International Journal of
Operations Research

International Journal of Operations Research Vol. 3, No. 2, 109−119 (2006)

A Hybrid Tabu Search Heuristic for the Two-Stage Assembly Scheduling Problem

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 2006; *Revised April 2006*; *Accepted May 2006*

Abstract—In this paper, we address the two-stage assembly scheduling problem where there are *m* machines at the first stage and an assembly machine at the second stage. The objective is to schedule the jobs on the machines so that total completion time of all *n* jobs is minimized. Optimal solutions are obtained for two special cases. A simulated annealing heuristic, a tabu search heuristic, and a hybrid tabu search heuristic are proposed for the general case. The proposed heuristics are compared with the existing heuristics and shown to be more efficient. The computational analysis shows that the proposed hybrid tabu search heuristic improves the error rate by about 60 and 90 percent over tabu search and simulated annealing heuristics, respectively, where the CPU time of all the three heuristics is almost the same.

Keywords—Scheduling, Assembly flowshop, Total completion time, Simulated annealing, Tabu search

1. INTRODUCTION

In a two-stage assembly flowshop scheduling problem, there are *n* jobs where each job has *m* + 1 operations and there are *m* + 1 different machines to perform each of these operations. Each machine can process only one job at a time. For each job, the first *m* operations are conducted at the first stage in parallel and a final operation in the second stage. Each of m operations at the first stage is performed by a different machine and the last operation at the second stage may start only after all m operations at the first stage are completed. The two-stage assembly scheduling problem has several applications in industry. Potts et al. (1995) described an application in personal computer manufacturing where central processing units, hard disks, monitors, keyboards, and etc. are manufactured at the first stage, and all the required components are assembled to customer specification at a packaging station (the second stage). Lee et al. (1993) described another application in a fire engine assembly plant. The body and chassis of fire engines are produced in parallel, in two different departments. When the body and chassis are completed and the engine has been delivered (purchased from outside), they are fed to an assembly line where the fire engine is assembled.

Another practical application of this problem is possibly in the area of distributed database systems. In recent years, there has been a rapid trend toward the distribution of computer systems over multiple sites that are interconnected via a communication network, Elmasri and Navathe (1999). It is common with current technology to develop forms or reports that require tens of embedded queries that retrieve information from different sites on the networks and assemble them in one final report, Ceri and Pelagatti (1984). For this scheduling problem, it may be possible to look at the problem from a higher level of abstraction. The details about database and multimedia servers that are typically addressed at the server level, e.g., memory caching, disk scheduling etc., is not considered, see Figure 1.

This is an on-line problem where requests keep on arriving. However, a static version of the problem can be assumed where there is a fixed number of requests for a given period of time. This assumption is not restrictive since the requests are collected until the system becomes available from the previous batch of requests. Once it becomes available, the batch of accumulated requests are considered for processing next. Hence, this can be considered as a static system within a window of time that is equivalent in duration to the time taken to process the previously collected batch of requests.

The two-stage assembly flowshop scheduling problem was introduced independently by Lee et al. (1993) and Potts et al. (1995). Lee et al. (1993) considered the problem with *m* = 2 while Potts et al. (1995) considered the problem with an arbitrary *m*. Both studies addressed the problem with respect to makespan minimization and both proved that the problem with this objective function is NP-hard in the strong sense for *m* = 2. Lee et al. (1993) discussed a few polynomially solvable cases and presented a branch and bound algorithm. Moreover, they proposed three heuristics and analyzed their error bounds. Potts et al.

[*] Corresponding author's email: alanzif@eng.kuniv.edu.kw

(1995) showed that the search for an optimal solution may be restricted to permutation schedules. They also showed that any arbitrary permutation schedule has a worst-case ratio bound of two, and presented a heuristic with a worst-case ratio bound of $2 - 1/m$. Hariri and Potts (1997) also addressed the same problem, developed a lower bound and established several dominance relations. They also presented a branch and bound algorithm incorporating the lower bound and dominance relations. Another branch and bound algorithm was proposed by Haouari and Daouas (1999). Sun et al. (2003) also considered the same problem with the same makespan objective function and proposed heuristics to solve the problem. Koulamas and Kyparisis (2001) generalized the two-stage problem to a three-stage assembly scheduling problem. They proposed several heuristics and analyzed the worst-case ratio bounds of the proposed heuristics for the makespan problem.

Tozkapan et al. (2003) considered the two-stage assembly scheduling problem but with the total weighted flowtime performance measure. They showed that permutation schedules are dominant for the problem with this performance measure. They developed a lower bound and a dominance relation, and utilized the bound and dominance relation in a branch and bound algorithm. They also proposed two heuristics to find an upper bound for their branch and bound algorithm. They indicated by computational analysis that problems with up to 20 jobs and $m = 10$ can be solved in a reasonable time with their proposed branch and bound algorithm. They suggested developing efficient heuristics for large sized problems.

In this paper, we consider the same problem that Tozkapan et al. (2003) addressed. We propose two algorithms and show that one algorithm is optimal with total completion time criterion under certain conditions. We also propose a tabu search and a simulated annealing heuristic for the problem. Moreover, we propose a hybrid tabu search heuristic and show by computational analysis that the proposed hybrid tabu search heuristic is more efficient and can easily be used for large sized problems.

2. FORMULATION AND THEORETICAL RESULTS

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. Each job consists of a set of $m + 1$ operations. The first m operations are completed at stage one in parallel while the last operation is performed at stage two. Let

$t_{i,j}$:	operation time of job i on machine j, $i = 1, \ldots, n$, $j = 1, \ldots, m$,
$t_{[i,j]}$:	operation time of the job in position i on machine j,
p_i:	operation time of job i on assembly machine,
$p_{[i]}$:	operation time of the job in position i on assembly machine,
$C_{[i]}$:	completion time of the job in position i.

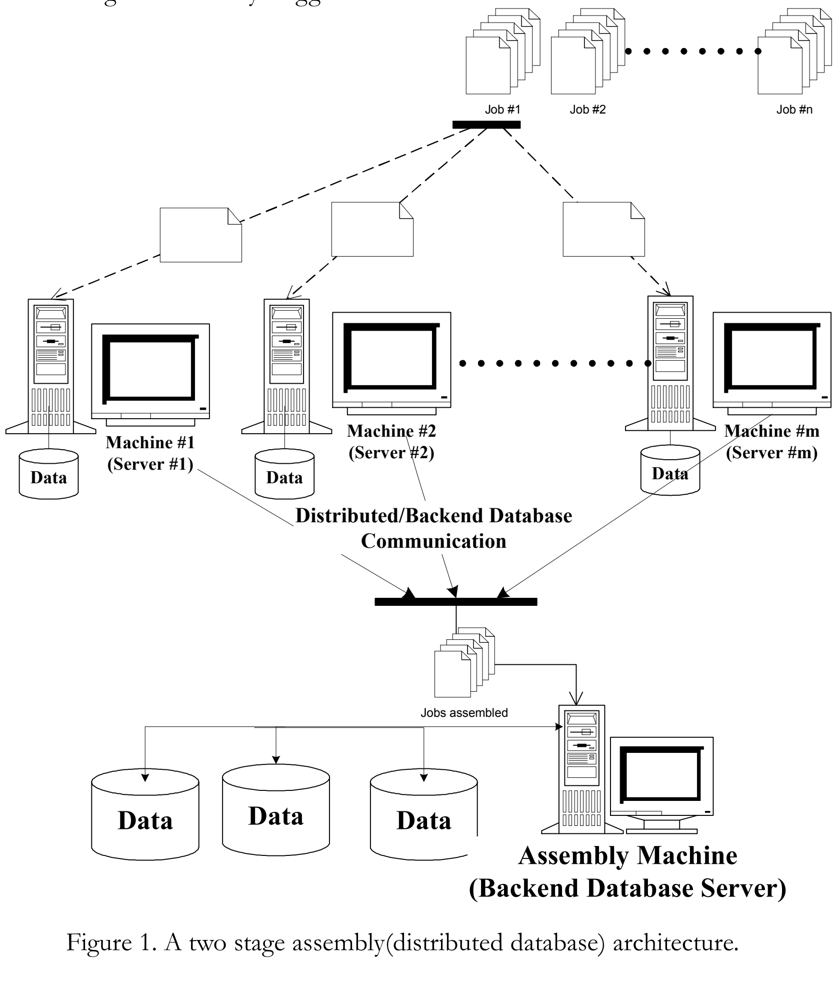

Figure 1. A two stage assembly(distributed database) architecture.

Note 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. Tozkapan et al. (2003) showed that permutation schedules are dominant with respect to total flowtime (completion time) criterion. Therefore, we restrict our search for the optimal solution to permutation schedules. In other words, the sequence of jobs on all of the machines, including the assembly machine, is the same.

It can be shown that the completion time of the job in position j is as follows:

$$C_{[j]} = \max\left\{\max_{k=1,\ldots,m}\left\{\sum_{i=1}^{j} t_{[i,k]}\right\}, C_{[j-1]}\right\} + p_{[j]}, \text{where } C_{[0]} = 0.$$

In the following we give two algorithms to find a solution for the problem. For the general case, their performances will be compared with those of the heuristics to be proposed in the next section as well as those of the previously known two upper bounds of Tozkapan et al. (2003) that was used in their branch and bound algorithm.

Algorithm 1.

Step 1. Set π_1 = {all the jobs}, π_2 = {empty}

Step 2. For j = 1..n − 1 do

Step 3. For all $i \in \pi_1$, compute

$$T_i = \max_{k=1,\ldots,m}\left\{\sum_{r=1}^{j-1} t_{[r,k]}(\pi_2) + t_{i,k}\right\},$$

where $t_{[r,k]}(\pi_2)$ denotes operation time of the job in position r on machine k in sequence π_2. Assign the job with the smallest T_i to the j^{th} position of the sequence π_2, and remove this job from the sequence π_1. In case of ties, assign the job with the smallest p_i to the j^{th} position of the sequence π_2.

Step 4. End for

Step 5. The sequence π_2 is the solution.

The above algorithm assigns the job, among the unassigned jobs, to the next available position of a partial sequence such that the maximum completion time on the first stage machines is minimum given that a partial sequence has already been obtained. The intuition behind this is that a job cannot start processing on the second stage unless all its operations on the first stage have been completed. Therefore, it is desired that this duration is as small as possible. It is important to note that the above algorithm ignores the processing time of a job on the second stage. The following algorithm (Algorithm 2) takes that also into account.

Algorithm 2.

Algorithm 2 is similar to Algorithm 1 except at *Step* 3 where T_i's are computed as follows:

$$T_i = \max_{k=1,\ldots,m}\left\{\sum_{r=1}^{j-1} t_{[r,k]}(\pi_2) + t_{i,k}\right\} + p_i.$$

Notice that Algorithm 1 ignores the processing times of the jobs on the assembly machine while Algorithm 2 takes this into account. It is important to note that when the processing times of the jobs on the assembly machine are dominant, then, one expects that it has to explicitly be taken into account.

If the conditions given in the following theorem (Theorem 1) are satisfied, then the first stage dominates the second one, and hence, an optimal sequence can be obtained by considering the first stage processing times only. On the other hand, when the second stage dominates the first one, then an optimal sequence can be obtained by sequencing the jobs based on their processing times of the second stage, which is described in Theorem 2.

Theorem 1. Algorithm 1 minimizes the total completion time if

$$\min_{i=1,\ldots,n}\left[\max_{k=1,\ldots,m}\{t_{i,k}\}\right] \geq \max_{j=1,\ldots,n}\{p_j\}.$$

Proof. Let

$$\sigma_j = \max_{k=1,\ldots,m}\left\{\sum_{i=1}^{j} t_{[i,k]}\right\} - \sum_{i=1}^{j-1} p_{[i]}, \text{ and } \Delta_j = \max\{\sigma_1, \sigma_2, \ldots, \sigma_j\}.$$

Then it can be shown that

$$C_{[j]} = \sum_{i=1}^{j} p_{[i]} + \Delta_j$$

Notice that

$$\sigma_j = \sigma_{j-1} + \max_{k=1,\ldots,m}\left\{t_{[j,k]}\right\} - p_{[j-1]}$$

Therefore, $\sigma_j \geq \sigma_{j-1}$, since $\max_{k=1,\ldots,m}\left\{t_{[j,k]}\right\} \geq p_{[j-1]}$ as a result of the assumption that $\min_{i=1,\ldots,n}\left[\max_{k=1,\ldots,m}\{t_{i,k}\}\right] \geq \max_{j=1,\ldots,n}\{p_j\}$.

This means that $\Delta_j = \sigma_j$.

Therefore,

$$\begin{aligned} C_{[j]} &= \sum_{i=1}^{j} p_{[i]} + \Delta_j = \sum_{i=1}^{j} p_{[i]} + \sigma_j \\ &= \sum_{i=1}^{j} p_{[i]} + \max_{k=1,\ldots,m}\left\{\sum_{i=1}^{j} t_{[i,k]}\right\} - \sum_{i=1}^{j-1} p_{[i]} \\ &= \sum_{i=1}^{j} \max_{k=1,\ldots,m}\left\{\sum_{i=1}^{j} t_{[i,k]}\right\} + p_{[j]}. \end{aligned}$$

Given the completion times of the jobs, the total completion time (*TCT*) can be computed as:

$$
\begin{aligned}
TCT &= \sum_{j=1}^{n} C_{[j]} \\
&= \sum_{j=1}^{n}\sum_{i=1}^{j} \max_{k=1,\ldots,m}\left\{\sum_{i=1}^{j} t_{[i,k]}\right\} + \sum_{j=1}^{n} p_{[j]} \\
&= \sum_{j=1}^{n}\sum_{i=1}^{j} \max_{k=1,\ldots,m}\left\{\sum_{i=1}^{j-1} t_{[i,k]} + t_{[j,k]}\right\} + \sum_{j=1}^{n} p_{[j]}.
\end{aligned}
$$

Note that the term $\sum_{j=1}^{n} p_{[j]}$ is constant and independent of a sequence. Hence, the minimization of *TCT* is equivalent to the minimization of the term $\sum_{j=1}^{n}\sum_{i=1}^{j} \max_{k=1,\ldots,m}\left\{\sum_{i=1}^{j-1} t_{[i,k]} + t_{[j,k]}\right\}$. Algorithm 1 step by step builds a sequence such that this double summation is minimized. Therefore, the sequence obtained by Algorithm 1 is optimal.

Theorem 2.
Arranging jobs in non-decreasing (increasing) order of p_i minimizes *TCT* if $\max_{i=1,\ldots,n}\left[\max_{k=1,\ldots,m}\{t_{i,k}\}\right] \le \min_{j=1,\ldots,n}\{p_j\}$ and if job i where $p_i = \min_{j=1,\ldots,n}\{p_j\}$ satisfies $\max_{k=1,\ldots,m}\{t_{i,k}\} \le \max_{\substack{j=1,\ldots,n\\ k=1,\ldots,m}}\{t_{j,k}\}$.

Proof.
If $\max_{i=1,\ldots,n}\left[\max_{k=1,\ldots,m}\{t_{i,k}\}\right] \le \min_{j=1,\ldots,n}\{p_j\}$, then $\Delta_j = \sigma_1 = \max_{k=1,\ldots,m}\{t_{[1,k]}\}$. Therefore,

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

$$
\begin{aligned}
TCT = &\sum_{j=1}^{n}\sum_{i=1}^{j} p_{[i]} + \sum_{j=1}^{n} \max_{k=1,\ldots,m}\{t_{[1,k]}\} \\
&= \sum_{j=1}^{n} (n-j+1) p_{[j]} + \sum_{j=1}^{n} \max_{k=1,\ldots,m}\{t_{[1,k]}\}.
\end{aligned}
$$

Clearly the first term of *TCT* is minimized by arranging the jobs in increasing order of $p_{[i]}$ and the second term is minimized by the fact that job i where $p_i = \min_{j=1,\ldots,n}\{p_j\}$ satisfies $\max_{k=1,\ldots,m}\{t_{i,k}\} \le \max_{\substack{j=1,\ldots,n\\ k=1,\ldots,m}}\{t_{j,k}\}$.

3. HEURISTICS

The two-stage assembly scheduling problem with total flowtime (completion time) criterion was addressed only by Tozkapan et al. (2003). They proposed two heuristics to find an upper bound for their proposed branch and bound algorithm. We refer to their heuristics in this paper as TKC1 and TKC2 denoting their first and second upper bounds, respectively.

In this section, we propose three heuristics for the problem. The first proposed heuristic is a simulated annealing heuristic, the second is a tabu search heuristic, and the third is a hybrid tabu search heuristic. All three heuristics start with a given initial sequence and iteratively improve on it until a stopping criterion is met. Hence, it is important to start with a good sequence. In the following we give the description of initial sequences that will be used in all the three heuristics.

3.1 Initial sequences

One initial sequence is obtained by ordering all the jobs in increasing order of p_i. This initial sequence is called *S*1. It is expected that when assembly machine dominates the first stage machines (i.e., when processing times on assembly machine are larger than those of the first stage machines), then, ordering the jobs based on Shortest Processing Time (SPT) on the assembly machine will yield a good solution. The SPT rule is known to perform well in general for total completion time criterion. The second initial sequence is obtained by considering the case that the first stage machines dominate the assembly machine. In this case, the sequence is obtained by ordering the jobs in increasing order of $\max_{k=1,\ldots,m}\{t_{i,k}\}$ which is called *S2*. A third sequence is obtained by ordering the jobs in increasing order of $\max_{k=1,\ldots,m}\{t_{i,k}\} + p_i$ where both stages are taken into account. This sequence is called *S*3.

3.2 Simulated annealing heuristic (SA)

Simulated annealing has been used to solve scheduling problems, e.g., Sadegheih (2006), Low (2005), and Mika et al. (2005). The main idea behind the proposed simulated annealing heuristic is to have a number of iterations where in every iteration of the heuristic there is a single random pair exchange in the sequence (other neighborhoods are used in different versions of simulated annealing). If the exchange improves the objective function, then it accepts the exchange and the new sequence is preserved. If the objective function does not improve, then it is only allowed to accept the exchange with some small probability *p*. As the number of iterations increases, the probability *p* for which the heuristic is allowed to accept an exchange that does not improve the objective function is reduced exponentially. This reduction in the probability is usually expressed as a function of a start temperature (*T*1) that is reduced by a cooling factor to reach a final (freezing) temperature. Notice that the temperature cooling factor used in the proposed heuristic is exponential. However, in a general SA heuristic, it needs not to have an exponential cooling factor. This technique of reducing the probability of accepting non-improving exchanges has proven to be very useful in escaping local optimum's during the course of search for global optimum. The following is an algorithmic description of the heuristic.

Simulated Annealing Heuristic (SA)

```
Begin
  Let $T1$ = 0.1
  Select the best sequence among $S1$, $S2$, and $S3$ as the
  current sequence
  While $T1 \geq 0.0001$
  Begin
    Repeat 50 times
    Begin
    Let $L1$ = value of the objective function with current
    sequence

    Pick two random positions $j$ and $k$
    Swap jobs in the positions of $j$ and $k$
    Let $L2$ = value of the objective function after the swap

    If $L2<L1$ then accept swap
    If $L2>L1$ then accept the move with probability $f$
    where $d = \left|\frac{L2-L1}{L1}\right|$
          $f = e^{-d/T1}$
  End Repeat
  Let $T1 = T1 \times 0.98$
  End While
End Heuristic
```

Setting the parameters for the proposed simulated annealing heuristic is essential in achieving a good performance. An initial estimate for the best value of a given parameter is obtained by changing the values of that parameter while keeping all other parameters as constant. We used the following values as initial estimates of the parameters; ($T1$ = .5, .1, and .01), (cooling factor = 0.99, 0.98, 0.97, 0.96, and 0.95), and (final temperature = 0.005, 0.001, 0.0005, 0.0001, 0.00005, 0.00001). Once these initial values are determined, then, the method of factorial experimental design (three values for each parameter including the initial best value of that parameter, one value above and one value below that value) is used to fine tune the values of the parameters. After these experimentations, the parameters for the simulated annealing heuristic are set as follows; the initial temperature $T1$ is set to 0.1, the cooling factor is set to 0.98, the final temperature is set to 0.0001, and the number of iterations per fixed temperature is set to 50 since not significant improvement has been observed beyond this value.

Finding an initial sequence for simulated annealing heuristics is common in the scheduling literature. For example, Sridhar and Rajendran (1993) used a similar approach for the hybrid flowshop scheduling problem with total completion time criterion. Sridhar and Rajendran (1993) first obtained an initial sequence, then, used a similar simulated annealing heuristic to find a solution. Note that the two-stage assembly flowshop problem has some similarity with the two-stage hybrid flowshop problem with m machines on the first stage and one machine on the second stage. In the hybrid problem, a job can be processed by any one of the m machines at the first stage before it is processed on the machine at the second stage. On the other hand, in a two-stage assembly flowshop problem each job has to be processed on all the m machines on the first stage before it can be processed at the assembly machine.

3.3. Tabu search heuristic (Tabu)

Some scheduling problems have been solved by using different algorithms including Tabu search heuristic, e.g., Al-Turki et al. (2001), Al-Fawzan and Haouari (2005), Ruiz and Maroto (2005), and Liaw (2003). The main idea behind a tabu search heuristic is to have a large number of iterations, where in every iteration of the heuristic there is a choice of a best sequence from neighborhood of the current sequence. The heuristic is only allowed to choose the new sequence if this sequence has the best objective function of the neighborhood of the current sequence and the new move is not in the previous h iterations. The last h moves are kept in a list for checking. This list is called the *tabu list.* This technique has been proven to be very useful for escaping loops as well as escaping a local optimum during the course of search for a global optimum.

We use an example of scheduling five jobs to illustrate the concept of neighborhood of a sequence for our problem (in general, for any scheduling problem). Let us assume that at some point of time we have three sequences *Seq-1, Seq-2*, and *Seq-3* as follows:

Seq-1 = [2, 3, 4, 1, 5]
Seq-2 = [2, 5, 4, 1, 3]
Seq-3 = [2, 4, 1, 3, 5]

In the above example, it is easy to see that sequences *Seq-1* and *Seq-2* are closer to each other than sequences *Seq-1* and *Seq-3.* This is because, we can obtain *Seq-2* from *Seq-1* by exchanging jobs 3 and 5 in the sequence while to obtain *Seq-3* from *Seq-1* 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 *Seq-1* and *Seq-2* is 2 while that of *Seq-1* and *Seq-3* is 3. Notice that the minimum distance we can achieve according to this definition is 2 for any sequence. Hence, in our tabu 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.
The following is an algorithmic description of the heuristic.

Tabu Search Heuristic (*Tabu*)

```
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 $j2 = j$ and $k2 = k$
    Swap back jobs in the positions of $j$ and $k$

    For all possible combinations of $j$ and $k$ (i.e., explore
    all neighborhood)
      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 $L2 = L3$, $j2 = j$ and $k2 = k$
        End If

        Reverse swap
      End If
    End For
    Swap jobs in the positions of $j2$ and $k2$
    Add $(j2, k2)$ 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
```

In the above description of the heuristic, a neighborhood of distance 2 is explored. This step requires examining all possible combinations of pairs of two jobs. The neighborhood concept can be extended to explore a distance more than 2. For example, in a neighborhood of distance 3, it is needed to inspect all possible combinations of triplicates of three jobs. Since a neighborhood of distance more than 2 requires significantly more computational time, we have chosen to use a neighborhood of distance 2 in this paper.

Note that the parameter $T1$ is only used as a geometric iteration index. $T1$ specifies the number of iterations that is required by the outer most loop of the Tabu algorithm. This is used in order to have the exact number of iterations that the other two heuristics have in order to have a fair comparison (i.e., to have the same computational time). For the proposed tabu search heuristic, setting the parameters is essential in achieving a good performance. After some experimentations as explained earlier in section 3.2, the parameters for the tabu search heuristic are set as follows; the total number of iterations is set to the same value of simulated annealing heuristic (for a fair comparison), and the tabu list size is set to four. The tabu list size of four was found to be the best performing value for the tested range of 1 to 7.

3.4 A hybrid tabu search heuristic (H-tabu)

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. This concept was introduced into the tabu search by looking at the main concept behind the simulated annealing heuristic. This concept was integrated into the tabu search, which we call a hybrid tabu search heuristic. The hybrid tabu search heuristic is allowed to accept exchanges that are not in the tabu list. As in tabu search, an exchange is allowed if this sequence has the best objective function of the neighborhood of the current sequence and the new sequence is not in the previous h iterations. It is also allowed in the hybrid tabu search heuristic to have exchanges of a second type. The second type is an exchange that is not in the tabu list and does not necessarily have the best value of the objective function with a small probability p. This probability is reduced exponentially as the search for the global optimum progresses. The following is an algorithmic description of the heuristic.

Replace the following lines of code form the regular tabu search heuristic:

```
If $L3 < L2$ then
Begin
    Set $L2 = L3$, $j2 = j$ and $k2 = k$
End If
```

by the following lines of code in the hybrid tabu search heuristic:

```
If $(L3 < L2)$ then
Begin
    Set $L2 = L3$, $j2 = j$ and $k2 = k$
Else
  Compute $d$ and $f$, where
  $d = \left| \frac{L3 - L1}{L1} \right|$
  $f = e^{-100*d/T1}$
  if ($L3 > L2$ and with probability $f$)
  Begin
      Set $L2 = L3$, $j2 = j$ and $k2 = k$
  End
End If
```

For the proposed hybrid tabu search heuristic, parameters are set to the same values as in simulated annealing and tabu search heuristics. Notice that the value of f has been fine tuned by multiplying the exponent of the simulated annealing by 100 which has been found to have a better performance. It should be noted that when computing the objective functions for sequences after swapping jobs in, say positions i and j ($i < j$), there is no need to compute $C_{[1]}$ up to $C_{[i-1]}$ again since they are the same before and after the swap. This can contribute to a significant savings in computational time. This is true for all the three algorithms.

4. HEURISTIC COMPARISON

The two existing heuristics of TKC1 and TKC2 and the three proposed SA, Tabu, and H-tabu heuristics along with the proposed two algorithms (Algorithm 1 and Algorithm 2) 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. Heuristics and algorithms are evaluated with respect to average error, standard deviation of the error, and the percentage of times yielding the best solution.

The processing times were randomly generated from a uniform distribution (0, 100) on all the m machines on the first stage while from a uniform distribution (1, 100) on the last stage. The reason for using a uniform distribution (0, 100), rather than (1, 100), on the first stage machines is that it could be that some of the jobs may not necessarily need to be processed by all the machines at the first stage. In the scheduling literature, the use of uniform distribution is common, e.g., Wang et al. (1997), Pan and Chen (1997), Al-Anzi and Allahverdi (2001), and Allahverdi and Al-Anzi (2002). 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 will most likely perform well with other distributions. However, in order to test the heuristics and algorithms for other types of data, we also consider the cases when processing times follow exponential distribution.

Problem data were generated for a different number of jobs for the range of 20 to 120 in increment of 20. Different number of machines at the first stage have been considered for experimentation to observe the behavior of the heuristic and algorithms. We vary the number of machines at the first stage as 2, 4, 6, or 8. We compare the performance of the heuristics using three measures: average error (Error), standard deviation (Std), and the percentage of the number of the best solutions (NBS). The error is defined as (Heuristic Solution − Best Solution)/(Worst Solution − Best Solution). Notice that according to this definition, the best performing heuristic will have an error of zero while the worst one will have an error of 100 %. Thirty replicates were generated for each instance of the twenty four (6 × 4) combinations of the number of jobs and machines.

Tables 1 and 2 show the results of running the existing and the proposed heuristics and algorithms for different number of jobs and machines with respect to the error and standard deviation, respectively. Each entry in the tables represents the average of the thirty replicates. Each heuristic and algorithm is evaluated for the same configuration to ensure accurate assessment of the different heuristics. It is clear from the tables that, as expected, TCK1 and TCK2 did not perform well since they were developed as upper bounds for a branch and bound algorithm.

Algorithm 1 and Algorithm 2 also did not perform well since they were designed for special cases. Table 2 shows that the standard deviation of Algorithm 1 is high compared to that of TCK1, TCK2, and Algorithm 2. This means that for some replicates where the special conditions (stated in Theorem 1) were nearly satisfied, the algorithm performed well (error was small) while for some others it performed badly. In order to further investigate the performance of the heuristics where it is more likely that the conditions are satisfied, two other processing time distributions have been tested. Table 3 illustrates the results of these distributions for 40 jobs where U(a, b; c, d) means the processing time distributions of the jobs on the first stage is uniform between a and b and on the second stage (i.e., the assembly machine) uniform between c and d. The average errors of Algorithm 1 for U(0, 100; 1, 100), U(0, 100; 1, 50), and U(0, 100; 1, 20) are 9.536, 1.755, and 1.676, respectively. Note that it is more likely that the conditions of Theorem 1 are satisfied in the cases of U(0, 100; 1, 20) and U(0, 100; 1, 50) than that of U(0, 100; 1, 100). Therefore, the averages of the former cases are significantly less than that of the latter case for Algorithm 1.

The same initial sequence (i.e., the best of $S1$, $S2$, and $S3$) is used in Tabu, SA, and H-tabu. The performance of $S1$, $S2$, and $S3$ was also evaluated and it has been observed that about 1, 81, and 18% of the time $S1$, $S2$, *and* $S3$ generated the best initial solution, respectively.

Figures 2-4 illustrate the performance of all heuristics and algorithms with respect to the error, std, and NBS, respectively. It is obvious that, as expected, SA, Tabu, and H-tabu significantly perform better than TKC1, TKC2, Algorithm 1, and Algorithm 2. Observe that logarithmic scales are used in Figures 2 and 3 for clarity. Given that three proposed heuristics (SA, Tabu, and H-tabu) take almost the same CPU time to run (see Table 5), it can be seen from the figures that Tabu and H-tabu perform better than SA, in general. It is also clear that the performance of Tabu and H-tabu gets better as the number of jobs increases, while that of SA deteriorates significantly. Moreover, H-tabu performs consistently better than Tabu.

For all 24 instances (4 values of m and 6 values of n), a test of hypotheses has been conducted at a significance level of 0.05 for comparing the mean errors of Tabu and H-tabu. It has been found that for all the 24 instances, the mean for H-tabu was less than that of tabu. Therefore, H-tabu is the best heuristic.

Table 1. Average percentage deviation from the best solution for all heuristics and algorithms

n	*m*	Algorithm 1	Algorithm 2	H-tabu	SA	Tabu	TKC1	TKC2
20	2	10.375	10.592	0.395	0.152	0.864	19.639	9.143
	4	6.123	9.811	0.447	0.095	0.716	16.634	9.838
	6	5.229	8.917	0.523	0.104	0.867	14.041	9.648
	8	4.186	9.205	0.539	0.095	0.795	13.503	8.868
40	2	15.142	12.410	0.259	0.264	0.304	24.236	9.084
	4	9.597	12.575	0.176	0.294	0.613	20.182	9.734
	6	6.836	12.569	0.171	0.330	0.541	15.687	9.939
	8	6.570	11.561	0.186	0.430	0.572	15.354	10.445
60	2	14.306	11.954	0.091	1.232	0.248	24.702	9.068
	4	10.086	13.073	0.068	1.084	0.273	20.463	9.552
	6	6.701	12.553	0.060	1.177	0.292	17.384	9.694
	8	6.088	11.295	0.077	1.140	0.458	16.147	9.345
80	2	17.046	12.299	0.078	2.135	0.259	25.441	8.555
	4	10.199	13.544	0.089	2.032	0.305	20.093	9.746
	6	7.356	12.898	0.119	2.014	0.351	17.500	8.833
	8	6.763	12.458	0.085	1.934	0.478	15.962	8.791
100	2	18.245	12.133	0.092	2.944	0.214	25.555	7.234
	4	9.152	13.535	0.067	2.735	0.249	19.896	7.540
	6	7.405	12.661	0.037	2.547	0.320	17.618	7.789
	8	6.692	12.916	0.043	2.553	0.375	16.160	8.704
120	2	18.131	12.286	0.058	3.488	0.195	25.501	7.193
	4	11.241	13.893	0.043	3.436	0.362	20.541	7.674
	6	8.666	13.285	0.025	3.101	0.391	17.602	7.997
	8	5.938	12.409	0.037	2.842	0.348	15.941	8.594
Overall Avg.		9.503	12.118	0.157	1.590	0.433	18.991	8.875

Table 2. Standard deviation for all heuristics and algorithms

n	*m*	Algorithm 1	Algorithm 2	H-tabu	SA	Tabu	TKC1	TKC2
20	2	6.130	4.380	0.420	0.280	0.920	6.150	3.160
	4	4.530	4.050	0.500	0.160	0.750	4.140	3.700
	6	4.330	3.520	0.680	0.230	0.780	3.560	4.240
	8	3.610	3.380	0.580	0.160	0.840	2.330	2.530
40	2	7.540	4.330	0.320	0.240	0.360	5.360	3.090
	4	4.980	3.340	0.330	0.270	0.580	2.600	2.170
	6	4.110	3.480	0.380	0.360	0.590	2.720	2.170
	8	4.730	3.440	0.390	0.390	0.510	2.590	3.150
60	2	6.220	3.870	0.150	0.380	0.240	3.580	2.810
	4	4.680	3.480	0.170	0.430	0.310	2.570	2.820
	6	3.290	2.200	0.130	0.390	0.400	2.350	2.820
	8	3.770	2.800	0.210	0.470	0.420	2.340	2.770
80	2	6.070	2.470	0.170	0.490	0.260	3.430	2.520
	4	4.620	2.590	0.180	0.600	0.380	2.730	3.230
	6	3.900	3.050	0.220	0.410	0.300	1.940	1.610
	8	3.740	2.400	0.190	0.370	0.440	1.680	2.240
100	2	5.130	2.560	0.150	0.530	0.260	3.530	2.500
	4	4.850	2.640	0.090	0.410	0.300	1.910	2.380
	6	4.660	2.110	0.110	0.410	0.330	1.810	1.860
	8	3.850	2.630	0.140	0.480	0.390	1.840	2.410
120	2	4.310	1.960	0.130	0.580	0.240	4.030	2.360
	4	4.850	2.480	0.100	0.500	0.340	2.670	2.380
	6	4.700	2.350	0.060	0.520	0.350	1.670	1.970
	8	3.330	1.880	0.150	0.610	0.310	1.290	2.700
Overall Avg.		4.664	2.975	0.248	0.403	0.442	2.868	2.650

Table 3. Error and standard deviation comparison for different uniform distribution ranges

Distribution		TKC1		TKC2		Algorithm 1		Algorithm 2		Tabu		SA		H-tabu	
range	*m*	Error	Std	Error	Std	Error	Std	Error	Std	Error	Std	Error	Std	Error	Std
U(0,100);	2	24.236	5.360	9.084	3.090	15.142	7.540	12.410	4.330	0.304	0.360	0.264	0.240	0.259	0.320
U(1,100)	4	20.182	2.600	9.734	2.170	9.597	4.980	12.575	3.340	0.613	0.580	0.294	0.270	0.176	0.330
	6	15.687	2.720	9.939	2.170	6.836	4.110	12.569	3.480	0.541	0.590	0.330	0.360	0.171	0.380
	8	15.354	2.590	10.445	3.150	6.570	4.730	11.561	3.440	0.572	0.510	0.430	0.390	0.186	0.390
Avg.		18.865	3.318	9.801	2.645	9.536	5.340	12.279	3.648	0.508	0.510	0.330	0.315	0.198	0.355
U(0,100);	2	18.546	5.620	5.199	1.690	2.146	1.060	5.393	1.720	0.133	0.110	0.393	0.120	0.016	0.050
U(1,50)	4	16.876	3.130	7.259	2.300	1.623	0.710	5.363	1.200	0.212	0.200	0.458	0.210	0.019	0.040
	6	14.816	2.380	8.516	2.040	1.731	0.660	5.307	1.840	0.319	0.260	0.448	0.250	0.021	0.060
	8	12.881	2.590	8.397	2.660	1.521	0.630	5.102	1.380	0.325	0.290	0.424	0.270	0.053	0.120
Avg.		15.780	3.430	7.343	2.173	1.755	0.765	5.291	1.535	0.247	0.215	0.431	0.213	0.027	0.068
U(0,100);	2	20.138	6.010	4.326	1.770	1.741	0.730	2.212	0.810	0.063	0.050	0.418	0.130	0.006	0.010
U(1,20)	4	16.602	3.140	6.810	2.590	1.862	0.690	2.528	0.900	0.252	0.190	0.499	0.190	0.000	0.000
	6	15.293	2.200	7.030	2.170	1.639	0.750	2.521	0.870	0.278	0.300	0.454	0.220	0.032	0.080
	8	13.665	2.310	7.929	2.750	1.461	0.580	2.401	0.920	0.352	0.320	0.523	0.210	0.011	0.040
Avg.		16.425	3.415	6.524	2.320	1.676	0.688	2.416	0.875	0.236	0.215	0.474	0.188	0.012	0.033

Table 4. Error and standard deviation comparison for exponential and uniform distributions

Distribution		TKC1		TKC2		Algorithm 1		Algorithm 2		Tabu		SA		H-tabu	
range	*m*	Error	Std	Error	Std	Error	Std	Error	Std	Error	Std	Error	Std	Error	Std
Uniform	2	24.702	3.580	9.068	2.810	14.306	6.220	11.954	3.870	0.248	0.240	1.232	0.380	0.091	0.150
	4	20.463	2.570	9.552	2.820	10.086	4.680	13.073	3.480	0.273	0.310	1.084	0.430	0.068	0.170
	6	17.384	2.350	9.694	2.820	6.701	3.290	12.553	2.200	0.292	0.400	1.177	0.390	0.060	0.130
	8	16.147	2.340	9.345	2.770	6.088	3.770	11.295	2.800	0.458	0.420	1.140	0.470	0.077	0.210
Avg.		19.674	2.710	9.415	2.805	9.295	4.490	12.219	3.088	0.318	0.343	1.158	0.418	0.074	0.165
Exponential	2	33.983	6.450	10.375	2.740	22.261	8.600	12.486	2.840	0.528	0.600	1.285	0.570	0.029	0.100
	4	27.701	4.340	11.356	3.740	13.308	7.790	12.856	3.000	0.447	0.460	1.182	0.540	0.092	0.210
	6	23.043	3.510	10.771	2.670	10.025	7.290	11.988	3.540	0.444	0.430	0.926	0.590	0.150	0.280
	8	20.795	2.660	11.835	3.320	8.682	7.410	11.995	2.930	0.492	0.490	1.109	0.670	0.143	0.330
Avg.		26.381	4.240	11.084	3.118	13.569	7.773	12.331	3.078	0.478	0.495	1.126	0.593	0.104	0.230

Table 5. CPU time (in seconds) for all heuristics and algorithms

n	Algorithm 1	Algorithm 2	H-tabu	SA	Tabu	TKC1	TKC2
20	0.000	0.000	0.471	0.470	0.455	0.000	0.000
40	0.001	0.001	1.702	1.724	1.686	0.001	0.000
60	0.003	0.003	3.720	3.781	3.705	0.002	0.001
80	0.006	0.006	6.527	6.643	6.512	0.003	0.002
100	0.011	0.012	10.119	10.305	10.105	0.005	0.002
120	0.019	0.019	14.498	14.771	14.485	0.007	0.004
Overall Avg.	0.007	0.007	6.173	6.282	6.158	0.003	0.001

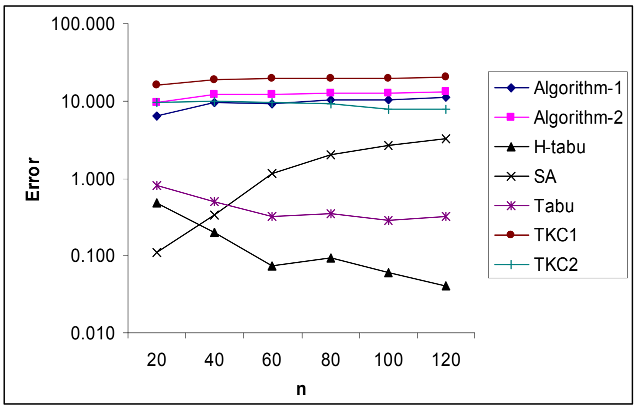

Figure 2. Error comparison for different *n* values.

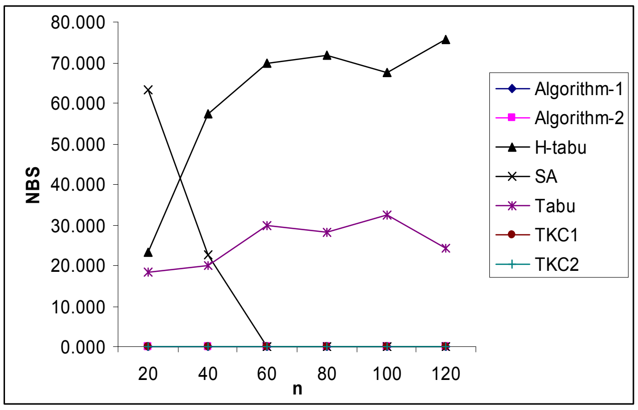

Figure 4. NBS comparison for different *n* values.

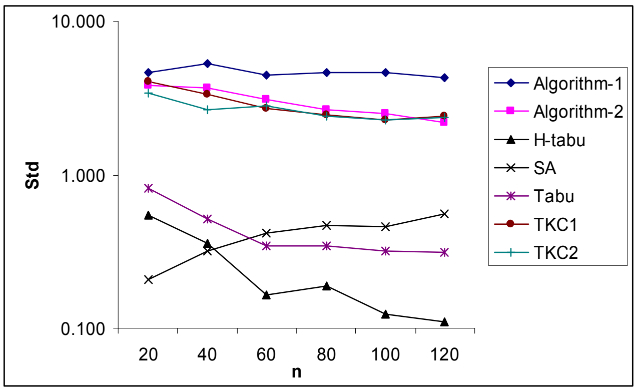

Figure 3. Std comparison for different *n* values.

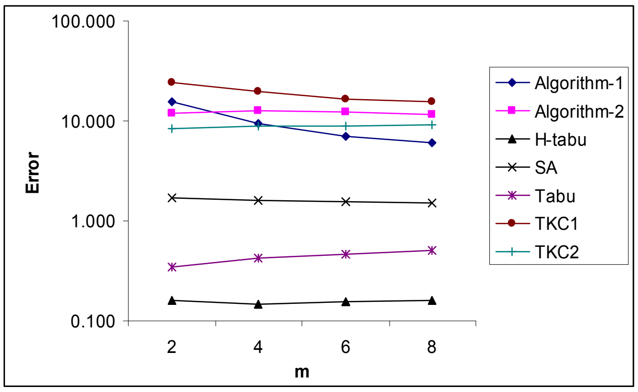

Figure 5. Error comparison for different *m* values.

Figure 5 shows the performance of the heuristics and algorithms for different number of machines at the first stage. It can be seen from the figure that the performance of Algorithm 2, TCK2, and H-tabu is not sensitive to m values. It can also be seen that the performance of SA slightly decreases while that of Tabu slightly increases as m increases. On the other hand, the performance of Algorithm 1 and TKC1 decreases as m increases. This is expected since Algorithm 1 only takes into account the processing times of the jobs on the first stage while TKC1 mainly takes these processing times into account.

The comparison of heuristics and algorithms has also been performed by generating processing times from a skewed distribution, exponential distribution with a mean of 70. Table 4 shows the results when n = 60 for both uniform distribution (U(0, 100) for machines on the first stage, and U(1, 100) for the machine on the second stage) and exponential distribution. As can be seen from the table, the performance of the heuristics and algorithms, in general, is consistent for both distribution. The results for other n values were similar and not reported in the paper due to space limitation.

5. CONCLUSION

The two-stage assembly flowshop scheduling problem is addressed in this paper. The objective is to schedule jobs on machines so that the total completion time of all n jobs is minimized. Optimal solutions are obtained for special cases, and two algorithms are presented. Moreover, three heuristics are proposed for the general case: a simulated annealing heuristic, a tabu search heuristic, and a hybrid tabu search heuristic. The computational analysis shows that the CPU time of all the three proposed heuristics is almost the same. The analysis further shows that the hybrid tabu search heuristic improves the error rate significantly over tabu search and simulated annealing heuristics. Therefore, the hybrid tabu search heuristic could be applied to other scheduling problems.

We assumed that setup times are negligible, and hence, can be considered as part of job processing times. While this assumption reflects certain applications, it adversely affects the solution quality of many applications of scheduling research that require an explicit treatment of setup times, Allahverdi et al. (1999, 2006). Therefore, a possible extension is to consider the problem addressed in this paper where setup times are treated as separate from processing times. Another possible extension is to consider the problem with re-entry of jobs on some of the machines on the first stage.

ACKNOWLEDGEMENTS

This research was supported by Kuwait University Research Administration project number EO 02/05.

REFERENCES

1. Al-Anzi, FS. and Allahverdi, A. (2001). The relation between three-tired client-server internet database and two-machine flowshop. *International Journal of Parallel*

and Distributed Systems and Networks, 4: 94-101.

2. Al-Fawzan, MA. and Haouari, M. (2005). A bi-objective model for robust resource-constrained project scheduling. *International Journal of Production Economics*, 96: 175-187.
3. Al-Turki, U., Fedjki, C., and Andijani, A. (2001). Tabu search for a class of single-machine scheduling problems. *Computers & Operations Research*, 28: 1223-1230.
4. Allahverdi, A. and Al-Anzi, FS. (2002). Using two-machine flowshop with maximum lateness objective to model multimedia data objects scheduling problem for WWW applications. *Computers & Operations Research*, 29: 971-994.
5. Allahverdi, A., Gupta, JND., and Aldowaisan, T. (1999). A review of scheduling research involving setup considerations. *OMEGA The International Journal of Management Sciences*, 27: 219-239.
6. Allahverdi, A., Ng, CT., Cheng, TCE., and Kovalyov, MY. (2006). A survey of scheduling problems with setup times or costs. *European Journal of Operational Research* , (to appear).
7. Ceri, S. and Pelagatti, G. (1984). *Distributed Databases: Principles and Systems*. New York: McGraw-Hi.
8. Elmasri, R. and Navathe, B.(1999). *Fundamentals of Database Systems, 3rd edition.* New York: Addison-Wesley.
9. Haouari, M. and Daouas, T. (1999). Optimal scheduling of the 3-machine assembly-type flow shop. *RAIRO Recherche Operationnelle*, 33: 439-445.
10. Hariri, AMA. and Potts, CN. (1997). A branch and bound algorithm for the two-stage assembly scheduling problem. *European Journal of Operational Research*, 103: 547-556.
11. Koulamas, C. and Kyparisis, GJ. (2001). The three-stage assembly flowshop scheduling problem. *Computers & Operations Research*, 28: 687-704.
12. Lee, CY., Cheng, TCE. and Lin, BMT. (1993). Minimizing the makespan in the 3-machine assembly-type flowshop scheduling problem. *Management Science*, 39: 616-625.
13. Liaw, CF. (2003). An efficient tabu search approach for the two-machine preemptive open shop scheduling problem. *Computers & Operations Research*, 30: 2081-2095.
14. Low, C. (2005). Simulated annealing heuristic for flow shop scheduling problems with unrelated parallel machines. *Computers & Operations Research*, 32: 2013-2025.
15. Mika, M., Waligóra, G., and Weglarz, J. (2005). Simulated annealing and tabu search for multi-mode resource-constrained project scheduling with positive discounted cash flows and different payment models. *European Journal of Operational Research*, 164: 639-668.
16. Pan, CH. and Chen, JS. (1997). Scheduling alternative operations in two-machine flow-shops. *Journal of the Operational Research Society*, 48: 533-540.
17. Potts, CN., Sevast'janov, SV., Strusevich, VA., Van Wassenhove, LN., and Zwaneveld, CM. (1995). The two-stage assembly scheduling problem: complexity and approximation. *Operations Research*, 43: 346-355.
18. Ruiz, R. and Maroto, C. (2005). A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*, 165: 479-494.
19. Sadegheih, A. (2006). Scheduling problem using genetic algorithm, simulated annealing and the effects of parameter values on GA performance. *Applied Mathematical Modelling*, 30: 147-154.
20. Sridhar, J. and Rajendran, C. (1993). Scheduling in a cellular manufacturing system: a simulated annealing approach. *International Journal of Production Research*, 31: 2927-2945.
21. 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.
22. Tozkapan, A., Kirca, O., and Chung, CS. (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.
23. Wang, MY., Sethi, SP., and Van De Velde SL. (1997). Minimizing makespan in a class of reentrant shops. *Operations Research*, 45: 702-7.