7e Conférence Internationale de MOdélisation et SIMulation - MOSIM'08 - du 31 mars au 2 avril 2008 – Paris- France

« Modélisation, Optimisation et Simulation des Systèmes : Communication, Coopération et Coordination.».

MINIMIZING TOTAL COMPLETION TIME IN TWO-MACHINE FLOWSHOP SUBJECT TO RELEASE DATES

M. A. RAKROUKI

Unité de recherche ROI
Ecole Polytechnique de Tunisie
BP 743, 2078 La Marsa, Tunisia
rakroukidali2003@yahoo.fr

T. LADHARI

Unité de recherche ROI
Ecole Polytechnique de Tunisie
BP 743, 2078 La Marsa, Tunisia
talel_ladhari2004@yahoo.fr

ABSTRACT : *We consider the problem of minimizing the sum of completion times in a two-machine permutation flowshop subject to release dates. Despite its theoretical and practical importance, this NP-hard problem has not been investigated before. We present some lower bounds for the problem under consideration. We present a new priority rule and we propose some efficient constructive heuristics. Moreover, a genetic local search algorithm is presented. Computational experiments on a large set of randomly generated instances provide evidence that one of the proposed constructive heuristics as well as the genetic algorithm perform consistently well.*

KEYWORDS : *Flowshop; Constructive Heuristics; Flowtime; Release dates; Genetic Algorithm.*

1. DESCRIPTION AND NOTATION

The problem can be defined as follows. We are given two machines M_1 and M_2 and n jobs (j=1,...,n). Each machine is available at time zero and can process at most one job at a time. Each of n jobs must be processed during p_{1j} time units without preemption firstly on machine M_1, and then during p_{2j} time units on machine M_2. The processing of job j can not be started before a release date r_j. The order by which the jobs are processed on any machine is identical and is not known. The objective is to minimize the sum of job completion time. Using notation of Lawler et al. (1993), this problem is denoted by F2 | r_j | $\sum C_j$.

Let $\pi=(\pi(1),\pi(2),...,\pi(n))$ be a permutation of {1,...,n}. Let $\theta=\{\pi \mid \pi$ is a feasible permutation$\}$ be the family of all the feasible permutations. A permutation π defines a processing order of jobs $(J_{\pi(1)},J_{\pi(2)},...,J_{\pi(n)})$ on each machine. Define C_{kj} as the completion time of job j (j=1,...,n) on M_k (k=1,2). We have $\pi^* \in \theta$ to find such permutation, such that

$$\sum_{j=1}^{n} C_{2j}(\pi^*) = \min_{\pi\in\theta} \sum_{j=1}^{n} C_{2j}(\pi)$$

Let $\pi \in \theta$, be a permutation defining a processing order of jobs on machines M_1 and M_2 then, we have

$$\sum_{j=1}^{n} C_{2j}(\pi) = \sum_{j=1}^{n} \max_{1\le h\le l\le j} \left[r_{\pi(h)} + \sum_{k=h}^{l} p_{1\pi(k)} + \sum_{k=l}^{j} p_{2\pi(k)} \right]$$

Remark 1 The F2|r_j|$\sum C_j$ is NP-hard in the strong sense. This is an immediate consequence of the NP-hardness result of the F2||$\sum C_j$ (Garey and Johnson (1979)). Consequently, the existence of a polynomial algorithm to solve the F2|r_j|$\sum C_j$ is unlikely.

Remark 2 The F2|r_j, $p_{1j}=p_{2j}=p$|$\sum C_j$ can be polynomialy solved by ranking the jobs in nondecreasing order of their release dates.

2. LITERATURE REVIEW

To the best of our knowledge, this is the first paper dealing with the F2|r_j|$\sum C_j$. In this section, we briefly review the related literature to this problem. The theoretical and practical importance of minimizing total completion time in flowshop scheduling has motivated several researchers to investigate this important optimality criterion. A machine based lower bounding scheme for the two-machine case was first proposed by Ignall and Schrage (1965), which was later extended to the *m*-machine case by Bansal (1977). An exact method was proposed by Szwarc (1983). Ahmadi and Bagachi (1990) derived a new machine based lower bound for the m-machine case. The calculations for Ahmadi and Bagachi's bound entail minimizing the total flow time for each of m single-machine problems with preemption and release times, which implies that the time requirement for this bound is O(*mn* log *n*). The computational experiments show that their bound performs Bansal's one. Chung et al. (2002) proposed a branch-and-bound algorithm to solve both the weighted and unweighted version of the problem. Their lower bound is based on the relaxation of the constraint that each machine can process at most on job at a time, for all machines but one. Chung et al.'s algorithm incorporates a new dominance test for pruning nodes in the search tree. This bound is calculated in O(m^2n) time.

The lower bounding procedures developed by Van Velde (1990) and Hoogeveen and Van Velde (1995) are based on a Lagrangian relaxation of the problem. Della Croce et al. (1996) present a detail analysis of a number of alternative lower bounding procedures. A computational study of these procedures in Della Croce et al. (1996) demonstrates that Van Velde's lower bounding procedure and a new lower bounding procedure presented by Della Croce et al. generate good bounds. Della Croce et al. (2002) improve the lower bound of Van Velde by exploring sufficient conditions for the optimality of a solution in the Lagrangian dual problem, and propose dominance criteria that improve the performance the branch-and-bound algorithm. Della Croce et al.'s (2002) branch-and-bound algorithm can solve up to 45 (30) job problems when processing times are uniformly distributed in the [1,10] ([1,100]) range.

Akkan and Karabati (2004) proposed a new branch-and-bound algorithm. The main feature of their algorithm is a new lower bounding scheme that is based on a network formulation of the problem. They formulate an exponential-size network resembling a search tree, in which a schedule corresponds to a path in a network, and then compute a lower bound by applying Lagrangian relaxation. This lower bound is tightened by the use of the dominance criteria already established in the literature. Akkan and Karabati's (2004) algorithm can solve problems with as many as 65 (45) jobs when processing time times are uniformly distributed in the [1,10] ([1,100]) range.

Recently Hoogeveen et al. (2006) formulate the problem by the using of so-called positional completion time variables and investigate the application of Lagrangian relaxation to the problem formulation. They presented two equivalent polynomial-time lower bounds based on the integer linear programming formulations with $O(n^2)$ variables and $O(n)$ constraints. These bounds dominate the previously published bounds.

Due to the NP-hardness of the problem many heuristics are proposed in the literature. In the case of the two machines Della Croce et al. (1996) proposed a heuristic which is divided in two phases. In the first phase, the jobs are first sorted in non-decreasing order of the first machine processing times then a greedy approach based on the preference relations between two jobs. The second phase is an $O(n^2)$ time descent neighborhood search based on pairwise interchanges. The heuristic algorithm has an average error of less than 0.5% from the optimal value and less than 2.7% from the lower bound. Wang et al. (1996) proposed three heuristics. The first heuristic focuses on avoiding idle times or reducing (if unavoidable) idle times on the second machine by choosing the job with the shortest processing time on the first machine. The basic idea of the second heuristic is to avoid or reduce (if unavoidable) waiting times of jobs at the first machine by choosing the job with the shortest processing time on the second machine. The third heuristic is based on the work of Chu (1992). Experimental results show that the two first algorithms are considerably efficient to find near optimal solutions.

Allahverdi and Aldowaisan (2002) and Framinan et al. (2005), present a classification, comparison and evaluation of existing heuristics for the $F|prmu|\sum C_j$. Moreover, a few metaheuristics were developed for the $F|prmu|\sum C_j$ including genetic algorithms (Vempati et al. (1993), Tang and Liu (2002)) and tabu search (Gupta et al. (2000)).

As a particular case of our problem, when there is a single machine, is the $1|r_j|\sum C_j$. This problem is known to be NP-hard in the strong sense (Rinnooy kan 1976). In the particular case with equal release dates the problem is solved in polynomial time by SPT priority rule (Shortest Processing Time, Smith 1956). When the preemption is allowed, the SRPT priority rule (Shortest Remaining Processing Time) provides an optimal solution in $O(n \log n)$ time (Baker 1974). Ahmadi and Bagchi (1990) have proven that the optimal solution of $1|r_j,pmtn|\sum C_j$ given by SRPT is the best available lower bound for the $1|r_j|\sum C_j$. Della Croce and T'kindt (2003) have proposed an improvement on this bound by exploiting the properties of the preemptive solution.

The $1|r_j|\sum C_j$ problem has been extensively studied in the literature. Several branch-and-bound algorithms and/or dominance properties have been designed for the single machine total completion time subject to release dates (see Chandra (1979); Dessouky and Deogum (1981); Deogum (1983); Chu (1991); Chand et al. (1996); Chang and Chen (2006); Jouglet et al. (2008)). Many efforts have been developed to obtain near optimal solutions (see for instances Chu (1992); Liu and MacCarthy (1991); Reeves (1995); Chand et al. (1996) and (1997); Della Croce and T'kindt (2002); Jouglet et al. (2008)). For an exhaustive survey on approximation algorithms for the $1|r_j|\sum C_j$ the reader is referred to Chekuri and Khanna (2004).

3. LOWER BOUNDS

Lower bounds for the $F2|r_j|\sum C_j$ could be computed by solving the subproblem that is derived by relaxing the constraint that each machine can process at most one job at a time (see Ladhari and Haouari 2005). More precisely, if we relax the capacity of one machine, then we obtain a one machine problem, where we have to sequence a set of jobs subject to release dates so as to minimize the total completion time. This problem is denoted $1|r_j|\sum C_j$ and is known to be NP-hard (Rinooy Kan 1976). Three lower bounds could be directly derived from this relaxation.

3.1. LB1

If we relax the capacity of the second machine M2, then we get a relaxed problem which is a single machine

scheduling problem $1|r_j, q_j|\sum C_j$ where $q_j=p_{2j}$ for $j\in J$. A lower bound on the optimal value of this problem is $LB1_1=|J|\min_{j\in J} r_j+\sum_{j\in J} C_{1j}+\sum_{j\in J} p_{2j}$ where $\sum_{j\in J} C_{1j}$ is the total completion time of an optimal solution of $1||\sum C_j$ problem defined on M_1. This later is solved in polynomial time using the so-called Smith's rule: schedule the jobs in order of nondecreasing processing times.

Similarly, if we relax the capacity of the first machine M_1, then we get a relaxed problem which is $1 \mid r_j \mid \sum C_j$ where for each $j\in J$, the release date is r_j+p_{1j}, respectively. A lower bound on the optimal value of this latter problem is $LB1_2=|J|\min_{j\in J}\left(r_j+p_{1j}\right)+\sum_{j\in J} C_{2j}$ where $\sum_{j\in J} C_{2j}$ is the total completion time of an optimal solution of $1||\sum C_j$ problem defined on M_2. Hence, a valid lower bound for $F2|r_j|\sum C_j$ is $LB1=\max\left(LB1_1, LB1_2\right)$

3.2. LB2

Let r denote the value of a fixed release date. We consider a subset of jobs $S_r\subseteq J$, where $S_r=\{j\in J;\ r_j>r\}$. A valid lower bound is $LB1_1\left(S_r\right)=|S_r|\min_{j\in S_r} r_j+\sum_{j\in S_r} C_{1j}+\sum_{j\in S_r} p_{2j}$ as well as $LB1_2\left(S_r\right)=|S_r|\min_{j\in S_r}\left(r_j+p_{1j}\right)+\sum_{j\in S_r} C_{2j}$

Thus, a lower bound is $LB1\left(S_r\right)=\max\left(LB1_1\left(S_r\right), LB1_2\left(S_r\right)\right)$

Taking the maximum over all possible subsets yield $LB2=\max_{S_r\subseteq J}\left(LB1\left(S_r\right)\right)$. This maximum is obtained by successively considering the different possible values of r_j. Thus, LB2 is a valid lower bound for $F2|r_j|\sum C_j$.

3.3. LB3

A second relaxation of $1|r_j|\sum C_j$ is obtained by allowing preemptive schedules. An optimal solution for the preemptive version (denoted $1|r_j, pmtn|\sum C_j$) can be found in polynomial time by using the SRPT rule: at any time schedule an available job with shortest processing time. Let $LB3_i$ (i=1,2) denote the value of the sum of the completion of the optimal preemptive schedule obtained after solving the $1|r_j, pmtn|\sum C_j$ problem on M_i (i=1,2). Before the computation of $LB3_2$, we set the release date of each job j (j=1,...,n) as (r_j+p_{1j}). Then the valid lower bound is $LB3=\max\left(LB3_1, LB3_2\right)$.

LB3 is calculated in O(n log n) time.

4. A PRTCT PRIORITY RULE

In this section, we provide a new priority rule for the total completion time based on the generalization of the sufficient and necessary condition for local optimality described in Chu (1992) for the $1|r_j|\sum C_j$.

Let i and j two jobs that have to be scheduled on two machines M_1 and M_2. M_1 and M_2 are available after times v_1 and v_2 respectively.

Definition A function $PRTCT(i,v_1,v_2)$ of a job i under v_1 and v_2 is defined as $PRTCT(i,v_1,v_2)=2\max(v_2,C_{1i})+p_{i2}$ (Priority Rule for Total Completion Time) with $C_{1i}=\max(v_1,r_i)+p_{1i}$.

5. CONSTRUCTIVE HEURISTICS

In this section we present eight constructive heuristics for the problem under consideration. The proposed heuristics belong to two families. The heuristics of the first family (H_1, H_2, H_3, H_4) are based on the adaptation of the well-known NEH (1983) procedure for the completion time objective function. The heuristics of the second family (H_5, H_6, H_7, H_8) are based on the PRTCT priority rule presented in Section 4.

First, we recall the NEH procedure which has been originally proposed for the $F||C_{max}$.

NEH Heuristic

Step 1: Rank the jobs in decreasing order of the total processing time on all machines.
Step 2: Consider the first two jobs and schedule them in order to minimize the partial makespan as if there were only these two jobs.
Step 3: Select an unscheduled job with the largest total processing time.
Step 4: Insert this job (selected in Step 3) into the scheduled jobs assuming the preceding job order in position which minimizes the makespan.
The third and the fourth step are repeated until all jobs are scheduled.

The adaptation of the according NEH heuristic requires modifying its step1 to the following four alternatives yielding four different variants:

H_1: Rank the jobs in nondecreasing order of r_j.

H_2: Sort the jobs in nondecreasing order of $T_j=(r_j+p_{1j}+p_{2j})$.

H_3: Rank the jobs in nondecreasing order of $Tj=2(r_j+p_{1j})+p_{2j}$. If two jobs have equal T_j then choose the job having $\min(T_j)$.

H_4: Rank the jobs in nondecreasing order of $T_j=2p_{1j}+p_{2j}$. Break ties by choosing the job having the smallest value of T_j.

In the second family we propose four approximate algorithms. These algorithms are based on the assumption that a job with low PRTCT should be given higher priority than a job with high PRTCT. The basic idea for this set of heuristics is building the solution in an iterative way, adding at each iteration the job with higher priority and finding the best partial solution. We denote by σ : a partial sequence of the scheduled jobs, J : the set of unscheduled jobs, and $C_{i\sigma}$: the minimum completion time of σ on the machine i (i=1,2)

H_5

Step 0: $\sigma=\emptyset$; $\bar{J}=J$.
Step 1: Schedule the first job j_1 from $\bar{J}$ having the minimum value of $T_j=2(r_j+p_{1j})+p_{2j}$ and set $\bar{J}=\bar{J}\setminus\{j_1\}$.
Step 2: Compute $S_j=2\max(C_{2\sigma},\max(r_j,C_{1\sigma})+p_{1j})+p_{2j}$ for each job $j\in\bar{J}$.
Step 3: Find the job j^* having the minimum S_{j^*}.
Step 4: Schedule j^* set $\sigma=\sigma j^*$ and $\bar{J}=\bar{J}\setminus\{j^*\}$.
Step 5: If $\bar{J}=\emptyset$ then stop otherwise goto Step 2.

H_6

This heuristic is similar to **H_5** except that in step 1 the job from $\bar{J}$ having the minimum value of $T_j=2p_{1j}+p_{2j}$ is selected.

H_7

Step 0: $\sigma=\emptyset$; $\bar{J}=J$.
Step 1: Let $T_j=2(r_j+p_{1j})+p_{2j}$ and $T=\min_{j\in\bar{J}} T_j$.
Step 2: Construct a subset $S=\{j\in\bar{J} \mid T_j\le T\}$. Rank the jobs in S in nondecreasing order of T_i, S′=S. Set $\bar{J}=\bar{J}\setminus\{S'\}$.
Step 3: Generate from S' a new sequence S″ using the NEH heuristic.
Step 4: Set σ=σS″.
Step 5: If J=∅ then stop otherwise update $\forall j\in\bar{J}$, $T_j=2\max(C_{2\sigma},\max(r_j,C_{1\sigma})+p_{1j})+p_{2j}$ and goto Step 1

H_8

Step 0 : $\sigma=\emptyset$; $\bar{J}=J$.
Step 1 : Select the two first jobs j_1 and j_2 from $\bar{J}$ having the minimum value of $T_j=2(r_j+p_{1j})+p_{2j}$.
Step 2 : Among the partial sequences σ = j_1j_2 and σ = j_2j_1, select the one with the minimum partial total completion time. Set $\bar{J}=\bar{J}\setminus\{j_1,j_2\}$.
Step 3: Calculate $S_j=2\max(C_{2\sigma},\max(r_j,C_{1\sigma})+p_{1j})+p_{2j}$ for each job $j\in\bar{J}$.
Step 4: Find the job j* having the minimum S_{j^*}.
Step 5: Select the job j* and insert it in k+1 possible positions of σ. Among k+1 sequences, select the one with the minimum partial total completion time and set it as the current σ.
Step 6: If J=∅ then stop otherwise goto Step 3.

6. GENETIC LOCAL SEARCH ALGORITHM

In this section we propose genetic local search algorithm as metaheuristic for the problem under consideration.

6.1. Genetic algorithms

6.1.1 Encoding scheme

The most frequently used encoding for optimization problem with permutation property is a simple permutation strings, i.e. strings of integers, with each string corresponding to a feasible solution (permutation). The relative order of the jobs in the permutation indicates the order of the jobs on machines. We adopt this representation in our genetic local search algorithm.

6.1.2 Population initialization

In our algorithm the initial population consists of M permutations (chromosomes). M-1 chromosomes are generated randomly while a single chromosome is generated using H_8 described above. M is referred to as the population size.

6.1.3 Fitness computation

The fitness evaluation function (Fitness_Function (chromosome)) assigns to each chromosome (permutation) in the population a value reflecting their relative superiority (or inferiority). The fitness value assigned to each chromosome is the total completion time of the corresponding permutation.

6.1.4 Crossover operators

Two-point crossover: A pair of crossing points is randomly selected along the length of the first parent chromosome. Two versions of the operator are implemented. In the first version (Figure 1 (a)), the jobs outside the selected two points are copied into the proto child and the remaining jobs are copied from the second parent in the order of their appearance. In figure 1 (a) the two crossing points are on the jobs 4 and 1, respectively. In the second version of the operator (Figure 1 (b)), the sequence of the jobs between the selected crossing points are copied into the child, and the remaining jobs are taken from the second parent in the order of their appearance.

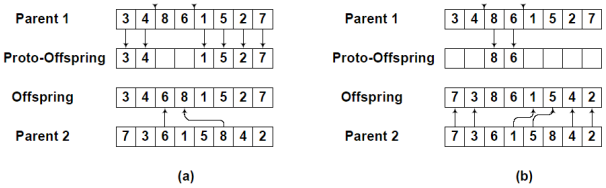

Figure 1. Two-point Crossover

Three-point crossover: Three crossing points are randomly selected along the length of the first parent. The chromosome is then divided into four distinct sections. Copy the first (second) and third (fourth) sub-sections of jobs into the same places of the offspring. Fill up the remaining empty locations of the offspring with jobs from the second parent according to their order of appearance. Again, no duplication of jobs is permitted. Figure 2 displays an example of the operation of these operators with crossing points at the jobs 4, 6, and 5 of the first parent.

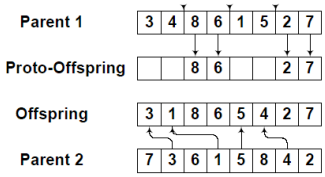

Figure 2. Three-point Crossover

6.1.5 Mutation schemes

In our experiments, we found it useful to set a higher probability (0.7). The exchange and the insertion are adopted as mutations operators. These two operators work as follows.

Random exchange mutation (M_{u1}): Two positions are selected at random along the chromosome and the jobs contained in these positions are exchanged as illustrated in figure 3.

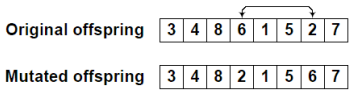

Figure 3. Random exchange mutation

Insertion mutation (M_{u2}): A single job is selected randomly and inserted in a random position. Figure 4 illustrates an example of this operator.

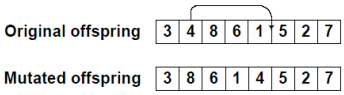

Figure 4. Insertion mutation

6.1.6 Parent selection

In our genetic local search algorithm, the parent selection is a binary tournament selection scheme based on individual fitness. Each execution of this selection scheme provides one individual to play the role of parent.

6.1.7 Population replacement

We have chosen the following strategy for the population replacement. The improved offspring that is better than any current individual in the population is replaced by the worst one.

6.1.8 Control parameters

After the experimental study of the behavior of the different parameters of the proposed algorithm, we found that solutions of higher quality are achieved using the following settings and control schemes:

Population size: M=200.
Crossover probability (Pc=0.9)
- The Two-Point Crossover (form 1) is used with a probability (P_{c1}=0.3)
- The Two-Point Crossover (form 2) is used with probability (P_{c2}=0.3)
- The Three-Point Crossover is used with probability (P_{c3}=0.4)

Mutation probability (P_{Mu}=0.7)
- The Exchange Mutation is used with probability (P_{Mu1}=0.5)
- The Insertion Mutation is used with probability (P_{Mu2}=0.5)

6.1.9 Termination condition

In our implementation of GA, the termination criterion depends on the size of the problem. The GA stops either when a maximum number of 100n generation has been surpassed or when the best solution of the population has not been improved on over 10n consecutive generations.

6.2 Genetic local search

In order to derive high quality solutions, we propose a hybridization of our genetic algorithm with a local search procedure, thus yielding a genetic local search algorithm. The main objective of the local search is to enhance the intensification process in the genetic search. In local search methods, the definition of the neighborhood is by far the most important ingredient. So far, we have used two different neighborhoods. These neighborhoods are the following:

Exchanging(π,i,j) : Given a permutation π, a neighbor π' is obtained by interchanging the jobs in positions i and j. The positions i and j are selected randomly.

Swapping(π,i,j) : Given a permutation π, a neighbor π' is obtained by interchanging the jobs in positions i and j. The positions i and j are enumerated in some systematic ways such as adjacent pairwise interchange.

These neighborhoods present the advantage of being of polynomial sizes, and easy to enumerate.

To integrate the local search phase in the GA, several alternatives are possible. Murata et al. (1996) propose an improvement step by applying a local search procedure before selection and crossover in a GA. In our implementation, we apply local search phase after using the mutation operators and this for each individual in the population. The used local search scheme is based on two different procedures. The first procedure (Find_best_neighbor) generates k neighbors for each chromosome in the population. The second one (Improvement_procedure) is an improvement procedure.

The proposed local search scheme is described as follows.

Local_Search_Procedure(k)
Begin
Generate a probability P_{LS}
If (P_{LS}<x)→(x=0.5) **then**
Call Find_best_neighbor(k, permutation)
Else
Call Improvement_procedure(k, permutation)
End if
End

Find_best_neighbor(k, π)
Begin
Step 1 Generate k (k=30) neighbors of a given solution π.(k/2) of the neighbors are generated by Exchanging(π,i,j). (k/2) of the neighbors are generated by Swapping(π,i,j).
Step 2 Insert the best neighbor in the population.
End

Improvement_procedure(k, π)
Begin
Step 1
For 1 **to** 100 **do**
π′= Exchange(π,i,j).
If Fitness_Function(π′) < Fitness_Function(π) **then**
replace π by π′(π←π′)
End if
End For
Step 2 Insert π in the population.
End

7. COMPUTATIONAL RESULTS

This section describes the computational tests which have been used to evaluate the empirical performance of the proposed lower bounds and heuristics. All the proposed procedures have been coded in C and compiled with the Microsoft Visual C++ (version 6.0). All the computational experiments were carried out on a Pentium IV 3.0 GHz PC with 1GB RAM.

The test problems are generated as follows. For each size n∈{20,30,50,100,200,300,400,500,700,1000} 30 instances were generated. The processing times are drawn from a discrete uniform distribution on [1,100]. The release dates are uniformly distributed between [1,100R]. In this way we obtain four subsets A1, A2, A3 and A4 respectively where R={1,2,n,2n}.

7.1. Performance of the proposed algorithms

The performance analysis is based on two measures : average relative gap and average relative percentage deviation (ARPD) from the best-known solution. The relative gap provides the gap between the solution provided by H_i (i=1,...,8) and the best lower bound. (i.e. $100*(((UB_i-LB)/(LB)))$ where UB_i denote the solution value of H_i and LB is a lower bound on the optimal solution). The percentage deviation is defined as $100*(((UB_i-UB^*)/(UB^*)))$, where $UB^*=\min(UB_i)$ (i=1,...,8). The results of the computational study of the proposed constructive heuristics are summarized in Table 1. The headings have the following meanings: n: number of jobs, Gap: average relative gap, ARPD : average relative percentage deviation from the best-known solution and Time: mean CPU time (in s).

Table 1 provides strong evidence that the heuristic based on the PRTCT priority rule (H_8) yields consistently near-optimal solutions and far much better than all the proposed heuristics in term of quality of the solution. For all problem sizes less than n=500 the mean CPU time required by H_8 is negligible (<1s). H8 is able to solve large problems in a few seconds. For instance, the results reveal that all the instances with 1000 jobs were solved within a mean CPU time nearly equal to 7s.

The GLS algorithm is tested for n∈{20,30,50,100,200,300,400,500}. Table 2 shows that the GLS consistently performs all the constructive heuristics, but requires longer CPU times. Indeed for all test problems, GLS provided the best average relative gap and average relative percentage deviation from the best-known solution over all proposed methods. We observe from Table 2, that our GLS algorithm is able to solve very large instances with up to 500 jobs within a relatively moderate CPU time. For instance, we found that all the 120 500-job instances were solved within a mean CPU time nearly equal to 12 mn. The GLS improves the performance of the H_8 heuristic and provides superior results compared to the all proposed constructive algorithms

Subset	*n*	H_1			H_2			H_3			H_4		
		Gap	*ARPD*	*Time*	*Gap*	*ARPD*	*Time*	*Gap*	*ARPD*	*Time*	*Gap*	*ARPD*	*Time*
A1	*20*	9.23	1.56	0.00	8.56	1.02	0.00	8.96	1.32	0.00	8.47	0.85	0.00
	30	11.51	2.43	0.00	9.85	0.99	0.00	10.23	1.28	0.00	9.57	0.70	0.00
	50	13.55	3.03	0.00	11.31	1.05	0.00	12.19	1.80	0.00	10.69	0.46	0.00
	100	15.88	4.19	0.01	13.29	1.87	0.01	14.27	2.74	0.01	11.89	0.61	0.01
	200	18.12	5.48	0.06	14.34	2.11	0.07	15.89	3.49	0.06	12.70	0.65	0.06
	300	19.92	6.10	0.19	15.74	2.40	0.25	17.57	4.02	0.19	14.05	0.91	0.19
	400	19.84	6.12	0.44	15.74	2.50	0.74	17.50	4.05	0.44	14.00	0.95	0.44
	500	20.07	6.32	0.90	15.91	2.64	1.43	17.64	4.17	0.88	14.07	1.01	0.89
	700	20.54	6.58	2.55	16.18	2.73	3.33	17.99	4.32	2.36	14.23	1.01	2.58
	1000	21.50	6.89	7.21	16.92	2.87	10.28	18.76	4.48	6.86	14.94	1.12	6.91
A2	*20*	11.53	2.27	0.00	11.17	1.96	0.00	11.47	2.22	0.00	10.27	1.14	0.00
	30	14.05	3.24	0.00	13.01	2.31	0.00	14.07	3.26	0.00	11.29	0.76	0.00
	50	15.09	3.21	0.00	14.12	2.35	0.00	14.93	3.06	0.00	12.01	0.46	0.00
	100	15.51	4.21	0.01	13.77	2.64	0.01	14.69	3.47	0.01	11.28	0.40	0.01
	200	18.88	5.49	0.06	16.51	3.39	0.06	17.81	4.54	0.06	13.58	0.79	0.06
	300	19.94	6.01	0.25	17.39	3.75	0.24	18.66	4.87	0.28	14.14	0.89	0.21
	400	20.58	6.30	0.61	18.08	4.10	0.59	19.43	5.29	0.74	14.55	0.99	0.48
	500	20.78	6.42	1.22	18.11	4.07	1.20	19.53	5.32	1.55	14.64	1.01	0.93
	700	21.14	6.58	3.59	18.53	4.29	3.70	19.92	5.51	4.07	14.85	1.05	2.35
	1000	21.37	6.73	10.73	18.66	4.34	11.54	20.10	5.61	11.66	14.97	1.10	6.81
A3	*20*	1.16	0.07	0.00	1.19	0.10	0.00	1.21	0.11	0.00	1.20	0.10	0.00
	30	0.89	0.08	0.00	0.89	0.08	0.00	0.87	0.06	0.00	0.87	0.06	0.00
	50	0.67	0.08	0.00	0.67	0.09	0.00	0.67	0.08	0.00	0.64	0.05	0.00
	100	0.37	0.03	0.01	0.37	0.03	0.01	0.37	0.03	0.01	0.37	0.02	0.01
	200	0.26	0.02	0.06	0.26	0.03	0.07	0.26	0.02	0.06	0.25	0.01	0.06
	300	0.16	0.01	0.27	0.16	0.02	0.28	0.16	0.02	0.30	0.16	0.01	0.19
	400	0.18	0.01	0.71	0.19	0.02	0.75	0.19	0.02	0.75	0.18	0.01	0.45
	500	0.38	0.04	1.36	0.39	0.05	1.41	0.40	0.06	1.44	0.35	0.01	0.92
	700	3.05	0.92	3.72	2.90	0.77	3.60	3.03	0.90	3.78	2.17	0.06	2.41
	1000	8.73	4.05	10.36	8.65	3.97	10.57	8.76	4.08	11.33	5.18	0.65	7.30
A4	*20*	0.24	0.01	0.00	0.25	0.01	0.00	0.25	0.01	0.00	0.24	0.01	0.00
	30	0.18	0.00	0.00	0.18	0.00	0.00	0.18	0.00	0.00	0.20	0.02	0.00
	50	0.13	0.01	0.00	0.14	0.01	0.00	0.13	0.01	0.00	0.14	0.01	0.00
	100	0.08	0.00	0.01	0.08	0.00	0.01	0.08	0.00	0.01	0.08	0.00	0.01
	200	0.05	0.00	0.06	0.05	0.00	0.08	0.05	0.00	0.06	0.05	0.00	0.06
	300	0.10	0.00	0.29	0.10	0.01	0.27	0.10	0.00	0.33	0.09	0.00	0.19
	400	0.18	0.01	0.75	0.19	0.02	0.64	0.19	0.02	0.77	0.18	0.01	0.44
	500	0.38	0.04	1.44	0.39	0.05	1.26	0.40	0.06	1.45	0.35	0.01	0.89
	700	3.05	0.92	3.59	2.90	0.77	3.61	3.03	0.90	3.89	2.17	0.06	2.58
	1000	8.73	4.05	10.48	8.65	3.97	10.46	8.76	4.08	11.78	5.18	0.65	6.91

Table 1. Computational performance of the constructive heuristics

Subset	*n*	H_5			H_6			H_7			H_8		
		Gap	*ARPD*	*Time*	*Gap*	*ARPD*	*Time*	*Gap*	*ARPD*	*Time*	*Gap*	*ARPD*	*Time*
A1	*20*	11.04	3.26	0.00	17.00	8.82	0.00	10.98	3.20	0.00	8.51	0.91	0.00
	30	12.73	3.59	0.00	17.17	7.67	0.00	12.66	3.52	0.00	9.53	0.66	0.00
	50	13.51	3.03	0.00	16.31	5.56	0.00	13.66	3.17	0.00	10.40	0.20	0.00
	100	14.41	2.89	0.01	16.36	4.64	0.01	14.42	2.89	0.02	11.27	0.07	0.01
	200	14.84	2.56	0.07	15.62	3.25	0.07	14.80	2.53	0.16	11.97	0.00	0.06
	300	15.69	2.36	0.25	16.34	2.93	0.25	15.71	2.38	0.49	13.02	0.00	0.19
	400	15.48	2.27	0.56	15.90	2.64	0.56	15.47	2.26	1.16	12.92	0.00	0.44
	500	15.28	2.09	1.15	15.68	2.44	1.16	15.31	2.11	2.66	12.92	0.00	0.90
	700	15.19	1.85	3.15	15.45	2.09	3.15	15.21	1.87	7.10	13.09	0.00	2.55
	1000	15.49	1.61	9.14	15.69	1.78	9.49	15.51	1.62	20.86	13.66	0.00	7.59
A2	*20*	10.46	1.30	0.00	15.57	6.03	0.00	10.20	1.06	0.00	9.81	0.72	0.00
	30	13.04	2.35	0.00	16.48	5.41	0.00	13.27	2.55	0.00	10.76	0.29	0.00
	50	13.72	1.99	0.00	18.25	6.06	0.00	13.93	2.18	0.00	11.69	0.18	0.00
	100	13.26	2.19	0.01	15.76	4.45	0.01	13.20	2.14	0.02	10.90	0.06	0.01
	200	14.77	1.85	0.07	16.65	3.52	0.07	14.91	1.98	0.15	12.69	0.01	0.06
	300	15.88	2.43	0.53	17.30	3.67	0.54	15.17	1.80	0.74	13.13	0.00	0.19
	400	15.81	2.10	1.31	16.88	3.04	1.29	15.22	1.58	1.81	13.42	0.00	0.44
	500	15.60	1.86	2.75	16.41	2.58	2.72	15.11	1.43	4.05	13.49	0.00	0.88
	700	15.68	1.78	8.34	16.46	2.47	8.04	15.25	1.40	11.51	13.66	0.00	2.34
	1000	15.48	1.55	25.16	15.97	1.98	24.39	15.16	1.27	26.10	13.72	0.00	7.18
A3	*20*	1.41	0.31	0.00	14.61	13.37	0.00	1.41	0.32	0.00	1.23	0.13	0.00
	30	1.14	0.33	0.00	20.25	19.27	0.00	1.14	0.33	0.00	0.86	0.05	0.00
	50	0.88	0.29	0.00	41.43	40.61	0.00	0.88	0.29	0.00	0.62	0.04	0.00
	100	0.56	0.22	0.01	39.07	38.60	0.01	0.56	0.22	0.02	0.37	0.02	0.01
	200	0.50	0.26	0.07	52.27	51.91	0.07	0.51	0.27	0.15	0.25	0.01	0.06
	300	0.29	0.14	0.55	58.78	58.55	0.58	0.29	0.14	0.76	0.15	0.01	0.19
	400	0.34	0.17	1.33	52.28	52.02	1.37	0.35	0.18	1.87	0.18	0.01	0.45
	500	1.47	1.13	3.06	62.55	62.00	2.80	1.43	1.09	3.79	0.36	0.02	0.89
	700	10.65	8.36	9.08	81.27	77.57	8.33	10.10	7.82	11.18	2.22	0.10	2.40
	1000	13.20	8.33	27.65	80.12	72.45	25.68	12.07	7.25	25.53	4.61	0.11	7.26
A4	*20*	0.27	0.04	0.00	9.28	9.02	0.00	0.27	0.04	0.00	0.24	0.01	0.00
	30	0.22	0.04	0.00	23.92	23.69	0.00	0.22	0.04	0.00	0.18	0.01	0.00
	50	0.18	0.05	0.00	28.77	28.60	0.00	0.18	0.05	0.00	0.13	0.00	0.00
	100	0.11	0.03	0.01	41.05	40.95	0.01	0.11	0.03	0.02	0.08	0.00	0.01
	200	0.06	0.01	0.07	51.35	51.27	0.07	0.06	0.01	0.15	0.05	0.00	0.06
	300	0.14	0.04	0.57	50.91	50.77	0.54	0.13	0.04	0.77	0.09	0.00	0.19
	400	0.34	0.17	1.41	52.28	52.02	1.32	0.35	0.18	1.96	0.18	0.01	0.44
	500	1.47	1.13	2.90	62.55	62.00	2.81	1.43	1.09	3.86	0.36	0.02	0.90
	700	10.65	8.36	8.75	81.27	77.57	8.34	10.10	7.82	11.00	2.22	0.10	2.39
	1000	13.20	8.33	27.45	80.12	72.45	25.87	12.03	7.21	25.90	4.61	0.11	7.30

Table 1. Computational performance of the constructive heuristics

Subset	*n*	***GLS***			
		Gap	*ARPD*	*gapH_8*	*Time*
A1	*20*	6.4892	0.0000	1.8987	0.11
	30	7.5778	0.0000	1.7986	0.23
	50	8.5625	0.0000	1.6819	0.66
	100	10.2359	0.0047	1.3116	4.33
	200	11.3394	0.0000	0.7670	36.00
	300	12.5770	0.0000	0.5081	151.69
	400	12.5927	0.0031	0.3493	461.13
	500	12.6521	0.0020	0.2974	1065.35
A2	*20*	8.1138	0.0000	1.5676	0.11
	30	9.0516	0.0000	1.5561	0.22
	50	10.0210	0.0000	1.5130	0.61
	100	9.8127	0.0000	1.3631	3.50
	200	12.0910	0.0000	0.6775	37.06
	300	12.6728	0.0047	0.5108	152.77
	400	13.1114	0.0017	0.3577	410.88
	500	13.2079	0.0032	0.3096	1060.49
A3	*20*	1.0328	0.0000	0.1906	0.11
	30	0.7721	0.0000	0.0863	0.20
	50	0.5299	0.0000	0.0941	0.54
	100	0.3092	0.0000	0.0557	2.86
	200	0.2028	0.0002	0.0438	24.40
	300	0.1247	0.0003	0.0283	93.57
	400	0.1437	0.0010	0.0325	291.81
	500	0.2489	0.0010	0.1079	746.31
A4	*20*	0.2299	0.0000	0.0116	0.10
	30	0.1773	0.0000	0.0068	0.20
	50	0.1249	0.0000	0.0074	0.52
	100	0.0746	0.0000	0.0054	2.79
	200	0.0465	0.0001	0.0028	20.10
	300	0.0830	0.0007	0.0111	87.79
	400	0.1301	0.0009	0.0325	313.69
	500	0.2252	0.0009	0.0963	804.08

Table 2. Computational performance of the GLS

Moreover we run our algorithms (GLS and H_8) on additional set of small sized instances (n=10). For these problems (120 test problems), the two algorithms are compared with the optimal solution. An important observation is that the GLS gives the optimal solution in negligible CPU time (≃0s) for all generated instances (120 instances) and 62 instances out of 120 were optimally solved by H_8. For the instances that remained unsolved by H_8 the maximal, minimal, and average gap are reported in Table 3.

Subset	US	*Gap*		
		Max	*Avg.*	*Min*
A1	26	6.265	1.367	0.037
A2	22	7.586	1.707	0.034
A3	9	0.940	0.263	0.034
A1	1	0.322	0.322	0.322

Table 3. Performance of H_8 on small instances (n=10)

8. CONCLUSION

This paper investigates the two-machine permutation flowshop with the objective of minimizing the total completion time subject to release dates. Despite its theoretical and practical importance, this NP-hard problem has not been investigated before. We proposed a new priority rule, lower bounds, constructive heuristics, as well as a genetic local search algorithm. Our computational experiments that were carried out on a large set of randomly instances provide strong evidence that a constructive heuristic based on the proposed priority rule as well as the GLS algorithm consistently yield near-optimal solutions.

An interesting issue which deserves further investigation is to develop an exact method for the problem under consideration.

REFERENCES

Ahmadi, R.H. and U. Bagchi, 1990. Lower bounds for single-machine scheduling problems, *Naval Research Logistics*, 37, p. 967-979.

Akkan, C. and S. Karabati, 2004. The two-machine flowshop total completion time problem: Improved lower bounds and a branch-and-bound algorithm. *European Journal of Operational Research*, 159, p. 420-429.

Allahverdi, A. and T. Aldowaisan, 2002. New heuristics to minimize total completion time in m-machine flowshops, *International Journal of Production Economics*, 7, p. 71-83.

Baker, K.R., 1974. Introduction to Sequencing and Scheduling, Wiley Publishing, New York.

Bansal, S.P., 1977. Minimizing the sum of completion times of n jobs over m machines in a flowshop - a branch and bound approach, *American Institute of Industrial Engineers Transaction*, 9, p. 306-317.

Chand, S., Traub, R. and R. Uzsoy, 1996. Single-Machine Scheduling with Dynamic Arrivals: Decomposition Results and an Improved Algorithm, *Naval Research Logistics*, 43, p. 709-716.

Chand, S., Traub, R. and R. Uzsoy, 1997. Rolling horizon procedures for the single machine deterministic total completion time scheduling problem with release dates, *Annals of Operations Research*, 70, p. 115-125.

Chandra, R., 1979. On n/1/F dynamic determistic Systems, *Naval Research Logistics*, 26, p. 537-544.

Chang, P.C. and S.H. Chen, 2006. A Decomposition Method for Single-Machine Scheduling With Dynamic Arrivals, *International Journal of Computer Science and Network Security*, 6(3A), p. 115-121.

Chekuri, C. and S. Khanna, 2004. Approximation Algorithms for Minimizing Average Weighted Completion Time, *Chapter in Handbook of*

Scheduling: Algorithms, Models, and Performance Analysis, edited by Joseph Leung, CRC Press.

Chu, C., 1991. A branch and bound algorithm to minimize total flow time with unequal release dates, *Naval Research Logistics*, 39, p. 859-875.

Chu, C., 1992. Efficient heuristics to minimize total flow time with release dates, *Operations Research Letters*, 12, p. 321-330.

Chung, C.S., J. Flynn, and O. Kirca, 2002. A branch and bound algorithm to minimize the total flow time for m-machine permutation flowshop problems, *International Journal of Production Economics*, 79, p. 185-196.

Della Croce, F., V. Narayan, and R. Tadei, 1996. The two-machine total completion time flowshop problem, *European Journal of Operational Research*, 90, p. 227-237.

Della Croce, F., M. Ghirardi, and R. Tadei, 2002. An improved branch-and-bound algorithm for the two machine total completion time flowshop problem, *European Journal of Operational Research*, 139, p. 293-301.

Della Croce, F. and V. T'Kindt, 2002. A recovering beam search algorithm for the one-machine dynamic total completion time scheduling problem, *Journal of Operational Research Society*, 53, p. 1275-1280.

Della Croce, F. and V. T'Kindt, 2003. Improving the preemtive bound for the one-machine dynamic total completion time scheduling problem, *Operations Research Letters*, 31, p. 142-148.

Deogum, D.S. 1983. On scheduling with ready times to minimize mean flowtime, *Computer Journal*, 26, p. 320-328.

Dessouky, M.I. and D.S. Deogum, 1981. Sequencing jobs with unequal ready times to minimize mean flowtime, *SIAM Journal on Computing*, 10, p. 192-202.

Framinan, J.M., R. Leisten, and R. Ruiz-Usano, 2005. Comparaison of heuristics for flowtime minimisation in permutation flowshops, *Computers and Operations Research*, 32, p. 1237-1254.

Garey, M.R. and D.S. Johnson, 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, CA.

Gupta, J.N.D., C.L. Chen, L.Y., Yap, and H. Deshmukh, 2000. Designing a tabu search algorithm to minimize total flow time in a flow shop, *The Arabian Journal for Science and Engineering*, 25, p. 79-93.

Hoogeveen, J.A. and S. Van Velde, 1995. Minimizing total completion and maximum cost simultaneously is solvable in polynomial time, *Operations Research Letters*, 17, p. 205-208.

Hoogeveen, H., L. van Norden, and S. Van Velde, 2006. Lower bounds for minmizing total completion time in a two-machine flow shop, *Journal of Scheduling*, 9, p. 559-568.

Ignall, E. and L. Schrage, 1965. Application of the branch-and-bound technique to some flowshop scheduling problems, *Operations Research*, 13, p. 400-412.

Jouglet, A., D. Savourey, J. Carlier, and P. Baptiste, 2008. Dominance-based heuristics for one-machine total cost scheduling problems, *European Journal of Operational Research*, 184, p. 879-899.

Ladhari, T. and M. Haouari, 2005. A computational study of the permutation flow shop problem based on a tight lower bound, *Computers and Operations Research*, 32 p. 1831-1847.

Lawler, E.L., J.K. Lenstra, A.H.G Rinnooy Kan, and D.B. Shmoys, 1993. *Sequencing and scheduling: algorithms and complexity*, in: Graves, C.G., Rinnooy Kan A.H.G. and Zipkin, P., eds., *Handbooks in Operations Research and Management Sciences*, Logistics of Production and Inventory, North-Holland, Amsterdam, 4, p. 445-522.

Liu, J. and B.L. MacCarthy, 1991. Effective heuristics for the single machine sequencing problem with ready times, *International Journal of Production Research*, 29, p. 1521-1533.

Murata, T., H. Ishibuchi, and H. Tanaka, 1996. Genetic algorithms for flowshop scheduling problems, *Computers and Industrial Engineering*, 30(4), p. 1061-1071.

Nawaz, M, E.E. Enscore, and I. Ham, 1983. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem, *OMEGA*, 11, p. 91-96.

Reeves, C., 1995. Heuristics for scheduling a single machine subject to unequal job release times, *European Journal of Operational Research*, 80, p. 397-403.

Rinnooy Kan, A.H.G., 1976. *Machine sequencing problem: classification, complexity and computation*, Nijhoff, The Hague.

Smith, W.E., 1956. Various Optimizers for Single Stage Production, *Naval Research Logistics Quarterly*, 3, p. 59-66.

Szwarc, W., 1983. The flow shop problem with mean completion time criterion, *IIE Transaction*, 15, p. 172-176.

Tang, L. and J. Liu, 2002. A modified genetic algorithm for the flow shop sequencing problem to minimize mean flow time, *Journal of Intelligent Manufacturing*, 13, p. 61-67.

Van de Velde, S., 1990. Minimizing the sum of job completion times in the two-machine flowshop by Lagrangean relaxation, *Annals Of Operations Research*, 26, p. 257-268.

Vempati, V.S., C.L. Chen, and S.F. Bullington, 1993. An effective heuristic for flow-shop problems with total flow time as criterion, *Computers and Industrial Engineering*, 25, p. 219-240.

Wang, C.G., C.B. Chu, and J.M. Proth, 1996. Efficient heuristic and optimal approaches for $n|2|F|\sum C_i$ scheduling problems, *International Journal of Production Economics*, 44, p. 225-237.