International Journal of
Operations Research

International Journal of Operations Research Vol. 9, No. 2, 66−75 (2012)

Better Heuristics for a Two-Stage Multi-Machine Assembly Scheduling Problem to Minimize Total Completion Time

Fawaz S. Al-Anzi[1] **and Ali Allahverdi**[2*]

[1]Department of Computer Engineering, Kuwait University, P.O. Box 5969, Safat Kuwait

[2]Department of Industrial and Management Systems Engineering, Kuwait University, P.O. Box 5969, Safat Kuwait

Received January 2012; Revised April 2012; Accepted April 2012

Abstract **—** We address the two-stage multi-machine assembly scheduling problem to minimize total completion times of all jobs. The first stage consists of m independently working machines where each machine produces its own component. The second stage consists of two independent and identical assembly machines. The processing of a job at the second stage cannot start until its m components, produced by the m machines at the first stage, are complete. This problem has been recently addressed in the literature for which an efficient heuristic, called SAK, was developed for the case when m=2. In this paper, we address the problem for the case $m\geq 2$. We propose a hybrid tabu search (HTS) heuristic and show that the overall average error of SAK is more than twice that of HTS while the average CPU time of SAK is five times that of HTS. This clearly indicates that the heuristic HTS is much better than the only existing heuristic available for the problem, i.e., SAK. We also propose two more heuristics, called SDE and NSDE, and show that the overall average error of SDE is about half of that of HTS. Furthermore, we show that the overall average error of NSDE is about one third that of SDE.

Keywords **—** Scheduling, assembly flowshop, total completion time, heuristic

1. INTRODUCTION

The two-stage assembly flowshop problem consists of two stages where there are m machines at the first stage while there is only a single assembly machine at the second stage. There are n jobs to be scheduled and each job has $m+1$ operations. For each job, the first m operations are conducted at the first stage by m machines in parallel and a final operation in the second stage by the assembly machine. The last operation at the second stage may start only after all m operations at the first stage are completed. This problem has many applications in industry, and hence, has received an increasing attention of researchers. Lee *et al.* (1993) described an application in a fire engine assembly plant while Potts et al. (1995) described an application in personal computer manufacturing. Another application of the problem is in the area of queries scheduling on distributed database systems, Allahverdi and Al-Anzi (2006a). In short, many real life problems can be modeled as a two-stage assembly flowshop scheduling problem. In particular, manufacturing of almost all items may be modeled as a two-stage assembly scheduling problem.

The two-stage assembly flowshop scheduling problem has been addressed with respect to different performance measures. The problem was addressed with respect to makespan performance measure by Lee *et al.* (1993), Potts et al. (1995), Hariri and Potts (1997), Haouari and Daouas (1999), Sun et al. (2003), and Allahverdi and Al-Anzi (2006b) while it was addressed with respect to maximum lateness performance measure by Allahverdi and Al-Anzi (2006a) and Al-Anzi and Allahverdi (2007).

For some scheduling environments, each completed job is needed as soon as it is processed. In such environments, one is interested in minimizing total completion times of all jobs. This objective is particularly important in environments where reducing inventory or holding cost is of primary concern. The literature survey reveals that the only researchers addressing total completion time criterion in a two-stage flowshop problem are Tozkapan et al. (2003), Al-Anzi and Allahverdi (2006), and Allahverdi and An-Anzi (2009). Tozkapan *et al.* (2003) developed a lower bound and a dominance relation, and utilized the lower bound and the dominance relation in a branch and bound algorithm. They also proposed two heuristics to find an upper bound for their branch and bound algorithm. On the other hand, Al-Anzi and Allahverdi (2006) proposed two algorithms and showed that one algorithm is optimal under certain conditions. They also proposed a tabu search and a simulated annealing heuristic for the problem. Moreover, they proposed a hybrid tabu search heuristic and showed by computational analysis that their proposed hybrid tabu search heuristic is more efficient and can easily be used for large sized problems. Allahverdi and An-Anzi (2009) addressed the same problem with the same objective function but where

* Corresponding author's email: ali.allahverdi@ku.edu.kw

setup times are considered as separate from processing times. They proposed a dominance relation and presented heuristics to solve the problem.

Some research has also been conducted on the problem with a weighted sum of makespan and mean completion time performance measures. Allahverdi and Al-Anzi (2008) proposed three heuristics and showed that one of them performed better than the others. Torabzadeh and Zandieh (2010) proposed a new heuristic which was indicated to perform better than the best heuristic of Allahverdi and Al-Anzi (2008).

The literature aforementioned addressed the two-stage assembly flowshop scheduling problem where there exists only one machine at the second stage, i.e., at the assembly stage. Recently, Sung and Kim (2008) addressed the two-stage assembly flowshop problem consisting of two stages where there are two machines at the first stage (i.e., m=2) while there are two independent and identical assembly machines at the second stage. There are n jobs to be scheduled and each job has three operations. For each job, the first two operations are conducted at the first stage by the two machines at the first stage independently and a final operation in the second stage by one of the two assembly machines, working independently, at the second stage. The last operation at the second stage may start only after the two operations at the first stage are completed. Sung and Kim (2008) presented a branch-and-bound algorithm and provided an efficient heuristic for the problem with the total completion time minimization objective function. Their heuristic is called SAK heuristic in this paper. Sung and Kim (2008) also presented various applications of the problem. In this paper, we consider a generalization of the problem with the same objective function and propose three heuristics; a hybrid tabu search (HTS), a self-adaptive differential evolution (SDE), and a new self-adaptive differential evolution (NSDE). We show that our proposed heuristics outperform that of Sung and Kim (2008) (SAK). In the generalization of the problem, we assume that there are two stages where there are m ($m \geq 2$) machines in parallel at the first stage while there are two machines in parallel at the second stage which is the assembly stage. Each job consists of a set of m+1 operations. The first m operations are completed at stage one in parallel (operation 1 on machine 1, operation 2 on machine 2, …, operation m on machines m) while the last operation is performed on one of the two machines at the second (assembly) stage.

In Section 2, we present the mathematical model of the problem. In Section 3, the proposed heuristics are presented while the evaluation of the proposed and earlier developed heuristics is conducted in Section 4. Finally, a summary of the work and direction for the future research are given in Section 5.

2. PROBLEM FORMULATION

There are two stages where there are m machines in parallel at stage one while there are two machines in parallel at the second (assembly) stage. Each job consists of a set of m+1 operations. The first m operations are completed at stage one in parallel (operation 1 on machine 1, operation 2 on machine 2, …, operation m on machines m) while the last operation is performed on one of the two machines at the second (assembly) stage.

We assume that n jobs are simultaneously available at time zero and that preemption is not allowed, i.e., any started operation has to be completed without interruptions.

Let

$t_{i,j}$: operation time of job i on machine k (at stage one), i=1, …, n, k=1, …, m where job i consists of k operations and each operation is performed by one of the m machines at the first stage.

$t_{[j,k]}$: operation time of the job in position j on machine k (at stage one), where job i consists of k operations and each operation is performed by one of the m machines at the first stage.

p_i: operation time of job i on assembly machine (at stage two), which can be processed by either machine 1 or machine 2 at the second stage with the same speed

$p_{[j]}$: operation time of the job in position j on assembly machine (at stage two), which can be processed by either machine 1 or machine 2 at the second stage with the same speed

A_1: Availability time of machine 1 at the assembly stage, which indicates when the machine becomes idle so that it can handle the incoming job

A_2: Availability time of machine 2 at the assembly stage, which indicates when the machine becomes idle so that it can handle the incoming job

$C_{[j]}$: completion time of the job in position j of a given sequence, which denotes the time when all operations at the first stage, and the operation at the second stage are completed.

TCT: Total completion time of all jobs on the second stage

It should be noted that the operation time of job i is the same regardless of which of the two assembly machines it is going to be processed on. It should be also noted that job k is complete once all of its operations $t_{k,j}$ (j=1, …, m) and p_k are completed where the operation p_k may start only after all operations $t_{k,j}$ (j=1, …, m) have been completed. It is known that permutation schedules are dominant. In other words, the sequences of the jobs at the first and second stages are the same. Therefore, only permutation schedules are considered.

The completion time of the job in position j of a given sequence can be computed as:

Step 1: Set A_1=A_2=0, j=1

Step 2: If $j>n$, stop
Step 3: If $A_1<=A_2$

$$C_{[j]} = \max\left\{ \max_{k=1,\dots,m} \left\{ \sum_{i=1}^{j} t_{[i,k]} \right\}, A_1 \right\} + p_{[j]}$$

$$A_1 = A_1 + p_{[j]}$$

Step 4: If $A_1>A_2$

$$C_{[j]} = \max\left\{ \max_{k=1,\dots,m} \left\{ \sum_{i=1}^{j} t_{[i,k]} \right\}, A_2 \right\} + p_{[j]}$$

$$A_2 = A_2 + p_{[j]}$$

Step 5: If $j=j+1$, go to Step 2

Step 3 states that machine 2 at the assembly stage is busy while machine 1 at the assembly stage is available, therefore, the current job is assigned to machine 1 at the assembly machine. Accordingly, the completion time of that job is computed by the first equation given in the Step 3 and the availability of machine 1 is updated next. Otherwise, the availability of time machine 2 is smaller than that of machine 1, and hence, the current job is assigned to machine 2. Again, then the completion time of this job is computed by the first equation in Step 4, and next the availability of machine 2 is updated.

Once completion times of the jobs are computed as described above, the total completion time is computed as

$$\text{TCT} = \sum_{i=1}^{n} C_{[i]}$$

3. PROPOSED HEURISTICS

It should be noted that when there is only one machine at the first stage, i.e., $m=1$, then the problem reduces to the regular two machine flowshop scheduling problem. It is known that the regular two machine flowshop scheduling problem with the objective function considered in this paper is NP-hard, Garey et al. (1976). Therefore, our problem is also NP-had. Hence, we propose heuristics to solve the problem. We propose three heuristics; a hybrid tabu search (HTS), a self-adaptive differential evolution (SDE), and a new self-adaptive differential evolution (NSDE). These three heuristics are described in subsections 3.2-3.4 after the description of the only existing heuristic of SAK for the problem in the next subsection.

3.1 Sung and Kim Heuristic (SAK)

The problem considered in this paper was addressed by Sung and Kim (2008) for the special case of having only two machines at the first stage, i.e., $m=2$. Sung and Kim (2008) proposed a branch-and-bound algorithm and also developed an efficient heuristic algorithm for the problem. The heuristic proposed by Sung and Kim (2008) is a processing-time-based pairwise exchange heuristic mechanism which is called SAK in this paper. Since we consider the problem with an arbitrary m value, we generalize their heuristic to handle m ($m \geq 2$) machines scenario at the first stage as their heuristic was developed for the case of $m=2$. However, for a fair comparison, we will also consider the case of $m=2$ in the computational experiments in addition to $m>2$.

The processing-time-based pairwise exchange relaxed heuristic mechanism SAK can be summarized as follows:

- Step 1: Compute $\text{psum}_j = (\sum_{i=1}^{m} t_{[j,i]} + p_{[j]})$ for each job at position j. Let K denote the initial job sequence obtained by arranging the associated psum_j values in the Shortest Processing Time (SPT) order. Then, let H denote all the intermediate sequences that can result from the pairwise exchange operations implemented on the initial heuristic sequence K. The best of these sequences can be obtained by repeatedly updating of the pairwise exchange operations.
- Step 2: Set $x = 1, y = x+1$, exchange the xth job and the yth job in the initial sequence K. If TCT is improved, declare it as the current H.
- Step 3: Update $y = y+1$, exchange the xth job and yth job in the sequence K. If TCT is improved, declare it as the current H.
- Step 4: If $y = n$, go to Step 5, else go to Step 3.
- Step 5: Update $x = x+1$. Set $y = x+1$, exchange the xth job and the yth job in the sequence H. If TCT is improved, declare it as the current H.
- Step 6: If $x = n$, declare the current H as the final sequence K and stop, otherwise go to Step 5.

3.2 Hybrid Tabu Search (HTS)

Al-Anzi and Allahverdi (2006) proposed three heuristics, namely, a simulated annealing, a tabu search, and a hybrid tabu search (HTS) for the problem addressed in this paper with only one assembly machine at the second stage. They showed that HTS outperforms both of the other heuristics by a large margin. They also showed that HTS significantly outperforms the two earlier developed heuristics by Tozkapan et al. (2003). Since HTS is known to be the best heuristic for the problem, we consider a generalized version of the HTS as one of the heuristics for the problem addressed in this paper with two independent assembly machines at the second stage. The following is an algorithmic description of the generalized version of the heuristic HTS. It should be noted that among the two assembly machines, the first available one is assigned to a job having all of its m components on the first stage completed. The sequences S1, S2, and S3 in the following algorithm are the same sequences that are described by Al-Anzi and Allahverdi (2006), and are described in the Appendix.

The main idea behind the hybrid tabu search heuristic is to introduce the concept of probability of accepting exchanges that are not necessarily of the best objective function of the neighborhood of the tabu search heuristic. In other words, the hybrid tabu search heuristic is allowed to accept exchanges that are not in the tabu list.

Hybrid Tabu Search Heuristic (HTS)

```
Begin
Initialize Tabu $h$ list with maximum size of 4
Select the best sequence among $S1$, $S2$, and $S3$ as the current sequence
Let $L1$ = value of the objective function with initial sequence
Let $T1 = 0.1$
While $T1 \geq 0.0001$
Begin
    Repeat 50 times
    Begin
        Pick two random positions $j$ and $k$ where $(j,k)$ is not in the Tabu list $h$
        Swap jobs in the positions of $j$ and $k$
        Let $L2$ = value of the objective function with the sequence after the swap
        Set $j_2=j$ and $k_2=k$
        Swap back jobs in the positions of $j$ and $k$
    For all possible combinations of $j$ and $k$ (i.e., explore all neighborhood)
    Begin
            If $(j,k)$ is not in the Tabu list then
            Begin
                Swap jobs in the positions of $j$ and $k$
                Let $L3$ = value of the objective function after the swap
                If $(L3 < L2)$ then
                Begin
                    Set $L2 = L3$, $j_2=j$ and $k_2=k$
                Else
                    Compute d and f where
                        $d = \left|\frac{L3 - L1}{L1}\right|$
                        $f = e^{-100*\frac{d}{T1}}$
                        if ($L3 > L2$ and with probability $f$) then
                          Begin
                            Set $L2 = L3$, $j_2=j$ and $k_2=k$
                        End If
                End If
                Reverse swap
            End If
        End For
        Swap jobs in the positions of $j_2$ and $k_2$
```

```
                Add ($j_2$,$k_2$) to front of Tabu
                If the Tabu maximum list size is exceeded, then delete the item at the end of the list $h$
                Update $L1$ = value of the objective function with current sequence
        End Repeat
        Let $T1$ = $T1$*0.98
End While
End Heuristic
```

It should be noted that we have fine tuned the parameters of the HTS for the current problem, see Table 1.

3.3 A Self-Adaptive Differential Evolution (SDE)

Differential evolution (DE) heuristics have been applied to solve a wide range of problems in different areas including scheduling, e.g., Onwubolu and Davendra (2006). Finding the best values for the control parameters in DE heuristics is a time consuming task. Therefore, a version of DE, where the control parameters are self-adaptive, was proposed by Omran *et al.* (2005). This new version is called *Self-adaptive Differential Evolution* (SDE). Al-Anzi and Allahverdi (2007) adapted this SDE to the two-stage assembly flowshop scheduling problem to minimize maximum lateness. They showed that SDE performed much better than other heuristics. Since the problem addressed in this paper is also two-stage assembly flowshop problem, we use this SDE as one of the heuristics for the current problem. Since we have a different objective function and the fact that there are two assembly machines at the second stage rather than one, we have to adapt the SDE to the current problem. First of all the objective function that Al-Anzi and Allahverdi (2007) considered was maximum lateness (L_{max}). However, the objective function considered in the current paper is total completion time TCT), and hence, in the steps of the SDE algorithm proposed by Al-Anzi and Allahverdi (2007) "L_{max}" should be replaced with "TCT". In other words, the partial sequences are evaluated based on TCT values rather than L_{max} values. The second difference is that Al-Anzi and Allahverdi (2007) considered the problem with setup times while in this paper we have not considered setup times. Hence, setup time values should be set to zero which indicates setup times are ignored. The third difference is that there are two or more parallel machines at the second stage, and hence, a job completing all of its operations at the first stage can be assigned to any of the k parallel machines at the second stage whichever one becomes available first.

The steps of the SDE are not described in this paper in order to avoid repetition. It can easily be obtained from Al-Anzi and Allahverdi (2007). It should be noted that among the two assembly machines, the first available one is assigned to a job having all of its m components on the first stage completed. Similar to HTS, we have fine tuned the parameters of SDE for the current problem, see Table 1.

3.4 A New Self-Adaptive Differential Evolution (NSDE)

Allahverdi and Al-Anzi (2009) proposed a modification to the SDE that was proposed by Al-Anzi and Allahverdi (2007), and called it New Self-Adaptive Differential Evolution (NSDE). The difference between the SDE that was proposed by Al-Anzi and Allahverdi (2007) and the NSDE proposed by Allahverdi and Al-Anzi (2009) is the introduction of a new step in the SDE. In this step, a random pair wise exchange is conducted which results in children of crossover operator with a certain probability. The problem addressed in this paper is similar to the one addressed by Allahverdi and Al-Anzi (2009), we use this NSDE as one of the heuristics for the current problem. However, since we do not consider setup times in this paper and the fact that there are two assembly machines at the second stage rather than one, we have to adapt the NSDE of Allahverdi and Al-Anzi (2009) to the current problem.

It should be noted that in the current paper there are two or more parallel machines at the second stage unlike only one machine at the second stage of the problem addressed by Allahverdi and Al-Anzi (2009). Therefore, a job completing all of its k operations at the first stage can be assigned to any of the machines at the second stage whichever one becomes available first. The second difference is that Al-Anzi and Allahverdi (2009) considered the problem with setup times while in this paper we have not considered setup times. Hence, setup time values should be set to zero which indicates setup times are ignored.

The steps of the NSDE are not described in this paper in order to avoid repetition. It can be obtained from Allahverdi and Al-Anzi (2009). It should be noted that among the two assembly machines, the first available one is assigned to a job having all of its m components on the first stage completed. Similar to HTS and SDE, we have fine tuned the parameters of NSDE for the current problem, see Table 1.

4. COMPUTATIONAL EXPERIMENTS

In this section, we first describe how the parameters of the proposed three heuristics are set in subsection 4.1**.**, and then the evaluation of heuristics is described in subsection 4.2.

4.1 Setting Heuristic Parameters

To optimize the performance of the proposed heuristics, fine tuning of their parameters has been performed. An initial estimate for the best value of a given parameter of a heuristic is obtained by changing the values of that parameter while keeping all other parameters of the heuristic as constant. After some experimentations and after no major changes in the performance have been noticed, the parameters were set as given in Table 1.

Table 1. Parameter fine tuning for the proposed heuristics

Best value	**Range**	**Parameter**	**Heuristic**
38,000	20,000 – 50,000 with an increment of 5,000	I_{max}	HTS
3	2 – 7 with an increment of 1	h	
3n	n – 5n with an increment of n	*POP*	SDE
3n	n – 5n with an increment of n	*GEN*	
2n	n – 5n with an increment of n	*CP*	
4/6	1/6 - 5/6 with an increment of 1/6	y	
0.125	0.1 – 0.2 with an increment of 0.005	Pr_i	
3n	n – 5n with an increment of n	*POP*	NSDE
3n	n – 5n with an increment of n	*GEN*	
2n	n – 5n with an increment of n	*CP*	
4/6	1/6 - 5/6 with an increment of 1/6	y	
0.125	0.2 – 0.2 with an increment of 0.005	Pr_i	

4.2 Proposed Heuristic Evaluation

The proposed HTS, NSDE, and SDE heuristics and SAK heuristic were implemented in C under GCC-3.4.2 compiler using the built-in math library. The machine used was a Sun Fire V880 with 4 CPU processors of 900MHz running under Solaris Version 9.0 operating system with 8GB RAM. To measure the effectiveness of the heuristics, we compared the performance of the four heuristics against each other and against a random solution.

The processing times were randomly generated from a uniform distribution [1, 100] on all m machines at the first stage as well as on the two assembly machines at the second stage. In the scheduling literature, most researchers have used this distribution in their experimentation. The reason for using a uniform distribution with a wide range is that the variance of this distribution is large and if a heuristic performs well with such a distribution, it is likely to perform well with other distributions.

Problem data were generated for different number of jobs: 30, 40, 50, 60, and 70. The experimentation was conducted for the number of machines at the first stage being 2, 4, 6 or 8. We compared the performance of the heuristics using two measures: average percentage error (Error) and standard deviation (Std) out of thirty replicates. The percentage error is defined as 100* (*TCT of the heuristic* – *TCT of the best heuristic*)/(*TCT of the best heuristic*) where TCT denotes total completion time.

There are 20 combinations for different values of *n* (30, 40, 50, 60, 70) and *m* (2, 4, 6, 8). Thirty replicates were generated for each combination, and therefore, a total of 600 instances were generated and evaluated. For the sake of brevity, the results will not be tabulated. The summary of the results are presented in Figures 1-4. A random solution was also considered for comparison purposes. However, the average error for the random solution was very large (on average, more than 30 times of the error of the worst heuristic) compared with the other heuristics, and therefore, is not reported in the figures.

The overall average errors and the standard deviation of the errors of all the heuristics are summarized in Figure 1 and Figure 2, respectively. Figure 1(2) illustrates the overall average errors (standard deviation of the errors) with respect to the number of jobs (*n*). The figures indicate that the performance of the heuristics get a bit closer to each other as the number of jobs increases. Figure 3 shows the results versus the number of machines at the first stage. Figure 3 indicates that as the number of machines at the first stage increases, the performances of the heuristics get closer to each other. The figures 1-3 indicate that HTS performs much better than SAK (as expected) including for the case of *m*=2, and SDE and NSDE perform better than HTS. Comparison of the performances of SDE and NSDE reveals that NSDE outperforms SDE.

The overall average errors of NSDE, SDE, HTS, and SAK over all *n* and *m* values were 0.08, 0.24, 0.46, and 1.01, respectively. It should be noted that the overall average error of SAK was more than double that of HTS while the average CPU time of SAK was five times that of HTS, see Figure 4. This clearly indicates that the heuristic HTS is much better than the only existing heuristic available for the problem, i.e., SAK.

The performances HTS and SAK heuristics were statistically tested by using a *t* test. The following hypothesis testing was conducted for all replica combinations for comparing the performances of HTS and SAK statistically:

Null Hypothesis: The average error of HTS = The average error of SAK

Alternative Hypothesis: The average error of HTS < The average error of SAK

The null hypotheses were rejected for 100% of the combinations at 99% significance level. This implies that the average error of HTS is statistically smaller than that of SAK.

Furthermore, since the best performing heuristics are NSDE and SDE, the results for these two heuristics were statistically tested by using a *t* test. More specifically, the following hypothesis testing was conducted for all replica combinations;

Null Hypothesis: The average error of NSDE = The average error of SDE

Alternative Hypothesis: The average error of NSDE < The average error of SDE

The null hypotheses were rejected for 95% of the combinations at 99% significance level. This implies that the average error of NSDE is statistically smaller than that of SDE. Before concluding that NSDE outperforms SDE one has to also consider CPU time in addition to the average error. The CPU times of all the heuristics are summarized in Figure 4. As can be seen from the figure, the CPU times of SDE and NSDE are close to each other. Moreover, even for the largest size of the problem (*n*=70), the CPU time of NSDE is less than 45 seconds. Therefore, it can now be stated that NSDE outperforms SDE.

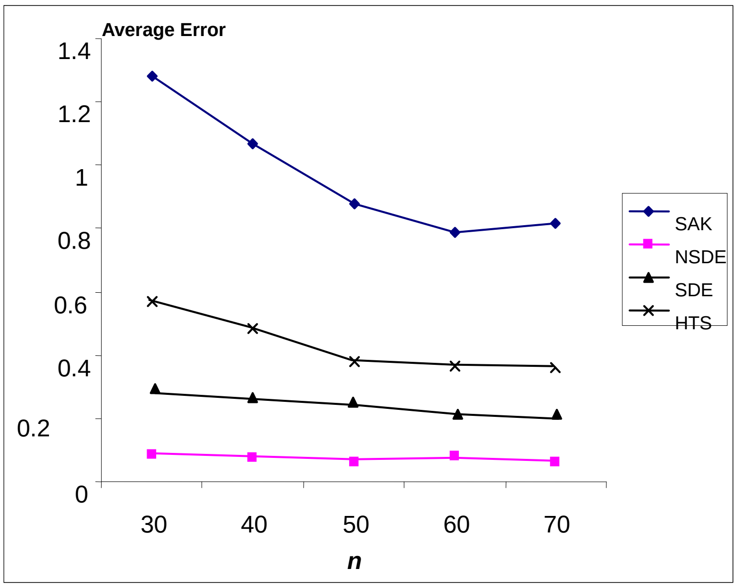

Figure 1. The average error versus the number of jobs

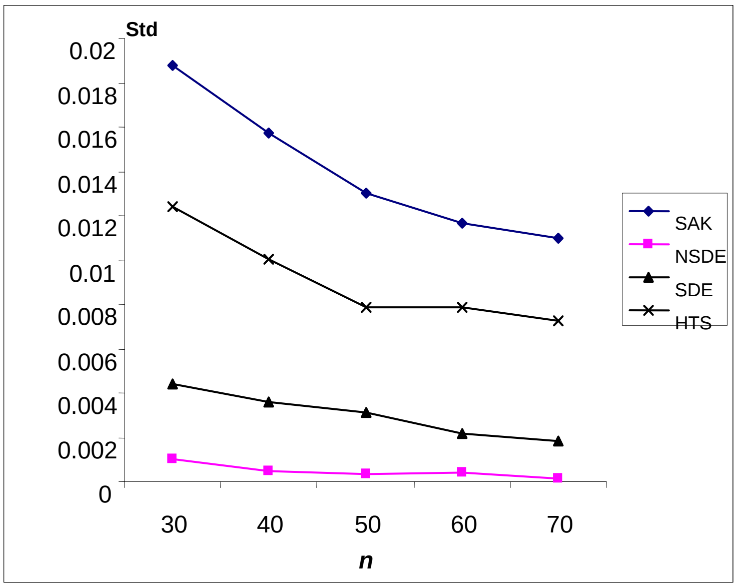

Figure 2. The standard deviation of the error versus the number of jobs

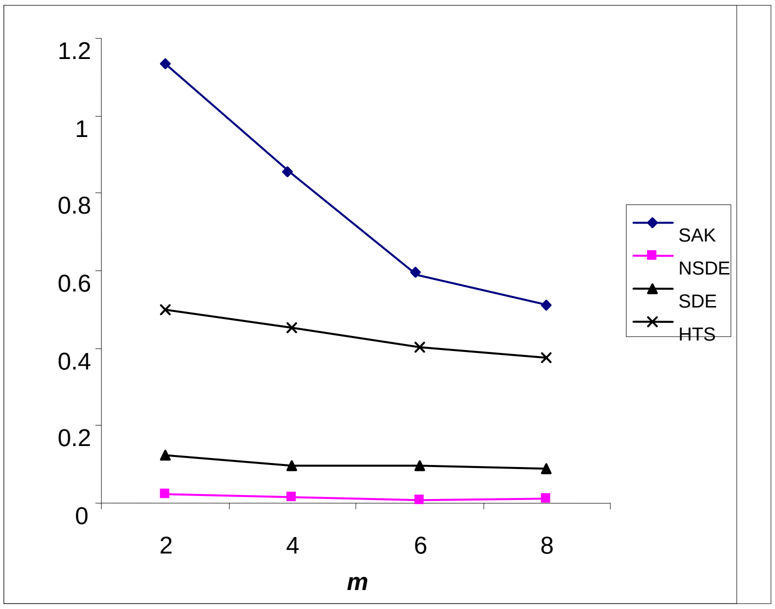

Figure 3. The average error versus the number of machines at the first stage

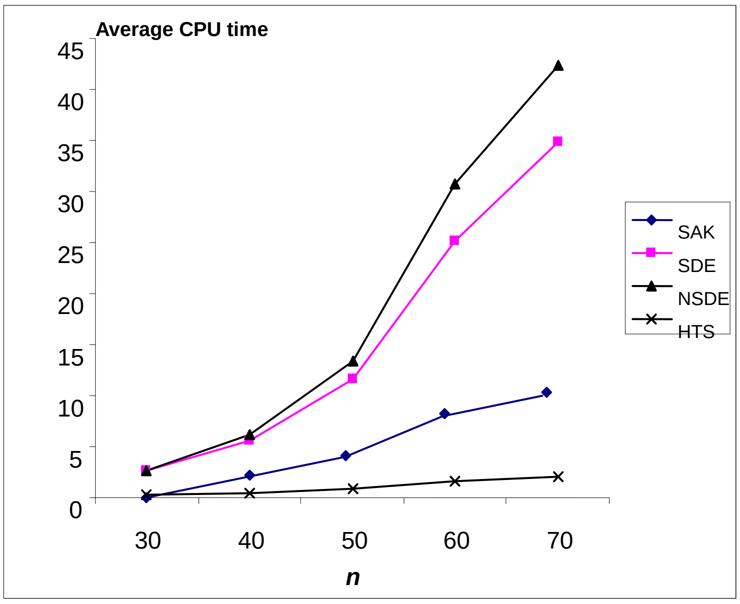

Figure 4. The average CPU time (in seconds) versus number of jobs

5. SUMMARY AND FUTURE RESEARCH

The scheduling problem of a two-stage assembly flowshop is considered with the objective of minimizing total completion time of all n available jobs. The first stage consists of m machines which produce their own products. Then, these m components are then processed by one of the two parallel machines at the second stage. The problem is NP-hard since it is known that the problem is NP-hard when there exists only one machine at the first stage and one machine at the second stage. This problem was addressed earlier in the literature where an efficient heuristic, called SAK, was developed for the case when m=2. We addressed the same problem for the case when m≥2. We proposed three heuristics, called SDE, NSDE, and HTS. The proposed three heuristics along with the only existing heuristic SAK were evaluated through randomly generated sets of data. The computational analysis indicated that the overall average errors of NSDE, SDE, HTS, and SAK over all n and m values were 0.08, 0.24, 0.46, and 1.01, respectively. Therefore, the overall average error of SAK was more than twice that of HTS while the average CPU time of SAK was five times that of HTS. This clearly indicates that the heuristic HTS is much better than the only existing heuristic available for the problem, i.e., SAK. The compuatioanl analysis further indicated that the heuristic SDE performed better than the heuristic HTS and that the heuristic NSDE perfomed better than that of SDE.

We assumed that setup times are included in the processing times in this paper. However, this may not be necessarily the case for some scheduling environments, e.g., Allahverdi and Soroush (2008) and Allahverdi et al. (2008). For such scheduling problems, the heuristics developed in this paper may not yield desirable results. Therefore, a possible extension is to address the problem considered in the paper where setup times are treated as independent from processing times.

ACKNOWLEDGMENT

This research was supported by Kuwait University Research Administration Grant No. EO02/10. The authors would like to thank two anonymous referees for their comments which greatly improved the quality and presentation of the paper.

REFERENCES

1. Al-Anzi, F.S. and Allahverdi, A. (2006). A hybrid tabu search heuristic for the two-stage assembly scheduling problem. *International Journal of Operations Research,* 3:109-119.
2. 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: 80-94.
3. Allahverdi, A. and Al-Anzi, F.S. (2006a). A NSDE and a tabu Search Heuristics for Assembly Scheduling Problem of the Two-Stage Distributed Database Application. *Computers & Operations Research*, 33: 1056-1080.
4. Allahverdi, A. and Al-Anzi, F.S. (2006b). Evolutionary heuristics and an algorithm for the two-stage assembly scheduling problem to minimize makespan with setup times. *International Journal of Production Research*, 44: 4713-4735.
5. Allahverdi, A. and Al-Anzi, F.S. (2008). The two-stage assembly flowshop scheduling problem with bicriteria of makespan and mean completion time. *International Journal of Advanced Manufacturing Technology*, 37: 166-177.

6. Allahverdi, A. and Al-Anzi, F.S. (2009). The two-stage assembly scheduling problem to minimize total completion time with setup times. *Computers & Operations Research,* 36: 2740-2747.
7. Allahverdi A., Ng, C.T., Cheng, T.C.E. and Kovalyov, M.Y. (2008). A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187: 985-1032.
8. Allahverdi, A., Soroush, H.M. (2008). The significance of reducing setup times/setup costs. *European Journal of Operational Research*, 187: 978-984.
9. Garey, M.R., Johnson, D.S. and Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics and Operations Research*, 1: 117-129.
10. Haouari M. and Daouas T. (1999). Optimal scheduling of the 3-machine assembly-type flow shop. *RAIRO Recherche Operationnelle*, 33: 439-445.
11. 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: 547-556.
12. 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: 616-625.
13. Omran, M., Salman, A. and Engelbrecht, A. (2005). Self-Adaptive Differential Evolution. *Proceedings of the International Conference on Computational Intelligence and Security*, December, Xi'an, China, pp. 192-199.
14. Onwubolu, G. and Davendra, D. (2006). Scheduling flow shops using differential evolution algorithm. *European Journal of Operations Research*, 171: 674-692.
15. 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: 346-355.
16. 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: 498-516.
17. 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: 1038-1048.
18. Torabzadeh, E. and Zandieh, M. (2010). Cloud theory-based simulated annealing approach for scheduling in the two-stage assembly flowshop. *Advances in Engineering Software*, 41: 1238-1243.
19. Tozkapan, A., Kirca, O. 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: 309-320.

APPENDIX

We use an example of scheduling five jobs to illustrate the concept of neighborhood of a sequence for our problem. Let us assume that at some point of time we have three sequences $S1$, $S2$, and $S3$ as follows.

$S1$ = [2, 3, 4, 1, 5]
$S2$ = [2, 5, 4, 1, 3]
$S3$ = [2, 4, 1, 3, 5]

In the above example, it is easy to see that sequences $S1$ and $S2$ are closer to each other than sequences $S1$ and $S3$. This is because, we can obtain $S2$ from $S1$ by exchanging jobs 3 and 5 in the sequence while to obtain $S3$ from $S1$ one needs to reorder jobs 3, 4 and 1. In this context, we define the distance between two sequences as the number of mismatches between the sequences. In the above example, the distance between $S1$ and $S2$ is 2 while that of $S1$ and $S3$ is 3. Notice that the minimum distance we can achieve according to this definition is 2 for any sequence. Hence, in our heuristic, the neighborhood of a sequence can be defined as all sequences that have a distance of 2 from the current sequence. A complete set of neighborhood of distance two can be achieved by simply swapping all pairs of jobs in a sequence.