



Hybrid flowshop with unrelated machines, sequence-dependent setup time, availability constraints and limited buffers [☆]

Victor Yaurima ^{a,*}, Larisa Burtseva ^b, Andrei Tchernykh ^c

^a CESUES Superior Studies Center, Carretera a Sonoyta Km. 6.5, San Luis Rio Colorado, Sonora 83450, Mexico

^b Autonomous University of Baja California, Calle de la Normal, Col. Insurgentes Este, Mexicali, B.C 21270, Mexico

^c CICESE Research Center, Carretera Tijuana – Ensenada Km. 107, Ensenada, B.C 22860, Mexico

ARTICLE INFO

Article history:

Received 27 November 2007

Received in revised form 31 August 2008

Accepted 6 September 2008

Available online 13 September 2008

Keywords:

Hybrid flowshop

Scheduling

Setup time

Availability

Buffer

Genetic algorithm

ABSTRACT

This paper presents a genetic algorithm for an important production scheduling problem. Since the problem is NP-hard, we focus on suboptimal scheduling solutions for the hybrid flowshop with unrelated machines, sequence-dependent setup time, availability constraints, and limited buffers. The production environment of a television assembly line for inserting electronic components is considered. The proposed genetic algorithm is a modified and extended version of the algorithm for a problem without limited buffers. It takes into account additional limited buffer constraints and uses a new crossover operator and stopping criteria. Experimental results carried out on real production settings show an improvement in scheduling when the proposed algorithm is used.

© 2008 Elsevier Ltd. All rights reserved.

1. Introduction

Hybrid flowshop (HFS) scheduling problems arise in many practical situations. It is a generalization of the classical flowshop problem by permitting multiple parallel processors in a stage (Morita & Shio, 2005). The HFS differs from the flexible flow line (Kochhar & Morris, 1987) and the flexible flowshop (Santos, Hunsucker, & Deal, 1995) problems. In a flexible flowline as well as in a flexible flowshop, machines available in each stage are identical. The HFS does not have this restriction (Aghezzaf & Artiba, 1998; Guinet & Solomon, 1996; Gupta & Tunc, 1998; Portmann & Vignier, 1998; Riane, Artiba, & Elmaghraby, 1998).

Arthanary and Ramaswamy (1971) introduced the HFS and studied a branch and bound algorithm for two stages. Salvador (1973) presented one of the earliest works with m stages, where a dynamic programming algorithm for the no-wait flowshop with multiple processors was proposed.

The real production environment was considered in various papers. Adler et al. (1993) developed Bagpak Production Scheduling System (BPSS) to support paper bags production. It takes into ac-

count setup times, and, in some stages, unrelated parallel machines. The BPSS uses priority rules. Aghezzaf, Artiba, Moursli, and Tahon (1995) proposed several methods to solve a problem in carpet manufacturing industry. Three stages and sequence-dependent setup times were considered. The solutions are based on the problem decomposition, heuristics, and mixed-integer programming models. Gourgand, Grangeon, and Norre (1999) presented several Simulated Annealing based algorithms applied to a real industrial problem. Allaoui and Artiba (2004) dealt with the HFS scheduling problem with maintenance constraints to optimize several objectives. Setup, cleaning and transportation times were taken into consideration.

This paper addresses HFS with unrelated machines, sequence-dependent setup time, machine availability constraints, and limited buffers for a television printed circuit-board (PCB) production.

The remainder of the paper is organized as follows. Section 2 provides a short description of the PCB manufacturing process, along with a characterization of the related production scheduling problem. Section 3 describes the HFS problem. Section 4 introduces the scheduling solution encoding scheme, selection, crossover, mutation operations, restart and stopping criteria of the genetic algorithm (GA). Section 5 describes parameters of GA calibration. Section 6 introduces the proposed algorithm. Section 7 presents the experimental setup and results. Finally, Section 8 summarizes the paper.

[☆] This manuscript was processed by Area Editor Maged M. Dessouky.

* Corresponding author. Tel./fax: +52 653 5356168.

E-mail address: vyaurima@yahoo.com (V. Yaurima).

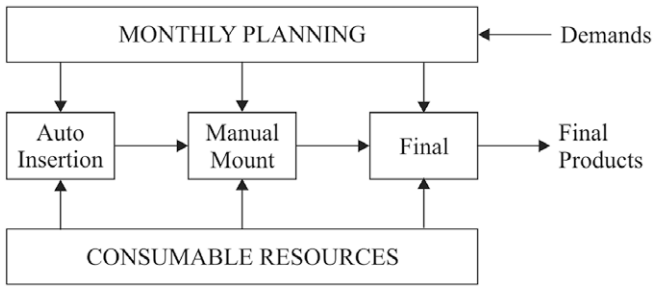


Fig. 1. PCB manufacturing process.

2. The hybrid flowshop in printed circuit-board assembly lines

The real case of the television production environment is considered. Different television models are distinguished by their set of PCBs. In the current factory setup, an assembly line contains three sections (Fig. 1):

1. Auto Insertion, where various PCB types are manufactured with automated machines.
2. Manual Mount, where big components are assembled by operators.
3. Final one, where the quality control, packing and shipping are carried out.

The monthly production plan is developed based on current requirements, machines availability and resource constrains. It is updated daily depending on the final section requirements (Fig. 1).

We address the auto-insertion section, where PCBs for 70 television models, 45 machines and production units of different brands are dealt with.

The auto-insertion section is represented by a hybrid flowshop with six stages (operations) in following order:

1. Eyelet (Ey), perforation of the orifices to insert component.
2. Jumper (Jv), insertion of jumpers to connect circuits.
3. Axial (Ax), insertion of axial components of fixed length.
4. Axial variable (Av), insertion of axial components of variable length.
5. Radial (Rd), insertion of radial components.
6. Surface Mount (Sm), soldering.

These operations are common for all PCB types. However, some PCBs do not require all six operations.

Fig. 2 shows an example of such a layout, and gives some insight on how the work flow is organized. Each stage consists of several insertion machines in parallel, and they are dedicated to the certain types of components processing. At each instant of time, each machine works on at most one PCB, and each PCB is processed by

at most one machine. The PCBs are moving along the assembly line, from one machine to another until it became a complete unit. The flow is determined by technological constraints. Machines of different brands with identical functionality but with different speeds or capabilities are included in the stage. The processing time depends on the machine brand. For instance, in stage 2 (Jv, Fig. 2), two machines (JVK) are able to perform 8500 insertions per hour, whereas three others (JVK2) 12,400 insertions, completing one PCB in 87 and 60 seconds, respectively.

We consider scheduling in the presence of machine eligibility restrictions when not all machines can process all PCBs, and machine availability restrictions when the use of machines depends on their current state: active or in maintenance service.

Adjustment of the machine and the preparation of its feeder are required when the board type is changed. The feeders have different capacities (number of slots). For example, machines could have 60 slots or 80 slots. The time needed for adjustment essentially depends on the board type previously processed in the machine. It cannot be neglected in the television PCB production environment. Hence, a sequence-dependent setup time is needed.

Each machine has a limited capacity buffer for storing “work in process” (WIP). If the storage is filled to full capacity, the production on this machine is blocked.

The problem is modeled as a HFS with the following constraints: (1) From two to six successive stages with the common flow pattern for all PCB types; (2) Stages with unrelated machines; (3) Machines eligibility/availability; (4) Sequence-dependent setup time; (5) limited buffers. The goal is to find a schedule that minimizes the total production time.

3. Problem statement

Let a set N of n jobs, $N = \{1, 2, \dots, n\}$ given at time 0 has to be processed in a set M of m consecutive production stages, $M = \{1, 2, \dots, m\}$, without preemption, with the objective of minimizing the total completion time. In stage $i \in M$, a set $M_i = \{1, 2, \dots, m_i\}$ of unrelated parallel machines is given, where $|M_i| \geq 1$. Each job has to be processed by exactly one machine in each stage. Let $p_{i,l,j}$ be the processing time of job $j \in N$, at machine $l \in M_i$, in stage i . A machine based sequence-dependent setup time is considered. Let $S_{i,l,j,k}$ be the setup time at machine l , in stage i , when processing job $k \in N$, after processing job j . We denote a set of eligible machines that can process job j , in stage i , as E_{ij} , $1 \leq |E_{ij}| \leq m_i$. For each machine $l \in M_i$ a limited buffer for jobs is given. A maximal storage capacity in front of each machine l is $b_{i,l}$, where $1 \leq |b_{i,l}| \leq n$.

Gourgand et al. (1999) showed that for a given problem the total number of possible solutions is $n!(\prod_{i=1}^m m_i)^n$. Moreover, Gupta and Tunc (1998) proved that the flexible flowshop problem with $m = 2$ is NP-hard even if one of two stages contains a single machine. Since HFS is a general case of the flexible flowshop, HFS is also NP-hard.

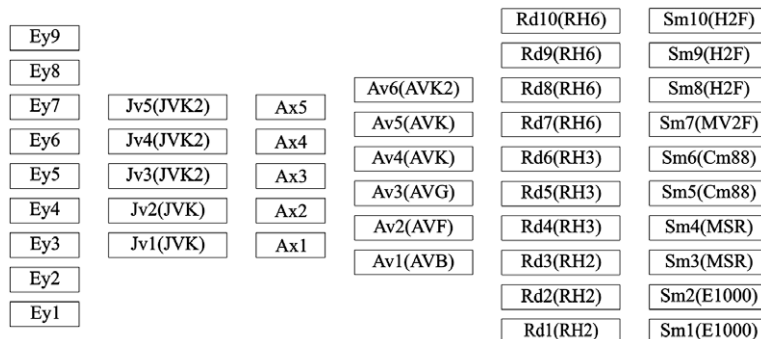


Fig. 2. An example of machines layout.

Using the well-known three field notation $\alpha | \beta | \gamma$ for scheduling problems and its extension for HFS proposed by Vignier, Billaut, and Proust (1999), the problem is denoted as:

$$FHm, ((RM^{(i)})_{i=1}^{(m)} | S_{sd}, M_j, Block | C_{max}).$$

Detailed surveys of the research on the problem are given by Vignier et al. (1999), Linn and Zhang (1999), and Ruiz and Maroto (2006). Quadri and Kuhn (2007) presented taxonomy for flexible flowline scheduling procedures. Many papers address simplified cases of the HFS with two and three stages. Seven studies about unrelated parallel machines with multiples stages are known (Adler et al., 1993; Aghezzaf et al., 1995; Allaoui & Artiba, 2004; Gourgand et al., 1999; Ruiz & Maroto, 2006; Ruiz, Şerifoğlu, & Urlings, 2008; Zandieh, Fatemi Ghomi, & Moattar Hussein, 2006). Four of them take into account sequence-dependent setup times (Aghezzaf et al., 1995; Ruiz & Maroto, 2006; Ruiz et al., 2008; Zandieh et al., 2006). Ruiz and Maroto (2006) and Ruiz et al. (2008) considered mentioned constraints and availability constraints together inside the same problem formulation. The availability constraint adds a new dimension to the scheduling problem (Lee, 2004). One can refer to Schmidt, 2000 for some results in this field.

Tang and Zhang (2005) considered the HFS scheduling problem with job dependent setup time without availability constraints. In the model, all jobs pass the same route, and at least one stage has more than two machines. A modification of the traditional Hopfield network formulation and the improved strategy are proposed. Zandieh et al. (2006) proposed an immune algorithm for the considered problem, where identical machines are considered in each stage. Obtained results are compared with those of Random Key Genetic Algorithm (RKGA) (Ryan, Atif, Azad, & Ryan, 2004). It was shown that the immune algorithm outperforms RKGA. Jin, Ohno, Ito, and Elmaghraby (2002) considered three-stage HFS for the production of printed circuit boards. Several types of PCBs are scheduled without preemption. Each type has to be assembled serially through three stages of identical machines. An off-line problem is considered, and the setup time between different types of PCBs is neglected. The objective is to find a schedule minimizing the makespan.

Ruiz and Maroto (2006) developed a genetic algorithm to a complex generalized flowshop scheduling problem applied to the production of textiles and ceramic tiles that takes into account unrelated parallel machines in each stage, sequence-dependent setup times, and availability constraints. It was shown that the proposed algorithm is more effective and efficient than known ones (Aldowaisan & Allahvedi, 2003; Chen, Vempati, & Aljaber, 1995; Murata, Ishibuchi, & Tanaka, 1996; Nawaz, Ensore, & Ham, 1983; Osman & Potts, 1989; Rajendran & Ziegler, 2004; Reeves, 1995; Widmer & Hertz, 1989). Recently, Ruiz et al. (2008) introduced a mixed-integer programming model and heuristics for a complex and realistic flowshop problem. Several factors are jointly considered: release dates for machines, existence of unrelated parallel machines in each stage, machine eligibility, sequence-dependent setup times, positive/negative time lags between operations, and generalized precedence relationships between jobs. Yaurima et al. (2008) proposed an algorithm for solving similar problem in real industry environment of the televisions production. The experimental results obtained on the benchmark data set show the advantages of using new crossover operator, yielding high quality solutions.

Papadimitriou and Kanellakis (1980) presented one of the first results related to the limited buffers constraints in the flowshop scheduling problem. The authors proved NP-completeness of two-machine problem with limited intermediate storage buffers. Weng (2000) presented a detailed analysis of earliest results of problems with storage buffers. Different aspects of the down-

stream machine blocking in consequence of the limited buffer overflow are considered. Norman (1999), Sawik (2000, 2002), and Wang et al. (2006) considered the makespan minimization problems in such systems. Weng (2000) studied the optimal buffer capacity for machines. Simulation results show that for up to 100 jobs and 20 machines the optimal buffer size is no more than 4. It is found that the buffer size is more significant with increasing the number of jobs. Witt and Voï (2007) presented three heuristics to control the work in process. Different consumptions of space by the jobs inside the limited storage are considered.

Heuristics rules (Witt & Voï, 2007), mixed-integer programming (Sawik, 2000, 2002), tabu search and simulated annealing (Ishibuchi, Misaki, & Tanaka, 1995; Weng, 2000) are applied to solve the makespan minimization problem taking into account limited buffers. Wang, Zhang, and Zheng (2006) proposed the use of a hybrid genetic algorithm. Genetic operators are combined with a local search to enhance the solution quality and performance. A neighborhood structure is based on the graph model.

Norman (1999) studied more complex problem that combines finite buffers and sequence-dependent setup times. A tabu search based algorithm is proposed to find the solution.

4. Genetic algorithm

A genetic algorithm (GA) is a well-known search technique used to find solutions to optimization problems. It was proposed by Holland (1975). Candidate solutions are encoded by chromosomes (also called genomes or individuals). The set of initial individuals forms the population. Fitness values are defined over the individuals and measures the *quality* of the represented solution. The genomes are evolved through the genetic operators generation by generation to find optimal or near-optimal solutions. Three genetic operators are repeatedly applied: selection, crossover, and mutation. The selection picks chromosomes to mate and produce offspring. The crossover combines two selected chromosomes to generate next generation chromosomes. The mutation reorganizes the structure of genes in a chromosome randomly so that a new combination of genes may appear in the next generation. The individuals evolve until some stopping criterion is met.

Many of the authors separate sequencing and assignment decisions in the HFS problems (Rajendran & Chaudhuri, 1992; Sherali, Sarin, & Kodialam, 1990). We follow the way proposed by Ruiz and Maroto (2006), where the assignment of jobs to machines in each stage is done by the evaluation function. In the HFS with no setup times and no availability constraints assignment of the job to the first available machine would result in the earliest completion time of the job. In the HFS with unrelated parallel machines it is demonstrated that if the first available machine is very slow for a given job, assigning the job to this machine can result in a later completion time compared with assignment to other machines. With the consideration of the setup times this problem becomes worse. To solve it, in our algorithm, a job is assigned to the machine that can finish the job at the earliest time at a given stage, taking into consideration different processing speeds, setup times, machine availability and buffer size.

4.1. Encoding

Each individual (candidate scheduling solution) is encoded as a string (permutation) of n integers, where each integer represents a job. Value j at position i in the chromosome means that job j is at position i in the job solution sequence.

The calculation of the total completion time C_{max} is as follow: Let π be a job permutation or sequence; $\pi_{(j)}$ be the job at the j th position in the sequence, $j \in N$. Each job has to be processed at each

stage, so m tasks per job are considered. Let L_i be the last job assigned to machine l in stage i , $l \in M_i$. Let $S_{i,L_i,\pi_{(j)}}$ be the setup time of machine l in stage i when processing job $\pi_{(j)}$ after having processed the previous work assigned to this machine l (L_i). Let $C_{i,\pi_{(j)}}$ be the completion time of job $\pi_{(j)}$ in stage i , $i \in M$.

$$C_{i,\pi_{(j)}} = \min_{l=1}^{m_j} \{ \max \{ C_{i,L_i} + S_{i,L_i,\pi_{(j)}}; C_{i-1,\pi_{(j)}} \} + p_{i,\pi_{(j)}} \},$$

The makespan is calculated as follows:

$$C_{\max} = \max_{j=1}^n \{ C_{m,\pi_{(j)}} \}$$

4.2. Selection, crossover and mutation

The binary tournament selection, known as an effective variant of the parents' selection is considered. Two individuals are drawn randomly from the population, and the more fit of the two "wins" the tournament. This process is repeated twice in order to generate two parents.

A mutation is incorporated into genetic algorithms to avoid convergence to local optimum, to reintroduce lost genetic material and variability in the population. Three mutation operators (insert, swap and switch) widely used in the literature are considered (Gen & Cheng, 1997; Michalewicz, 1996; Ruiz & Maroto, 2006).

A crossover generates new sequences by combining two other sequences. The goal is to generate new solutions with better C_{\max} values. Five operators known in the literature (OBX, PPX, OSX, TP, SB2OX) together with three new operators (ST2PX, TPI, OBSTX) are considered.

OBX – Order Based Crossover (Gen & Cheng, 1997). This crossover operator is based on a binary mask. The mask values equal to one indicate that the corresponding sequence elements are copied from parent 1 to the child. The rest of elements are copied from parent 2. The mask values are generated randomly and uniformly in all crossover operations.

PPX – Precedence Preservative Crossover (Bierwirth, Mattfeld, & Kopfer, 1996). It is a binary mask based. The mask values equal to one indicates that corresponding sequence elements are copied from parent 1 to the child, and the values equal to zero indicate that elements are copied from parent 2.

OSX – One Segment Crossover (Allaoui & Artiba, 2004; Guinet & Solomon, 1996). It chooses two points randomly. Elements from position 1 to the first point are copied from parent 1. Elements from first point to second point are copied from parent 2. Finally, from the second point to last element, are copied from parent 1, considering not copied elements.

TP – Two Point (Michalewicz, 1996). It chooses two crossover points randomly. Elements from position 1 to the first point and from the second point to the last position are copied from parent 1. The elements from the first point to second point are copied from parent 2.

SB2OX – Similar Block 2-Point Order Crossover (Ruiz & Maroto, 2006). The common blocks (at least two consecutive jobs) of the parents are copied to the offspring; then two random cut points are drawn and the section between these two points is directly copied to the offspring. The missing elements of the offspring are copied in the relative order from the parents.

Three new crossover operators are proposed: TPI, OBSTX, ST2PX.

TPI (Two Point Inverse). It chooses two points randomly. Elements from position 1 to the first point are copied from parent 2. The positions from the first point to the second one are copied from parent 1, unlike to TP where the elements are copied from parent 2. The positions from the second point to last one are copied from parent 2, unlike to TP where the elements are copied from parent 1.

OBSTX (Order based Setup Time Crossover). It comes from the original OBX crossover taking into consideration sequence-dependent setup times according to the binary mask. The value one of the mask indicates that the corresponding element of the parent is copied to the child. The mask with value zero indicates that the element of parent 2 is copied according the minimal sequence-dependent setup time of the first stage machine chosen randomly.

ST2PX (Setup Time Two Point Crossover). It takes into consideration the sequence-dependent setup time. It chooses two crossover points randomly. Elements from the positions 1 to the first point are copied from parent 1. Elements from the second point to last one are copied from parent 1. Elements from the first point to second one are copied from parent 2 according to the minimal sequence-dependent setup time of the first stage machine chosen randomly (see Algorithm 2, chap. 6).

4.3. Restart and stopping criterion

In order to escape from local optima, it is standard practice to periodically restart genetic algorithms and reinitialize the population according to some restart policy. Alcaraz, Maroto, and Ruiz (2003) proposed the following scheme: keep 20% best individuals, replace first 40% by simple Insert mutations of the one chosen randomly from first 20%, and replace reminding (worst) 40% by randomly generated individuals. Reeves (1995) shown that a steady state genetic algorithm, where the worst individuals in the offspring are replaced, yields much better results than regular genetic algorithms.

Our restart policy introduced to improve quality of the solutions is a simple modification of the second step: replaces first 40% individuals by simple Insert mutations of the best individual.

There are few guidelines for determining when to terminate the search. One of the criteria is to stop when the fitness value is not improved over the certain number of generations. The termination condition for our genetic algorithm is 25 iterations without improvement of the makespan, and 10 restarts. This condition leads to good quality solutions and reasonable computation times.

5. GA parameters calibration

A method of experimental design is adapted from Ruiz and Maroto (2006), where the following steps are defined: (a) test all instances produced with possible combinations of parameters; (b) obtain the best solution for each instance; (c) apply the Multi-factor Variance Analysis (ANOVA) with 95% confidence level to find the most influential parameters; (d) set algorithm parameters based on selected parameters values; (e) calculate relative difference of the calibrated algorithm and other adapted algorithms over the best solutions.

The following parameters were set for the calibration: Population size: 100, 150 and 200; Crossover operators: OBX, PPX, OSX, TP, ST2PX, TPI, OBSTX, SB2OX; Crossover probability (P_c): 0.5, 0.6, 0.7, 0.8, 0.9; Mutation operators: Insert, Swap, Switch; Mutation probability (P_m): 0.03, 0.05, 0.1, 0.2, 0.4. Hence, $3 * 8 * 5 * 3 * 5 = 1800$ different setups were considered to calibrate GA_{SBC} algorithm.

The following parameters that correspond to real-life television production settings were considered. The number of stages was set to 2, 3, and 6. Number of machines was uniformly distributed between 1 and 10 machines per stage. Number of jobs (PCB lots) in the range from 50 to 100 was considered. The job processing times were uniformly distributed in the range of between 50 and 99.

The workloads were based on the instances studied by Taillard (1993) for flowshops, and augmented by Ruiz and Maroto (2006) for hybrid flowshops. Sixty instances that include 10 loads for each of 6 groups (2, 3 and 6 stages with 50 and 100 jobs, respectively) were generated and 108,000 experiments were conducted (1800 different algorithms alternatives for each instance).

The sequence-dependent setup times were drawn from the uniform distribution $U[25, 50]$ that corresponds to the range of 25–50% of the processing times. The probability of occurring the situation when machine l in stage i is not available or eligible for processing job j was set to 0.25. The size of the limited buffers were set in the range of $U[25, 50]$: 25–50% of the number of jobs for each machine.

The largest setup was as follows: 100 jobs, 6 stages, with 10 machines per stage. Thus, 100×60 matrix of processing times, 60 matrices of 100×100 of the setup times, and a vector of buffers for 60 machines were used.

The performance measure of the proposed algorithm was calculated as the percentage of the relative distance from the obtained solution to the best one:

$$\frac{Heu_{sol} - Best_{sol}}{Best_{sol}} \times 100$$

where Heu_{sol} is the value of the objective function obtained by considered algorithm, and $Best_{sol}$ is the best value obtained during the testing all possible parameter combinations.

All experiments were performed on a PC with Pentium 4 2.8 GHz processor, 512 Mbytes RAM, and Windows XP operating system.

To assess the statistical difference among the experimental results, to observe effect of different parameters on the result quality, the ANOVA technique was applied. The analysis of variance was used to determine do any of the factors have a significant effect, and which are the most important factors. Parameters of the HFS problem were considered as factors, and their values as levels. We assume that there is no interaction between the factors.

Figs. 3–7 show means and 95% LSD intervals of the most influential factors using the following setup: 30 instances, 100 jobs, 2, 3 and 6 stages.

Fig. 3 shows the results obtained for crossover operators. We can see that our crossover ST2PX is the best crossover among the eight ones tested. This is due to the fact that the ST2PX takes into account the sequence-dependent setup time, which is an important factor of the job assignment to unrelated parallel machines. Fig. 4 presents plots for the mutation operations, where the Swap is shown to be the best. Fig. 5 shows the results for the crossover probability. It can be seen that the best probability of crossover occurring is 0.8. Fig. 6 shows plots for the mutation probability. The best probability of mutation occurring is 0.1. Fig. 7 presents plots for the population size. The population of 200 individuals is statistically more significant.

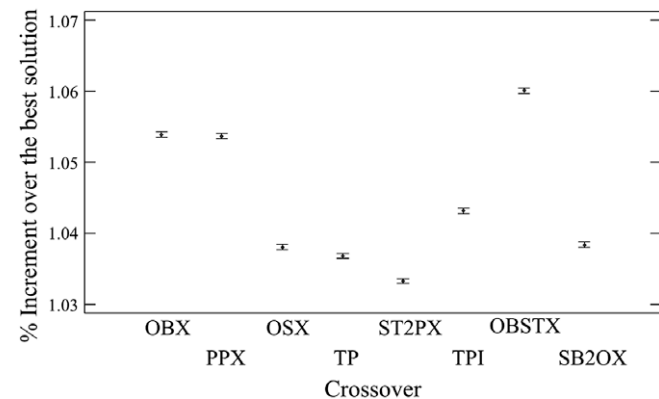


Fig. 3. Means and 95% LSD intervals of crossover operations.

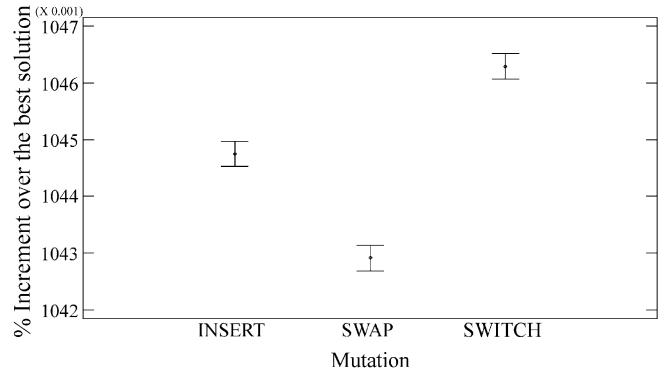


Fig. 4. Means and 95% LSD intervals of mutation operations.

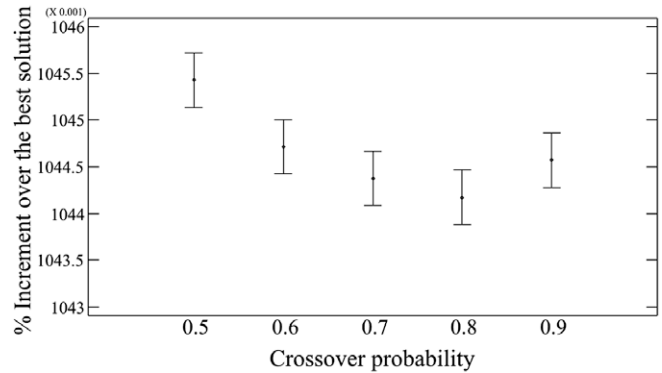


Fig. 5. Means and 95% LSD intervals of crossover probability.

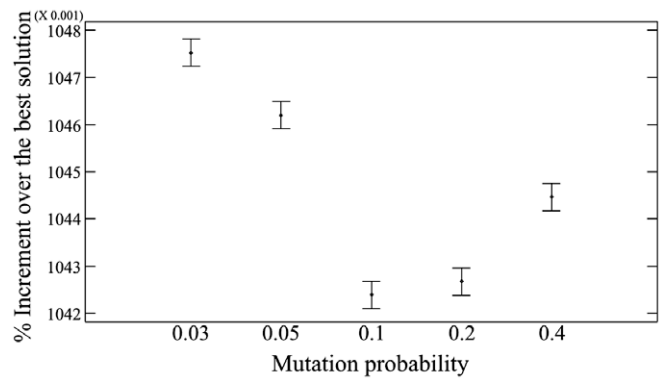


Fig. 6. Means and 95% LSD intervals of mutation probability.

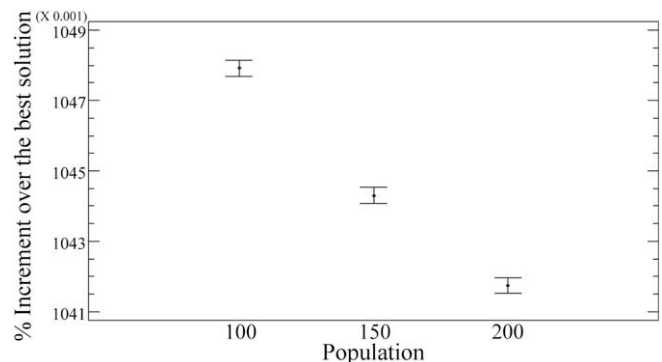


Fig. 7. Means and 95% LSD intervals of population size.

6. Proposed genetic algorithm GA_{SBC}

Our genetic algorithm, GA_{SBC} , presented in this section is tuned up by the following parameters obtained during the calibration step: crossover: ST2PX; mutation: Swap; crossover probability: 0.8; mutation probability: 0.1; population size: 200.

The execution steps of GA_{SBC} are presented in Algorithm 1.

Algorithm 1. GA_{SBC}

```

Input: The population of  $P_{size}$  individuals.
Output: An individual of length  $n$ .
01. generate_population
02. regeneration = 1
03. while not stopping_criterion do
04.   for  $i = 0$  to  $P_{size}$ 
05.     evaluate_objective_function(i)
06.   keep_the_best_individual_found()
07.   if actual_best_makespan >= previous_best_makespan
08.     iterations_without_improvement = iterations_without_
improvement + 1
09.   if iterations_without_improvement = 25
10.     if regeneration = 10
11.       stopping_criterion = true
12.     else
13.       sort_the_population_in_ascending_order_of_Cmax()
14.       regenerate_population()
15.       regeneration = regeneration + 1
16.       iterations_without_improvement = 0
17.   select_individuals_by_the_binary_tournament_selection
18.   crossover ST2PX with probability 0.8
19.   mutation SWAP with probability 0.1
    
```

Algorithm 2. ST2PX Crossover

```

Input: Two individuals ( $parent_1, parent_2$ )
Output: One individual ( $child$ )
01. set the first_point and second_point randomly
02. for  $i = 1$  to first_point
03.   do copy element  $i$  from  $parent_1$  to  $child$ 
07. for  $j = second\_point$  to last position
08.   do copy element  $j$  from  $parent_1$  to  $child$ 
04. for  $k = first\_point$  to  $second\_point$ 
05.   do choose the machine randomly (from the machines of the
first stage)
06.   do pick up the best position (from the first_point to
second_point) of parent 2 for position  $k$  of the child
according to the sequence-dependent setup time of the
chosen machine.
    
```

Table 1
A set of eligible machines that can process job j in stage i

Stage i	1	2	3
Job j	1	{1}	{1}
	2	{2}	{1,2}
	3	{1,2}	{1,2}
	4	{1,2}	{2}
	5	{1,2}	{1,2}
	6	{1}	{2}
	7	{2}	{2}

Table 2
The processing time $P_{i,l,j}$ of job j , at machine l , in stage i

Stage i	1	1	2	2	3
Machine l	1	2	1	2	1
Job j	1	54	-1	69	-1
	2	-1	76	75	67
	3	58	93	51	82
	4	59	95	-1	52
	5	75	62	58	73
	6	50	-1	-1	52
	7	-1	57	-1	66

Table 3
Sequence-dependent setup times for the first machine

Job k	1	2	3	4	5	6	7
Job j	1	0	41	50	28	27	29
	2	38	0	25	38	47	48
	3	29	35	0	38	25	29
	4	42	26	37	0	26	33
	5	28	45	47	31	0	47
	6	36	29	27	44	31	0
	7	42	28	49	49	32	49

Table 4
Limited buffer

Stage i	1	1	2	2	3
Machine l	1	2	1	2	1
$b_{i,l}$	2	2	3	2	3

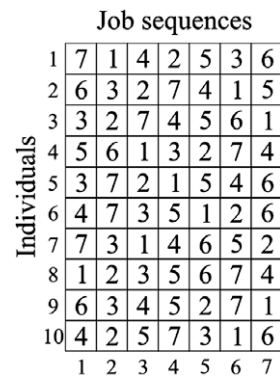


Fig. 8. Initial population.

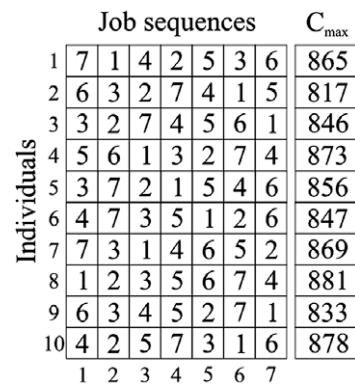


Fig. 9. Fitness value of each individual.

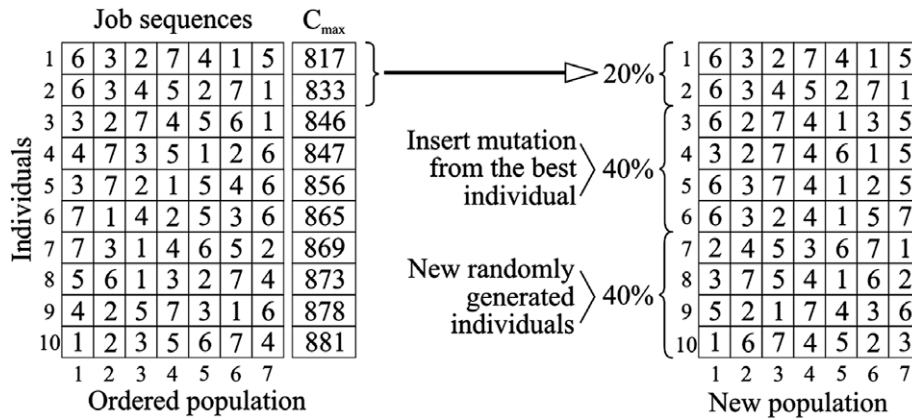


Fig. 10. Regeneration procedure.

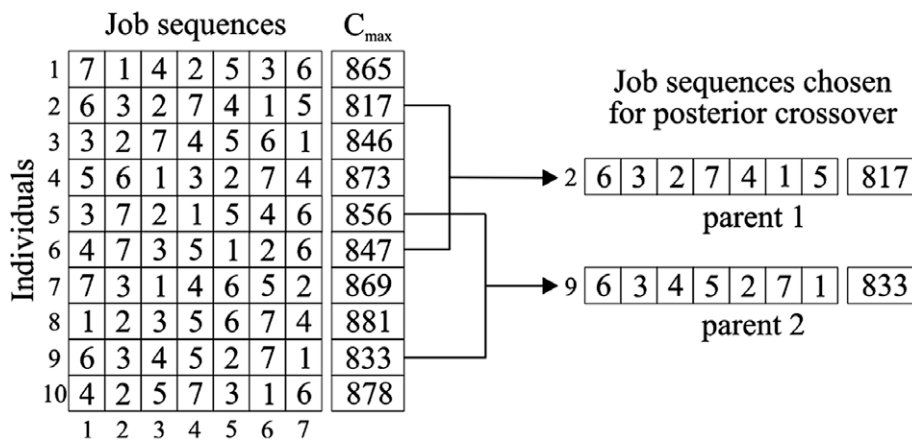
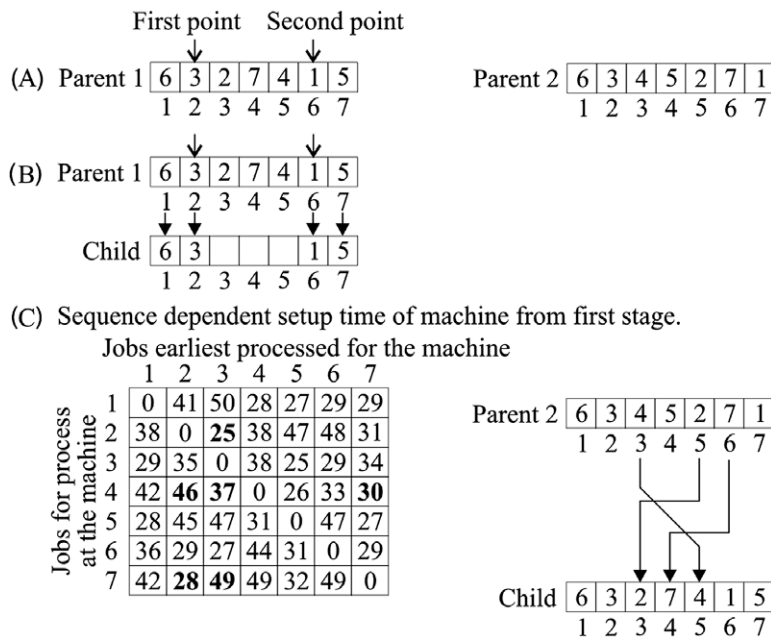


Fig. 11. Binary selection.



After job 3: $\min[37(4), 25(2), 49(7)] = 25$, then job 2 is chosen.
 After job 2: $\min[46(4), 28(7)] = 28$, then job 7 is chosen.
 After job 7: $\min[30(4)] = 30$, finally job 4 is copied.

Fig. 12. ST2PX crossover.

The following example illustrates the use of our algorithm for the HFS problem solution. Let us consider an instance with $n = 7$, $m = 3$, $m_1 = m_2 = 2$, and $m_3 = 1$. Let Table 1 sets up eligibility, and Table 2 processing times. The number -1 means that the machine l is not eligible or not available for the job j . Table 3 shows sequence-dependent setup times of job k if job j precedes job k . Table 4 shows the limited buffer sizes.

Let a population with 10 individuals is generated (Fig. 8). Fig. 9 presents the fitness value of each individual. The best solution is represented by the individual 2 with makespan 817.

The population is ordered and regenerated: 20% best individuals are kept, 40% are replaced by simple Insert mutation of the best individual, and reminding worst 40% are replaced by randomly generated individuals. Fig. 10 shows the regeneration result.

Fig. 11 shows results of the binary selection. The ST2PX crossover is applied with a probability of 0.8 (Fig. 12). Let us assume that the first point is at position 2, and the second point is at position 6 (Fig. 12A). Elements from position 1 to position 2 of parent 1 are copied to the child. Elements from position 6 to position 7 (last position) are copied from parent 1 (Fig. 12B). The remaining positions of the child are filled with best elements from parent 2, taking into

account the sequence-dependent setup times (Fig. 12C). Three jobs (4, 2 and 7) can be processed at position 3 after processing job 3 at position 2. Hence, three setup times (37, 25, 49) are compared, and job 2 with minimal setup time 25 is chosen. Two setup times (46 and 28) are compared for position 4, and job 7 is chosen. The last job (4) is copied to position 5. Finally, the SWAP mutation is applied with a probability 0.1 (Fig. 13). Fig. 14 shows the Gantt chart of the final result.

7. Computational experiments

Up to date, no algorithm was proposed for a given problem. Ruiz and Maroto (2006) proposed a genetic algorithm GA_H for solving a similar problem without limited buffers. It was compared with nine variants of metaheuristic methods (NEH heuristic, simulated annealing, tabu search, ant-based algorithms, etc.) and showed to be the best for solving the problem without limited buffers.

To compare our GA_{SBC} algorithm we implemented the adaptation of the GA_H to a limited buffers version. We refer to the adapted algorithm as GA_{HBC} .

Figs. 15 and 16 show the percentages of the relative distance of the GA_{SBC} and GA_{HBC} performance to the best solutions obtained for 50 and 100 jobs. It can be seen that GA_{SBC} outperforms GA_{HBC} in all experiments.

Table 5 presents the mean of relative distance of the GA_{SBC} and GA_{HBC} to the best results. GA_{SBC} improves GA_{HBC} in all experiments by a margin of between 99% and 378% depending on the case.

Table 6 shows the mean of relative distance of the GA_{SBC} to GA_{HBC} in terms of the solution quality of the best results. We can see that the GA_{SBC} outperforms the GA_{HBC} in all experiments by a range of between 2.58% and 4.60%.

Table 7 shows the number of times the best solutions (out of 10) is found by GA_{SBC} and GA_{HBC} . It can be seen that GA_{HBC} was not able to find best solutions in all cases.

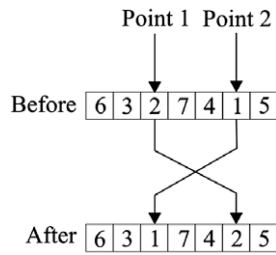


Fig. 13. SWAP mutation.

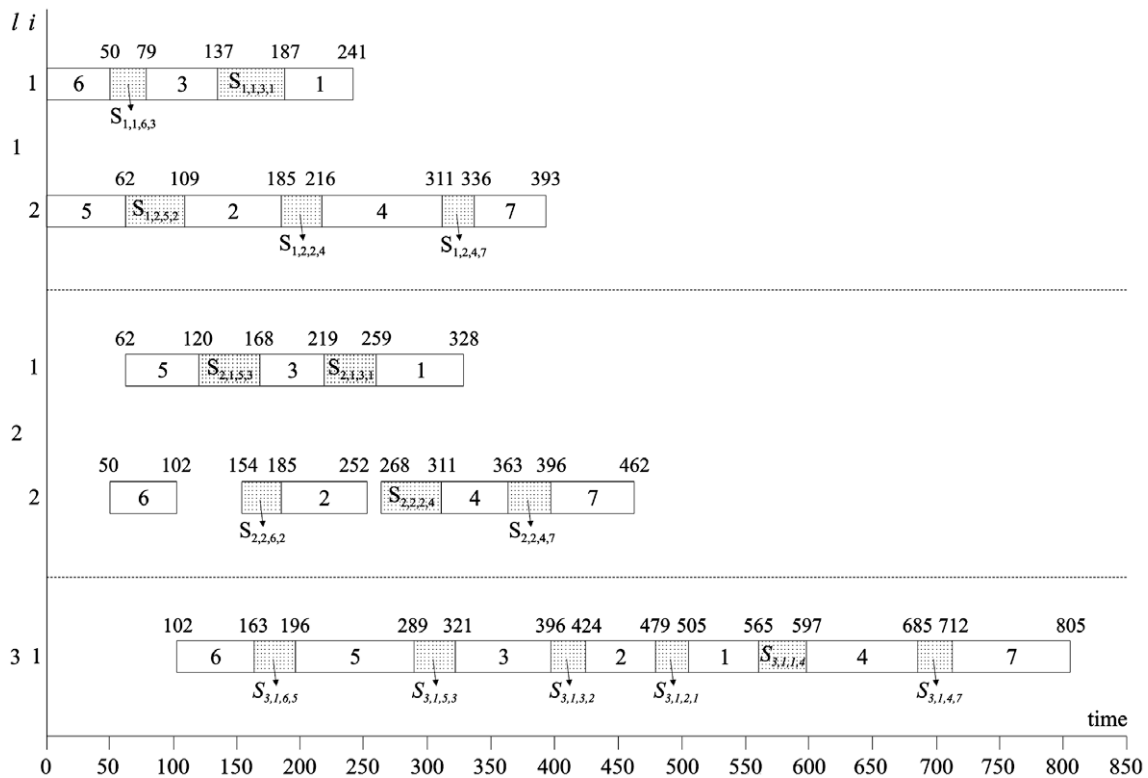


Fig. 14. Gantt chart for the problem solution ($C_{max} = 805$).

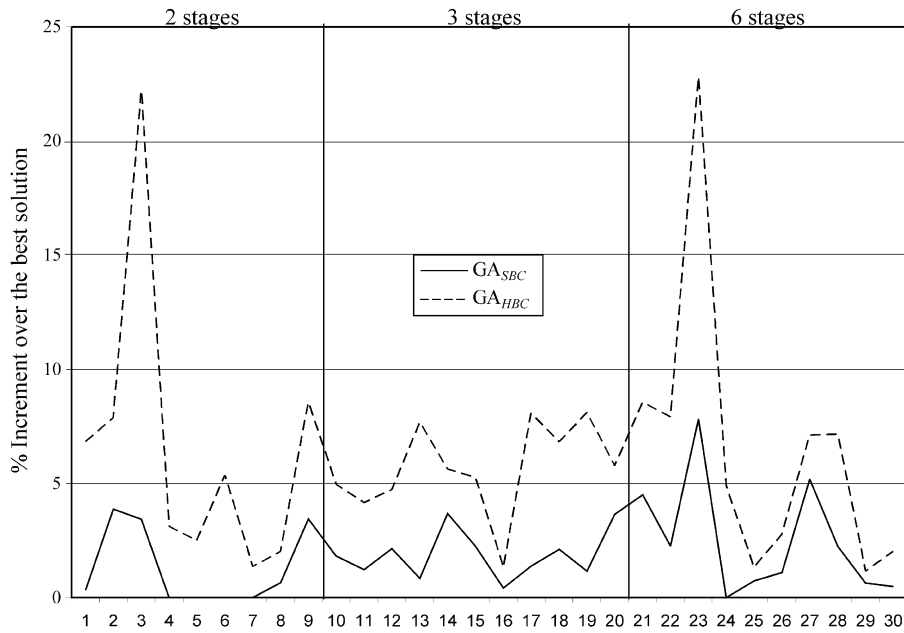


Fig. 15. Percentages of relative distance to the best solutions, 50 jobs.

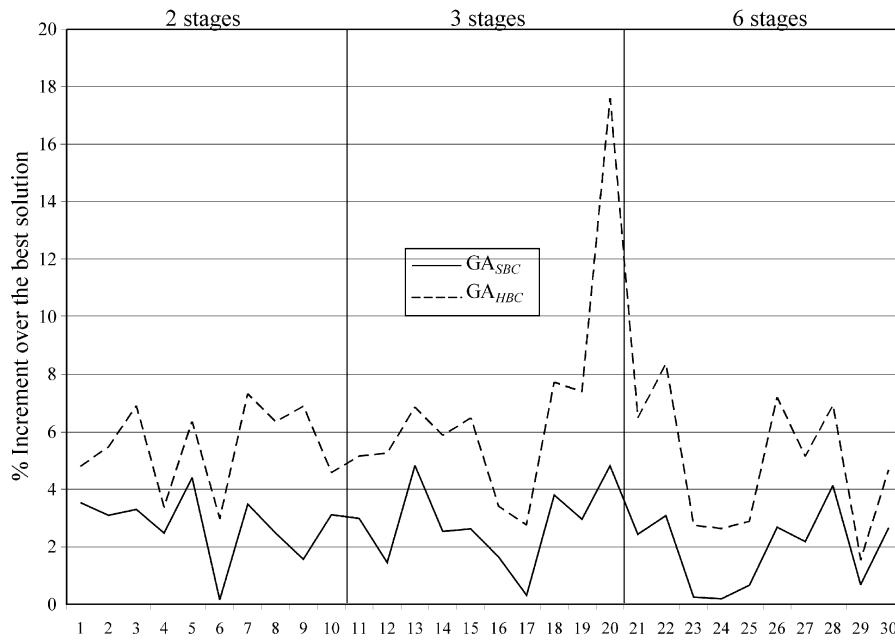


Fig. 16. Percentages of relative distance to the best solutions, 100 jobs.

The reason for success of the proposed algorithm is conjectured to be the new crossover operator and novel regeneration scheme.

CPU times of the algorithms are presented in Figs. 17 and 18. It can be seen that GA_{SBC} is slightly slower than GA_{HBC} , however, the computation time is not significantly high.

Table 5
Average percentages of the relative distance to the best solutions

50 jobs instances			100 jobs instances		
Instance	GA_{SBC}	GA_{HBC}	Instance	GA_{SBC}	GA_{HBC}
50 × 2	1.3557	6.4789	100 × 2	2.7658	5.5054
50 × 3	1.8877	5.7557	100 × 3	2.8016	6.8553
50 × 6	2.4963	6.5712	100 × 6	1.8996	4.8614
Average	1.9132	6.2686	Average	2.489	5.7407

Table 6
Average percentage of the relative distance to the GA_{HBC} algorithm

Instance	%	Instance	%
50 × 2	-4.60	100 × 2	-2.58
50 × 3	-3.62	100 × 3	-3.70
50 × 6	-3.64	100 × 6	-2.80
Average	-3.95	Average	-3.03

Table 7
The number of best solutions found (out of 10)

50 jobs instances			100 jobs instances		
Instance	GA _{SBC}	GA _{HBC}	Instance	GA _{SBC}	GA _{HBC}
50 × 2	4	0	100 × 2	0	0
50 × 3	0	0	100 × 3	0	0
50 × 6	1	0	100 × 6	0	0

Another series of experiments were performed in order to observe the behavior of the algorithms for a given run time. The aim of such a slight modification of the stopping criterion is to study how the quality of the solutions depends on CPU time. Four

fixed CPU times were considered based on CPU time of the previous experiments: minimum CPU time of algorithms reduced by 30%; minimum CPU time; maximum CPU time; maximum CPU time increased by 30%.

Table 8 shows the advantage of the GA_{SBC} performance when both algorithms take the same CPU time. Moreover, we can see how as the time increases the GA_{SBC} obtains better results, whereas GA_{HBC} does not show the significantly high improvement. Table 8 also shows the number of times the best solution is found (out of 10) in the same time intervals. The number returned by the GA_{SBC} is greater than the one returned by the GA_{HBC} in all non-zero cases.

The above results are further confirmed by an analysis of variance. Fig. 19 plots the means and 95% LSD intervals of GA_{SBC} and

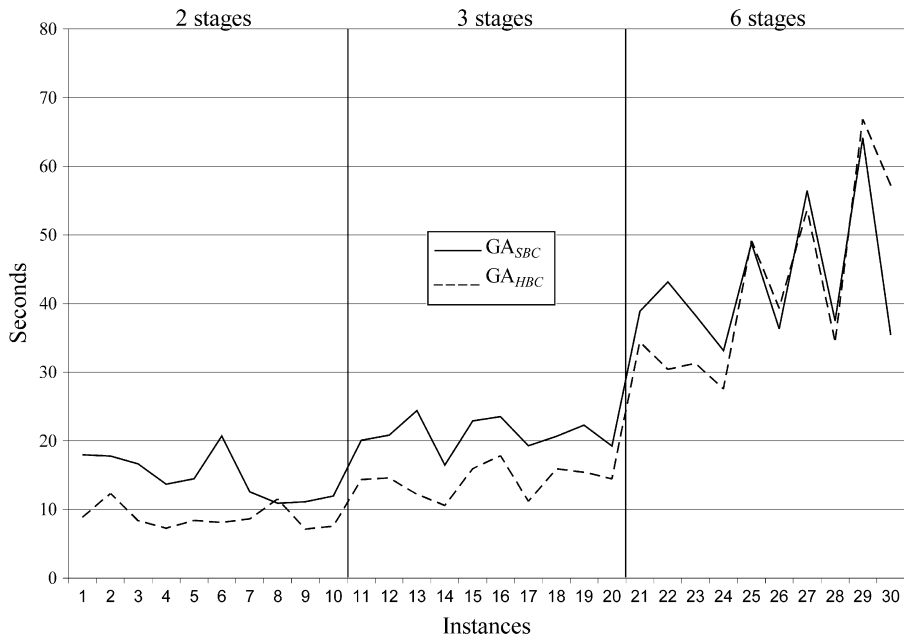


Fig. 17. CPU time, 50 jobs.

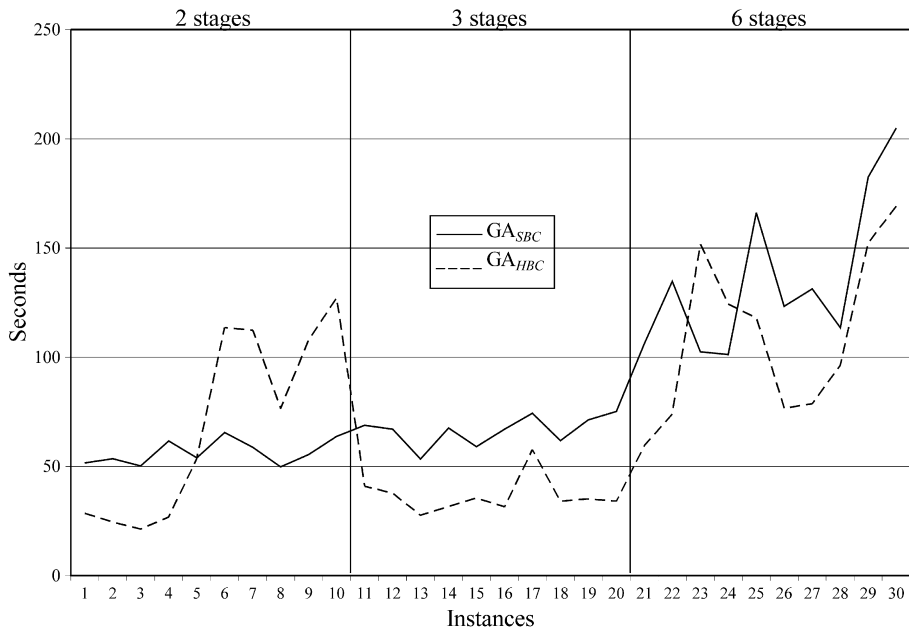


Fig. 18. CPU time, 100 jobs.

Table 8

Average percentages of the relative distance to the best solutions obtained with the same CPU time and number of best solutions found

Instance	Time (s)	Average percentages		Number of best solutions found	
		GA _{SBC}	GA _{HBC}	GA _{SBC}	GA _{HBC}
50 × 2	5.5	2.39	6.15	2	0
	7.1	2.12	6.79	2	0
	20.7	1.09	5.36	3	1
	26.9	0.51	5.79	4	0
50 × 3	8.1	3.04	6.58	0	0
	10.6	2.66	6.98	0	0
	24.4	1.84	6.2	0	0
	31.7	1.16	6.16	1	0
50 × 6	21.2	2.96	6.59	1	0
	27.5	2.59	6.68	1	0
	66.8	2.82	5.26	1	0
	86.9	2.88	5.67	1	0
100 × 2	16.4	4.07	5.43	0	0
	21.3	3.32	5.86	0	0
	127.1	1.21	4.63	0	0
	165.2	0.81	4.7	3	0
100 × 3	21.2	4.31	7.09	0	0
	27.6	3.6	7.02	0	0
	75.2	1.54	6.63	0	0
	97.8	2.04	6.34	1	0
100 × 6	45.8	2.97	5.68	0	0
	59.6	2.4	5.02	0	0
	204.9	1.39	4.03	1	0
	266.3	1.18	3.67	0	0

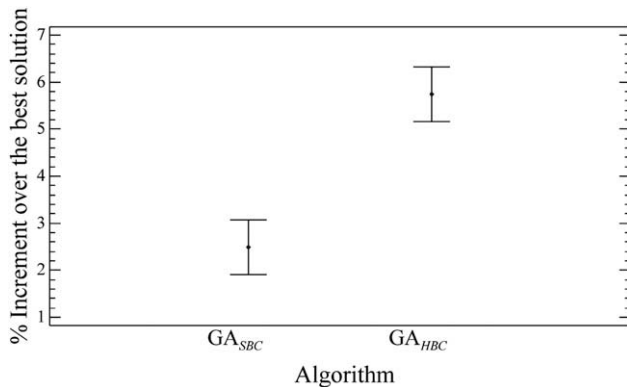


Fig. 19. The means and 95.0% LSD intervals of GA_{SBC} and GA_{HBC} (100 jobs).

GA_{HBC}. The tested algorithms are statistically different. Our algorithm GA_{SBC} is statistically better than GA_{HBC}.

To conclude the above analysis we state that our algorithm behaves quite well and may be useful in practice.

8. Conclusions

We presented an efficient genetic algorithm GA_{SBC} that solves the HFS with sequence-dependent setup time, unrelated parallel machines, machine availability, and limited buffers constraints together inside the same problem formulation. This complex problem is common in real-life industry environment of the printed circuit-board assembly line for inserting electronic components. A new crossover operator and stopping criterion were introduced to improve the solution quality. The algorithm was calibrated by means of extensive experiments. The ANOVA technique was used. With 95% confidence level important parameters and parameters' values that have a significant effect were determined. GA_{SBC} was tuned according to the calibration results.

Computational experiments were performed to compare GA_{SBC} with the best algorithm reported in the literature. Experimental results carried out on real-life settings show that the proposed algorithm yields high quality solutions. The algorithm is better than the known one in all experiments.

Acknowledgements

The authors thank the anonymous referees whose valuable remarks and comments helped to improve the paper. This work is partly supported by CONACYT (Consejo Nacional de Ciencia y Tecnología de México) under Grant No. 48385, UABC (Autonomous University of Baja California, Mexico) under Grant No. 2389, and PROMEP CESUES-020.

References

- Adler, L., Fraiman, N., Kobacker, E., Pinedo, M., Plotnecoff, J., & Wu, T. P. (1993). BPSS: A scheduling support system for the packaging industry. *Operations Research*, 41(4), 641–648.
- Aghezzaf, E., & Artiba, A. (1998). Aggregate planning in hybrid flowshops. *International Journal of Production Research*, 36(9), 2463–2477.
- Aghezzaf, E., Artiba, A., Moursli, O., & Tahon, C. (1995). Hybrid flowshop problems, a decomposition based heuristic approach. In *Proceedings of the international conference on industrial engineering and production management, IEPM'95, Marrakech. FUCAM – INRIA* (pp. 43–56).
- Alcaraz, J., Maroto, C., & Ruiz, R. (2003). Solving the multi-mode resource-constraints project scheduling problem with genetic algorithms. *Journal of the Operational Research Society*, 54, 614–626.
- Aldowaisan, T., & Allahvedi, A. (2003). New heuristics for no-wait flowshops to minimize makespan. *Computers & Operations Research*, 30, 1219–1231.
- Allaoui, H., & Artiba, A. (2004). Integrating simulation and optimization to schedule a hybrid flow shop with maintenance constraints. *Computers & Industrial Engineering*, 47, 431–450.
- Arthanary, L., & Ramaswamy, K. (1971). An extension of two machine sequencing problem. *OPSEARCH. Journal of the Operational Research Society of India*, 8(4), 10–22.
- Bierwirth, C., Mattfeld, D., & Kopfer, H. (1996). On permutation representations for scheduling problems. In H.-M. Voigt, W. Ebeling, I. Rechenberg, & H.-P. Schwefel (Eds.), *Parallel problem solving from nature, LNCS* (Vol. 1141, pp. 310–318). Berlin, Germany: Springer.
- Chen, C., Vempati, V., & Aljaber, N. (1995). An application of genetic algorithm for flow shop problems. *European Journal of Operational Research*, 80, 389–396.
- Gen, M., & Cheng, R. (1997). *Genetic algorithms & engineering optimization*. New York: John Wiley & Sons.
- Gourgand, M., Grangeon, N., & Norre, S. (1999). Metaheuristics for the deterministic hybrid flow shop problem. In *Proceedings of the international conference on industrial engineering and production management, Glasgow. FUCAM – INRIA* (pp. 136–145).
- Guinet, A., & Solomon, M. (1996). Scheduling hybrid flowshops to minimize maximum tardiness or maximum completion time. *International Journal of Production Research*, 34(6), 1643–1654.
- Gupta, J., & Tunc, E. (1998). Minimizing tardy jobs in a two-stage hybrid flowshop. *International Journal of Production Research*, 36(9), 2397–2417.
- Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor: The University of Michigan Press.
- Ishibuchi, H., Misaki, S., & Tanaka, H. (1995). Modified simulated annealing algorithms for the flow shop sequencing problem. *European Journal of Operational Research*, 81(2), 388–398.
- Jin, Z. H., Ohno, K., Ito, T., & Elmaghraby, S. E. (2002). Scheduling hybrid flowshops in printed circuit board assembly lines. *Production and Operations Management*, 11(2), 216–230.
- Kochhar, S., & Morris, R. (1987). Heuristic method for flexible flow line scheduling. *Journal of Manufacturing Systems*, 6(4), 299–314.
- Lee, C.-Y. (2004). Machine scheduling with availability constraints. In J. Y.-T. Leung (Ed.), *Handbook of scheduling* (pp. 22.1–22.13). New York: CRC Press.
- Linn, R., & Zhang, W. (1999). Hybrid flow shop scheduling: A survey. *Computers & Industrial Engineering*, 37(1–2), 57–61.
- Michalewicz, Z. (1996). *Genetic algorithms + data structures = evolutions programs* (3rd ed.). New York: Springer-Verlag.
- Morita, H., & Shio, N. (2005). Hybrid branch and bound method with genetic algorithm for flexible flowshop scheduling problem. *JSME International Journal C*, 48(1), 46–52.
- Murata, T., Ishibuchi, H., & Tanaka, H. (1996). Genetic algorithms for flowshop scheduling problems. *Computers and Industrial Engineering*, 30(4), 1061–1071.
- Nawaz, M., Ensco, E., Jr., & Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *OMEGA. The International Journal of Management Science*, 11(1), 91–95.

- Norman, B. A. (1999). Scheduling flowshops with finite buffers and sequence-dependent setup times. *Computers and Industrial Engineering*, 36(1), 163–177.
- Osman, I., & Potts, C. (1989). Simulated annealing for permutation flow-shop scheduling. *OMEGA. The International Journal of Management Science*, 17(6), 551–557.
- Papadimitriou, Ch. H., & Kanellakis, P. C. (1980). Flowshop scheduling with limited temporary storage. *Journal of the ACM (JACM)*, 27(3), 533–549.
- Portmann, M., & Vignier, A. (1998). Branch and bound crossed with GA to solve hybrid flowshops. *European Journal of Operational Research*, 107, 389–400.
- Quadt, D., & Kuhn, H. (2007). A taxonomy of flexible flow line scheduling procedures. *European Journal of Operational Research*, 178(3), 686–698.
- Rajendran, C., & Chaudhuri, D. (1992). A multi-stage parallel processor flowshop problem with minimum flowtime. *European Journal of Operational Research*, 57, 11–122.
- Rajendran, C., & Ziegler, H. (2004). Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. *European Journal of Operational Research*, 155, 426–438.
- Reeves, C. (1995). A genetic algorithm for flowshop sequencing. *Computers & Operations Research*, 22(1), 5–13.
- Riane, F., Artiba, A., & Elmaghraby, S. (1998). A hybrid three-stage flowshop problem: Efficient heuristics to minimize makespan. *European Journal of Operational Research*, 109, 321–329.
- Ruiz, R., & Maroto, C. (2006). A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. *European Journal of Operational Research*, 169, 781–800.
- Ruiz, R., Şerifoğlu, F., & Urlings, T. (2008). Modeling realistic hybrid flexible flowshop scheduling problems. *Computers & Operations Research*, 35(4), 1151–1175.
- Ryan, E., Atif Azad, R. M., & Ryan, C. (2004). On the performance of genetic operators and the random key representation. In *Genetic Programming, LNCS* (Vol. 3003, pp. 162–173). Berlin: Springer.
- Salvador, M. (1973). A solution to a special case of flow shop scheduling problems. In S. E. Elmaghraby (Ed.), *Symposium of the theory of scheduling and applications* (pp. 83–91). New York: Springer-Verlag.
- Santos, D., Hunsucker, J., & Deal, D. (1995). Global lower bounds for flow shops with multiple processors. *European Journal Operational Research*, 80, 112–120.
- Sawik, T. (2000). Mixed integer programming for scheduling flexible flow lines with limited intermediate buffers. *Mathematical and Computer Modelling*, 31(13), 39–52.
- Sawik, T. (2002). An exact approach for batch scheduling in flexible flow lines with limited intermediate buffers. *Mathematical and Computer Modelling*, 36(4–5), 461–471.
- Sherali, H., Sarin, S., & Kodialam, M. (1990). Models and algorithms for a two-stage production process. *Production Planning and Control*, 1(1), 27–39.
- Shmidt, G. (2000). Scheduling with limited machine availability. *European Journal of Operational Research*, 121, 1–15.
- Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64, 278–285.
- Tang, L., & Zhang, Y. (2005). Heuristic combined artificial neural networks to schedule hybrid flow shop with sequence dependent setup times. In *Advances in neural networks, LNCS* (Vol. 3496, pp. 788–793). Berlin: Springer.
- Vignier, A., Billaut, J., & Proust, C. (1999). Les Problèmes D'Ordonnement de Type Flow-Shop Hybride: État de L'Art. *RAIRO Recherche opérationnelle*, 33(2), 117–183 [in French].
- Wang, L., Zhang, L., & Zheng, D. (2006). An effective hybrid genetic algorithm for flow shop scheduling with limited buffers. *Computers and Operations Research Archive*, 33(10), 2960–2971.
- Weng, M. X. (2000). Scheduling flow-shops with limited buffer spaces. In J. A. Joines, R. R. Barton, K. Kang, & P. A. Fishwick (Eds.), *Proceedings of the 2000 winter simulation conference* (pp. 1359–1363).
- Widmer, M., & Hertz, A. (1989). A new heuristic method for the flowshop sequencing problem. *European Journal of Operational Research*, 41, 186–193.
- Witt, A., & Voí, S. (2007). Simple heuristics for scheduling with limited intermediate storage. *Computers and Operations Research*, 34(8), 2293–2309.
- Yaurima, V., Burtseva, L., & Tchernykh, A. (2008). Hybrid flowshop with unrelated machines, sequence dependent setup time and availability constraints: An enhanced crossover operator for a genetic algorithm. In *Parallel processing and applied mathematics*. In Wyrzykowski et al. (Eds.), *LNCS* (Vol. 4967, pp. 608–617). Springer.
- Zandieh, M., Fatemi Ghomi, S., & Moattar Husseini, S. (2006). An immune algorithm approach to hybrid flow shops scheduling with sequence-dependent setup times. *Applied Mathematics and Computation*, 180, 111–127.