A hybrid genetic algorithm for solving no-wait flowshop scheduling problems

Article *in* International Journal of Advanced Manufacturing Technology · June 2010

DOI: 10.1007/s00170-010-3009-4

CITATIONS
36

READS
213

3 authors:

Bassem Jarboui
University of Sfax
118 PUBLICATIONS **1,869** CITATIONS
SEE PROFILE

Mansour Eddaly
Higher Institute of Business Administration in Gafsa, Tunisia
36 PUBLICATIONS **386** CITATIONS
SEE PROFILE

Patrick Siarry
Université Paris-Est Créteil Val de Marne - Université Paris 12
504 PUBLICATIONS **12,170** CITATIONS
SEE PROFILE

Int J Adv Manuf Technol (2011) 54:1129–1143
DOI 10.1007/s00170-010-3009-4

ORIGINAL ARTICLE

A hybrid genetic algorithm for solving no-wait flowshop scheduling problems

Bassem Jarboui · Mansour Eddaly · Patrick Siarry

Received: 17 June 2010 / Accepted: 1 November 2010 / Published online: 24 November 2010

Abstract A hybrid genetic algorithm is proposed in this paper to minimize the makespan and the total flowtime in the no-wait flowshop scheduling problem, which is known to be NP-hard for more than two machines. The Variable Neighborhood Search is used as an improvement procedure in the last step of the genetic algorithm. First, comparisons are provided with respect to several techniques that are representative of the previous works in the area. Then, we compare the results given by three proposed algorithms. For the makespan criterion as well as for the total flowtime, the computational results show that our algorithms are able to provide competitive results and new best upper bounds.

Keywords No-wait flowshop · Genetic algorithm · Variable Neighborhood Search · Makespan

1 Introduction

Because of the multitude of its applications in many real-world problems, including the chemical, pharmaceutical, metal, steel, plastic, and hotrolling industries [1], the no-wait flowshop problem has attracted much attention of researchers. In this variant of the flowshop scheduling problem, each job is to be processed without interruption on or between machines. In such way, if necessary, the start of a job on a given machine must be delayed such that the completion of the operation coincides with the start of the operation on the further machine. The no-wait flowshop problem is known to be NP-hard for more than two machines [2]. So, unless P = NP, the complexity of the problem increases exponentially with the increase of the instance's size. Indeed, most of research focused on approximation methods to solve this problem. Hall and Sriskandarajah [3] have presented a detailed survey on no-wait flowshop scheduling problems. In addition, no-wait flowshop scheduling presents a particularity vis a vis other scheduling problems: it can be transformed into the asymmetric traveling salesman problem (ATSP) [4].

We can classify the resolution techniques used for solving the no-wait flowshop scheduling problem into three classes: first, the constructive heuristics, such as [1, 5–10]; second, local search-based methods, which include Simulated Annealing Algorithm [6, 11], Tabu Search algorithm [6, 12], and Variable Neighborhood Search [13]. The last class consists of population-based algorithms, including Genetic Algorithms [11, 13, 14], Ant Colony algorithm [15], and Particle Swarm algorithm [16, 17].

Genetic algorithm (GA) is a stochastic search procedure based on mechanisms of natural selection, genetics, and evolution; it was invented by John Holland in 1960 [18]. Later, it became the most popular technique in evolutionary computation [19]. Therefore, the area of applications of GAs is very large. Husbands [20] has provided a survey on the applications of GA to scheduling.

The main framework of the genetic algorithm based on the selection and recombination operators is used for global exploration of the search space. Therefore, the genetic algorithm converges slowly toward local optima. To overcome this disadvantage, many works have proposed hybrid-

B. Jarboui · M. Eddaly
FSEGS,
route de l'aéroport km 4,
Sfax 3018, Tunisia

P. Siarry (✉)
LiSSi, Université de Paris 12,
61 avenue du Général de Gaulle,
94010, Créteil, France
e-mail: siarry@univ-paris12.fr

Fig. 1 Basic GA

```
Basic GA()
Generate an initial population of P individuals at random;
Fix crossover probability (P_c) and mutation probability (P_m)
loop
    Select two individuals P_1 and P_2 from the population;
    Create offspring O_1 and O_2 by applying crossover operator to P_1 and P_2 with a probability P_c;
    Mutate offspring O_1 and O_2 with a probability P_m;
    Add O_1 and O_2 to population by applying replacement operator;
until(stopping criterion is reached)
```

izing approaches, generally with local search techniques, for refining the search space, and performing local exploration [24]. In the flowshop sequencing context, several works have proposed to add a local search procedure to GA [21–23].

In this paper, we develop an effective hybrid genetic algorithm where Variable Neighborhood Search (VNS) Algorithm is introduced as an improvement phase to GA. Based on the main characteristics of the considered problem, we propose a new crossover operator that takes advantage of common sub-sequences provided by the best solutions.

Therefore, the paper is organized as follows. Section 2 presents a description of the no-wait flowshop scheduling problem. Section 3 describes the genetic algorithm. The Variable Neighborhood Search Algorithm is presented in Section 4. Section 5 presents our proposed algorithm and the computational results are presented in Section 6. Section 7 is the conclusion.

2 No-wait flowshop scheduling problem

In no-wait flowshop scheduling, there is a set of n jobs to be processed through a set of m machines. Let p_{ij} be the processing time for the job i on the machine j and $P_{[i]j}$ denote the processing time of the job in the position i on the machine j in the sequence s. where $i = 1, 2, ..., n$, $j = 1, 2, ...m$, $s = (s_1, s_2, ..., s_n)$. p_{ij} are fixed, known in advance, and have non-negative values.

The no-wait constraint implies that the operations of all jobs must be processed from start to end without any interruption on or between machines. The objective is to find a schedule that minimizes the makespan ($C_{\max}$) criterion.

```
Basic VNS()
Generate an initial solution x_best;
Select the set of neighbourhood structures η_k, k = 1,2,...,k_max;
do
    k = 1;
    do
        pick at random x' from η_k(x_best);
        Apply a local search method to x';
        if (x' is better than x_best)
            update x_best;
            k = 1;
        else k = k + 1;
        endif
    while(k ≤ k_max)
while(stopping conditions are not met)
```

Fig. 2 Basic VNS

Also, the no-wait flowshop scheduling problem can be transformed into ATSP [4]. According to this transformation, Pan et al. [16] and Liu et al. [17] have formulated the problem as follows: we denote by d_{ik} the minimum delay between the beginnings of job i and job k on the first machine with respect to the no-wait restriction, where the job k is processed immediately before the job i in the sequence.

$$d_{ik} = p_{i1} + \max\left\{\max_{2\le j\le m}\left[\sum_{l=2}^{j} p_{il} - \sum_{l=1}^{j-1} p_{kl}\right], 0\right\}$$

We denote by $C_{[i]}$ the completion time of the job located on position i in the sequence s.

$$C_{[1]} = \sum_{j=1}^{m} p_{[1]j}$$

$$C_{[i]} = \sum_{l=2}^{i} d_{[l-1][l]} + \sum_{j=1}^{m} p_{[i]j} \quad i = 2, 3, ..., n$$

So, the maximum completion time, also known as the makespan, can be obtained as follows:

$$C_{\max} = \max_{1in}\left\{C_{[i]}\right\}$$

The total flowtime (TFT) can be given by this formula:

$$\mathrm{TFT} = \sum_{i=1}^{n} C_{[i]}$$

$$\mathrm{TFT} = \sum_{i=2}^{n} (n+1-i) d_{[i-1][i]} + \sum_{i=1}^{n}\sum_{j=1}^{m} p_{ij}$$

3 Genetic algorithm

GA was created by John Holland in 1960 [18]. It is a kind of evolutionary algorithm that can be defined as a stochastic process based on a population of individuals. It operates as follows. Initially, each individual represents a potential

10	3	1	6	8	2	5	9	4	7

Fig. 3 Encoding scheme of a sequence of 10 jobs

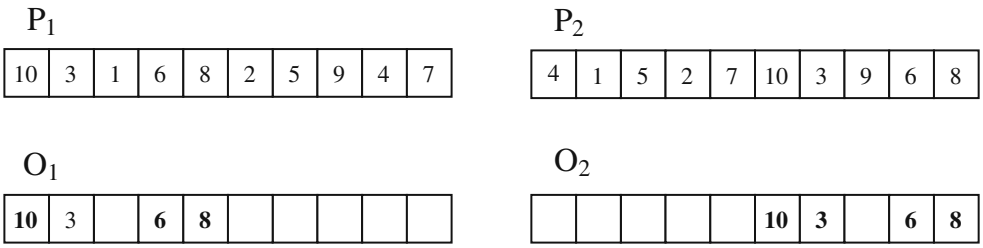

Fig. 4 Similar blocks of jobs

solution for the given problem obtained through an encoding mechanism. Next, an initial population of individuals is generated at random. Then, the individuals are ranked in the population according to a fitness function. Among individuals, selection, crossover, and mutation are applied relatively to calculated probabilities. Finally, replacement is usually performed by generating new individuals, called offspring [19]. Figure 1 shows the basic steps of a GA.

Since its discovery, GA has attracted the attention of the researchers. In fact, the area of applications of GAs is very large, especially in combinatorial optimization [24]. However, scheduling problems constitute a field of application of GAs. Husbands [20] has provided a survey of the applications of GAs to scheduling problems and Reeves [25] has presented the basic concepts of GAs developed for the permutation flowshop scheduling problem in order to minimize the makespan criterion. In particular, for the no-wait flowshop scheduling, Chen et al. [14] and Aldowaisan and Allahvardi [11] have adopted this algorithm to solve the problem.

4 Variable neighborhood search algorithm

VNS is a recent metaheuristic for combinatorial and global optimization. It was introduced by Mladenovic and Hansen [26]. The basic idea of this technique is to allow a systematic change in neighborhood structures of the current best (incumbent) solution within randomized local search. Its effectiveness was tested on several combinatorial problems, such as clustering, TSP [27], scheduling [28, 29]. The fundamental steps of this procedure are shown in Fig. 2.

Also, the no-wait flowshop scheduling constitutes an area of application of the VNS algorithm [13]. In the VNS algorithm of Schuster and Framinan, [13], the initial solution is generated by employing several heuristics and choosing the sequence with the best value of the objective function. The obtained sequence is then introduced into a local search procedure, which consists in selecting a job from the current sequence and inserting it in all positions. The best solution will be retained. To escape from the local optimum, k consecutive jobs are inserted in the same order in all positions.

5 Hybrid genetic algorithm for the no-wait flowshop scheduling problem

This section discusses the structure and rationale of the proposed hybrid genetic algorithm (GA-VNS). This algorithm will be depicted by the following steps.

5.1 Encoding scheme

For solving the no-wait flowshop scheduling problem by a GA, the first task is to represent a solution of a problem as a chromosome. The most known encoding scheme used for this problem is a simple permutation of jobs as a chromosome. The order of the jobs in the permutation denotes the processing order of the jobs by the machines. Figure 3 illustrates the encoding scheme of a sequence of 10 jobs.

5.2 Initial population

The initial population is commonly selected randomly in GAs as for Schuster and Framinan [13]. Nevertheless, this

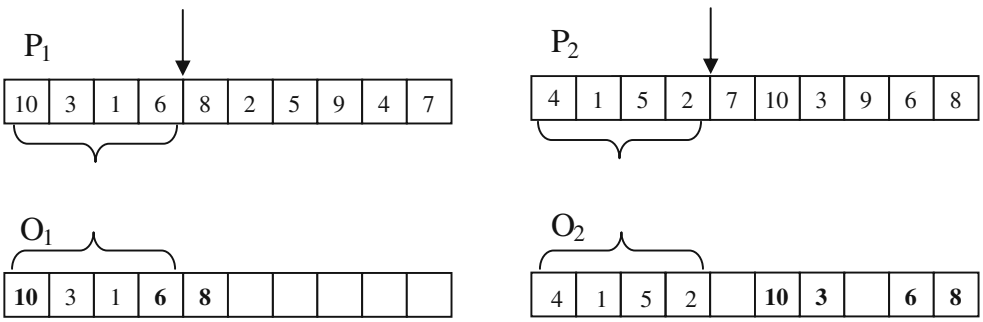

Fig. 5 Cut points

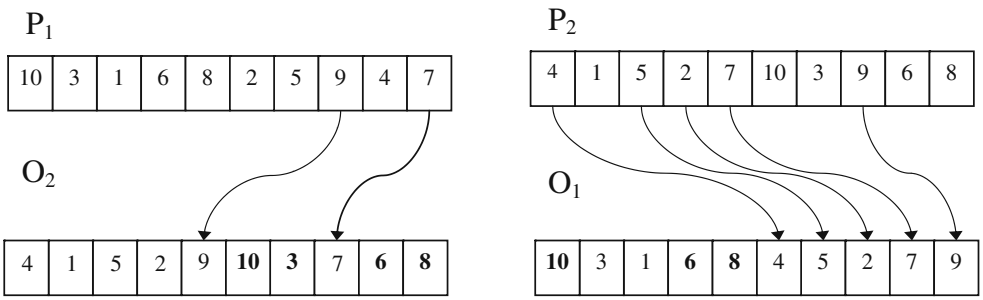

Fig. 6 Adding the missed jobs

procedure does not provide a good result [22]. In Chen et al. [14] and Aldowaisan and Allahverdi [11], the half of individuals are randomly generated and the other half are generated by using known heuristics of the problem.

In our paper, we propose an initialization procedure based on the NEH heuristic [30]. We adopt the variant of NEH proposed by Framinann et al. [31]. This procedure can be described as follows:

Step1: Generate a sequence of n jobs at random.
Step2: Take the first two jobs and evaluate the two possible schedules containing them. The sequence with better objective function value is taken for further consideration.
Step3: Take every remaining job in the permutation given in Step 1 and find the best schedule by placing it at all possible positions in the sequence of jobs that are already scheduled.

We notice that the P individuals of the initial population were all generated using the above procedure where P is the size of the initial population.

5.3 Selection

Selection provides the driving force in GAs: too much selection may induce GA finishing prematurely, and not enough selection may induce slow convergence of the GA.

In our algorithm we adopted the same procedure of selection employed by Reeves [25] for solving the flowshop scheduling problem under the makespan criterion. We describe this procedure as follows.

First, for each individual p, calculate the fitness value $f(p) = \frac{1}{C_{\max}(p)}$; second the individuals of the initial population are sorted in ascending order according to their fitness, i.e., the individual with a higher makespan value will be at the top of the list. Finally, the selection of parents will be made relatively to this probability:

$$\text{prob}(r) = \frac{2r}{P(P+1)}$$

where r is the rank of the rth individual in the sorted list.

Under the total flowtime criterion, the selection procedure used is the known truncation selection. Among the list of sorted individuals according to the TFT, 30% from the best solutions are selected as parents for the crossover operator.

5.4 Crossover

The crossover (recombination) operator is the main operator in a genetic algorithm. It produces new individuals called "offspring" from the selected parents, in order to obtain "better" offspring (in our case, better sequences). For the sequencing problem, various crossover operators have been proposed: Partially Mapped Crossover (PMX) [32], Order Crossover [33], Position-Based Crossover [34], Order-Based Crossover [35], Cycle Crossover [35], Linear Order Crossover [36], Subsequent Exchange Crossover [37], and Partial Schedule Exchange Crossover [38].

Ruiz et al. [22] have proposed a crossover operator for the permutation flowshop scheduling, Similar Block Order Crossover. It considers only similar blocks that occupy the same position in both parents.

The PMX operator is used by Chen et al. [14] and Aldowaisan and Allahverdi [11]. Schuster and Framinann [13] use a specific crossover operator which guarantees the feasibility of the obtained offspring.

In addition, the crossover probability P_c is the basic parameter in the crossover procedure. It is a parameter deciding if crossover will be performed or not. If there is no crossover, offspring and parents are similar. Else, offspring are made from parts of both parents' chromosomes.

In our algorithm, we proposed a new crossover operator, called "Block Order Crossover", based on the blocks of jobs in both parents in any position, i.e., not necessarily in the same position. Our operator comprises of the following steps:

Step1: Find the similar blocks of consecutive jobs (containing at least two jobs), regardless of their positions, from both parents (P_1 and P_2) and copy them to both offspring (O_1 and O_2; Fig. 4). If there are no similar blocks, go to step 2.
Step2: Choose at random a cut point. O_1 (respectively O_2) inherits all jobs from P_1 (respectively P_2) up to the chosen cut point (Fig. 5).
Step3: Add the missed jobs to O_1 (respectively O_2) with respect to their orders in P_2 (respectively P_1; Fig. 6).

```
Crossover()
input P1, P2;
output O1, O2;
for i = 1 to N
    position1(P1(i)) = i;                  // the position of job P1(i) in parent1
    position2(P2(i)) = i;                  // the position of job P2(i) in parent2
    assigned_job1(i) = false;              // the assigned job in offspring1 (take true if job i is already assigned
                                           // to offspring1 and false otherwise)
    assigned_job2(i) = false;              // the assigned job in offspring2 (take true if job i is already assigned
                                           // to offspring2 and false otherwise)
    assigned_position1(i) = false;         // the assigned position in offspring1 (take true if there existe a job
                                           // already scheduled in the position i in offspring1 and false otherwise)
    assigned_position2(i) = false;         // the assigned position in offspring2 (take true if there exist a job
endfor                                     // already scheduled in the position i in offspring2 and false otherwise)
for i = 1 to N − 1
    j=position2(P1(i))                     // j take the position of job P1(i) in parent2 where P1(i) = P2(j)
    if (j < N)
        if (P2(j + 1) == P1(i + 1))        // if the successors of the job (P1(i) = P2(j)) are the in the two parent then
            O1(i) = P1(i);                 // assign job P1(i) and P1(i + 1) in the positions i and i + 1 in offspring1 respectively
            O1(i + 1) = P1(i + 1);
            O2(j) = P2(j);                 // assign job P2(j) and P2(j + 1) in the positions j and j + 1 in offspring2 respectively
            O2(j + 1) = P2(j + 1);
            assigned_job1(P1(i)) = true;           // mark that the job P1(i) is already assigned in offspring1
            assigned_job1(P1(i + 1)) = true;       // mark that the job P1(i + 1) is already assigned in offspring1
            assigned_job2(P2(j)) = true;           // mark that the job P2(j) is already assigned in offspring2
            assigned_job2(P2(j + 1)) = true;       // mark that the job P2(j + 1) is already assigned in offspring2
            assigned_position1(i) = true;          // mark that there exist a job already scheduled in the postion i in O1
            assigned_position1(i + 1) = true;      //mark that there exist a job already scheduled in the postion i + 1 in O1
            assigned_position2(j) = true;          //mark that there exist a job already scheduled in the postion j in O2
            assigned_position2(j + 1) = true;      //mark that there exist a job already scheduled in the postion j + 1 in O2
        endif
    endif
endfor
k = rand(N − 1);                           //select at random a cut point
for i = 1 to k
    O1(i) = P1(i);                         // assign the job P1(i) on the position i in offspring1
    assigned_position1(i) = true;          // mark that a job is alredy scheduled the position i in offspring1
    assigned_job1(P1(i)) = true;           // mark that the job P1(i) is already scheduled in the offspring1
    O2(i) = P2(i);                         // assign the job P2(i) on the position i in offspring2
    assigned_position2(i) = true;          // mark that a job is alredy scheduled the position i in offspring2
    assigned_job2(P2(i)) = true;           // mark that the job P2(i) is already scheduled in the offspring2
endfor
j = k + 1;
for i = 1 to N
    if (assigned_job1(P2(i)) == false)     // if the job P2(i) is not already scheduled in offspring1 then
        while(assigned_position1(j) == true & j ≤ N) j++;      // find the first empty position j in offspring1 and
        if (j ≤ N)
            O1(j) = P2(i);                 //assign the job P2(i) in the position j in offspring1
            j++;
        else break;
        endif
    endif
endfor
j = k + 1;
for i = 1 to N
    if (assigned_job2(P1(i)) == false)     // if the job P1(i) is not already scheduled in offspring2 then
        while(assigned_position2(j) == true & j ≤ N) j++;      // find the first empty position j in offspring2 and
        if (j ≤ N)
            O2(j) = P1(i);                 //assign the job P1(i) in the position j in offspring2
            j++;
        else break;
        endif
    endif
endfor
```

Fig. 7 Pseudo code of crossover

Fig. 8 Swap local search

```
Swap local search()
input x_0                    // the input solution
output x_best                // the output solution
x_best = x_0;
set i = 1;
loop
    set j = i+1;
    do
        create a new solution by exchanging the jobs in position i and j in the best sequence;
        if (the new solution is better than x_best) then
            update x_best;
            i = i - 1;
            break;
        else j = j+1;
        endif
    while(j ≤ n)
    i = i+1;
    if (i > n-1) set i = 1;
until (no possible improvement)
```

The pseudo code of the crossover operator, given in Fig. 7, shows that this operation can be performed in linear time.

5.5 Mutation

After crossover, obtained offspring are subject to mutation. This operator helps the algorithm to escape from a local optimum by the process of diversification. It is viewed as a background operator that introduces new genetic structures in an individual by a random modification of its string. For permutation encoding, there exist several mutation operators:

Inversion	two positions are chosen at random and the section of jobs between these two positions is inverted
Insertion	a job is chosen randomly and inserted at a random position
Swap	two positions are chosen at random and the jobs on these positions are exchanged
Shift	selects a job at random and shifts it to a random position right or left of the job's position.

Chen et al. [14] and Aldowaisan and Allahverdi [11] use the same mutation operator by swapping two randomly picked jobs in a sequence. In Schuster and Framinann [13], the inversion operator is used.

Like for crossover, the mutation probability (P_m) plays an important role in the step of mutation. It indicates if the modification of the chromosome will be performed or not. We adopted the insertion operator in our algorithm.

5.6 Variable neighborhood search

During the past decade, various hybridization methods have been applied to GA aiming at preventing it from being stuck into a local optimum. The hybrid approach was extensively exploited in combinatorial optimization. We propose to apply a VNS algorithm as an improvement procedure after mutation. Let $p^c = \exp\left(\frac{\mathrm{RD}}{\alpha}\right)$ be the calculated probability for application of VNS where $\mathrm{RD} = \left(\frac{f(x_{\mathrm{current}}) - f(x_{\mathrm{best}})}{f(x_{\mathrm{best}})}\right)$, x_{current} denotes the created offspring, x_{best} denotes the best solution found by the algorithm and α is a negative constant. More the absolute value of α is large, more the created offspring has a large probability of being subjected to an application of a VNS.

For each individual, we draw at random a number between 0 and 1. If this number is less than or equal to p^c, then we apply VNS to the individual under consideration.

Fig. 9 Insert local search

```
Insert local search()
input x_0                    // the input solution
output x_best                // the output solution
x_best = x_0;
set i = 1;
loop
    set j = 1;
    do
        if (j = i)  j = j+1;
        create a new solution by inserting the job in position i into position j in the best sequence;
        if (the new solution is better than x_best) then
            update x_best;
            i = i - 1;
            break;
        else j = j+1;
        endif
    while(j ≤ n)
    i = i+1;
    if (i > n) set i = 1;
until (no possible improvement)
```

Fig. 10 VNS algorithm

```
VNS algorithm()
input x_0                          // the input solution
output x_best                      // the output solution
x_best = x_0;
x_current = x_0;
loop
    do
        find the optimum local x' by applying swap local search to x_current;
        find the optimum local x_current by applying insert local search to x';
    while (x_current is better than x')
    if (x_current is better than x_best ) then
        update x_best;
    endif
    x_current = x_best;
    if (rand ≤ 0.5) then
        select i and j at random and insert the job in position i into position j in x_current;
    else
        select i and j at random and exchange the jobs in position i and j in x_current;
    endif
until(stopping criterion is reached)
```

We select two structures of neighborhoods, called *swap_local_search* and *insert_local_search*. The first one leads to all possible swaps of pairs of job's positions (i,l), where $1 \leq il \leq n$, within all parts of solutions (Fig. 8).

After performing the swap moves, the obtained local optimum is introduced to the second structure. The latter consists in doing all possible insert moves of jobs within all parts of the obtained solution (Fig. 9).

Table 1 Results of local search-based algorithms

		Best known	RAJ	DS	DS + M	TS	TS + M	TS + MP	VNS(1)[a]	VNS(2)[a]		
Instances	m*n									Min	Average	Max
REC01	5*20	1526	4.19	0.33	0.46	0.00	0.07	0.07	1.31	0.00	0.02	0.07
REC03	5*20	1361	7.05	3.38	3.38	0.00	0.00	0.00	2.42	0.00	0.10	1.91
REC05	5*20	1511	8.34	2.25	2.25	0.33	0.07	0.00	0.73	0.00	0.04	0.20
REC07	10*20	2042	3.77	2.64	2.64	0.00	0.00	0.00	1.37	0.00	0.02	0.19
REC09	10*20	2042	4.85	1.08	1.08	0.00	0.05	0.05	2.35	0.00	0.03	0.15
REC11	10*20	1881	3.46	1.97	1.97	0.00	0.00	0.00	1.86	0.00	0.26	0.95
REC13	15*20	2545	6.44	2.79	2.79	0.00	0.00	0.00	0.31	0.00	0.23	1.88
REC15	15*20	2529	6.41	1.27	1.27	0.12	0.00	0.12	0.12	0.00	0.17	1.19
REC17	15*20	2587	5.91	0.08	0.08	0.00	0.00	0.00	0.46	0.00	0.12	1.00
REC19	10*30	2850	10.77	2.25	2.53	0.00	0.53	0.39	2.39	0.00	0.49	1.22
REC21	10*30	2821	6.88	2.94	2.94	0.07	0.14	0.28	2.38	0.00	0.25	0.53
REC23	10*30	2700	12.22	4.04	4.04	0.15	0.19	0.00	0.15	0.00	0.40	1.07
REC25	15*30	3593	6.74	3.45	3.45	0.36	0.00	0.11	0.92	0.00	0.11	0.70
REC27	15*30	3431	6.53	2.65	2.65	0.52	0.03	0.32	0.32	0.00	0.23	1.02
REC29	15*30	3291	8.87	1.00	1.00	0.36	0.00	0.36	1.00	0.00	0.40	0.82
REC31	10*50	**4307**	7.52	3.48	3.46	1.18	0.93	0.84	2.46	0.28	0.65	1.11
REC33	10*50	**4424**	7.82	5.70	5.65	1.88	1.02	0.95	2.06	0.63	1.10	1.80
REC35	10*50	**4397**	7.30	2.00	2.00	0.77	0.68	0.96	1.39	0.00	0.50	1.54
REC37	20*75	**8017**	12.00	3.27	3.13	1.46	1.37	1.52	0.80	0.47	0.96	1.55
REC39	20*75	**8420**	8.76	3.36	3.19	1.15	1.16	1.25	2.98	0.51	1.01	1.74
REC41	20*75	**8437**	10.75	4.02	4.02	1.03	1.26	0.98	2.55	0.34	0.87	1.42
HEL01	10*100	**704**	10.80	5.11	4.69	1.85	1.70	1.56	2.13	0.71	1.33	2.55
HEL02	10*20	179	5.59	1.68	1.68	0.00	0.00	0.00	0.56	0.00	0.08	0.56
Average				2.64	2.62	0.49	0.40	0.42	1.44	0.13	0.41	1.09

[a] VNS(1) and VNS(2) refer respectively to VNS algorithm of Schuster and Framinan [13] and our VNS algorithm

Table 2 Results of population-based algorithms

		Best known	GA-SA	DPSO			GA			$DPSO_{VND}$			GA-VNS		
Instances	m*n			min	average	max	min	average	max	min	average	max	min	average	max
REC01	5*20	1526	0.07	0.46	0.61	1.18	0.00	0.96	1.96	0.00	0.00	0.00	0.00	0.00	0.07
REC03	5*20	1361	2.28	1.92	2.09	2.43	0.00	1.77	3.37	0.00	0.00	0.00	0.00	0.00	0.00
REC05	5*20	1511	0.86	3.84	1.65	2.25	0.33	0.82	1.32	0.00	0.02	0.20	0.00	0.01	0.07
REC07	10*20	2042	0.20	0.05	0.45	0.98	0.19	0.68	1.32	0.00	0.00	0.00	0.00	0.00	0.00
REC09	10*20	2042	0.15	0.54	1.58	3.87	0.05	0.68	1.17	0.00	0.00	0.00	0.00	0.02	0.05
REC11	10*20	1881	0.00	0.00	0.84	1.86	0.58	1.47	2.12	0.00	0.00	0.00	0.00	0.00	0.00
REC13	15*20	2545	0.43	0.43	0.83	1.92	0.43	0.84	1.92	0.00	0.00	0.00	0.00	0.00	0.00
REC15	15*20	2529	0.00	0.00	0.51	1.03	0.00	0.40	0.86	0.00	0.00	0.00	0.00	0.00	0.00
REC17	15*20	2587	0.12	0.08	0.33	0.58	0.07	0.16	0.42	0.00	0.00	0.00	0.00	0.00	0.00
REC19	10*30	2850	4.74	1.69	2.14	2.66	1.50	2.21	2.98	0.00	0.19	0.38	0.00	0.11	0.38
REC21	10*30	2821	4.50	1.66	2.20	2.59	0.77	1.86	2.41	0.00	0.11	0.42	0.00	0.09	0.35
REC23	10*30	2700	4.70	2.22	2.37	3.11	2.00	2.62	3.22	0.00	0.05	0.92	0.00	0.07	0.63
REC25	15*30	3593	3.87	1.11	1.83	2.44	0.58	1.58	2.28	0.00	0.05	0.11	0.00	0.00	0.00
REC27	15*30	3431	3.76	1.75	2.77	3.00	0.20	1.71	2.59	0.00	0.25	0.73	0.00	0.08	0.32
REC29	15*30	3291	4.53	1.61	1.99	2.37	0.75	1.52	1.97	0.00	0.09	0.64	0.00	0.13	0.36
REC31	10*50	**4307**	10.45	2.25	2.91	3.62	1.97	2.99	4.13	0.11	0.43	0.67	0.18	0.51	0.76
REC33	10*50	**4424**	12.97	3.26	4.46	5.61	3.27	3.90	4.65	0.18	0.62	0.97	0.09	0.71	1.49
REC35	10*50	**4397**	11.23	2.48	3.26	3.89	2.16	2.85	3.68	0.09	0.29	0.84	0.00	0.40	0.72
REC37	20*75	**8017**	18.60	2.86	3.62	4.32	2.95	3.89	4.61	0.17	0.73	1.10	0.15	0.78	1.24
REC39	20*75	**8420**	18.34	2.71	3.27	3.78	3.00	3.54	4.29	0.54	0.80	1.18	0.10	0.75	1.19
REC41	20*75	**8437**	18.26	3.42	3.99	4.58	2.57	3.43	4.13	0.47	0.71	0.97	0.15	0.68	1.01
HEL01	10*100	**704**	24.57	3.98	4.60	4.97	4.11	4.77	5.68	0.42	0.64	0.99	0.57	1.12	1.56
HEL 02	10*20	179	0.56	1.12	3.21	4.47	0.00	0.58	1.67	0.00	0.00	0.00	0.00	0.00	0.00
Average			6.31	1.71	2.24	2.94	1.19	1.97	2.73	0.09	0.22	0.44	0.05	0.24	0.44

If a local optimum is obtained after the application of the *insert_local_search*, we return to the *swap_local_search* with the improved solution as a current solution. We reapply this procedure until there is no improvement in the solution.

If the obtained solution is better than the best solution known until now, before performing the local search procedures, then we replace the latter with the new solution and we reinitialize the number of iterations at 1, else we increment the number of iterations.

Next, we draw at random a number between 0 and 1. If this number is less than or equal to 0.5, then we select two distinct positions at random from x_{best} and the jobs on these positions are exchanged. Else, we select a job randomly from x_{best} and insert it at a random position. The whole procedure is repeated until reaching the maximal number of iterations ($iter_{max}$; Fig. 10).

5.7 Replacement

Replacement is the last phase in a GA; it consists in updating the population. Therefore, at each iteration, two offspring are generated from two parents but not all four individuals will be returned to the population, so two individuals must be replaced. There are many techniques to decide which individuals may stay in the population.

In our algorithm, for each offspring, we propose to select an individual at random among a subset of the sorted list of parents. This subset is composed of 20% of worst individuals in the population. If the offspring is best than this individual, then the latter is removed from the population and replaced with the new individual.

5.8 Stopping criterion

The stopping condition indicates when the search will be terminated. Various stopping criteria may be listed, such as maximum number of generations, bound of time, maximum number of iterations without improvement, etc.

6 Computational results

The algorithm was coded in C++ programming language. All experiments with GA-VNS for no-wait flowshop

Table 3 CPU times of local search-based algorithms

		DS	DS + M	TS	TS + M	TS + MP	VNS(1)	VNS(2)		
Instances	m*n							min	average	max
REC01	5*20	0.00	0.00	0.20	0.20	0.20	0.00	0.00	0.02	0.96
REC03	5*20	0.00	0.00	0.20	0.20	0.20	0.00	0.00	0.02	1.77
REC05	5*20	0.00	0.00	0.20	0.20	0.20	0.00	0.00	0.05	0.82
REC07	10*20	0.00	0.00	0.20	0.20	0.20	0.00	0.00	0.09	0.68
REC09	10*20	0.00	0.00	0.20	0.20	0.20	0.00	0.00	0.06	0.68
REC11	10*20	0.00	0.00	0.20	0.20	0.20	0.00	0.00	0.05	1.47
REC13	15*20	0.00	0.00	0.30	0.30	0.30	0.00	0.00	0.09	0.84
REC15	15*20	0.00	0.00	0.30	0.30	0.30	0.00	0.00	0.06	0.40
REC17	15*20	0.00	0.00	0.30	0.30	0.30	0.00	0.00	0.03	0.16
REC19	10*30	0.00	0.00	0.40	0.40	0.40	1.00	0.00	0.11	2.21
REC21	10*30	0.00	0.00	0.40	0.40	0.40	1.00	0.00	0.22	1.86
REC23	10*30	0.00	0.00	0.40	0.40	0.40	0.00	0.00	0.13	2.62
REC25	15*30	0.00	0.00	0.50	0.50	0.50	1.00	0.00	0.16	1.58
REC27	15*30	0.00	0.00	0.50	0.50	0.50	1.00	0.00	0.18	1.71
REC29	15*30	0.00	0.00	0.50	0.50	0.50	1.00	0.00	0.21	1.52
REC31	10*50	0.00	0.00	1.10	1.10	1.10	5.00	0.06	0.41	2.99
REC33	10*50	0.00	0.00	1.10	1.10	1.10	7.00	0.06	0.71	3.90
REC35	10*50	0.00	0.00	1.10	1.10	1.10	7.00	0.15	0.61	2.85
REC37	20*75	0.10	0.00	2.50	2.60	2.60	122.00	0.09	1.39	3.89
REC39	20*75	0.10	0.00	2.50	2.60	2.60	106.00	0.11	1.36	3.54
REC41	20*75	0.10	0.00	2.50	2.60	2.60	110.00	0.15	1.36	3.43
HEL01	10*100	0.10	0.00	3.80	3.90	3.90	185.00	0.29	2.19	4.77
HEL 02	10*20	0.00	0.00	0.20	0.20	0.20	0.00	0.00	0.03	0.58
average		0.02	0.00	0.85	0.87	0.87	23.78	0.04	0.41	1.97

VNS(1) is run on Athlon 1,400 MHz [13]. DS, DS + M, TS, TS + M, and TS + MP are run on Pentium 1,000 MHZ [12]

scheduling problem, with respect to the makespan and the total flowtime criteria, were run in Windows XP on a desktop PC with Intel Pentium IV, 3.2 GHz processor and 512 MB memory. In total, GA-VNS was tested for 133 instances distributed as follows: two instances of Heller [39], 21 instances of Reeves [25], and 110 instances from Taillard [40] with sizes $m = 5, 10, \text{and} 20$ and $n = 20, 50, 100, \text{and } 200$.

Until now, Taillard's instances have not been tested for the no-wait flowshop scheduling problem to minimize the makespan. Therefore, we proposed new upper bounds of the makespan for these instances. All results were obtained after 20 replications for each instance.

The corresponding parameters of GA-VNS were set, after a series of preliminary experiments, to the following values: population size (P) 10, crossover probability (P_c) 1.00, and mutation probability (P_m) 0.8. We fixed the parameter α, according to the relative deviation RD between the current solution and the best solution found by the algorithm, and the calculated probability p^c. Numerically, p^c=0.5 leads to accepting a sequence with an objective function value superior by 5% relatively to the best value of makespan found. So, $\alpha = \frac{\text{RD}}{\log(p^c)} = \frac{0.05}{\log(0.5)}$, thereafter we determined p^c according to this formula $p^c = \exp\left(\frac{\text{RD}}{\alpha}\right)$ and maximum number of iterations of VNS algorithm ($iter_{\max}$). In our case, we set a bound of time equal to $\left(\frac{n^2}{2}\right)ms$ as a stopping criterion.

The performance measure employed in our numerical study was average relative percentage deviation in makespan or total flowtime Δ_{average}:

$$\Delta_{\text{average}} = \frac{\sum_{i=1}^{R}\left(\frac{\text{Heu}_i - \text{Best}_{\text{known}}}{\text{Best}_{\text{known}}} \times 100\right)}{R}$$

where Heu_i is the solution given by any of the R=10 replications of the considered algorithms and $\text{Best}_{\text{known}}$ is the best solution provided by our algorithm through extensive experiments for each instance. In order to obtain the sequences of jobs relative to these upper bounds, the interested readers can contact us.

Table 4 CPU times of population-based algorithms

		GA-SA	DPSO			GA			$DPSO_{VND}$			GA-VNS		
Instances	m*n		Min	Average	Max	Min	Average	Max	Min	Average	Max	Min	Average	Max
REC01	5*20	6.00	0.00	0.00	0.02	0.00	0.00	0.02	0.00	0.01	0.03	0.00	0.006	0.08
REC03	5*20	6.00	0.00	0.00	0.02	0.00	0.00	0.02	0.00	0.00	0.02	0.00	0.00	0.01
REC05	5*20	7.00	0.00	0.00	0.02	0.00	0.00	0.03	0.00	0.02	0.06	0.00	0.00	0.01
REC07	10*20	12.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.02	0.00	0.01	0.06
REC09	10*20	11.00	0.00	0.00	0.02	0.00	0.00	0.02	0.00	0.00	0.02	0.00	0.02	0.09
REC11	10*20	10.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.01	0.03	0.00	0.01	0.09
REC13	15*20	17.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.01	0.03	0.00	0.01	0.09
REC15	15*20	17.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.02	0.00	0.01	0.09
REC17	15*20	16.00	0.00	0.00	0.02	0.00	0.00	0.02	0.00	0.00	0.02	0.00	0.01	0.08
REC19	10*30	34.00	0.00	0.01	0.02	0.00	0.00	0.03	0.00	0.02	0.08	0.00	0.00	0.01
REC21	10*30	35.00	0.00	0.01	0.02	0.00	0.00	0.03	0.00	0.04	0.16	0.00	0.02	0.10
REC23	10*30	35.00	0.00	0.00	0.02	0.00	0.00	0.03	0.01	0.05	0.17	0.00	0.05	0.21
REC25	15*30	55.00	0.00	0.00	0.02	0.00	0.00	0.05	0.00	0.03	0.17	0.00	0.02	0.12
REC27	15*30	51.00	0.00	0.00	0.02	0.00	0.01	0.05	0.00	0.06	0.16	0.01	0.07	0.21
REC29	15*30	54.00	0.00	0.01	0.02	0.00	0.00	0.01	0.00	0.02	0.14	0.00	0.06	0.18
REC31	10*50	147.00	0.00	0.02	0.03	0.00	0.04	0.14	0.06	0.33	0.56	0.10	0.34	0.62
REC33	10*50	145.00	0.00	0.02	0.03	0.00	0.07	0.24	0.05	0.34	0.56	0.18	0.36	0.59
REC35	10*50	146.00	0.00	0.02	0.05	0.00	0.05	0.23	0.16	0.34	0.58	0.09	0.30	0.60
REC37	20*75	907.00	0.00	0.03	0.06	0.00	0.20	0.96	0.34	1.16	1.58	1.13	0.35	1.40
REC39	20*75	890.00	0.00	0.04	0.11	0.00	0.15	0.55	0.17	0.99	1.63	0.36	0.98	1.43
REC41	20*75	904.00	0.00	0.04	0.08	0.00	0.12	0.20	0.20	0.88	1.56	0.28	1.12	1.42
HEL01	10*100	1088.00	0.00	0.05	0.22	0.00	0.13	0.42	0.67	1.86	2.66	0.82	1.68	2.50
HEL 02	10*20	10.00	0.00	0.00	0.02	0.00	0.00	0.03	0.00	0.00	0.01	0.00	0.00	0.01
Average		200.13	0.00	0.01	0.04	0.00	0.03	0.14	0.07	0.27	0.45	0.13	0.24	0.44

DPSO and $DPSO_{VND}$ are run on Pentium IV 3.0 GHz [16]

6.1 Results for GA, VNS, and GA-VNS based on instances of Reeves and Heller under the makespan criterion

We coded the proposed GA and VNS algorithm independently of GA-VNS algorithm and we compared their results against other approaches used for this problem.

This comparison is based on the instances of Reeves and Heller (23 instances). The results corresponding to the local search based algorithms are shown in Table 1 and the results from the population-based algorithms are presented in Table 2. The corresponding CPU times of these algorithms are measured in seconds and are given in Tables 3 and 4.

It can be pointed out that the bold numbers in Tables 1 and 2 indicate that the best known solutions were provided by one of our proposed algorithms.

The methods used for comparison are: heuristic algorithm provided by Rajendran [41], five local search algorithms based on several variants of descending search and tabu search algorithms (DS, DS + M, TS, TS + M and TS + MP) employed by Garbowski and Pempera [12], the VNS and GA-SA of Schuster and Framinan [13], and DPSO and $DPSO_{VND}$ of Pan et al.[16].

Table 1 shows that our VNS algorithm is better than all other local search-based algorithms with the exception of TS + M in terms of $\Delta_{average}$, but it is also better than TS + M with respect to minimum value. In particular it outperforms VNS(1). Indeed, in average, VNS(2) provides a $\Delta_{average}$ equal to 0.41 and a CPU time equal to 1.97 s (Table 3), whereas the VNS algorithm of Schuster and Framinan [13] provides a $\Delta_{average}$ equal to 1.44 and a CPU time equal to 23.78 s (Table 3).

Regarding the results of population-based algorithms, Table 2 shows that the searching quality of our GA is very superior to that of GA-SA with respect of quality of solutions and CPU time. Also, it can be seen that the proposed GA outperforms the DPSO algorithm in terms of minimum, average, and maximum relative deviations. However, our GA is competitive in terms of CPU time.

Analysis of Table 2 reveals also that the performance of GA-VNS remains competitive. Especially, the comparison between GA-VNS and $DPSO_{VND}$ shows that the results of these algorithms are very near. But it is clear that GA-VNS dominates $DPSO_{VND}$ in minimum deviation. In fact, the former reaches a minimum deviation, in average, equal to

Table 5 Upper bounds of makespan criterion for Taillard's benchmark problems

Instances	UB	Instances	UB	Instances	UB	Instances	UB
tai01	1486	tai31	3161	tai61	6402	tai91	15319
tai02	1528	tai32	3432	tai62	6240	tai92	15126
tai03	1460	tai33	3211	tai63	6133	tai93	15398
tai04	1588	tai34	3339	tai64	6025	tai94	15240
tai05	1449	tai35	3356	tai65	6198	tai95	15259
tai06	1481	tai36	3347	tai66	6087	tai96	15116
tai07	1483	tai37	3231	tai67	6255	tai97	15415
tai08	1482	tai38	3235	tai68	6130	tai98	15279
tai09	1469	tai39	3072	tai69	6381	tai99	15135
tai10	1377	tai40	3317	tai70	6384	tai100	15340
tai11	2044	tai41	4274	tai71	8079	tai101	19740
tai12	2166	tai42	4177	tai72	7886	tai102	20112
tai13	1940	tai43	4099	tai73	8028	tai103	19937
tai14	1811	tai44	4399	tai74	8348	tai104	19961
tai15	1933	tai45	4322	tai75	7958	tai105	19849
tai16	1892	tai46	4289	tai76	7814	tai106	19942
tai17	1963	tai47	4420	tai77	7866	tai107	20112
tai18	2057	tai48	4318	tai78	7913	tai108	20064
tai19	1973	tai49	4155	tai79	8166	tai109	19918
tai20	2051	tai50	4283	tai80	8117	tai110	19942
tai21	2973	tai51	6129	tai81	10700		
tai22	2852	tai52	5725	tai82	10594		
tai23	3013	tai53	5862	tai83	10611		
tai24	3001	tai54	5788	tai84	10607		
tai25	3003	tai55	5886	tai85	10539		
tai26	2998	tai56	5863	tai86	10677		
tai27	3052	tai57	5962	tai87	10835		
tai28	2839	tai58	5926	tai88	10840		
tai29	3009	tai59	5876	tai89	10723		
tai30	2979	tai60	5958	tai90	10798		

Table 6 Relative percentage deviation in makespan relative to Taillard's instances

	GA			VNS			GA-VNS		
n*m	Min	Average	Max	Min	Average	Max	Min	Average	Max
20*05	0.57	1.62	2.84	0.00	0.10	0.27	0.00	0.03	0.10
20*10	0.45	1.60	2.89	0.00	0.13	0.52	0.00	0.06	0.15
20*20	0.27	1.14	2.09	0.00	0.13	0.48	0.00	0.03	0.07
50*05	3.22	4.43	5.54	0.27	0.64	1.09	0.17	0.55	0.94
50*10	2.52	3.65	4.76	0.16	0.52	1.00	0.09	0.47	0.87
50*20	2.05	3.14	4.21	0.14	0.55	1.15	0.07	0.47	1.04
100*05	4.68	5.75	6.89	0.45	0.91	1.39	0.42	0.84	1.27
100*10	4.19	4.94	5.66	0.42	0.91	1.38	0.43	0.85	1.32
100*20	3.36	4.09	4.72	0.46	0.89	1.43	0.42	0.91	1.44
200*10	5.18	5.80	6.33	0.68	1.08	1.54	1.16	1.68	2.24
200*20	3.93	4.54	5.03	0.62	1.00	1.43	1.08	1.52	1.98
average	2.77	3.70	4.63	0.29	0.62	1.06	0.35	0.67	1.04

Table 7 CPU times relative to Taillard's instances

	GA			VNS			GA-VNS		
n*m	Min	Average	Max	Min	Average	Max	Min	Average	Max
20*05	0.00	0.00	0.00	0.00	0.00	0.04	0.00	0.01	0.06
20*10	0.00	0.00	0.00	0.00	0.00	0.04	0.00	0.00	0.05
20*20	0.00	0.00	0.00	0.00	0.01	0.06	0.00	0.01	0.07
50*05	0.01	0.04	0.15	0.05	0.32	0.61	0.15	0.42	0.62
50*10	0.00	0.02	0.11	0.03	0.28	0.60	0.12	0.37	0.62
50*20	0.00	0.02	0.09	0.04	0.26	0.59	0.07	0.35	0.62
100*05	0.16	0.38	0.95	1.06	1.98	2.48	1.48	2.23	2.53
100*10	0.04	0.21	0.58	0.77	1.86	2.49	1.43	2.18	2.52
100*20	0.02	0.20	0.52	0.91	1.90	2.48	1.53	2.18	2.50
200*10	0.71	1.74	3.77	6.06	9.02	10.01	6.22	9.14	10.20
200*20	0.48	1.37	3.05	6.16	8.95	9.97	7.26	9.34	10.18
average	0.13	0.36	0.84	1.37	2.23	2.67	1.66	2.38	2.72

Table 8 Best results found by our algorithms for total flowtime criterion

Instances	VNS	GA-VNS	Instances	VNS	GA-VNS	Instances	VNS	GA-VNS	Instances	VNS	GA-VNS
tai01	15674	15674	tai31	75684	75682	tai61	305164	304668	tai91	1495092	1496865
tai02	17250	17250	tai32	82874	82874	tai62	**297531**	297647	tai92	1482086	**1474411**
tai03	15821	15821	tai33	78103	78103	tai63	**291120**	291844	tai93	1493442	1494196
tai04	17970	17970	tai34	82492	**82422**	tai64	276644	**275721**	tai94	**1462009**	1464596
tai05	15317	15317	tai35	83679	83493	tai65	289933	**289728**	tai95	1478430	**1476137**
tai06	15501	15501	tai36	80713	**80655**	tai66	287084	**286219**	tai96	1476213	**1472509**
tai07	15706	15693	tai37	78755	78604	tai67	298175	**297591**	tai97	1512445	**1509555**
tai08	15955	15955	tai38	78892	**78726**	tai68	287493	**287107**	tai98	1497924	**1496867**
tai09	16385	16385	tai39	75724	**75679**	tai69	304806	**304195**	tai99	1478059	**1476884**
tai10	15329	15329	tai40	83678	83550	tai70	297937	**297427**	tai100	**1488747**	1490476
tai11	25205	25205	tai41	114382	**113908**	tai71	410161	**408905**	tai101	1993117	**1985273**
tai12	26342	26342	tai42	112591	112663	tai72	391020	**390443**	tai102	2026254	**2022831**
tai13	22910	22910	tai43	105706	105365	tai73	402596	402826	tai103	2024113	**2012629**
tai14	22243	22243	tai44	113404	**113337**	tai74	419699	**418027**	tai104	2023240	**2015455**
tai15	23150	23150	tai45	115425	115425	tai75	398599	398165	tai105	2007551	**2003080**
tai16	22011	22011	tai46	**112489**	112719	tai76	388621	**387324**	tai106	2023896	2017560
tai17	21939	21939	tai47	116564	**116492**	tai77	390969	**390912**	tai107	2021408	**2017680**
tai18	24158	24158	tai48	**115040**	115082	tai78	**398128**	398436	tai108	2022321	**2020334**
tai19	23501	23501	tai49	110413	110499	tai79	**411793**	413384	tai109	2012138	**2002309**
tai20	24597	24597	tai50	113427	113462	tai80	409012	407684	tai110	2023063	**2015827**
tai21	38597	38597	tai51	173051	172883	tai81	559404	**557120**			
tai22	37571	37571	tai52	161115	160836	tai82	562643	**562111**			
tai23	38312	38312	tai53	160916	160104	tai83	560250	**559813**			
tai24	38802	38802	tai54	161567	161813	tai84	560850	560590			
tai25	39060	39012	tai55	167555	**167319**	tai85	552600	552756			
tai26	38562	38562	tai56	**161599**	162028	tai86	559810	558617			
tai27	39663	39663	tai57	167507	167115	tai87	572101	571073			
tai28	37000	37000	tai58	168485	**168113**	tai88	576943	**573870**			
tai29	39228	39228	tai59	165292	165292	tai89	561955	**559855**			
tai30	37931	37931	tai60	168386	168766	tai90	569479	569631			

Table 9 Comparative results with respect to total flowtime criterion

	SA		TS (Chins)		TS (Pilot 10)		DPSOVND				VNS				GA-VNS			
Instances	Average	t	Average	t	Average	t	Min	Average	Max	t	Min	Average	Max	t	Min	Average	Max	t
20 jobs	0.00	1000	0.00	1000	0.00	1000	0.00	0.00	0.00	0.00	0.00	0.05	0.12	0.01	0.00	0.03	0.07	0.00
50 jobs	0.98	1000	0.88	1000	0.74	1000	0.4	0.57	0.79	0.36	0.50	0.76	1.07	0.32	0.42	0.63	0.9	0.44
100 jobs	2.64	1000	2.2	1000	1.88	1000	0.87	1.15	1.46	3.61	1.00	1.31	1.61	4.36	**0.85**	**1.14**	**1.39**	3.94
200 jobs	1.00	1000	1.19	1000	0.08	1000	−1.63	−1.38	−1.09	27.78	−1.56	−1.31	−1.00	23.24	**−1.76**	**−1.49**	**−1.19**	24.79
average	**1.16**	1000	**1.07**	1000	**0.68**	1000	**−0.09**	**0.09**	**0.29**	**7.94**	**−0.02**	**0.20**	**0.45**	**6.98**	**−0.12**	**0.08**	**0.29**	**7.29**

SA, TS(Chins) ans TS(Pilot 10) are run on Pentium II with 266 MHz [6]. DPSO and DPSOVND are run on Pentium IV 3.0 GHz [16]

0.05 whereas for the latter this deviation is equal to 0.09. For the CPU time requirement, the average time to reach the best solution of the GA-VNS algorithm was shorter than that of the $DPSO_{VND}$ (Table 4).

6.2 Results for GA, VNS, and GA-VNS based on instances of Taillard under the makespan criterion

All previous works, in the literature, that have addressed the no-wait flowshop problem did not test the benchmark problems from Taillard [40] in order to minimize the makespan criterion. These instances are available in OR-Library's web site. They consist of 11 different problem types and 10 instances of each type.

We proposed upper bounds of these benchmark problems by using the GA-VNS algorithm with an extensive search (Table 5).

From Tables 6 and 7, it can be seen that GA-VNS is better than GA and VNS both in terms of quality of solution (including minimum, average, maximum deviations, and the range of results between maximum and minimum deviations) and time requirements for the first eight instances (until 100 jobs, 10 machines). For the last three classes of instances, the VNS algorithm is better than GA-VNS, because the latter needs more computations for providing good solutions for large instances.

6.3 Experimental results based on instances of Taillard under the total flowtime criterion

In order to prove the efficiency of our algorithm according to competing approaches of this problem, we evaluate it with respect to the total flowtime criterion. This comparison is performed based on 110 instances of Taillard [40]. The compared approaches used include Simulated Annealing algorithm (SA), two Tabu Search algorithms (TS(chins) and TS(pilot10)) of Fink andVoß [6], and $DPSO_{VND}$ of Pan et al. [16].

The major part of parameters of our algorithm remains similar as for makespan criterion, except for the parameter α and the stopping criterion. α is set with a probability $p^c = 0.1$ for accepting a sequence with TFT superior by 1% relatively to the best value of TFT found. So, $\alpha = \frac{RD}{\log(p^c)} = \frac{0.01}{\log(0.1)}$. In such way, we determined p^c according to $p^c = \exp\left(\frac{RD}{\alpha}\right)$. The stopping criterion was fixed to $\left(\frac{n^2 \times 2}{3}\right) ms$ for this comparative study in order to align with the computational times of competing approaches.

The performance measures used are minimum, average, and maximum relative percentage deviations, over five replications, denoted by min, average, and max, respectively. These deviations are calculated according to the optimal solutions, for the case of 20 jobs, and the lower bound values, for the cases of 50 and 100 jobs, obtained by using the three-index formulation of Picard and Queyranne [42]. On the other hand, for large-scale problems (200 jobs), these deviations are calculated according to the best obtained solutions, with respect to the total flowtime criterion, provided by the algorithms of Fink and Voß [6]. Table 8 presents the best obtained solutions by our algorithms (GA-VNS or VNS). The number marked in bold face denotes the improved values of the total flowtime according to the best results of Fink and Voß [6] and Pan et al. [16]. Therefore, in total, 50 solutions among 110 have been improved by our algorithms where VNS algorithm participates by nine improved solutions. Also, it can be seen from this table that a large part of our improvements occurs in the large sizes (100 and 200 jobs) with 38 improved solutions. This proves the effectiveness of our algorithms for solving large-scale problems.

The results of the comparison with SA and TS algorithms of Fink and Voß [6] and $DPSO_{VND}$ of Pan et al. [16] are displayed in Table 9. In average of all instances, GA-VNS outperforms all competing approaches in terms of minimum, average, and maximum percentage deviations. Although, $DPSO_{VND}$ seems better than GA-VNS for the instances with 20 and 50 jobs, our algorithm is the best for the large sizes. Among the local search-based algorithms, our VNS algorithm provided the best results. Regarding the computational effort, in average over five replications, GA-VNS appears fast to find good results.

According to these results, we can see that the GA without hybridizing, since it performs global exploration of the space search, cannot converge rapidly toward local optima. On the other hand, the VNS algorithm can find good solutions, because it performs local exploration, but this result is extremely sensitive to the starting solutions of this algorithm. Therefore, the hybridization procedure between GA and VNS can coordinate the main properties of these algorithms, where VNS offers a local exploration tool to the GA and the latter offers a global exploration, by giving a good beginning for the VNS which covers a large part of the search space. Moreover, we can observe that this cooperation reduces the range of results between maximum and minimum deviations, regarding the results of GA and VNS. Thus, the hybrid approach can provide more robustness.

7 Conclusion

In this paper, we have proposed a hybrid genetic algorithm for the no-wait flowshop scheduling with respect to the makespan criterion. The algorithm starts with a population of individuals created by a heuristic procedure called RNEH, based on NEH heuristic. The used crossover operator, called Block Order Crossover, consists of finding the similar blocks of consecutive jobs from both parents regardless their positions. Then, we copied them into offspring by keeping the same positions. In order to prevent the algorithm from being stuck into local optima, we introduced an improvement procedure by using a VNS algorithm.

We have improved the bounds for the instances of Reeves and Heller with extensive experiments of the GA-VNS algorithm. Then we have compared our proposed algorithm with other approaches that have been used for the same problem. The results show that our algorithm remains efficient both in terms of quality of solution and time requirement. Also, we have tested our algorithm on 110 Taillard's benchmark problems. The results reveal that GA-VNS is also efficient for the instances with small sizes. But, for large size instances, VNS algorithm is better than GA-VNS. So the hybridization process has an unfavorable effect in this case. On the other hand, a comparative study is performed with respect to the total flowtime criterion. The computational results show, in average, the effectiveness, and the efficiency of our algorithms against the other techniques from the literature.

References

1. Framinan JM, Nagano MS (2008) Evaluating the performance for makespan minimisation in no-wait flowshop sequencing. J Mater Process Technol 197:1–9
2. Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. Freeman, San Francisco
3. Hall NG, Sriskandarajah C (1996) A survey of machine scheduling problems with blocking and no-wait in process. Oper Res 44:510–525
4. Bagchi TP, Gupta JND, Sriskandarajah C (2006) A review of TSP based approaches for flowshop scheduling. Eur J Oper Res 169:816–854
5. Rajendran C, Chaudhuri D (1990) Heuristic algorithms for continuous flow-shop problem. Nav Res Logist 37:695–705
6. Fink A, Voβ S (2003) Solving the continuous flowshop scheduling problem by metaheuristics. Eur J Oper Res 151:400–414
7. Aldowaisan T, Allahverdi A (2004) New heuristics for m-machine no-wait flowshop to minimize total completion time. Omega 32:345–352
8. Aldowaisan T (2001) A new heuristic and dominance relations for no-wait flowshops with setups. Comput Oper Res 28:563–584
9. Aldowaisan T, Allahverdi A (1998) Total flowtime in no-wait flowshops with separated setup times. Comput Oper Res 25:757–765
10. Gangadharan R, Rajendran C (1993) Heuristic algorithms for scheduling in the no-wait flowshop. Int J Prod Econ 32:285–290
11. Aldowaisan T, Allahverdi A (2003) New heuristics for no-wait flowshops to minimize makespan. Comput Oper Res 30:1219–1231
12. Grabowski J, Pempera J (2005) Some local search algorithms for no-wait flow-shop problem with makespan criterion. Comput Oper Res 32:2197–2212
13. Schuster CJ, Framinan JM (2003) Approximative procedure for no-wait job shop scheduling. Oper Res Lett 31:308–318
14. Chen CL, Neppalli RV, Aljaber N (1996) Genetic algorithms applied to the continuous flowshop problem. Comput Ind Eng 30:919–929
15. Shyu SJ, Lin BMT, Yin PY (2004) Application of ant colony optimization for no-wait flowshop scheduling problem to minimize the total completion time. Comput Ind Eng 47:181–193
16. Pan QK, Tasgetiren MF, Liang YC (2008) A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem. Comput Oper Res 35:2807–2839
17. Liu B, Wang L, Jin Y-H (2007) An effective hybrid particle swarm optimization for no-wait flow shop scheduling. Int J Adv Manuf Technol 31:1001–1011
18. Holland J (1975) Adaptation in natural and artificial systems. University of Michigan Press, MI
19. Sivanandam SN, Deepa SN (2008) Introduction to genetic algorithms. Springer, New York
20. Husbands P (1994) Genetic algorithms for scheduling, AISB Quaterly, 89
21. Murata T, Ishibuchi H, Tanaka H (1996) Genetic algorithms for flowshop scheduling problems. Comput Ind Eng 30:1061–1071
22. Ruiz R, Maroto C, Alcaraz J (2006) Two new robust genetic algorithms for the flowshop scheduling problem. Omega 34:461–476
23. Zhang Y, Li X, Wang Q (2009) Hybrid genetic algorithm for permutation flowshop scheduling problems with total flowtime minimization. Eur J Oper Res 196:869–876
24. Gen M, Cheng R (2000) Genetic algorithms and engineering optimization. Wiley, Hoboken
25. Reeves CR (1995) A genetic algorithm for flowshop sequencing. Comput Oper Res 22:5–13
26. Mladenovic N, Hansen P (1997) Variable neighborhood search. Comput Oper Res 24:1097–1100
27. Hansen P, Mladenovic N (2001) Variable neighborhood search: principles and applications. Eur J Oper Res 130:449–467
28. Liao C, Cheng C (2007) A variable neighborhood search for minimizing single machine weighted earliness and tardiness with common due date. Comput Ind Eng 52:404–413

29. Blazewicz J, Pesch EM, Sterna, Werner F (2008) Metaheuristic approaches for the two-machine flow-shop problem with weighted late work criterion and common due date. Comput Oper Res 35:574–599
30. Nawaz M, Enscore EEJ, Ham I (1983) A heuristic algorithm for the m-machine, n job flow-shop sequencing problem. Omega 11 (1):91–95
31. Framinan JM, Leisten R, Ruiz-Usano R (2002) Efficient heuristics for flowshop sequencing with the objectives of makespan and flowtime minimisation. Eur J Oper Res 141:559–569
32. Goldberg D, Lingle R Jr (1985) Alleles, loci, and the travelling salesman problem. In: Grefenstette JJ (ed) Proceedings of the first international conference on genetic algorithms and their applications. Lawrence Erlbaum associates, NJ, pp 154–159
33. Davis L (1985) Applying adaptive algorithms to epistatic domains. Proceedings of the Ninth International Joint Conference on Artificial Intelligence 1:162–164
34. Syswerda G (2) Uniform crossover in genetic algorithms. In: Schaffer J (ed) Proceedings of the third international conference on genetic algorithms. Morgan Kaufmann, San Mateo, pp 9–1989
35. Oliver I, Smith D, Holland J (1987) A study of permutation crossover operators on the traveling salesman problem. In: Grefenstette, J. (Ed.), Proceedings of second international conference on genetic algorithms and their applications. Hillsdale, NJ, 224–230
36. Falkenauer E, Bouffoix S (1991) A genetic algorithm for job shop. In: Proceedings of the 1991 IEEE international Conference on Robotics and Automation
37. Kobayashi S, Ono I, and Yamamura M (1995) An efficient genetic algorithm for job shop scheduling problems. In Proceedings of the sixth international conference on Genetic Algorithms, 506–511
38. Gen M, Tsujimura Y, Kubota E (1994) Solving job-shop scheduling problem using genetic algorithms. In: Proceedings of the 16th international conference on Computer and Industrial Engineering, Ashikaga, Japan, 576–579
39. Heller J (1960) Some numerical experiments for an M×J flow shop and its decision-theoretical aspects. Oper Res 8:178–184
40. Taillard E (1993) Benchmarks for basic scheduling problems. Eur J Oper Res 64:278–285
41. Rajendran C (1994) A no-wait flowshop scheduling heuristic to minimize makespan. J Oper Res Soc 45(4):472–478
42. Picard J-C, Queyranne M (1978) The time-dependent traveling salesman problem and its application to the tardiness problem in one-machine scheduling. Oper Res 26:86–110