A NEW HYBRID HEURISTIC ALGORITHM FOR SOLVING FLOWSHOP SCHEDULING PROBLEM

Mircea ANCĂU[1] and Liberato CAMILLERI[2]

ABSTRACT: This paper presents a new heuristic hybrid algorithm for solving flow shop scheduling problem. The algorithm is based on a general optimization method which works in two stages. In the first stage it performs a global search of the optimal solution by a Monte Carlo based approach, while in the second stage of the local search, the previous solution is improved. To exhibit the effectiveness of the proposed method, several computational tests are carried out complemented by statistical analysis.
KEY WORDS: heuristic, optimization, flow shop scheduling.

1 INTRODUCTION

According to group technology concept, pieces in industry may be classified depending on different criteria (geometry, functional role, size etc.) in families of pieces. A main feature of pieces, which belong to the same family, is the fact that it follows the same operating sequence on every machine. Due to geometry variation, different pieces from the same family may have different processing times on the same machine, but the operating sequence on all machines is the same. Consider a set of N_p pieces (or jobs) to be processed on M different machines. The main objective is to find the sequence in which the jobs must be processed so that the total completion time (the time between the beginning of the execution of the first job on the first machine and the completion of the execution of the last job on the last machine) or makespan, denoted C_{max}, to be minimum. The processing times of jobs on all machines, denoted $t_{ij} \in R_+$ (i = 1,2, ..., N_p, j = 1,2, ..., M), are nonnegative integer values that are known in advance. For this problem, the following assumptions should be made:

- all jobs are independent and available for processing when the process is initialized;
- every job has to be processed at most once on machine 1,2,...,M, in this order;
- all machines are permanently available;
- every job is processed at most on one machine at a time;
- every machine processes only one job at a time;
- once started an operation cannot be interrupted;
- the set-up times of the operations are included in the processing times and do not depend on the sequence;
- if the machine which follow in the operation sequence is not available (being busy with another job) next jobs will be allocated in a waiting list.

Many heuristic algorithms have been proposed to solve the flow shop scheduling problems.

They can be classified in two main groups: constructive heuristics and improvement heuristics. The first group builds a feasible schedule using a specific technique, while the second group tries to improve the previously generated schedule. The classic algorithm of Johnson (Johnson,1954) offers the optimal solution for the case of N_p jobs on two machines. These scheduling problems can be solved in polynomial time. For more than two machines, M $\geq$ 3, these problems become NP-Complete. The CDS heuristic algorithm of Campbell et al. (Campbell et al,1970) divides all M machines into two groups which are considered as two virtual machines. Therefore, the problem is solved by applying Johnson's algorithm. Subsequently, the best solution is chosen among (M-1) processing sequences. The HFC heuristic of Koulamas (Kuolamas,1988) uses the algorithm developed by Johnson in the first phase and then attempts at improving the feasible solution.

Many other heuristic methods assign a weight to each job, sorting the list of jobs using that weight as a sort key (Palmer,1965), (Gupta,1971), (Hundal & Rajgopal,1988).

The RA (Rapid Access) algorithm made by Dannenbring (Dannenbring,1977), is a combination between previous methods based on Johnson and Palmer algorithms.

[1]Technical University of Cluj-Napoca, Faculty of Machine Building, Dept. of Manufacturing Engineering B-dul Muncii 103 – 105, 400641 Cluj-Napoca, Romania

[2]University of Malta, Faculty of Science, Department of Statistics and Operations Research, Msida MSD06, Malta

E-mail: mircea.ancau@tcm.utcluj.ro; liberato.camilleri@um.edu.mt

According to Taillard (Taillard,1990), the NEH heuristic algorithm of Nawaz et al, (Nawaz et al,1983) is recognized as one of the efficient heuristic method in this field. NEH is neither based on Johnson's algorithm nor on techniques that assign weights. Initially, the algorithm calculates the total completion time for each job taken solely, on all machines. The jobs are then sorted in descending order depending on the size of these durations. For two jobs, there are two possible variants to be placed in the manufacturing sequence.

The permutation that yields the minimum completion time is selected. Similarly, for three jobs there are three possible variants to be placed in the manufacturing sequence. As in the preceding step, the variant with minimum completion time is selected. This iterative process is continued until all jobs are placed in the manufacturing sequence. In this way, the manufacturing sequence is generated, by placing each job J_k ($2 < k \leq N$) in the most favourable position in the sequence J_1, J_2, ..., J_{k-1} already formed. Consequently, there are basically $n \cdot (n-1)/2 - 1$ evaluation sequences to get the final result. Many other heuristic methods, based on NEH, propose different starting sequences (Framinan et al,2003). SPIRIT algorithm of Widmer and Hertz, (Widmer & Hertz,1989) is a constructive heuristic of the insertion type. The first two jobs J_i, J_k from the entire list of jobs are chosen such that the total completion time is minimum. The following jobs are inserted in the manufacturing sequence randomly based on the minimum increasing of the total completion time.

Unlike constructive heuristic algorithms, the improvement heuristic algorithms start from a specified manufacturing sequence and try to improve it by applying different procedures.

A practical method to improve a manufacturing sequence is to exchange the places of two neighbour jobs and to replicate this step until an improved manufacturing sequence is obtained.

Considering the time between the manufacturing start of a job on the M_k machine and it's manufacturing finish on M_{k+1} machine as a criterion, the heuristic algorithm of Ho şi Chang (Ho & Chang,1991) exchanges job positions in the manufacturing sequence, in order to decrease these intermediary times.

To solve the flow shop scheduling problems, other heuristic methods apply principles used to solve some other combinatorial optimization problems. The most popular and widely used heuristic methods are either based on Simulated Annealing (Osman & Potts,1989), (Widmer & Hertz,1989), (Taillard,1990), (Ogbu & Smith,1990), (Ishibuchi,1995), Tabu Search (Moccellin,1995), (Nowicki & Smutnicki,1996) , or genetic algorithms (Murata et al,1996, (Reeves & Yamada,1998), (Ponnambalam,2001) etc. Several comprehensive studies including those of Ruiz and Morato (Ruiz & Morato,2005) and Garrido et al, (Garrido et al,2000) discuss and compare the performances of these methods.

2 THE GENERAL PRINCIPLE OF THE NUMERIC ALGORITHM

Flow shop scheduling is one of the classic combinatorial optimization problems, which may have one or more of both local or global optimum points. The huge number of the possible variants (n!) makes the exhaustive exploration almost impossible. To avoid the trap of local optimum points, a reasonable choice would be a Monte Carlo search based technique, followed by a local search in the neighbourhood of the solution previously found (Goertzel,1993). To find the true optimum by means of a Monte Carlo technique, it is necessary to generate a large number of random points in the space of feasible problem solutions. If the number of random points is large enough, then the best solution found will be certainly a reasonable guess. It is important to note that by random points imply that manufacturing sequences are randomly generated. Therefore, in the case of global search procedure we will generate a random jobs permutation. Based on this random generated permutation, we will take the first two jobs as partial manufacturing sequence. Starting from this partial sequence and using the random generated permutation, each job will be inserted in the partial sequence in the proper position, based on the minimum C_{max} criterion. The phase of ordered manufacturing sequence is finalized when all the jobs from the list are inserted in the manufacturing sequence. The construction heuristic segment of the algorithm differ from NEH because it does not use a previously ordered jobs, according to the total manufacturing time of each job on all machines. In addition, it differs from the SPIRIT algorithm, as it does not select the first two jobs according to some criterion accomplishment.

However, this method starts from a random permutation of N_p integers. The manufacturing sequence is created based on a method of insertion type, in which every new job is inserted in the partial manufacturing sequence according to the minimum completion time principle. The job insertion order is carried out by the initial generated

random permutation. Since the phase of manufacturing sequence construction is repeated several times, we will follow the principles of a global search, based on Monte Carlo method.

```
while ( stopping condition not satisfied)
{
    // The random constructive heuristic (global
    search procedure)
    Generate a random permutation
        (p[i], i = 1, 2, ..., N);
    Append jobs in the order given by the random
    permutation,based on the condition of minimum
    Cmax;
    Calculate total completion time (Cmax);
    if (actual value of Cmax < previous value of Cmax)
        Minimum Cmax = Actual Cmax;

    // Permutation heuristic (local search procedure)
    Improve the actual order by permutations;
    Calculate total completion time (Cmax);
    if (actual value of Cmax < previous value of Cmax)
        Minimum Cmax = Actual Cmax;
}
```

Figure 1. The main steps performed by the heuristic algorithm

After reading the initial data corresponding to the manufacturing times of N_p jobs on M machines, the algorithm works in two phases (see figure 1). The first phase constructs manufacturing sequence, by initially generating a random permutation. In the second phase, the constructed manufacturing sequence is improved by means of a permutation scheme. These two phases are repeated iteratively until some of stopping criterion is achieved. This may be the iteration number or running time.

2.1 The Construction Heuristic Algorithm

To develop the jobs manufacturing sequence, the first step consists in generating a random permutation of N_p positive integers. The *rand()* function of an ANSI C compiler returns a value of type int (a two byte quantity on many machines), which must be at most 32767. This may be a worrying limitation, especially when conducting a Monte Carlo approach. In generating a random instance, the portable random numbers generator *ran1( )* of Park and Miller (Press,1997) was used, since this has a period larger than 10^8. The random permutation (see figure 2) is used for the job insertion order. Consider the random permutation (J_{r1}, J_{r2}, ..., J_{rNp}). We will take the first two jobs (J_{r1}, J_{r2}), according to the insertion order. The total completion time for the partial manufacturing sequence is calculated in two cases, to decide which of the arrangements (J_{r1}, J_{r2}), (J_{r2}, J_{r1}) is the best. Suppose this is (J_{r1}, J_{r2}). The subsequent task is to insert job J_{r3} in the manufacturing sequence using the random permutation. In other words, we need to test which of the variants (J_{r1}, J_{r2}, J_{r3}), (J_{r1}, J_{r3}, J_{r2}) or (J_{r3}, J_{r1}, J_{r2}) is the best. Jobs J_{r4}, .. J_{rNp} are inserted in the manufacturing sequence in a similar way.

```
// the random permutations generator
  for i = 0 to N
    r[i] = i;
  for i = 0 to N
  {
// call the random number generator ran1():
  Assign to q a random integer number,
    between i to (N-i);
  exchange r[i] with r[q];
  }
```

Figure 2. The random permutation algorithm

To construct the manufacturing sequence, an auxiliary variable labelled *njob* is used (see figure 3). Initially, *njob* takes the value 2 and is incremented each time a new job is inserted in the partial manufacturing sequence. The values of the manufacturing times are stored in a table labelled *t_ini[M][Np]*. The jobs partial sequence is stored in a vector labelled *PartialIndex[Np]*, whereas the optimum manufacturing sequence is stored in *OptimPartialIndex[Np]*.

The construction heuristic phase also call two routines: *calculateMatrixX( )*, which returns the values of the waiting times on each machine and *findPartialSequence( )*, which builds the partial optimum manufacturing sequence as well as the total completion time for the partial manufacturing sequence considered. The construction of the manufacturing sequence is completed when the condition njob = N_p is fulfilled.

```
njob = 2;          // chose first two jobs
for j=1 to M
        for i=1 to njob
        t[j][i] = t_ini[j][r[i]];
        //according to random permutation

for i=1 to njob
        PartialIndex[i] = r[i];
for j=1 to M
        for i=1 to njob
                t[j][i] = 0.0;
calculateMatrixX (x, t, njob);

tPartial = 0.0;
for j=1 to njob
        tPartial = tPartial + x[M][j] + t[M][j];
if ( tPartial < tPartialMin )
{
```

```
        tPartialMin = tPartial;
        for i=1 to njob
                OptimPartialIndex[i] =
        PartialIndex[i];
}

// reverse the order of the first two jobs
exchange PartialIndex[1] with PartialIndex[2]
for j=1 to M
        exchange t[j][1] with t[j][2];'
calculateMatrixX (x, t, njob);
tPartial = 0.0;
for j=1 to njob
        tPartial = tPartial + x[M][j] + t[M][j];
if( tPartial < tPartialMin )
{
        tPartialMin = tPartial;
        for i=1 to njob
                OptimPartialIndex[i] =
        PartialIndex[i];
}
while( njob <= Np )
{
        PartialIndex[njob] = r[njob];
        for j=1 to M
                t[j][njob] = t_ini[j][r[njob]];
        tPartialMin = findPartialSequence (t_ini,
        PartialIndex, njob);
        if ( tPartialMin < timpTotal && njob == Np
        )
        {
                timpTotal = tPartialMin;
                for i=1 to Np
                        index_optim[i] =
                PartialIndex[i];
        }
        njob++;
}
```

Figure 3. The random constructive heuristic

2.2 The Permutation Heuristic Algorithm

The manufacturing sequence generated during the construction heuristic phase is improved in the second phase by means of a permutation heuristic. The total completion time is computed for each set of jobs permutations. This completion time is updated when a manufacturing sequence yields a better result compared to the preceding sequences. The pseudo code of the permutation phase is presented in figure 4. At each permutation there is an exchange between two columns in the matrix $t[M][N_p]$ of jobs manufacturing times. Initially job J_{r1} is moved sequentially one place at a time, first following job J_{r2}, then following job J_{r3} and so on. By inserting job J_{r1} in the last position and job J_{r2} in the first position of the manufacturing sequence, we proceed by moving job J_{r3} in the same way as the job J_{r1}. This procedure is repeated for the remaining jobs.

```
for g = 1 to Np
  for k = 1 to Np-1
    for i = k to Np
      for j = 1 to M
        exchange t[j][i] with t[j][i+1]
      exchange index of job[i] with index of job[i+1]
      calculateX(x,t);
      calculate Actual Cmax
      if (Actual Cmax < Minimum Cmax)
        Minimum Cmax = Actual Cmax;
        for e = 0 to Np
          optimIndex[e] = index[e];
return Minimum Cmax;
```

Figure 4. The jobs permutation module

The first step of this permutation phase is completed when the job J_{r1} is inserted in the last position of the manufacturing sequence. There will be $n = N_p$ such steps which reduce the number of permutations from n! to n^2.

3 NUMERICAL RESULTS

The efficiency of the algorithm was tested on four groups of benchmark problems from OR Library (see http://mscmga.ms.ic.ac.uk/info.html) corresponding to Taillard's, Carlier's Heller's and Reeves. Tables 1 and 2 shows the computational results of these test instances.

Table 1. Taillard's benchmark problems

Nr. crt	Problem instance	Result	Upper bound	Gap [%]
Taillard's instances				
1	J20m5(1)	1283	1278	0.39
2	J20m5(2)	1359	1359	0.00
3	J20m5(3)	1100	1081	1.75
4	J20m5(4)	1323	1293	2.32
5	J20m5(5)	1250	1236	1.13
6	J20m5(6)	1210	1195	1.25
7	J20m5(7)	1256	1239	1.37
8	J20m5(8)	1237	1206	2.57
9	J20m5(9)	1256	1230	2.11
10	J20m5(10)	1127	1108	1.71
11	J20m10(1)	1636	1582	3.41
12	J20m10(2)	1732	1659	4.40
13	J20m10(3)	1563	1496	4.47
14	J20m10(4)	1440	1378	4.49
15	J20m10(5)	1491	1419	5.07
16	J50m5(1)	2755	2724	1.13
17	J50m5(2)	2905	2834	2.50
18	J50m5(3)	2676	2621	2.09
19	J50m5(4)	2843	2751	3.34
20	J50m5(5)	2887	2863	0.84

21	J50m10(1)	3328	3025	10.02
22	J50m10(2)	3213	2892	11.10
23	J50m10(3)	3214	2864	12.22
24	J50m10(4)	3365	3064	9.82
25	J50m10(5)	3323	2986	11.28
26	J100m5(1)	5572	5493	1.43
27	J100m5(2)	5380	5268	2.12
28	J100m5(3)	5328	5175	2.95
29	J100m5(4)	5140	5014	2.51
30	J100m5(5)	5380	5250	2.47
31	J200m10(1)	11556	10868	6.33
32	J200m10(2)	11410	10494	8.72
33	J200m10(3)	11669	10922	6.83
34	J200m10(4)	11447	10889	5.12
35	J200m10(5)	11467	10524	8.96

Both the number of machines used and the number of jobs processed affects the number of iterations required to get a good result. It was noted that the number of iterations increased more conspicuously by increasing the number of machines rather than increasing the number of jobs. In other words, the machine number has a greater influence than the job number on the number of iterations. The algorithm needs to execute a noticeable larger number of iterations, when the machine number is increased, in order to retain a similar gap between the result and the upper bound.

For Carlier's test instances, the algorithm reached the upper bound in all cases within 1000 iteration. This was mainly due to the smaller size of the problem. The average CPU time was 8.8 sec, on an Intell(R) Pentium at 3.01 GHz.

For the Heller's test instances we had only the results of Agarwal et al (Agarwal et al,2006) which were taken as upper bound. For Reeves's test instances the results are relative similar to those corresponding to Taillard's from Table 1.

Table 2. Makespans and gaps

Nr. crt	Problem instance	Result	Upper bound	Gap [%]
Carlier's instances				
1	Car1-11x5	7038	7038	0.00
2	Car1-13x4	7166	7166	0.00
3	Car3-12x5	7312	7312	0.00
4	Car4-14x4	8003	8003	0.00
5	Car5-10x6	7720	7720	0.00
6	Car6-8x9	8505	8505	0.00
7	Car7-7x7	6590	6590	0.00
8	Car8-8x8	8366	8366	0.00
Heller's instances				
9	Heller-20x10	145	136	6.62
10	Heller-100x10	550	516	6.59
Reeves instances				
11	ReC01-20x5	1279	1247	0.39
12	ReC02-20x5	1257	1247	0.80
13	ReC03-20x5	1117	1109	0.72
14	ReC04-20x5	1122	1109	1.17
15	ReC05-20x5	1247	1242	0.41
16	ReC06-20x5	1247	1242	0.41
17	ReC07-20x10	1599	1566	2.11
18	ReC08-20x10	1605	1566	2.49
19	ReC09-20x10	1598	1537	3.97
20	ReC10-20x10	1566	1537	1.89
21	ReC11-20x10	1513	1431	5.73
22	ReC12-20x10	1494	1431	4.40
23	ReC13-20x15	2017	1930	4.51
24	ReC15-20x15	2021	1950	3.64
25	ReC17-20x15	2016	1902	5.99
26	ReC19-30x10	2247	2093	7.36
27	ReC21-30x10	2176	2017	7.88
28	ReC23-30x10	2162	2011	7.51
29	ReC25-30x15	2732	2513	8.71
30	ReC27-30x15	2602	2373	9.65
31	ReC29-30x15	2553	2287	11.63
32	ReC31-50x10	3414	3045	12.12
33	ReC33-50x10	3354	3114	7.71
34	ReC35-50x10	3450	3277	5.28
35	ReC37-75x20	5731	4890	17.20
36	ReC39-75x20	5825	5043	15.51
37	ReC41-75x20	5782	4910	17.76

A different approach when making inferences about the total completion time is to use probability theory. Probability theory cannot establish explicitly the minimum makespan; however, given the distribution of the total completion time we can calculate the probability that this makespan is less than some specified duration. If the makespan has a Normal distribution with known parameters then the required probability is the area under this Normal curve beyond the specified duration. In the subsequent section, we explain the theoretical aspects of the Normal distribution and illustrate methods for checking the normality assumption using the results of Taillard's test instance j20m5(1). We also describe a procedure that transforms skewed distributions so that the normality assumption becomes more plausible.

The Normal distribution plays a very important role in statistical inference and is perhaps the most important distribution in statistical applications. The probability density function $f(t)$ for a Normal random variable t is given by:

$$f(t)=\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left[-\frac{(t-\mu)^2}{2\sigma^2}\right]; \qquad (1)$$

The parameters μ and σ are the mean and standard deviation of the Normal distribution. To determine probabilities relating to random variables

having a normal distribution, we make use of the following transformation:

$$z=\frac{t-\mu}{\sigma}; \tag{2}$$

Since the relationship between the values of t and z is linear, z must take on a value between $z_1=\frac{t_1-\mu}{\sigma}$ and $z_2=\frac{t_2-\mu}{\sigma}$, when t takes on a value between t_1 and t_2. Hence,

$$P(t_1<t<t_2)=$$

$$=\frac{1}{\sqrt{2\pi\sigma^2}}\int_{t_1}^{t_2}\exp\left[-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2\right]dt; \tag{3}$$

$$P(t_1<t<t_2)=\frac{1}{\sqrt{2\pi}}\int_{z_1}^{z_2}\exp\left[-\frac{1}{2}z^2\right]dz= \tag{4}$$

$$=P(z_1<z<z_2);$$

To illustrate the procedure, we the use the generated completion times of Taillard's test instance j20m5(1) to establish the probability that the makespan C_{max} is less than some specified duration. Preliminary investigation of the completion time distribution shows that it does not have an exact Normal distribution. The coefficient of skewness (0.208) indicates that the distribution is skewed to the right and the coefficient of kurtosis (-.185) specifies that the distribution is flatter than the Normal distribution (Francis et al,1993).

Table 3. Statistical measures for the skewed distribution of actual completion times

	Actual completion time
Coefficient of Skewness	.208
Coefficient of Kurtosis	-.185

Moreover, the Q-Q plot displaying the quantiles of the time completion distribution against the quantiles of the Normal distribution shows that the points near the left tail of the completion time distribution do not cluster around the straight line.

This implies that the smaller completion times are the observations that are most violating the normality assumption and these happen to be the data points that we are interested most.

One of the procedures that is normally employed to reduce departure from normality and make the time completion distribution less skewed is the Box-Cox transformation.

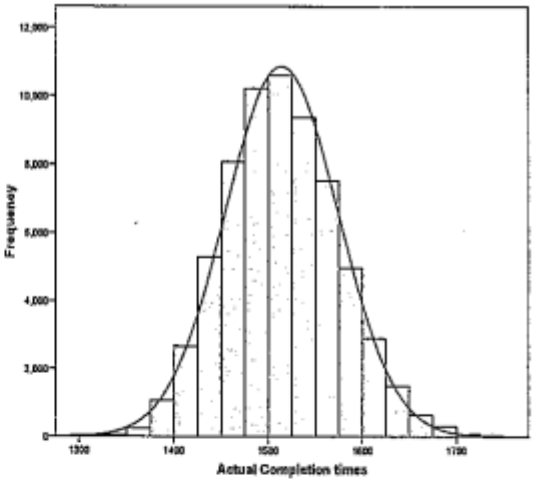

Figure 5. The distribution of the actual completion time, for the j20m5(1) instance, displayed on a normal curve

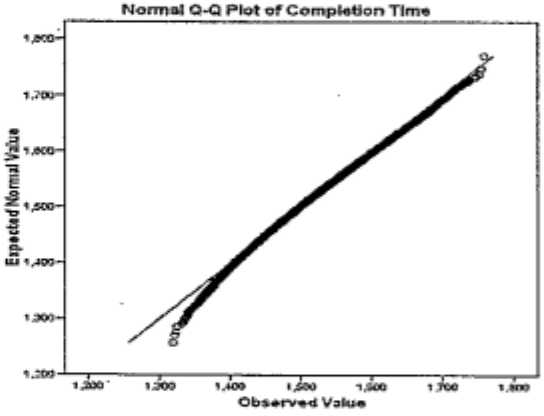

Figure 6. The normal Q-Q plot for the actual completion times

This transformation is defined as:

$$t(\lambda)=\begin{cases}\left(t^{\lambda}-1\right)/\lambda & for\ \lambda\neq 0\\ \log t & for\ \lambda=0\end{cases} \tag{5}$$

where λ is the transformation parameter and t is the completion time variable. Given the data t_i for $i=1,\ldots,n$ we assume that there exist some λ for which $t_i(\lambda)$ has a Normal distribution with mean μ and variance σ^2. For $\lambda\neq 0$, the density function can be written as:

$$f\left(t|\lambda,\mu,\sigma\right)=$$

$$=\frac{1}{\sqrt{2\pi\sigma^2}}|\lambda|t^{\lambda-1}\exp\left[-\frac{\left(t^{\lambda}-\mu\right)^2}{2\sigma^2}\right]; \tag{6}$$

For the given observations t_i, the log-likelihood function is given by:

$$l(\lambda,\mu,\sigma)=-\frac{n}{2}\log\left(2\pi\sigma^2\right)+n\log|\lambda|+$$
$$+(\lambda-1)\sum_i \log t_i-\sum_i \frac{\left(t_i^{\lambda}-\mu\right)^2}{2\sigma^2}; \quad (7)$$

The maximum likelihood estimate $\hat{\lambda}$ of the transformation parameter is obtained by setting $\partial l/\partial\lambda$ to 0. This is the value of λ that minimizes the deviance (-2 log-likelihood).

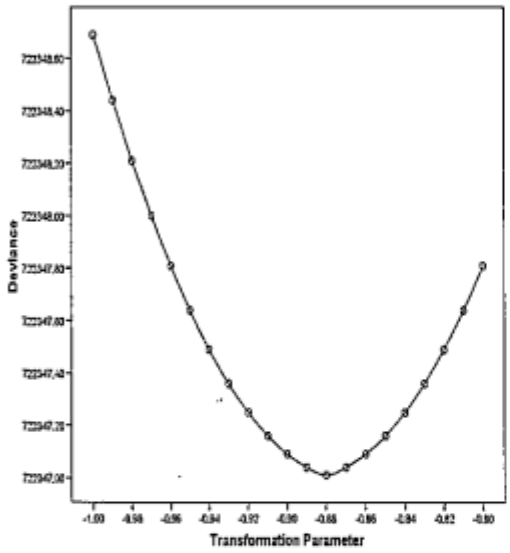

Figure 7. The deviance as a function transformation parameter

The deviance is minimized when $\hat{\lambda}=-0.88$. Using this value of λ, we can estimate the parameters μ and σ using maximum likelihood by setting both $\partial l/\partial\mu$ and $\partial l/\partial\sigma$ to 0.

$$\frac{\partial l}{\partial\mu}=\frac{1}{\mu^2}\sum_i\left(t_i^{\lambda}-\mu\right)=0 \quad and$$
$$\frac{\partial l}{\partial\sigma}=-\frac{n}{\sigma}+\frac{1}{\sigma^3}\sum_i\left(t_i^{\lambda}-\mu\right)^2 \quad (8)$$
$$\hat{\mu}=\frac{1}{n}\sum_i t_i^{\hat{\lambda}} \quad and \quad \hat{\sigma}^2=\frac{1}{n}\sum_i\left(t_i^{\hat{\lambda}}-\mu\right)^2$$

The coefficients of skewness (-0.002) and kurtosis (-0.019) are both close to 0 indicating that the transformed completion times have a Normal distribution.

Moreover, the data points near the tails of the transformed completion time distribution do cluster around the straight line.

Table 4. Parameter estimates

Parameter	Estimate
$\hat{\lambda}$	-0.88
$\hat{\mu}$	0.001592
$\hat{\sigma}^2$	$3.072\text{x}10^{-9}$

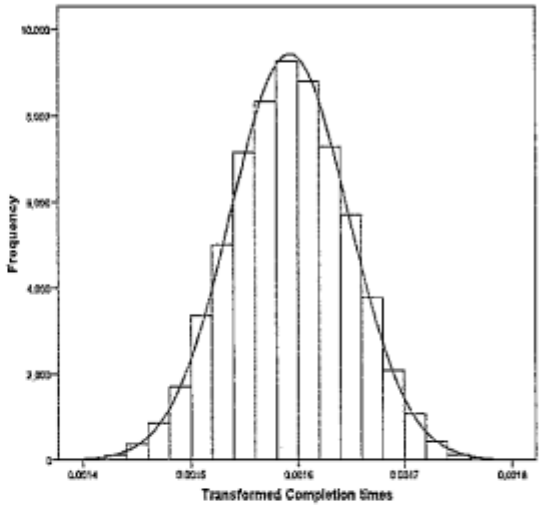

Figure 8. The distribution of the transformed completion times, for j20m5(1) instance, displayed on a normal curve

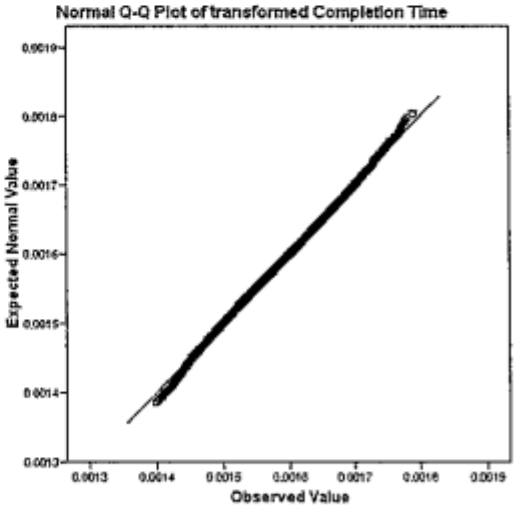

Figure 9. The normal Q-Q plot for the transformed completion times

Using these results it is possible to estimate the probability that the shortest processing time is less than a specified value α.

$$P(t<\alpha)=P\left(t^{\hat{\lambda}}>\alpha^{\hat{\lambda}}\right)=P\left(z>\frac{\alpha^{\hat{\lambda}}-\hat{\mu}}{\hat{\sigma}}\right); \quad (9)$$

The inequality sign is reversed because $\hat{\lambda}$ has a negative value. For example, the probability that the shortest processing time is less than 1300 is:

$$P(t<1300)=P\left(z>\frac{1300^{-0.88}-0.001592}{\sqrt{3.072\cdot10^{-9}}}\right)= \quad (10)$$

$$=P(z>4.5843)=2.279\cdot10^{-5};$$

The probability that the completion time is less than 1300 is equal to 0.000021786. This indicates that the event is very unlikely to occur. The probability that the shortest processing time is less than 1278 is:

$$P(t<1278)=P\left(z>\frac{1278^{-0.88}-0.001592}{\sqrt{3.072\cdot10^{-9}}}\right)= \quad (11)$$

$$=P(z>4.5843)=2.279\cdot10^{-6};$$

It is evident from these examples that as the specified duration approaches the minimum completion time the probability of finding a solution with a make span less than the specified value approaches zero.

4 CONCLUSIONS

This paper presents a new heuristic hybrid algorithm for solving the flowshop scheduling problem. The algorithm is based on a general optimization method of iterative type, which works in two stages. In the first stage, of a constructive heuristic type, a feasible schedule is generated, based on a Monte Carlo approach. In the second stage, the initial solution is improved by a local search procedure, based on adjacent jobs permutation.

The numerical tests show that the proposed algorithm works very well especially for problems of small and medium size. Concerning the performances of the algorithm, it seems that the number of machines used has a greater influence in both directions of CPU time and solution quality compared to the number of jobs processed.

The statistical analysis emphasise the limitations of the method. Further researches will take into consideration different strategies for the local search, in order to improve the solution and reduce the CPU time, especially for large sized problems.

5 APPENDIX: THE CALCULATION OF WAITING TIMES

Let t_{mn} be the manufacturing time of job J_n on machine m, where n = 1, 2, ..., N_p, and m = 1, 2, ..., M. In figure 10, $x_{mn} \geq 0$ denotes the waiting time that has to elapse before the execution of job J_n on machine m.

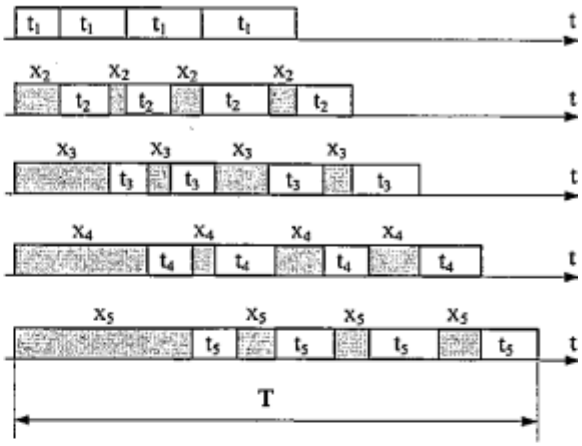

Figure 10. Gantt representation of the case: 4 jobs on 5 machines

By assuming that the execution of jobs on the first machine takes place without breaks, then by convention we will consider the waiting times on first machine to be null. So

$$x_{11}=x_{12}=...=x_{1N}=0; \quad (12)$$

Next, we determine the waiting time that elapses before the execution of each job, on any of the (M-1) remaining machines. For the first job in the list, the waiting time for machine M is equal to the sum of the manufacturing times corresponding to the previous machines:

$$x_{M1}=\sum_{j=1}^{M-1}t_{j1}; \quad (13)$$

Hence the required manufacturing time for the first job is:

$$T_{M1}=x_{M1}+t_{M1}; \quad (14)$$

For the next jobs J_n (n = 2, 3 , ..., N_p) in the manufacturing sequence, the waiting times corresponding to machine m ($1 \leq m \leq M$) can be determined using these equations:

$$x_{m2}=\max\left\{\sum_{i=1}^{1}t_{1i}+\sum_{j=1}^{m-1}t_{j2}-\sum_{i=1}^{1}(x_{mi}+t_{mi}),0\right\}; \quad (15)$$

$$x_{m3}=\max\left\{\sum_{i=1}^{2}t_{1i}+\sum_{j=1}^{m-1}t_{j3}-\sum_{i=1}^{2}(x_{mi}+t_{mi}),0\right\}; \quad (16)$$

.....

$$x_{mn}=\max\left\{\sum_{i=1}^{n-1}t_{1i}+\sum_{j=1}^{m-1}t_{jn}-\sum_{i=1}^{n-1}\left(x_{mi}+t_{mi}\right),0\right\}; \quad (17)$$

Equation (15) corresponds to the waiting times that elapse before the job J_2, in the manufacturing sequence, is executed on each machine m (m = 2, 3, ..., M). Similarly, the equations (16) and (17) correspond to the waiting times that elapse before the jobs J_3 and J_n respectively, in the manufacturing sequence, is executed on each machine m. It is noteworthy that in equations (15) to (17) there exists a relation of precedence which does not allow, for example, to calculate the waiting time x_{23} (which precede the manufacturing of J_3 on the second machine) before the waiting time x_{22} (which precede job J_2 on the same machine).

The total completion time for all N_p jobs on M machines is equal to the sum of manufacturing times corresponding to the jobs on the last machine added to the sum of the waiting times on the same machine. From equation (13), the waiting time that precedes the beginning of the execution of the first job on last machine is equal to the sum of manufacturing times of the job J_1 on the first (M-1) machines. The waiting time that precedes the execution of job J_2 on the last machine, can be obtained by setting the condition m = M in equation (15).

$$x_{M2}=\max\left\{\sum_{i=1}^{1}t_{1i}+\sum_{j=1}^{M-1}t_{j2}-\sum_{i=1}^{1}\left(x_{Mi}+t_{Mi}\right),0\right\}; \quad (18)$$

By summing x_{M1} and x_{M2} we get:

$$\sum_{i=1}^{2}x_{Mi}=\max\left\{\sum_{i=1}^{1}t_{1i}+\sum_{j=1}^{M-1}t_{j2}-\sum_{i=1}^{1}t_{Mi},x_{M1}\right\}; \quad (19)$$

Setting m = M in equation (16), we obtain the value of the waiting time that precede the execution of the third job on machine M:

$$x_{M3}=\max\left\{\sum_{i=1}^{2}t_{1i}+\sum_{j=1}^{M-1}t_{j3}-\sum_{i=1}^{2}\left(x_{Mi}+t_{Mi}\right),0\right\}; \quad (20)$$

By adding equations (19) and (20) we can obtain the sum of the waiting times that elapse before the execution of the third job on the last machine, so:

$$\sum_{i=1}^{3}x_{Mi}=\max\left\{\sum_{i=1}^{2}t_{1i}+\sum_{j=1}^{M-1}t_{j3}-\sum_{i=1}^{2}t_{Mi},\sum_{i=1}^{2}x_{Mi}\right\}; \quad (21)$$

Similarly, we can determine the sum of waiting times that elapse before the execution of all N_p jobs on machine M, by equation 22.

$$\sum_{i=1}^{Np}x_{Mi}=\max\left\{\sum_{i=1}^{Np-1}t_{1i}+\sum_{j=1}^{M-1}t_{jNp}-\sum_{i=1}^{Np-1}t_{Mi},\sum_{i=1}^{Np-1}x_{Mi}\right\}= \quad (22)$$

$$=\max\left\{\sum_{i=1}^{Np-1}t_{1i}+\sum_{j=1}^{M-1}t_{jNp}-\sum_{i=1}^{Np-1}t_{Mi},\sum_{i=1}^{Np-2}t_{1i}+\sum_{j=1}^{M-1}t_{jNp-1}-\sum_{i=1}^{Np-2}t_{Mi},\ldots\right.$$

$$\left.\ldots,\sum_{i=1}^{2}t_{1i}+\sum_{j=1}^{M-1}t_{j2}-\sum_{i=1}^{2}t_{Mi},\sum_{i=1}^{1}t_{1i}+\sum_{j=1}^{M-1}t_{j2}-\sum_{i=1}^{1}t_{Mi},\sum_{j=1}^{M-1}t_{j1}\right\};$$

Equation (22) is a general formulation of the sum of waiting times on the last machine, which can be easily proved by complete induction.

We denote the sum of waiting times on the last machine corresponding to the manufacturing sequence S, by $X_\alpha(S)$ ($1 \le \alpha \le N_p$).

$$X_\alpha(S)=\max_{1\le\alpha\le N_p}\left\{\sum_{i=1}^{\alpha-1}\left(t_{1i}-t_{Mi}\right)+\sum_{j=1}^{M-1}t_{j\alpha}\right\}; \quad (23)$$

The benefit of equation (23) lies in the fact that the waiting times are expressed solely in terms of the manufacturing times. $X_\alpha(S)$ is minimized only when the manufacturing sequence S is optimal.

Consider two different manufacturing sequences S_1 and S_2, of the N_p jobs:

$$S_1=\left(J_1,J_2,\ldots,J_k,J_{k+1},\ldots,J_{Np}\right); \quad (24)$$

$$S_2=\left(J_1,J_2,\ldots,J_{k+1},J_k,\ldots,J_{Np}\right); \quad (25)$$

In the manufacturing sequence S_2, unlike the initial sequence S_1, only the execution of jobs J_k and J_{k+1} are exchanged. The order in which the remaining jobs are executed is unchanged.

Let $L_\alpha(S)$ be defined by:

$$L_\alpha(S)=\left\{\sum_{i=1}^{\alpha-1}\left(t_{1i}-t_{Mi}\right)+\sum_{j=1}^{M-1}t_{j\alpha}\right\}; \quad (26)$$

Consequently, equation (23) can be expressed in a compressed form as:

$$X_\alpha(S) = \max_{1 \le \alpha \le N_p} \{L_\alpha(S)\}; \quad (27)$$

For the sequences S_1 and S_2 of job executions, the values L_α will be the same, except for those corresponding to $\alpha = k$ and $\alpha = k+1$ respectively. We can deduce that:

$$X_\alpha(S_1) = X_\alpha(S_2); \quad (28)$$

if and only if

$$\max\{L_k(S_1), L_{k+1}(S_1)\} = \max\{L_k(S_2), L_{k+1}(S_2)\}; \quad (29)$$

If condition (29) is not satisfied implies that one of the manufacturing sequences S_1, S_2 is better than the other.

6 REFERENCES

► S.N. Johnson, Optimal two - and three - stage production schedules with setup times included, Naval Research Logistics Quarterly. 1 (1954) 61-68.

► H.G. Campbell, R.A. Dudek, M.L. Smith, A heuristic algorithm for the n job, m machine sequencing problem, Management Science. Vol.16 No.10 (1970) B630-B637.

► C. Kuolamas, A new constructive heuristic for the flowshop scheduling problem, European Journal of Operational Research Society. 105 (1988) 66-71.

► D. Palmer, Sequencing jobs through a multi-stage process in the minimum total time - a quick method of obtaining a near optimum, Operational Research Quarterly. Vol.16 No.1 (1965) 101-107.

► J.N. Gupta, A functional heuristic algorithms for the flowshop scheduling problem, Operational Research Quarterly. Vol.22 No.1 (1971) 39-47.

► T.S. Hundal, J. Rajgopal, An extension of Palmer's heuristic for the flow shop scheduling problem, International Journal of Production Research. Vol.26 No.6 (1988) 1119-1124.

► D.G. Dannenbring, An evaluation of flow shop sequencing heuristics, Management Science. Vol 23 No.11 (1977) 1174-1182.

► E. Taillard, Some efficient heuristic methods for the flow-shop sequencing problem, European Journal of Operational Research. 47 (1990) 67-74.

► M. Nawaz, E.E. Enscore Jr., I. Ham, A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem, OMEGA, The International Journal of Management Science. Vol.11 No.1 (1983) 91-95.

► J.M. Framinan, R. Leisten, C. Rajendran, Different initial sequences for the heuristic of Nawaz, Enscore and Ham to minimize makespan, idletime or flowtime in the static permutation flowshop sequencing problem, International Journal of Production Research. Vol.41 No.1 (2003) 121-148.

► M. Widmer, A. Hertz, A new heuristic method for the flow shop sequencing problem, European Journal of Operational Research. 41 (1989) 186-193.

► J.C. Ho, Y. -L. Chang, A new heuristic for the n-job, m-machine flow-shop problem, European Journal of Operational Research. 52 (1991) 194-202.

► I. Osman, C. Potts, Simulated annealing for the permutation flow-shop scheduling, OMEGA, The International Journal of Management Science. Vol.17 No.6 (1989) 551-557.

► F. Ogbu, D. Smith, The application of the simulated annealing algorithm to the solution of the $n/m/C_{max}$ flowshop problem, Computers and Operations Research. Vol.17 No.3 (1990) 243-253.

► H. Ishibuchi, S. Misaki, H. Tanaka, Modified simulated annealing algorithms for the flow shop sequencing problem, European Journal of Operational Research. 81 (1995) 388-398.

► J.a.V. Moccellin, A new heuristic method for the permutation flow shop scheduling problem, Journal of the Operational Research Society. 46 (1995) 883-886.

► E. Nowicki, C. Smutnicki, A fast tabu search algorithm for the permutation flow-shop problem, European Journal of Operational Research. 91 (1996) 160-175.

► T. Murata, H. Ishibuchi, H. Tanaka, Genetic algorithms for flowshop scheduling problem, Computers and Industrial Engineering. Vol.30 No.4 (1996) 1061-1071.

► C. Reeves, T. Yamada, Genetic algorithms, path relinking and the flowshop sequencing problem, Evolutionary Computation. Vol.6 No.1 (1998) 45-60.

► S.G. Ponnambalam, P. Aravindan, S. Chandrasekaran, Constructive and improvement flow shop scheduling heuristics: An extensive evaluation, Production Planning and Control. Vol.12 No.4 (2001) 335-344.

► R. Ruiz, C. Morato, A comprehensive review and evaluation of permutation flowshop heuristics,

European Journal of Operational Research. 165 (2005) 479-494.
► A. Garrido, M.A. Salido, F. Barber, M.A. Lopez, Heuristic methods for solving job-shop scheduling problems. Proceedings of the Workshop on New Results in Planning, Scheduling and Design (PUK'00), ECAI00 Berlin, Germany, (2000) 44-51.
► B. Goertzel, The structure of intelligence, Berlin: Springer-Verlag, 1993.
► W.H. Press, et al., Numerical Recipes in C. The Art of Scientific Computing, second ed, New York: Cambridge University Press, 1997.
► A. Agarwal, S. Colak, E. Eryarsoy, Improvement heuristic for the flow-shop scheduling problem: An adaptive-learning approach, European Journal of Operational Research. 169 (2006) 801-815.
► M. Aitkin, D. Anderson, B. Francis, J. Hinde, Statistical Modelling in GLIM, Oxford Science Publications, 1994.
► P. McCullagh, J.A. Nelder, Generalized Linear Models, second ed. Chapman and Hall, London, 1989.
► B. Francis, M. Green, C. Payne, The GLIM 4 manual, Oxford Science Publications, 1993.