algorithm that can provide an optimal solution to the problem where the first stage is dominant, as the following corollary states:

Corollary 1. *The optimal solution of the $Am||\sum_j C_j$ problem under the conditions of Theorem 1 is the same than the optimal solution of the $PDm||\sum_j C_j$ problem.*

Proof. Note that the second term in Equation (3) does not depend on the specific sequence, so the

problem is equivalent to minimising the completion time in a $PDm||\sum_j C_j$ problem instance taking the processing times of the jobs in the first stage. □

Corollary 1 establishes conditions for the dominance of the first stage in the $Am||\sum_j C_j$ problem, and, since the $PDm||\sum_j C_j$ is NP-hard even for $m = 2$, then there is no polynomial algorithm that can yield the optimal solution for the former problem. However, when the second (assembly) stage is predominant, such polynomial algorithm does exist. Let us present first the conditions for the second stage to be dominant:

Theorem 2. *If* $\min_{\forall j}\{p_j\} \geq t_{max}$*, then the total completion time of a sequence can be expressed as:*

$$\sum C_j = n \max_{\forall i}\{t_{i[1]}\} + \sum_{j=1}^{n}(n-j+1)\cdot p_{[j]} \tag{4}$$

Proof. From Equation (1) it can be seen that $C_{[1]} = \max\{C_{[0]}; \max_{\forall i}\{t_{i[1]}\} + p_{[1]}\} = max_{\forall i}\{t_{i[1]}\} + p_{[1]}$. Regarding the second job:

$$C_{[2]} = \max\{C_{[1]}; \max_{\forall i}\{t_{i[1]} + t_{i[2]}\}\} + p_{[2]} =$$

$$= \max\{max_{\forall i}\{t_{i[1]}\} + p_{[1]}; \max_{\forall i}\{t_{i[1]} + t_{i[2]}\}\} + p_{[2]} =$$

$$= \max\{max_{\forall i}\{t_{i[1]} + p_{[1]}\}; \max_{\forall i}\{t_{i[1]} + t_{i[2]}\}\} + p_{[2]}$$

Since $\min_{\forall j}\{p_j\} \geq t_{ik}$ for all i, k, we have:

$$C_{[2]} = \max_{\forall i}\{t_{i[1]}\} + p_{[1]} + p_{[2]}$$

It is then easy to see that:

$$C_{[j]} = \max_{\forall i}\{t_{i[1]}\} + \sum_{k=1}^{j} p_{[k]}$$

Therefore, the total completion time $\sum C_j$ is:

$$\sum C_j = \sum_{j=1}^{n} C_{[j]} = \sum_{j=1}^{n} \left(\max_{\forall i} \{t_{i[1]}\} + \sum_{k=1}^{j} p_{[k]} \right) = n \cdot \max_{\forall i} \{t_{i[1]}\} + \sum_{j=1}^{n} (n - j + 1) \cdot p_{[j]} \qquad (5)$$

□

The results of Theorem 2 establish the basis for a polynomial optimal algorithm for the problem:

Corollary 2. *The optimal solution of* $Am||\sum_j C_j$ *under the conditions of Theorem 2 is given by the following procedure:*

1. *Sort all jobs in non decreasing order of* p_j*. Denote this sequence as* Π_{SPT}
2. *Set* $best := \sum C_j(\Pi_{SPT})$*, and* $\Pi_b := \Pi_{SPT}$
3. *For all jobs* $(j = 2, \ldots, n)$*:*
   (a) *Obtain a candidate sequence* Π_c *by removing job in position* j *in* Π_c *and insert it in position* 1*.*
   (b) *Set* $curr := \sum C_j(\Pi_c)$
   (c) *If* $curr < best$*, set* $best = curr$ *and* $\Pi_b := \Pi_c$

This procedure can run in $O(n \cdot \log n)$

Proof. The second term in Equation (5) shows that the sequence with minimal total completion time with respect to this term is obtained by sorting the jobs in ascending order of their processing times in the second stage. However, the first term states the influence of the job in position 1. Therefore, a manner to find the optimal solution is to try all jobs in the first position while the rest are sequenced in SPT order with respect to the second stage, and take the sequence with the lowest completion time. □

If additional conditions on the processing times of the first stage are imposed, then the SPT rule can be optimal:

Corollary 3. *Under the conditions of Theorem 2, let k be the job for which $p_k = \min_{1\leq j\leq N} p_j$. If*

$$\min_{1\leq j\leq n}\left\{\max_{1\leq i\leq m} t_{ij}\right\} = \max_{1\leq i\leq m} t_{ik} \tag{6}$$

then sorting the jobs in non decreasing order of their processing times in the assembly stage provides the optimal solution to the problem.

Proof. Under the conditions of Theorem 2, the total completion time can be calculated according to Equation (4). Note that, if condition (6) holds, sorting the jobs in non decreasing order of their processing times in the assembly stage provides the optimal solution to the problem. □

Corollary 3 expresses the specific conditions for the second-stage SPT rule to be optimal for the assembly problem, therefore it would correspond –albeit with different conditions due to a flaw in the proof– to Theorem 2 in Al-Anzi and Allahverdi (2006a).

The problem properties discussed in this section are useful to detect that there are two (extreme) cases where the problem is related to other problems. These cases would serve us in a twofold manner:

- First, to design a testbed where the instances do not belong to the extreme cases, as there are specific algorithms available for these problems: If the first stage is dominant, there are several efficient algorithms for the problem, most notably the algorithm by Framinan and Perez-Gonzalez (2017). If the second stage is dominant, Corollary 2 gives a polynomial algorithm to solve the problem. The testbed designed in Section 5 takes these conditions into account.

- Second, when the conditions for the extreme cases are not fulfilled –which would be the most common case–, it is clear that the quality of the solutions would be influenced by the dominance of the stages, therefore this idea can be used to develop some efficient heuristics for the problem. This aspect will be addressed in Section 3.

3 Constructive heuristics

In this section, we first analyse the existing constructive heuristics for the problem (Section 3.1), and then (Section 3.2) propose a new constructive heuristic based on the ideas discussed in the previous section.

3.1 Existing constructive heuristics

The first constructive heuristics for the problem have been proposed by Tozkapan et al. (2003). More specifically, these authors propose two heuristics: The first one –labelled in the following TCK1– obtains $m+1$ sequences, each one as a product of applying the SPT rule with respect to the processing times of each job in machine i in the first stage, and with respect to the processing times of each job in the assembly machine. These heuristics exploit the already mentioned connection between the single machine scheduling problem with total completion time as objective, and the problem under consideration. To be precise, $m+1$ indices are developed for each job:

- Processing Times of job j on machine i ($i = 1, \ldots, m$) in the First stage (PTF_{ij}), i.e. $PTF_{ij} = t_{ij}$.
- Processing Times of job j in the Second stage (PTS_j), i.e. $PTS_j = p_j$.

Then, $m+1$ sequences are obtained by sorting the jobs in non descending order of these indices, and the sequence yielding the lowest total completion time is selected.

The second heuristic –labelled in the following TCK2– obtains three indices for each job:

- Minimum Processing Times of job j (MPT_j), i.e. $MPT_j = \min\{t_{1j}, t_{2j}, \ldots, t_{mj}, p_j\}$.
- Average Processing Times of job j (APT_j), i.e. $APT_j = \frac{1}{m+1} \sum_{i=1}^{m} t_{ij} + p_j$.
- Maximum Processing Times of job j ($MXPT_j$), i.e. $MXPT_j = \max\{t_{1j}, t_{2j}, \ldots, t_{mj}, p_j\}$.

Then, three sequences are obtained by sorting the jobs in non descending order of these indices, and the sequence yielding the lowest total completion time is selected.

Al-Anzi and Allahverdi (2006a) propose three constructive heuristics for the problem: heuristic S1 is obtained by sorting the jobs in non descending order of p_j. Heuristic S2 sorts the jobs in

non descending order of $\max_{1\leq i\leq m} t_{ij}$. Finally, heuristic S3 considers the two stages to obtain the solution, i.e. it sorts the jobs in non descending order of $\max_{1\leq i\leq m} t_{ij} + p_j$. The rationale of these heuristics is that sorting the jobs according to the SPT order of the second machine would yield good results when the processing times in the assembly machine are larger than those in the first stage machines. The S1-S3 heuristics have not been compared with other heuristics, although they are used by Al-Anzi and Allahverdi (2006a) as starting seeds for two metaheuristics for the problem.

Finally, Al-Anzi and Allahverdi (2006a) propose two algorithms for the problem, labelled A1 and A2. These algorithms construct a sequence by iteratively inserting a non sequenced job at the end of the existing sequence. The unscheduled job to be inserted is chosen so the value of an indicator is minimised. More specifically, at iteration j ($j = 1, \ldots, n$), a partial sequence of $j-1$ jobs has been constructed and, for each unscheduled job k, the following indicator is computed for algorithm A1:

$$A1_k = \max_{i=1,\ldots,m} \left\{ \sum_{r=1}^{j-1} t_{i[r]} + t_{ik} \right\} \tag{7}$$

whereas for algorithm A2 the indicator is:

$$A2_k = \max_{i=1,\ldots,m} \left\{ \sum_{r=1}^{j-1} t_{i[r]} + t_{ik} \right\} + p_k \tag{8}$$

The A1 and A2 algorithms are compared with TCK1 and TCK2 by Al-Anzi and Allahverdi (2006a), resulting that the best performance is obtained by TCK2 and A1. The differences between both algorithms do not seem to be big enough to be considered statistically significant. Therefore, these two heuristics can be assumed to be the most efficient constructive procedures so-far.

Despite the fact that there are several heuristics available for the problem, further improvements could be obtained based on the following insights: First, A1 iteratively constructs a solution by appending unscheduled jobs at the end of the partial sequence. This strategy has been successfully employed for other related problems, such as the permutation flowshop with total completion time objective (see Fernandez-Viagas and Framinan, 2015), or the customer order scheduling problem (see Leung et al., 2005). However, in A1 the evaluation of the candidate jobs to be appended is based only in the partial contribution to the objective function, without taking into account the rest

of the unscheduled jobs. Estimating the contribution of the unscheduled jobs to the final objective function would potentially improve the results, as the resulting heuristic would not be completely greedy. Furthermore, A1 ignores the processing times in the second stage, therefore it does not seem to be well suited for these problem instances where the second stage is dominant. However, including the processing times of the second stage in a straightforward manner may not yield the desired results, as it can be shown by the bad results obtained by A2.

These two aforementioned aspects (estimation of the contribution of the unscheduled jobs to the completion time and a mechanism to properly assess the candidates depending on which stage is dominant) will be the basis for the proposal of a new heuristic discussed in the following section.

3.2 Proposed constructive heuristic

The idea of the heuristic proposed is, as in A1 or A2, to iteratively construct a solution by selecting one job among the unscheduled jobs and appending it to the end of a partial sequence. Therefore, the procedure starts with an empty schedule Π and a set Ω with all unscheduled jobs. At iteration j ($j = 1, \ldots, n$), an unscheduled job $\omega_l \in \Omega$ is analysed as candidate to be inserted in position j in Π.

The criterion to select the job among the candidates should take into account that, for the specific problem instance, the processing times of the first (second) stage may be much larger than those for the second (first) stage, therefore greatly affecting the (partial) completion time. To distinguish the two cases, $C_1(\omega_l)$ the maximum completion time of ω_l in the first stage is computed, i.e.:

$$C_1(\omega_l) := \max_{i=1,\ldots,m} \{C1_i^* + t_{i\omega_l}\} \tag{9}$$

where $C1_i^*$ denotes the completion time on machine i in the first stage of partial sequence Π, i.e. the completion time in the first stage of the already scheduled jobs.

Let C_2^* be the completion time of the jobs in Π in the second stage. If $C_1(\omega_l)$ is higher than C_2^*, it is clear that, for candidate w_l, the completion times in the first stage would largely influence the completion time of the candidate job and that of the subsequent (unscheduled) jobs. Therefore,

$C_1(\omega_l)$ would be a good indicator of the contribution of w_l to the completion times.

In contrast, if the completion time of the candidate job in the first stage is lower than the completion time of the previous jobs, then the completion time of the candidate job (and that of subsequent jobs) would be influenced by the completion times of the candidate job in the second stage. This influence decreases with the number of machines in the first stage since, with a higher number of machines, there are more possibilities for the subsequent jobs not to be affected by the second stage. Therefore, the influence of the second stage will be weighted according to the number of machines, so $\frac{p_{\omega_l}}{m}$ would be a good indicator of the contribution of w_l to the completion times.

The second aspect that the heuristic takes into account is the estimation of the completion time of the unscheduled jobs. To do so, we estimate the completion times of an *artificial* job $\bullet$ composed of the unscheduled jobs by assuming that these unscheduled jobs are sorted according to a given sequence $S := (s_1, s_2, \ldots, s_{|\Omega-\{\omega_l\}|})$ (we will discuss later how this sequence is established). Therefore, we propose to estimate $C_{1\bullet}$ the completion times of this artificial job in the first stage using the following equation:

$$C_{1\bullet} := \frac{1}{|\Omega - \{\omega_l\}|} \sum_{k \in \Omega - \{\omega_l\}} \max_{1 \leq i \leq m} \{C1_i^* + \sum_{j=1}^{k} t_{is_j}\} \tag{10}$$

and $p_\bullet$ the processing times of artificial job $\bullet$ in the second stage using the following equation:

$$p_\bullet := \frac{1}{|\Omega - \{\omega_l\}|} \sum_{k \in \Omega - \{\omega_l\}} p_k \tag{11}$$

Note that Equation (10) represents a sort of average completion time in the first stage of the unscheduled jobs if it is assumed that they are scheduled according to S, whereas Equation (11) is the average processing time of the unscheduled jobs in the second stage.

As with the scheduled jobs, if the completion times of the artificial job in the first stage are higher than the completion times (in the assembly stage) of the candidate job, then the first stage of the artificial job would largely determine the completion times. With an analogous reasoning as done before, we propose to weight the processing time of the artificial job in the second stage with respect to the number of machines in the first stage.

As already mentioned, the computation of $C_{1\bullet}$ depends on the sequence S assumed for the

unscheduled jobs. As our intention is to provide an estimate of the completion times of this artificial job, we suggest sorting the unscheduled jobs according to algorithm S2, which is the sorting algorithm providing the best results according to the computational experience by Al-Anzi and Allahverdi (2006a). Using this algorithm –or any other index-based algorithm– also has the advantage that the relative order of the jobs can be determined at the beginning of the constructive heuristic and it does not have to be recomputed for each iteration or candidate.

Finally, recall that the artificial job represents an estimation of the completion times of the unscheduled jobs, so the importance of this estimation would decrease with the iterations of the algorithm. Therefore, we propose to weight the contribution of the unscheduled jobs using $\frac{n-j+1}{n}$.

As a summary, the indicator ψ_l used to estimate the suitability of inserting a candidate job ω_l at the end of Π would be given by:

$$\psi_l := C_1(\omega_l) + \frac{(n-j+1)}{n}\left(C_{1\bullet} + \frac{p_\bullet}{m}\right) \tag{12}$$

if $C_1(\omega_l)$ is higher than C_2^* (i.e. the first stage determines the completion time), otherwise:

$$\psi_l := \frac{p_{\omega_l}}{m} + \frac{(n-j+1)}{n}\left(C_{1\bullet} + \frac{p_\bullet}{m}\right) \tag{13}$$

Therefore, the candidate job with the lowest value of ψ_l is selected and extracted from the list of unscheduled jobs. The procedure continues until all jobs have been scheduled. Figure 1 shows the pseudocode of the proposed heuristic.

4 Variable Local Search

As we will show in the computational experience in Section 5, the procedure proposed in the previous section allows us to obtain very fast (less than 0.01 seconds) solutions with an average deviation of around 2% with respect to the best known solutions. However, it can be also useful to embed this heuristic in a more sophisticated local search procedure to obtain solutions of higher quality, albeit at the expenses of higher computation times.

There are several metaheuristics available for the problem, all by Al-Anzi and Allahverdi (2006a):

Procedure *Proposed Heuristic*

// All jobs are initially unscheduled

$\Pi := \varnothing$;

// Completion times on stages 1 and 2 of sequence Π:

$C1_i^* := 0 \quad i = 1, \ldots, m$

$C2^* := 0$

Obtain a sequence $\Omega := (\omega_1, \ldots, \omega_n)$ by applying algorithm S2;

for $j = 1$ **to** n **do**

  for each $\omega_l \in \Omega$ **do**

    // Compute the completion times in the first stage after selecting ω_l as candidate:

    $C_1(\omega_l) := \max_{1 \le i \le m}\{C1_i^* + t_{i\omega_l}\}$

    // Compute completion times of the artificial job in the first stage by Eq. (10):

    $C_{1\bullet} := \frac{1}{|\Omega - \{\omega_l\}|} \sum_{k \in \Omega - \{\omega_l\}} \max_{1 \le i \le m}\{C1_i^* + \sum_{j=1; j \ne l}^{k} t_{iw_j}\}$

    // Compute processing times of the artificial job in the second stage by Eq. (11):

    $p_\bullet := \frac{1}{|\Omega - \{\omega_l\}|} \sum_{k \in \Omega - \{\omega_l\}} p_k$

    // Compute the indicator by Eqs. (12) or (13) depending on the dominating stage:

    if $C_1(\omega_l) > C_2^*$ **then**

      $\psi_l := C_1(\omega_l) + \frac{n-j+1}{n}(C_{1\bullet} + \frac{p_\bullet}{m})$

    else

      $\psi_l := \frac{p_{\omega_l}}{m} + \frac{n-j+1}{n}(C_{1\bullet} + \frac{p_\bullet}{m})$

    end

  end

  $r := \text{argmin}_{1 \le k \le n-i+1} \psi_k$;

  Append ω_r at the end of Π, i.e. $\Pi := (\pi_1, \ldots, \pi_{i-1}, \omega_r)$;

  Extract ω_r from Ω, i.e. $\Omega := (\omega_1, \ldots, \omega_{r-1}, \omega_{r+1}, \ldots, \omega_{n-i+1})$;

  // Update values of the constructive sequence:

  $C1_i^* := C1_i^* + t_{i\omega_r}$

  $C_2^* := \max\{\max_{1 \le i \le m}\{C1_i^*\}; C_2^*\} + p_{w_r}$

end

return C_2^*

end

Figure 1: Pseudo-code of the proposed heuristic

a Simulated Annealing algorithm, a Tabu Search algorithm, and a Hybrid Tabu Search algorithm (HTS in the following). Among these three algorithms, HTS obtains the solutions with the highest quality while approximately requiring the same computational effort than the other two metaheuristics. Therefore, it can be stated that HTS is the best-so-far metaheuristic for the problem.

HTS starts with the best solution provided by the constructive heuristics S1, S2, and S3, and performs a complete local search in the neighbourhood by systematically exploring all neighbour solutions of a given sequence by exchanging two job positions (General Pairwise Interchange or GPI -type of neighbourhood). The four last positions used for GPI are stored in a tabu list to prevent the algorithm to have a cyclic behaviour. Based in a simulated annealing -like cooling scheme, the algorithm updates the best-so-far solution if a better solution is found during the exploration of the neighbourhood. The solution is updated even if the neighbour solution is worse with a probability that depends on the distance between the best-so-far solution and the current solution, and the number of iterations of the algorithm, so this acceptance probability exponentially decreases as the algorithm progresses. The algorithm runs for a pre-defined number of iterations, therefore it does not have any external parameter.

Although the performance of HTS is excellent (according to Al-Anzi and Allahverdi, 2006a it provides an average 0.15% deviation from the best known solutions in an extensive testbed), we believe that there is room for improvement by allowing a higher diversification in the exploration of solutions. Our proposal to carry out this diversification consists of two aspects:

- To restart the GPI search process once a best-so-far solution is found, so the algorithm may quickly move to explore the most promising neighbourhoods.
- To change the type of neighbourhood if the algorithm is stuck in a local optima, borrowing ideas from Variable Neighbourhood Search or VNS (Mladenović and Hansen, 1997).

Along these ideas, we propose an algorithm –labelled Variable Local Search or VLS– that starts with an initial solution obtained using the heuristic proposed in Section 3.2. This solution is set to both the best-so-far sequence Π_b and the current candidate Π_c.

Once this solution is obtained, the algorithm performs a maximum of *maxit* iterations, consisting of the following phases:

1. **General Pairwise Interchange with Restart (GPIR) phase**: The most promising candidate in the neighbourhood of Π_c is found by evaluating all neighbours of Π_c obtained by exchanging the job in position l ($l = n, n-1, \ldots, 2$) in Π_c with that in position k ($k = l-1, l-2, \ldots, 1$). If a sequence Π_n with lowest total completion time than that of Π_c is found in the search, then Π_c is replaced by Π_n and the search is restarted so all neighbours of the new most promising candidate can be explored.

2. **Perturbation phase**: Once the most promising candidate in the neighbourhood has been obtained, it is compared with the best-so-far sequence Π_b. If the candidate is better than the best so-far sequence, then Π_b is updated, and the counter for iterations without improvement is restarted. Otherwise, the counter for iterations without improvement is increased in one unit. As long as the counter for iterations without improvement does not exceeds n, the job in position r (chosen at random from positions $[1, n-1]$) of the most promising candidate is removed and inserted in position $r+1$. The aim of the insertion performed here is to carry out a perturbation of the current solution in order to make the GPIR phase to explore a different section of the solution space.

   If the counter for iterations without improvement exceeds the number of jobs n, then there are little chances that the current candidate will improve the best-so-far solution in the current section of the solution space, therefore a greater perturbation would give more opportunities for improvement. In our proposal, this is done by successively performing the reinsertion a bigger number of times. More specifically, the number of times d that a random insertion is performed is 1 until n iterations have been elapsed without improvement. Then, $d = 2$ for another n iterations without improvement, and so forth until $d = n$ or the maximum number of iterations $maxit$ is reached. Every time a new best-so-far solution is found, d is reset to 1 to perform a systematic exploration of the new neighbourhood.

The pseudocode of the proposed algorithm is shown in Figure 2. The intensification in the local search is provided by the GPIR, as the neighbourhood of the current solutions is systematically explored. The diversification of the algorithm is provided by performing successively greater perturbations in the current solution in order to move the algorithm to explore different neighbour-

hood. In order to progressively widen the search space, perturbations of the current solutions are performed in a similar manner to the VNS, but in this case the neighbourhood remains the same and the idea of variable neighbours is used to diversify the search. We are not aware of a similar use of the concept of VNS. The resulting algorithm is rather simple as it has only one parameter (the maximum number of iterations allowed), so there is no need for callibration. In the next section, we conduct an extensive computational experience to test the effectiveness of our proposal, as well as the constructive heuristic proposed in Section 3.2.

5 Computational Experience

In this section, we analyse the efficiency of the constructive heuristic proposed in Section 3.2, and the variable local search proposed in Section 4. To do so, we generate a testbed using the parameters given by Allahverdi and Al-Anzi (2009). This testbed consist of 30 replications of instances generated for different combinations of n and m. More specifically, $n \in \{20, 40, 60, 80, 100, 120\}$, and $m \in \{2, 4, 6, 8\}$. For each instance size, the processing times of the jobs in the machines for both stages are drawn from a uniform distribution $[1, 100]$. In total, 720 instances have been generated. Furthermore, it has been checked that none of the instances fulfilled the conditions of Theorems 1 and 2.

Using the testbed described above, we conduct two experiments:

1. First, we compare the constructive heuristics existing for the problem with our proposal in Section 3.2. Since these heuristics provide the solution in real-time (0.01 seconds for the biggest instances), the comparison is done in terms of the quality of solution obtained by heuristic h on instance i measured as the Relative Percentage Deviation (RPD):

$$RPD_{hi} = \frac{O_{hi} - O_i^*}{O_i^*} \cdot 100 \tag{14}$$

where O_{hi} is the total completion time obtained by heuristic h when applied to instance i, and O_i^* is a reference value of the minimum total completion time for instance i. In order to have good reference values, we developed a MILP model for the problem under consideration,

```
Procedure Variable Local Search(maxit)
    d := 1; // Set the number of insertions to perform to 1
    it := 1; // Set the number of iterations to 1
    improvement := 0; // Set the counter for iterations without improvement
    Let Π_b be the sequence obtained by the heuristic in Figure 1;
    F_b := SumC(Π_b); // Set Π_b as the best so far sequence
    Π_c := Π_b; // Set the sequence as the most promising candidate
    F_c := F_b; // Set the completion time of the most promising candidate
    while it < maxit do
        // Perform local search
        for l = n, n − 1, . . . , 2 do
            Obtain Π_n by exchanging positions l and k (k = l − 1, l − 2, . . . , 1) in Π_c;
            // Check if Π_n is the most promising candidate
            if SumC(Π_n) < F_c then
                Set Π_n as the most promising candidate: Π_c := Π_n and F_c := SumC(Π_n);
                Restart the local search, i.e. set l = n to explore all possible exchanges;
            end
        end
        // Check if the most promising candidate improves the best-so-far sequence
        if F_c < F_b then
            Update best-so-far sequence, i.e. F_b := F_c and Π_b := Π_c;
            improvement := 0; // Restart the counter for iterations without improvement
            d = 1; // Reset the number of insertions to perform to 1.
        else
            improvement := improvement + 1;
        end
        if improvement > n then
            // Enhances the neighbourhood
            d := min{d + 1; n − 1};
            improvement := 0; // Restart the counter for iterations without improvement
        end
        // Performs d times a random insertion
        for i = 1, . . . , d do
            Remove job in a random position r in Π_c and insert it in position r + 1;
        end
        it := it + 1; // Increase the number of iterations
    end
    Return Π_b as the best solution;
end
```

Figure 2: Pseudo-code of the proposed variable local search

and tried to solve the instances using the Gurobi solver. For all instances with $n = 20$ and $m = 2, 4$ the solver was able to provide the optimal value within 900 seconds. For bigger sizes ($n = 40$), in most instances the solver was not able to find the optimal solution in the allowed CPU time. Still, the so-obtained value was used if resulted to be better than those obtained by any of the multiple approximate methods tested in the experiments, therefore providing a fairly tight upper bound of the optimal total completion time.

2. Second, we compare the best-so-far metaheuristic for the problem (the Hybrid Tabu Search by Allahverdi and Al-Anzi, 2009) with the Variable Local Search proposed in Section 4. In addition, we also compare the best-so-far metaheuristic for the multi-machine assembly problem (the Artificial Immune System by Al-Anzi and Allahverdi, 2013, see Section 1). In this case, the comparison is done both in terms of the quality of solutions –measured in terms of RPD– and the computational effort required to obtain the solution –measured in seconds of CPU time–.

These two experiments are discussed in the next subsections.

5.1 Comparison of constructive heuristics

In this experiment, the following constructive heuristics have been compared:

1. TCK1 and TCK2 heuristics by Tozkapan et al. (2003).

2. S1-S3 and A1, A2 heuristics by Al-Anzi and Allahverdi (2006a).

The results of the comparison are shown in Table 1. As it can be seen, the proposed heuristic clearly outperforms the existing constructive heuristics for all problem sizes. It also provides the lowest value of the standard deviation of RPD. The average RPD with respect to the reference solution is almost five times smaller than that of the following heuristic (TCK2). The differences can be also appreciated in Figure 3 where the 95% confidence intervals of ARPD are shown. It can be also checked in Figure 3 that there are statistically significant differences among most procedures, so it can be concluded that our proposal is the best constructive heuristic for the problem. Regarding

		A1		A2		S1		S2		S3		TCK1		TCK2		Proposed	
m	n	Aver.	St. Dev.	Aver.	St. Dev.	Aver.	St. Dev.	Aver.	St. Dev.	Aver.	St. Dev.	Aver.	St. Dev.	Aver.	St. Dev.	Aver.	St. Dev.
2	20	10.937	6.191	10.686	3.732	29.278	8.159	12.920	5.339	14.121	5.212	20.215	6.339	8.399	3.291	1.717	1.302
	40	16.422	5.560	13.732	4.191	36.558	6.548	17.207	4.970	17.349	4.800	24.891	5.871	9.660	3.394	1.604	0.967
	60	15.783	5.832	12.664	3.658	33.870	4.716	16.676	4.931	16.281	3.895	25.799	4.825	8.890	2.501	1.773	0.847
	80	18.394	5.842	11.716	2.888	36.194	4.946	19.469	5.161	16.076	3.863	25.645	4.806	9.110	3.203	1.409	0.623
	100	21.366	4.827	12.875	3.735	36.229	4.544	21.996	4.404	15.869	3.693	27.247	4.194	8.940	1.965	1.596	0.901
	120	19.698	4.409	11.912	2.991	34.878	3.984	20.053	4.063	15.122	3.257	26.172	3.161	7.530	2.811	1.647	0.700
4	20	7.500	6.290	9.834	4.218	24.558	6.946	13.383	4.315	17.783	6.363	17.697	4.711	10.167	3.461	1.805	0.870
	40	11.183	6.056	13.309	4.399	28.797	4.624	15.415	4.249	22.016	4.811	20.858	4.294	11.162	2.671	2.364	0.585
	60	12.747	5.145	14.390	2.496	29.611	3.840	15.302	3.436	22.424	3.210	22.319	2.880	10.125	1.994	2.209	0.651
	80	12.372	5.477	14.199	2.705	30.756	4.609	15.235	3.772	22.066	3.943	21.244	2.355	9.207	2.332	1.996	0.428
	100	11.801	4.708	14.100	2.395	29.327	3.893	14.955	3.420	21.588	3.169	21.825	2.628	9.332	2.549	2.010	0.561
	120	13.049	5.345	14.825	2.687	29.523	3.480	15.653	3.619	21.503	2.634	21.655	2.280	9.128	2.535	2.036	0.543
6	20	7.432	6.095	11.595	4.845	20.256	6.040	13.028	4.287	18.070	5.836	15.044	3.105	10.024	3.484	2.103	1.004
	40	8.770	3.966	12.359	3.615	26.974	5.049	14.730	2.765	21.942	3.256	17.687	2.598	11.043	2.836	2.333	0.984
	60	9.754	4.631	12.621	3.110	27.034	3.870	13.696	2.612	22.183	3.430	18.900	2.675	10.844	2.879	2.211	0.576
	80	10.374	4.041	14.305	1.987	26.328	2.778	14.960	2.987	22.236	2.382	18.971	2.165	10.029	2.604	2.341	0.658
	100	9.776	4.293	14.395	2.212	27.634	2.731	14.308	2.604	23.493	3.198	18.675	2.334	9.850	2.191	2.136	0.545
	120	9.846	4.720	13.615	2.519	26.794	3.601	14.834	2.620	22.750	2.955	18.749	2.156	9.764	1.933	1.964	0.409
8	20	6.649	3.785	9.715	3.187	19.423	6.110	13.803	4.312	17.464	4.451	12.712	2.700	10.327	2.821	2.272	0.948
	40	8.861	6.188	11.967	3.497	24.321	4.749	15.542	3.764	22.008	5.379	15.750	2.362	11.432	2.967	2.656	1.047
	60	7.250	4.679	11.879	2.586	25.161	4.710	14.381	2.699	22.393	4.055	16.385	1.993	10.085	2.830	2.239	0.684
	80	9.309	5.896	12.772	2.736	24.451	3.500	14.834	2.695	22.069	2.965	17.713	2.301	11.345	3.404	2.306	0.541
	100	8.721	4.069	13.173	2.071	26.488	4.027	14.186	2.482	23.566	3.877	17.330	1.794	9.859	3.125	2.229	0.507
	120	10.362	4.413	14.447	2.542	24.998	3.258	15.006	2.614	22.910	2.955	17.623	1.287	9.466	2.350	2.224	0.629
Total		11.598	6.430	12.795	3.466	28.310	6.571	15.482	4.309	20.137	4.964	20.046	5.037	9.822	2.902	2.049	0.808

Table 1: Average values and Standard Deviation of the RPD obtained by the constructive heuristics

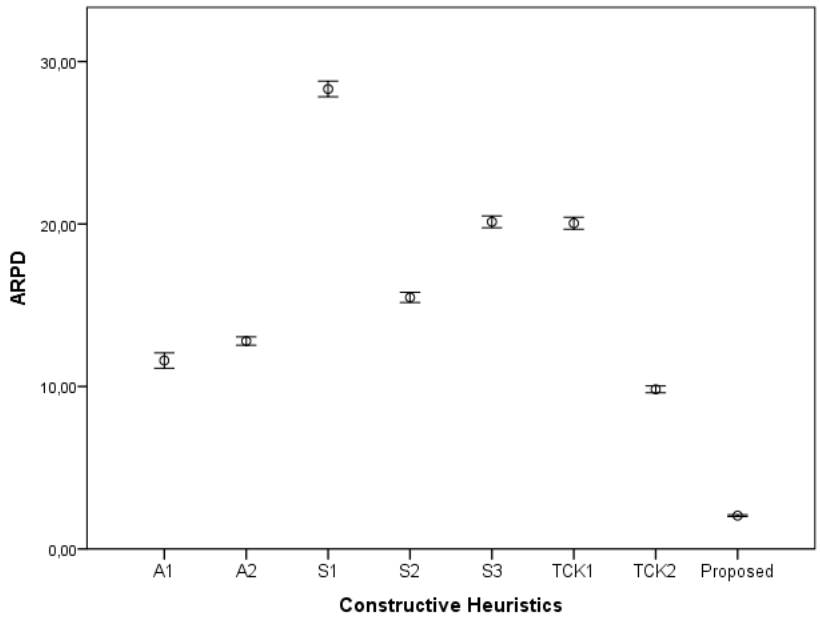

Figure 3: 95% CI RPD for the constructive heuristics

the rest of heuristics, the experiment confirms the results already obtained by Al-Anzi and Allahverdi (2006a), although the statistical analysis allows to assert that TCK2 is better than A1.

5.2 Comparison of metaheuristics

In the next experiment, the performance of the VLS proposed in Section 4 is tested against the best-so-far metaheuristic available for the problem, i.e. the HTS. In addition, we include the best-so-far metaheuristic available for the multi-machine assembly case, i.e. the AIS algorithm by Al-Anzi and Allahverdi (2013). As discussed in Section 1, the problem with several assembly machines can be seen as a generalisation of our problem, although it is not clear that efficient procedures for the latter would also yield good solutions when there is only one machine, due to a potentially different structure of the space of solutions.

Since VLS does not have a pre-defined stopping criterion (i.e. the maximum number of iterations in VLS is not fixed), different values of the maximum number of iterations have been tested to cover several scenarios ranging from an average computation time of around one second to one minute. These CPU time values correspond to $maxit \in \{100, 500, 1000, 2000, 5000\}$. The results in terms

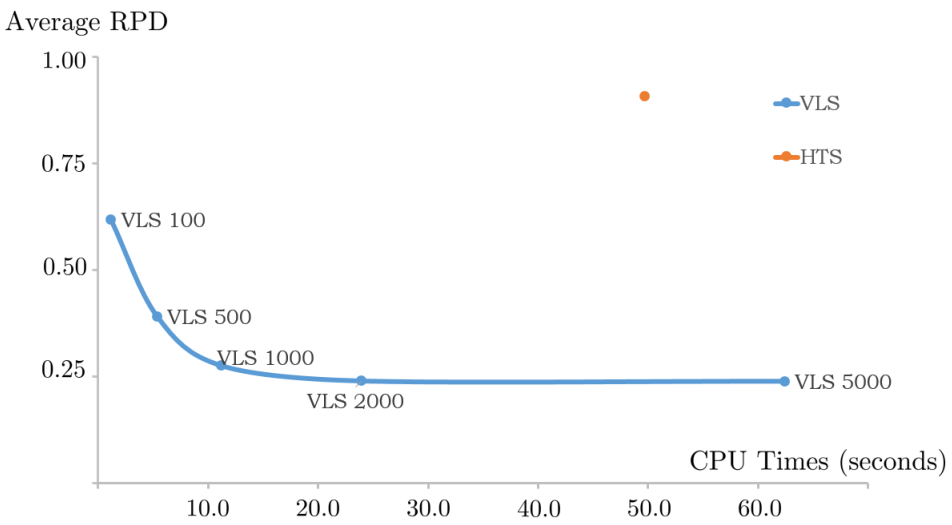

Figure 4: Trade-off between quality of solutions and computational effort

of Average RPD and CPU time are shown in Tables 2 and 3 respectively. As it can be seen in Table 2 even the version of VLS with $maxit = 100$ outperforms HTS for all problem sizes. This fact is remarkable taking into account that the average CPU effort for VLS in this case is around 50 times smaller than for HTS, as shown in Table 3. Also, the standard deviation in all problem sizes is smaller for the different versions of VLS than for HTS, which speaks for the robustness of the proposed method. The Average RPD values of VLS decrease with the number of iterations, although the ratio of improvement decreases with the number of iterations, perhaps due to the proximity of the solutions found to their best values. Regarding AIS, its performance is the worst among the algorithms under comparison, both regarding the quality of solutions and the required CPU time, which is probably pointing at the best suitability of the GPI neighbourhood scheme of VLS and HTS for the problem under consideration, as compared to the random swapping of AIS. This may be reinforced by the poor results obtained by some preliminary experiments in which VLS was modified to use the random swapping neighbourhood instead of GPI.

Finally, Figure 4 shows the trade-off between CPU time and Average RPD for the different methods (AIS is excluded in view of its poor results). If interpreted as a Pareto set regarding these two objectives, it could be stated that VLS dominates HTS for all values of $maxit$ (with the exception of $maxit = 5000$, which also takes longer CPU times).

		HTS		AIS		VLS 100		VLS 500		VLS 1000		VLS 2000		VLS 5000	
m	n	Aver.	St. Dev.	Aver.	St. Dev.	Aver.	St. Dev.	Aver.	St. Dev.	Aver.	St. Dev.	Aver.	St. Dev.	Aver.	St. Dev.
2	20	0.416	0.397	1.186	0.837	0.302	0.282	0.249	0.222	0.249	0.222	0.249	0.222	0.249	0.222
	40	0.791	0.528	1.618	0.554	0.441	0.285	0.277	0.228	0.277	0.228	0.277	0.228	0.277	0.228
	60	0.725	0.362	1.448	0.536	0.447	0.297	0.353	0.246	0.353	0.246	0.353	0.246	0.353	0.246
	80	0.654	0.281	1.344	0.453	0.366	0.235	0.246	0.157	0.196	0.136	0.196	0.136	0.196	0.136
	100	0.693	0.382	1.472	0.352	0.289	0.194	0.198	0.154	0.169	0.146	0.169	0.146	0.169	0.146
	120	0.630	0.283	1.355	0.298	0.331	0.189	0.239	0.156	0.141	0.129	0.135	0.123	0.135	0.123
4	20	0.577	0.371	1.463	0.819	0.444	0.407	0.347	0.396	0.347	0.396	0.347	0.396	0.347	0.396
	40	0.856	0.383	1.624	0.782	0.654	0.397	0.408	0.313	0.408	0.314	0.408	0.314	0.408	0.314
	60	1.006	0.492	1.801	0.639	0.643	0.477	0.259	0.272	0.212	0.214	0.212	0.214	0.212	0.214
	80	0.988	0.357	1.707	0.430	0.644	0.377	0.346	0.239	0.188	0.181	0.188	0.181	0.188	0.181
	100	1.140	0.409	1.777	0.505	0.572	0.310	0.399	0.270	0.217	0.171	0.187	0.177	0.187	0.177
	120	1.107	0.422	1.737	0.547	0.486	0.274	0.362	0.289	0.209	0.169	0.145	0.139	0.145	0.139
6	20	0.616	0.697	1.477	0.906	0.412	0.347	0.284	0.271	0.284	0.271	0.284	0.271	0.269	0.273
	40	0.908	0.478	1.565	0.617	0.771	0.419	0.293	0.240	0.293	0.240	0.293	0.240	0.293	0.240
	60	0.952	0.413	1.892	0.680	0.758	0.411	0.344	0.238	0.263	0.206	0.263	0.206	0.263	0.206
	80	1.165	0.540	2.153	0.713	0.999	0.447	0.628	0.410	0.319	0.263	0.288	0.259	0.288	0.259
	100	1.082	0.422	1.855	0.524	0.715	0.318	0.535	0.267	0.200	0.174	0.144	0.136	0.144	0.136
	120	1.010	0.347	1.822	0.554	0.720	0.250	0.575	0.246	0.250	0.214	0.120	0.131	0.119	0.132
8	20	0.644	0.513	1.547	0.868	0.446	0.388	0.332	0.335	0.332	0.335	0.286	0.294	0.286	0.294
	40	0.902	0.555	1.739	0.765	0.812	0.496	0.405	0.329	0.405	0.329	0.405	0.329	0.405	0.329
	60	1.054	0.497	1.629	0.730	0.845	0.459	0.312	0.305	0.208	0.189	0.208	0.189	0.208	0.189
	80	1.222	0.592	2.118	0.695	0.871	0.452	0.535	0.285	0.238	0.149	0.207	0.154	0.207	0.154
	100	1.238	0.427	2.095	0.492	0.865	0.360	0.637	0.296	0.362	0.288	0.202	0.214	0.200	0.215
	120	1.390	0.399	2.435	0.704	0.966	0.275	0.781	0.226	0.481	0.254	0.164	0.160	0.164	0.160
Total		0.907	0.505	1.702	0.696	0.617	0.412	0.389	0.307	0.275	0.248	0.239	0.235	0.238	0.235

Table 2: Average values and Standard Deviation of the RPD obtained by the VLS, AIS and the HTS

m	n	HTS	AIS	VLS 100	VLS 500	VLS 1000	VLS 2000	VLS 5000
2	20	0.972	19.331	0.016	0.076	0.147	0.289	0.717
	40	5.395	56.328	0.093	0.510	1.029	2.045	5.093
	60	15.279	122.972	0.304	1.635	3.392	6.992	16.934
	80	32.714	304.603	0.709	3.857	8.168	16.730	41.502
	100	57.750	467.824	1.471	7.273	15.502	31.973	80.040
	120	95.150	408.032	2.843	12.626	26.512	56.528	143.408
4	20	1.082	21.942	0.017	0.084	0.163	0.316	0.800
	40	6.485	62.913	0.103	0.578	1.186	2.372	5.968
	60	18.568	140.619	0.368	1.857	3.872	7.865	19.369
	80	40.502	336.654	0.826	4.280	8.857	18.755	47.536
	100	74.717	518.258	1.710	7.911	17.211	36.909	95.490
	120	123.684	491.445	3.528	14.099	29.527	64.574	168.969
6	20	1.217	22.287	0.019	0.095	0.185	0.367	0.928
	40	7.603	72.533	0.122	0.665	1.395	2.803	7.140
	60	22.651	150.049	0.444	2.194	4.728	9.584	23.711
	80	49.952	377.199	1.010	4.951	10.268	21.843	56.081
	100	94.249	595.731	2.166	9.779	20.374	44.646	117.831
	120	158.257	587.626	4.080	16.787	34.197	75.362	203.522
8	20	1.342	23.901	0.025	0.102	0.203	0.406	1.003
	40	8.493	79.715	0.141	0.765	1.575	3.140	8.083
	60	26.084	191.498	0.505	2.459	5.228	10.834	26.952
	80	58.295	419.795	1.201	5.682	11.845	25.448	65.093
	100	109.457	663.848	2.578	11.334	23.440	50.636	134.386
	120	183.047	671.365	4.790	19.693	40.335	83.771	227.995
Total		49.706	283.603	1.211	5.387	11.223	23.924	62.440

Table 3: Average CPU time (seconds) by the HTS, AIS and the different versions of VLS

6 Conclusions

In this paper we address the 2-stage assembly scheduling problem with the objective of minimising the total completion time. We first analyse some problem properties and correctly formulate some flaws found in the work by Al-Anzi and Allahverdi (2006a). Based on the ideas of the predominance of one of the stages in the total completion time, a new constructive heuristic has been developed. The computational experience carried out shows that this proposal results in a much better performance than the existing constructive heuristics, yielding an average deviation from the best known solution of around 2%, which is around five times smaller than that of the second best performing heuristic, and, at the same time, it requires negligible computation times (around than 0.01 seconds in the biggest instances). For those scenarios where a higher quality of solutions is required, a variable local search algorithm has been proposed. This algorithm outperforms the best-so-far metaheuristic for the problem, yielding solutions with higher quality in much lesser CPU time. Using an average CPU time of one second, the variable local search algorithm finds solutions of around 0.6% of the best-known solution. If an average minute of CPU time is allowed, the results are on average of around 0.2% of the best-known solution.

As a result, with the heuristics presented in the paper, the state-of-the-art of the approximate

methods for the 2-stage assembly scheduling problem with total completion time has been substantially modified. Regarding future research lines, the idea of the constructive heuristic with the estimation of the contribution of the unscheduled jobs could be extended to other objectives. Additionally, it can be enhanced for the multi-machine assembly scheduling problem, where the number of assembly machines in the second stage is higher than one, or to the case of multi-stage assembly scheduling problems.

References

Al-Anzi, F. and Allahverdi, A. (2012). Better heuristics for a two-stage multi-machine assembly scheduling problem to minimize total completion time. *International Journal of Operations Research*, 9:66–75. cited By 3.

Al-Anzi, F. S. and Allahverdi, A. (2006a). A Hybrid Tabu Search Heuristic for the Two-Stage Assembly Scheduling Problem. *International Journal of Operations Research*, 3(2):109–119.

Al-Anzi, F. S. and Allahverdi, A. (2006b). Empirically discovering dominance relations for scheduling problems using an evolutionary algorithm. *International Journal of Production Research*, 44(22):4701–4712.

Al-Anzi, F. S. and Allahverdi, A. (2007). A self-adaptive differential evolution heuristic for two-stage assembly scheduling problem to minimize maximum lateness with setup times. *European Journal of Operational Research*, 182(1):80–94.

Al-Anzi, F. S. and Allahverdi, A. (2013). An artificial immune system heuristic for two-stage multi-machine assembly scheduling problem to minimize total completion time. *Journal of Manufacturing Systems*, 32(4):825–830.

Allahverdi, A. and Al-Anzi, F. S. (2006). A PSO and a Tabu search heuristics for the assembly scheduling problem of the two-stage distributed database application. *Computers & Operations Research*, 33(4):1056–1080.

Allahverdi, A. and Al-Anzi, F. S. (2009). The two-stage assembly scheduling problem to minimize total completion time with setup times. *Computers and Operations Research*, 36(10):2740–2747.

Fernandez-Viagas, V. and Framinan, J. M. (2015). A new set of high-performing heuristics to minimise flowtime in permutation flowshops. *Computers & Operations Research*, 53:68–80.

Framinan, J., Leisten, R., and Ruiz, R. (2014). *Manufacturing Scheduling Systems: An Integrated View of Models, Methods, and Tools*. Springer.

Framinan, J. and Perez-Gonzalez, P. (2017). New approximate algorithms for the customer order scheduling problem with total completion time objective. *Computers and Operations Research*, 78:181–192.

Hariri, A. M. A. and Potts, C. N. (1997). A branch and bound algorithm for the two-stage assembly scheduling problem. *European Journal of Operational Research*, 103(3):547–556.

Koulamas, C. and Kyparisis, G. J. (2001). The three-stage assembly flowshop scheduling problem. *Computers & Operations Research*, 28:689–704.

Lee, C.-Y., Cheng, T. C. E., and Lin, B. M. T. (1993). Minimizing the makespan in the 3-machine assembly-type flowshop scheduling problem. *Management Science*, 39(5):616–625.

Leung, J. Y. T., Li, H., and Pinedo, M. (2005). Order Scheduling in an Environment with Dedicated Resources in Parallel. *Journal of Scheduling*, 8(5):355–386.

Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100.

Potts, C. N., Sevast'janov, S. V., Strusevich, V. A., Van Wassenhove, L. N., and Zwaneveld, C. M. (1995). The Two-stage Assembly Scheduling Problem: Complexity and Approximation. *Operations Research*, 43(2):346–355.

Shoaardebili, N. and Fattahi, P. (2015). Multi-objective meta-heuristics to solve three-stage assembly flow shop scheduling problem with machine availability constraints. *International Journal of Production Research*, 53(3):944–968.

Sun, X., Morizawa, K., and Nagasawa, H. (2003). Powerful heuristics to minimize makespan in fixed, 3-machine, assembly-type flowshop scheduling. *European Journal of Operational Research*, 146(3):498–516.

Sung, C. S. and Kim, H. A. (2008). A two-stage multiple-machine assembly scheduling problem for minimizing sum of completion times. *International Journal of Production Economics*, 113(2):1038–1048.

Tozkapan, A., Kirca, Ö., and Chung, C. S. (2003). A branch and bound algorithm to minimize the total weighted flowtime for the two-stage assembly scheduling problem. *Computers & Operations Research*, 30(2):309–320.

Xiong, F., Xing, K., Wang, F., Lei, H., and Han, L. (2014). Minimizing the total completion time in a distributed two stage assembly system with setup times. *Computers and Operations Research*, 47:92–105.