Applied Soft Computing 11 (2011) 1600–1613

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Machine scheduling in custom furniture industry through neuro-evolutionary hybridization

Juan C. Vidal*, Manuel Mucientes, Alberto Bugarín, Manuel Lama

Department of Electronics and Computer Science, University of Santiago de Compostela, 15782 Santiago de Compostela, Spain

ARTICLE INFO

Article history:
Received 29 November 2008
Received in revised form 7 April 2010
Accepted 29 April 2010
Available online 7 May 2010

Keywords:
Machine scheduling optimization
Multi-objective evolutionary algorithms
NSGA-II
Neural networks
Hybrid approach

ABSTRACT

Machine scheduling is a critical problem in industries where products are custom-designed. The wide range of products, the lack of previous experiences in manufacturing, and the several conflicting criteria used to evaluate the quality of the schedules define a huge search space. Furthermore, production complexity and human influence in each manufacturing step make time estimations difficult to obtain thus reducing accuracy of schedules. The solution described in this paper combines evolutionary computing and neural networks to reduce the impact of (i) the huge search space that the multi-objective optimization must deal with and (ii) the inherent problem of computing the processing times in a domain like custom manufacturing. Our hybrid approach obtains near optimal schedules through the Non-dominated Sorting Genetic Algorithm II (NSGA-II) combined with time estimations based on multilayer perceptron neural networks.

1. Introduction

Machine scheduling is a difficult problem in industrial environments such as custom furniture industry. This problem can be defined as finding an optimal sequence of operations for a set of resources and constraints. Machine scheduling is classified as NP-hard [1] because it has a combinatorial search space caused by the resources that must be allocated to maximize the utilization of machines and to minimize the time required to complete the scheduled process. Therefore, this problem is not feasible with exact methods such as branch and bound, dynamic programming or constraint logic programming. For this reason, solutions near the optimal are considered good solutions in this context and non-derivative methods, such as evolutionary algorithms, simulated annealing, tabu search or simplex method, are better suited [2].

Machine scheduling problem is characterized by the presence of many conflicting objectives. Therefore, it is natural to look at scheduling as a multi-objective optimization problem that raises the issue about how different objectives should be combined to yield a final solution. Unfortunately, the processing times that constitute the essence of some scheduling process are difficult to obtain in custom manufacturing. The computation of these times depends on several factors such as machine, material and piece characteristics. The latter is the key point because materials, dimensions and shape of custom-designed pieces may take a wide range of different values. Since these times consist of a mixture of numerical simulations, analytical calculations and catalogue selections, there is no precise way for calculating the basic information for scheduling.

Selection of the multi-objective optimization algorithm is closely related to the particular problem to be solved. This paper deals with a real-world scheduling problem that is close to *job-shop scheduling problems* family. However, the scheduling constraints in the furniture industry define a huge search space which may be even huger if the scheduling faced events such as machines breaking down, workers getting sick or new jobs appearing. In our approach we deal with these events through a new scheduling and not through rescheduling. That is, these events will only modify the conditions under which the new scheduling is performed. For this reason, our approach requires to be reliable but also time efficient since time is critical to have a faster response to customer orders. In this context, evolutionary algorithms are known to be a fast and robust solution in optimization problems such as scheduling [3,4] because they facilitate finding a global optima, and do not get trapped on local optima as gradient methods might do.

The machine scheduling solution described in this paper combines evolutionary computing and neural networks to reduce the impact of (i) the huge search space that our multi-objective scheduling process must deal with and (ii) the inherent problem of computing the processing times in a domain like custom manufacturing. Our approach is based on the use of a Multi-Objective Evolutionary Algorithm (MOEA) where schedule evaluation is based on processing times modeled by neural networks. In essence,

* Corresponding author. Tel.: +34 981563100x13564.
E-mail addresses: juan.vidal@usc.es (J.C. Vidal), manuel.mucientes@usc.es (M. Mucientes), alberto.bugarin.diz@usc.es (A. Bugarín), manuel.lama@usc.es (M. Lama).

1568-4946/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.asoc.2010.04.020

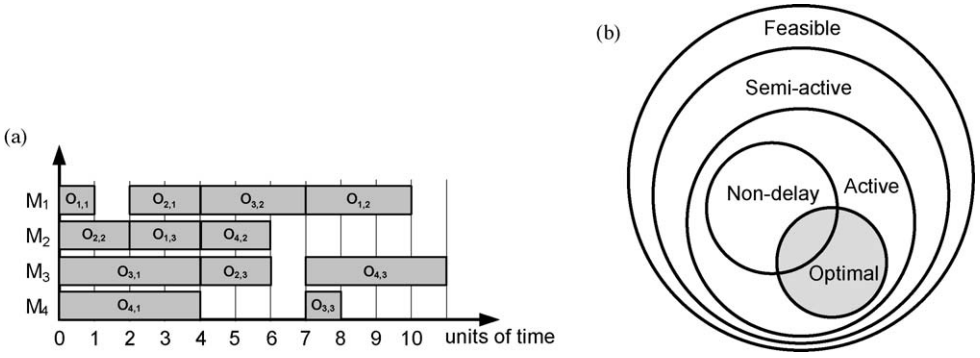


Fig. 1. Schedules representation and hierarchy. (a) Gantt chart of a schedule. $O_{i,j}$ stands for the jth operation for job J_i and (b) Venn diagram of the different types of schedules.

we apply the Non-dominated Sorting Genetic Algorithm II (NSGA-II) [5] for producing new schedules in each iteration, and a multilayer perceptron neural network [6] to estimate their processing times.

The paper is structured as follows: Section 3 introduces the problem of scheduling in custom wood-based furniture manufacturing industry. Then, Section 4 describes the NSGA-II algorithm that solves the scheduling problem, how the problem has been encoded and which objective functions evaluate the fitness of the solutions. Section 5 presents the neural network model we defined to compute the processing times of machines, while Section 6 shows some results and comparisons for experiments in a real-world production environment. Finally, Section 7 points out the conclusions.

2. Related work for machine scheduling problems

The scheduling problem consists of finding a schedule satisfying a set of restrictions. A schedule is the allocation of time intervals for the m machines $M = \{M_1, M_2, \ldots, M_m\}$ that perform the n jobs $J = \{J_1, J_2, \ldots, J_n\}$ of the problem. Each job $J_i \in J$ consists of a set of n_i operations $O_i = \{O_{i,1}, O_{i,2}, \ldots, O_{i,n_i}\}$ with a processing time $p_{k,i,j}$ on a machine M_k. For example, a schedule may be represented as Gantt charts as shown in Fig. 1(a). Each row represents a machine and each box an operation with a time interval.

A schedule represents the final allocation of resources to jobs over time. The schedules that satisfy all constraints are denominated feasible schedules. Feasible schedules can be classified as follows:

- *Semi-active schedules*: No operation can be started earlier without changing the processing order or violating some constraint.
- *Active schedules*: No operation can be started earlier without delaying at least another operation or violating some constraint.
- *Non-delay schedules*: No machine is ever idle if an operation is ready to be processed.

The relationship between these classes of schedules and optimal schedules is illustrated in Fig. 1(b). Usually, the corresponding scheduling problem only searches for the set of active (or sometimes semi-active) schedules, since this brings a huge reduction of the search-space while still guaranteeing that an optimal schedule can be found.

2.1. Machine scheduling problem

The scheduling literature is characterized by an unlimited number of problem types. Scheduling problems can be classified according to job characteristics, machine environment and optimality criteria [7]. Job characteristics may have:

- *Preemption*: A job may be interrupted and resumed at a latter time, even on another machine.
- *Precedence relations*: The precedence between jobs may be represented by acyclic graphs or rooted trees.
- *Release dates*: Each machine M_k has a *start time* $s_{k,i,j}$ before which no processing can be done on the machine for the operation $O_{i,j}$.
- *Processing times restrictions*: For example, setting a *unit processing time restriction*, $p_{i,j} = 1$, to all the operations $O_{i,j}$.
- *Due dates*: A job J_i must be achieved before a due date d_i.
- *Batch modes*: A set of operations may be grouped in batches. A batch must be processed jointly on a machine.

Machine environment defines the type of scheduling problem. The first four scheduling problems of the following list apply to one-operation models (jobs with only one operation) while the latter five to multi-operation models:

- *Dedicated machines*: Each job must be processed on a dedicated machine.
- *Identical parallel machines*: All the machines that process a job are identical. Thus, they have the same processing time, $p_{k,i,1} = p_i$.
- *Uniform parallel machines*: The processing time of the machine M_k is $p_{k,i,1} = p_i/s_k$ where s_k is the speed of the machine.
- *Unrelated parallel machines*: The processing time of the machine M_k depends on the job, $p_{k,i,1} = p_i/s_{i,k}$.
- *General-shop*: The machines are dedicated and there are precedence relations between the operations.
- *Open-shop*: Equivalent to general-shop except that there are no precedence relations. The operations of a job may not be processed in any particular order.
- *Job-shop*: Special case of the general-shop where the precedence relation states that the operation $O_{i,j}$ is the j-th operation of job i and cannot start until the operation $O_{i,j-1}$ has finished, $1 < j < n$.
- *Flow-shop*: Special case of the job-shop where the machine processing orders O_i are the same for all the jobs. This does not mean that jobs are identical, since their processing times may vary.
- *Mixed-shop*: A combination of job-shop and open shop.

Many optimality criteria exists for scheduling problems. Frequently used performance measures are:

- *Makespan*: The maximum completion time (length of the schedule).
- *Total flow-time*: The total time spent on all the jobs.
- *Total lateness*: The summed lateness of all the jobs, that is, how much later than the deadline each job finishes.
- *Total tardiness*: The summed tardiness of all the jobs, that is, the sum of the lateness of the jobs.
- *Total earliness*: The summed earliness of all the jobs, that is, the sum of the negative lateness of the jobs.

- *Maximum lateness*: The lateness of the latest job.
- *Maximum tardiness*: The tardiness of the tardiest job.

In this work, we address a job-shop scheduling problem (JSSP). Among the various types of scheduling problems, the JSSP problem is one of the most challenging. Problems of size $n \times m$ and $m \geq 2$ are NP-hard and are considered the worst among combinatorial problems [1]. The size of the search space for the traditional JSSP is $(n!)^m$.

2.2. Traditional approaches

Many approaches have been proposed to solve JSSP. In Refs. [8,9], sophisticated branch and bound (BB) methods are used to minimize the makespan in classical JSSP. Although these methods have proved to be very useful for small to medium sized problems, their excessive computing time makes their application to large problems difficult [10]. Many other exact approaches of operational research, such as mixed integer programming [11] or dynamic programming [12], have been used. However, the number of constraints and/or variables becomes very large even for small sized problems, and therefore those attempts are not very effective for large sized problems.

Many non-optimal heuristic approaches can deal with JSSP in reasonable computational time. Knowledge-based systems [13], dispatching rules [14] or neural networks [15] have been used in real JSSP. Nevertheless, meta-heuristic algorithms have proved a better performance for solving job-shop problems. Tabu search (TS) has proved to be very effective algorithm for the JSSP [16,17]. Although, when it is applied to hard optimization problems, such as real scheduling problems, the performance depends on the initial solution used. Another technique that has been applied to JSSP is simulated annealing (SA) [18,19]. SA can avoid local maxima/minima, but is unable to achieve good solutions quickly. Performance can be increased by combining SA with TS. The main principle of this approach is that SA is used to find the elite solutions so that TS can re-intensify search from the promising solutions [20]. Shifting bottleneck (SB) heuristic works by relaxing the problem into a number of one machine subproblems that are solved one at a time. This heuristic was among the first really efficient approximation methods [21,12]. SB search requires a high computing effort due to the re-optimizations that are necessary to achieve the results. Best solutions are achieved by tuning its several parameters.

Evolutionary algorithms (EA) have become a popular approach to solve JSSP since they usually achieve better performance than many traditional and heuristic approaches applied in JSSP [22]. Most of these EA aimed to solve the classical JSSP or small variations of the classical problem. They differ from one another mainly in the representation schemes, in the operators, in the hybridization level with other heuristics (to improve previous generated solutions), and in the performance measure adopted.

Many chromosome representation schemes for solving JSSP are reported in the literature [23]. Chromosomes can use a direct or an indirect representation. Direct representations encode the schedule in the chromosome. For example, an *operation-based* chromosome encodes the schedule as a sequence of operations, and each gene stands for one operation. However, complex crossover and mutation operators are required to create new solutions. In indirect representations, simple operators are allowed but the chromosome encodes a feasible schedule. For example, *dispatching rules* for job assignment can be encoded in each gene.

Multi-objective criteria have also been incorporated into EA models [24,25]. Most of the approaches are based on the combination of multiple objectives into a single scalar objective using weighted coefficients [26,27]. However, few MOEAs have been applied to scheduling in terms of Pareto dominance among individuals. For example, in [28] a MOEA is applied to the scheduling of drilling operations in printed circuit board industry with the objectives of minimizing makespan and total flow time. In [29] to the classical JSSP with the makespan and total tardiness as objectives. In [30] to cellular manufacturing systems with three objectives, makespan, total flow time and machine idleness. In Ref. [31] a MOEA is proposed to derive the optimal dispatching rules in the Giffer and Thompson algorithm and in [32] to flexible JSSP. Contemporary MOEA use selection and replacement based on a multi-objective domination criterion. Examples of this approach are MOGA [33], NPGA [34], PESA [35], SPEA [36] and NSGA-II [5]. Many other meta-heuristics strategies, such as Greedy Randomized Adaptive Search Procedure (GRASP) [37], Ant/Bee Colonies [38,39], and Jumping Genes Genetic Algorithm [40] have been applied to solve JSSP. However, limited results are available and have yet to prove their performance compared to current state-of-the-art algorithms [22].

2.3. Processing time estimation

In order to obtain a good solution for a JSSP, it is necessary to have an accurate time estimation of each of the operations involved in the scheduling. Processing time estimation is a regression problem. Therefore, given a dataset in which each example contains the values of several input variables and their corresponding output (processing time), any regression technique could be applied to obtain a time estimation. A number of soft computing-based techniques, such as fuzzy rule based systems, decision trees, support vector machines for regression, Bayesian regression, have been described for this work.

In particular, the most widely used methodology for processing time estimations in scheduling problems is neural networks. For example, in [41] a neural network has been used for shipbuilding scheduling. The system tries to accurately estimate the required welding man-hours of each one-of-a-kind block of the scheduling. They considered four groups of variables: ship type, block type, block's physical characteristics and shop type. Also, in [42] the processing time estimations for metal furniture assembly, welding and painting is done with a neural network. The processing times required to complete these operations vary significantly with the specific product variation and contain non-linearities and unspecified interactions.

Whenever accuracy is important, but also a certain degree of interpretability of the proposed regression system is required, fuzzy rule based systems are a good choice [43]. Rules have a variable structure in the consequent, as the regression functions can be completely distinct for different machines or, even, for different classes of inputs to the same machine. The TSK knowledge base was learned with genetic programming together with a context-free grammar to restrict the valid structures of the regression functions.

3. Machine scheduling problem in custom wood-based furniture manufacturing

The time interval for completing all of the operations of a work order is defined as throughput time. Its accurate estimation is hard in industries such as the furniture industry where custom-designed products are dominan. The lack of previous manufacturing experiences and the complexity in the production makes time estimations difficult.

The reduction of throughput time has many benefits: lower inventory, reduced costs, improved product quality (problems in the process can be detected more quickly), faster response to customer orders, and increased flexibility. Much effort is there-

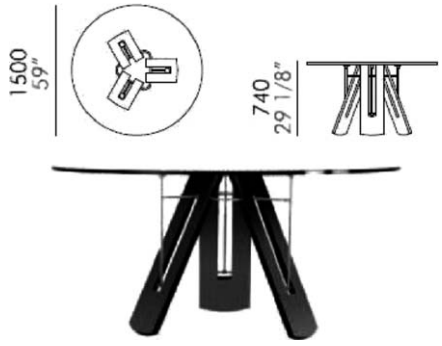


Fig. 2. Conceptual design of a hall office table from which furniture parts, manufacturing steps and finishings can be extracted.

fore spent to reduce throughput time by improving manufacturing planning, control systems, and developing more sophisticated scheduling procedures [44,45]. The objective is to define work plans that minimize the resources queuing and maximize resources capacity, constrained to the material availability and product requirements. In this context, the feasibility, time, and cost of the most promising plans is analyzed [46,47]. Note that the throughput time has many components, including move, queue, setup, and processing times [48–50]. In this work we will address all these components by improving the machine scheduling task to produce feasible work plans.

The real-world scheduling problem described in this work is close to the *job-shop scheduling problem* (JSSP) family:

- Each job J_i defines a precedence relation with its operations $O_{i,j}$.
- Each job J_i must be achieved before a due date d_i.
- No processing restrictions on the operations are defined.
- No batch mode is defined. Some of the operations of our domain can be considered batch modes. However, we abstract them as simple operations.
- Each machine has a start time $s_{k,i,j}$ before which no processing can be done on the machine for the operation $O_{i,j}$.
- Each machine can process only one operation at a time.
- No machine can free an operation until it is finished, that is, no preemption is allowed.
- The total number of machines of each type is fixed and greater than one.
- No job can start before the processing of its parts is available.

However, some of the constraints of our problem modify the classical JSSP, and make it closer to flexible JSSP:

- Two operations of the same job may be processed simultaneously. In our scheduling problem precedence relations are represented as acyclic graphs, while in classical JSSP these relations are represented as a sequence of operations. Furniture industry, like many others, manufacture products made up of many different types of raw material. For example, the table depicted in Fig. 2 is made of plywood, steel and glass. Therefore, the processing of the different materials until they are assembled can be performed simultaneously. Although the way in which the precedence relations are defined is out of the scope of this paper, just mention that this selection is based on a set of rules based on woodworking knowledge. These rules take into account the constructive decisions, joints used to assemble the furniture, and finishing or quality standards.
- Each job can be scheduled on the same machine more than once. In classical JSSP, it is usually required that for each job J_i, the sequence of operations $O_{i,j}$ contains exactly one operation to be processed on each of the machines. However, in our machine environment, the same machine can perform several manufacturing steps. For example, edge banding machines can be used or edge banding, trimming and sanding. Therefore the same machine may be used more than once for a job.
- Jobs do not have to visit every machine in M. Although in classical JSSP each machine must be visited in the schedule of a job J_i, this restriction does not apply to our real scheduling problem. Two products may have very different manufacturing steps. For example, a solid wood table and the table depicted in Fig. 2 do not share many operations.

Another difficulty our scheduling problem must deal with is the processing times estimation, which is a critical piece in each manufacturing step. The estimation of the processing time is one of the most important tasks in the product design life cycle. For example, time estimations are taken into account to redesign the product if the predicted time is longer than expected. There are many models and techniques for estimating the processing times of a manufacturing step based on the product design [51]. For a detailed design, highly detailed process planning, manufacturing process simulation, or time estimation models can be employed [51,47]. For a conceptual design, however, less detailed models must depend on a more limited set of critical design information [51]. Both approaches are applied in the furniture industry since the definition of a detailed design is cost and time expensive. Therefore, there are several ways of computing the processing time of a machine depending on the available input variables. For instance, several manufacturing processes can be extracted from the conceptual designs depicted in Fig. 2. The hall office table has a round top leaf with three tripod legs. The base material is phenolic plywood, the structure is bright stainless steel and the table top is transparent glass. Taking into account that the round glass material is purchased to an external provider, two different classes of operations can be inferred for (i) wood and (ii) steel pieces processing. For example, wood pieces must be cut from large wood boards, planned and calibrated to have a uniform surface and thickness, cut again at a 45° angle, and so on.

The processing time of an operation for a machine can depend on several input variables, like the dimensions of the product, the material, the speed of the machine for that kind of product, and so on. For example, the calibrating of wood pieces depends on the sanding machine that will perform the operation, apart from the piece variables. Calibrating machinery specifies feed speed, abrasive belt speed, abrasive belt size, working thickness, maximum and minimum workpiece dimensions and so on. Nevertheless, the processing time for a given operation may not always be influenced by all of these variables. Thus, a machine can have several regression functions for a class of operation and product characteristics. Therefore, a good estimation of the throughput time demands an accurate regression function as well as the selection of the appropriate function among all the regression functions of the machine. In this context, processing times estimation is difficult to obtain by standard methods or manufacturing experts and is a source of errors and uncertainty for the scheduling process.

4. NSGA-II multi-objective approach for machine scheduling

A multi-objective scheduling problem can be described as a multi-objective optimization problem:

$$minF(x) = \{f_1(x), f_2(x), \ldots, f_k(x)\} s.t.\, x \in S \quad (1)$$

where x is a solution, S is the set of feasible solutions, k is the number of objectives in the problem, $F(x)$ is the image of x in the k-objective space and each $f_i(x)$ for $i = 1, \ldots, k$ represents one objective. In

custom furniture manufacturing, like in many other real-world problems, there are conflicting $f_i(x)$ objectives. For example, objectives like *minimize the cost of furniture* and *minimize the completion time* may be in conflict since it is usual that faster machines have a higher recovery cost. So, in contrast to a single-objective optimization problem, there is not one best solution, but several solutions to choose from (non-dominated set of solutions).

Our approach to this problem uses the multi-objective scheduling optimization through the NSGA-II algorithm [5]. NSGA-II is one of the most efficient MOEAs using elitist approach: its computational complexity is in $O(MN^2)$, where M is the number of objectives and N the population size. The algorithm structure is described in Fig. 3. NSGA-II has a fitness assignment scheme that consists in sorting the population in different fronts using the non-domination order relation. Therefore, it has two objectives: (i) to find a set of non-dominated solutions as close as possible to the Pareto-optimal front in each iteration but (ii) maintaining the set of solutions as diverse as possible covering or nearby the Pareto-optimal front.

The main loop of the algorithm begins with the combination of the current and previous populations, and the calculation of the non-dominated fronts of R_t. This is done with the *fast-non-dominated-sort* function. In the first step of this function, for each individual p of the population P, two calculations are done: the set of individuals dominated by p (S_p), and the number of individuals that dominate p (n_p). All those individuals (p) that are non-dominated will have a rank of $p_{rank} = 1$ and will belong to the first Pareto front (non-dominated front) (V_1). In order to calculate the other Pareto fronts (step 3), for each individual (p) of the previous Pareto front, each element (q) of set S_p decreases its domination counter in 1 (n_q). Therefore, all those individuals q with $n_q = 0$ will belong to the i th-Pareto front.

Once all the Pareto fronts have been determined, the main loop of the algorithm adds (step 1.d) to the new population (P_{t+1}) all the individuals in the i-th Pareto front (V_i), starting with $i = 1$ and increasing i while the size of the population is under N. Moreover, for each individual belonging to V_i, a measure of the crowding distance (*crowding-distance-assignment*) is calculated taking into account all the individuals in V_i. This crowding distance is the sum of the crowding distances for each objective and gives and estimation of the density of solutions surrounding a particular solution.

Steps 1.e and 1.f of the main loop are used when not all the individuals of the ith Pareto front can be added to the new population (P_{t+1}) (as the total size of the population would exceed N). All the individuals of V_i are sorted in descending order using the crowded-comparison operator ($\prec_n$). This operator is used in all the selection processes of the algorithm (reduction of the population and tournament selection), thus it is necessary to calculate the crowding distance for all the individuals of P_{t+1} (step 1.d.i), and not only for the individuals in V_i (step 1.e). As defined in Fig. 3, solution i has better rank than solution j if it belongs to a lower order Pareto front ($i_{rank} < j_{rank}$), or if the Pareto front is the same and the crowding distance of i is higher than that of j ($i_{distance} > j_{distance}$).

Finally (step 1.g), and using population P_{t+1}, individuals are selected (tournament selection using $\prec_n$), crossed and mutated to create the new population Q_{t+1}.

Main Loop

1. *while* $t <$ *maxIterations*
   (a) $R_t = P_t \bigcup Q_t$
   (b) $V =$ *fast-non-dominated-sort* (R_t)
   (c) $P_{t+1} = \emptyset$ and $i = 1$
   (d) *until* $|P_{t+1}| + |V_i| \leq N$
      i. *crowding-distance-assignment* (V_i)
      ii. $P_{t+1} = P_{t+1} \bigcup V_i$
      iii. $i = i + 1$
   (e) *Sort* $(V_i, \prec_n)$
   (f) $P_{t+1} = P_{t+1} \bigcup V_i\,[1 : (N - |P_{t+1}|)]$
   (g) $Q_{t+1} =$ *make-new-pop* (P_{t+1})
   (h) $t = t + 1$

fast-non-dominated-sort (P)

1. *for each* $p \in P$
   (a) $S_p = \emptyset$, $n_p = 0$
   (b) *for each* $q \in P$
      i. *if* $(p \prec q)$ *then*
         A. $S_p = S_p \bigcup q$
      ii. *else if* $(q \prec p)$ *then*
         A. $n_p = n_p + 1$
   (c) *if* $\left(n_p = 0\right)$ *then*
      i. $p_{rank} = 1$
      ii. $V_1 = V_1 \bigcup \{p\}$
2. $i = 1$
3. *while* $(V_i \neq \emptyset)$
   (a) $Q = \emptyset$
   (b) *for each* $p \in V_i$
      i. *for each* $q \in S_p$
         A. $n_q = n_q - 1$
         B. *if* $\left(n_q = 0\right)$ *then*
            - $q_{rank} = i + 1$
            - $Q = Q \bigcup \{q\}$
   (c) $i = i + 1$
   (d) $V_i = Q$

$i \prec_n j$ *(crowded-comparison operator)*

- *if* $(i_{rank} < j_{rank})$
- *or* $((i_{rank} = j_{rank})$ *and* $(i_{dist} > j_{dist}))$

crowding-distance-assignment (I)

1. $l = |I|$
2. *for each i, set* $I[i]_{distance} = 0$
3. *for each objective m*
   (a) $I = sort\,(I,\ m)$
   (b) $I[1]_{dist} = I[l]_{dist} = \infty$
   (c) *for* $i = 2$ *to* $l - 1$
      i. $I[i]_{dist} = I[i]_{dist} + (I[i+1].m - I[i-1].m) / \left(f_m^{max}(x) - f_m^{min}(x)\right)$

Fig. 3. NSGA-II algorithm [5].

4.1. Problem encoding

Choosing a good representation is a vital component for solving search problems. In this paper, a composite representation of the chromosome is proposed. Part of this encoding is used to reduce the flexible JSSP to a classical JSSP and the remainder to define the priority dispatching rules used to create the schedule. Specifically, machine assignment is achieved through the *parallel job encoding* [52]. Fig. 4 represents this encoding for a scheduling problem of four jobs and four machines. Each row of the matrix is an ordered

	O_1	O_2	O_3
J_1	$(M_1, 0)$	$(M_1, 7)$	$(M_2, 2)$
J_2	$(M_1, 2)$	$(M_2, 0)$	$(M_3, 4)$
J_3	$(M_3, 0)$	$(M_1, 4)$	$(M_4, 7)$
J_4	$(M_4, 0)$	$(M_2, 4)$	$(M_3, 7)$

Fig. 4. Parallel job encoding of the 4×4 schedule example (first part of the chromosome).

	d_i	Order	$O_{i,j}$	M_1	M_2	M_3	M_4
J_1	9	1,3,2	$O_{1,1}$	1	4	6	9
			$O_{1,2}$	3	2	5	1
			$O_{1,3}$	4	2	1	3
J_2	7	2,1,3	$O_{2,1}$	2	8	7	1
			$O_{2,2}$	2	2	4	5
			$O_{2,3}$	6	11	2	7
J_3	8	1,2,3	$O_{3,1}$	8	5	4	9
			$O_{3,2}$	3	3	6	1
			$O_{3,3}$	7	1	8	1
J_4	11	1,2,3	$O_{4,1}$	5	10	6	4
			$O_{4,2}$	4	2	3	8
			$O_{4,3}$	7	3	4	1

Fig. 5. Due dates, operations order and processing times of operations on machines.

sequence of operations $O_{i,j}$, $i = 1, \ldots, 4$, $j = 1, \ldots, 3$ (j is the operation index). Each element in a row contains two terms: (i) the machine M_k, $k = 1, \ldots, 4$, which performs the operation and (ii) the starting time $t_{k,i,j}$ of operation $O_{i,j}$ in machine M_k. For example, the operation O_1 in job J_2 is performed on machine M_1 at time 2.

This encoding directly produces a feasible solution. It contains the machines that will execute each operation and at which time. Time is set to zero when the machine cannot perform the operation and the gene is in blank when the operation does not apply in the job. Therefore, a schedule is easily created knowing the operations routing and the processing times. Fig. 5 complements the 4×4 scheduling example and provides (i) the due date of each job J_i, (ii) the operations order in the job and (iii) the time (the sum of the setup and processing time) of each operation $O_{i,j}$ on each machine of the production plant. For example, job J_2 has a due date of 7 time units and defines the $O_{2,2}$, $O_{2,1}$, $O_{2,3}$ operations order. The Gantt chart depicted in Fig. 1(a) represents the schedule that can be inferred from the chromosome depicted in Fig. 4 and the data of Fig. 5.

Fig. 6 depicts the priority dispatching rules assigned to the 4 jobs and 4 machines scheduling problem. In this encoding the gene i represents the priority rule that must be applied to process the ith operation of the schedule. From these assignments, the schedule is constructed with the Giffler and Thompson algorithm [53] but using this priority dispatching rules heuristic.

The codes and priority rules considered [23] are given in Table 1. For example, the first two jobs will prioritize the operations with a shortest processing time (SOT).

4.2. Schedule generation

Schedules are generated with a modified version of the *Giffer–Thompson algorithm* [53]. The algorithm depicted in Fig. 7 generates active schedules from the chromosomes previously described. Firstly, the algorithm inserts in A all the operations that are initially ready to be scheduled, that is, the first operations of each job. Since the precedence relation of our scheduling is an acyclic graph, A may contain several operations of the same job. Then, in each iteration it takes the operation $O_{i,j}$ in A with the earliest potential completion time, and then selects an operation $O_{x,y}$ from the set of operations to be processed by the machine

φ_1	φ_2	φ_3	φ_4	...
0	0	3	4	...

Fig. 6. Priority dispatching rules encoding of the 4×4 schedule example (second part of the chromosome).

Table 1
Priority dispatching rules for operation assignment.

Code	Rule	Description
0	SOT Shortest Operation Time	Operation with shortest processing time on the considered machine
1	LOT Longest Operation Time	Operation with longest processing time on the considered machine
2	SPT Shortest Processing Time	Operation which job has the shortest total processing time
3	LPT Longest Processing Time	Operation which job has the longest total processing time
4	SRO Smallest Remaining Operations	Operation with smallest number of remaining job operations
5	LRO Largest Remaining Operations	Operation with largest number of remaining job operations
6	SRT Shortest Remaining Time	Operation with shortest remaining job processing time
7	LRT Longest Remaining Time	Operation with longest remaining job processing time

assigned to $O_{i,j}$. The latter selection is based on the priority dispatching rule assigned to this iteration in the chromosome. Finally, it adds this operation to the schedule, removes it from A and add its job successors to A.

Fig. 8 depicts the first steps of the algorithm for the chromosome represented in Figs. 4 and 6 and the time-table of Fig. 5. In this figure, a schedule is described by a directed graph $G = (V, P \cup T)$ in which each node in the set V represents an operation $O_{i,j}$ and each arc in $P \cup T$ represents a relation between the operations. Specifically, non-dashed arcs represent the set P of *precedence relations* while dashed arcs the set T of technological (machine) relations. Each job J_i of the chromosome represents a row of the graph and each node is marked with the number of the machine that will process the operation. Note that each row of the graph may also have parallel branches since several operations of a job may be processed in parallel. The A, S and Q sets are also represented in this figure. For instance, operations in A are marked by dashed circles, operations in Q are marked by squares and scheduled operations are gray colored.

1. set $S = \emptyset$
2. set $A = \{O_{i,j} | 1 \le i \le n, first(J_i) = j\}$
3. set $i = 1$
4. while (A is not $\emptyset$) do
   (a) find the operations $O_{i,j} \in A$ with the earliest potential completion time σ (if two or more operations are tied, pick one at random)
   (b) set M_k to the machine that is to process o
   (c) set Q to the set of operations from A to be processed on M_k with potential starting times earlier than σ
   (d) pick an operation $O_{x,y} = \phi_i(Q)$ from Q, where ϕ_i is the priority rule assigned to the iteration i (if two or more operations are tied, pick one at random)
   (e) remove $O_{x,y}$ from A
   (f) add $O_{x,y}$ to S with starting time $h(O_{x,y})$
   (g) add to A the job operations that are successor of $O_{x,y}$
   (h) set $i = i + 1$

Fig. 7. Modified Giffer–Thompson algorithm used for schedule generation. The algorithm has been modified to use the corresponding priority dispatching rule of each iteration.

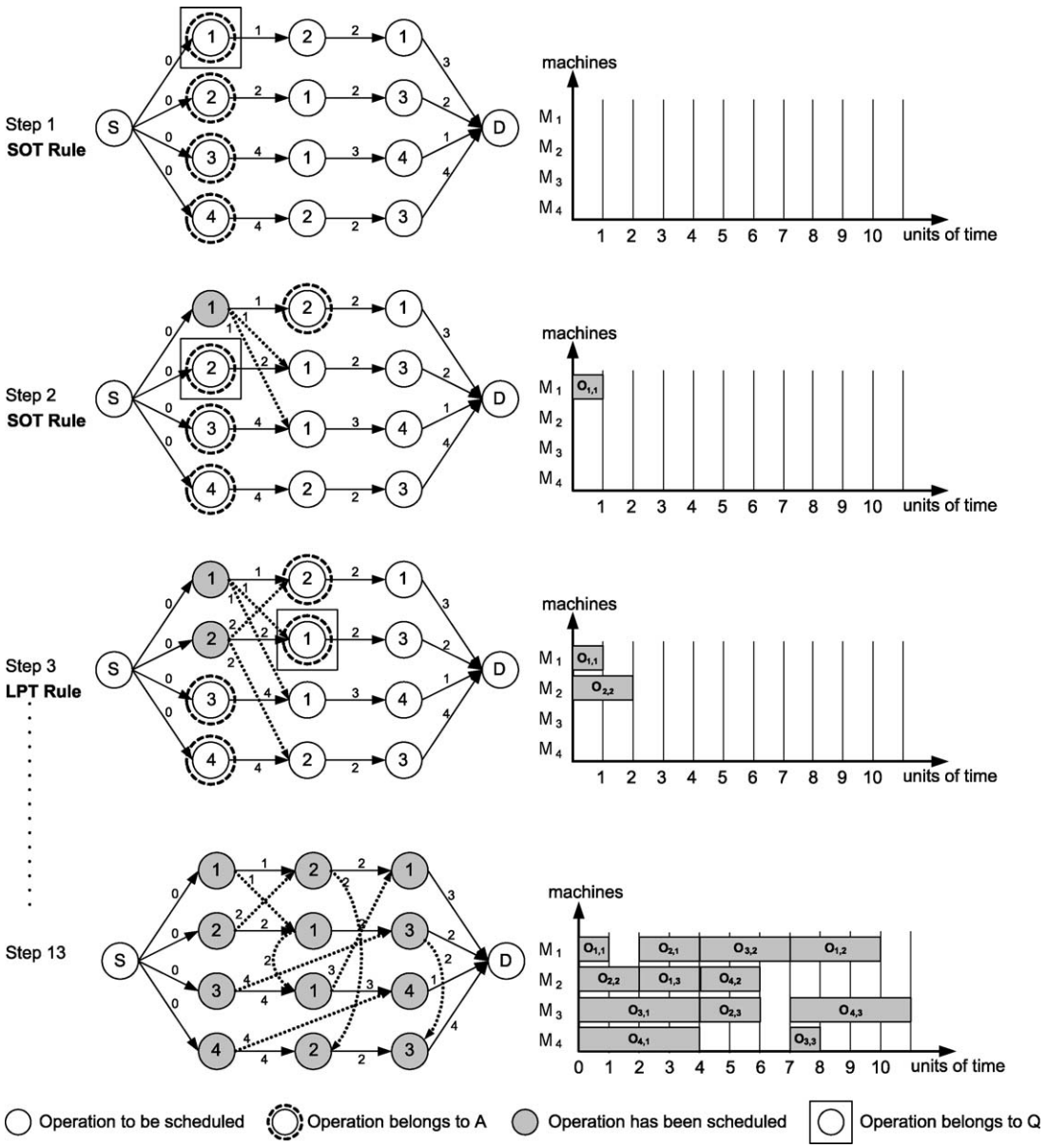


Fig. 8. Example of the modified Giffer–Thompson algorithm 7. Each step of the algorithm is represented by a graph representation with Gantt chart showing the schedule.

After the initial step, A contains the initial operations of each job $\{O_{1,1}, O_{2,2}, O_{3,1}, O_{4,1}\}$ and Q the operation $O_{1,1}$ which has the earliest completion time of all machines. According to the SOT strategy associated to the first iteration, only $O_{1,1}$ may be selected as the shortest processing time operation for machine M_1. Therefore, it is scheduled for processing, removed from A and its successor operation in J_1 is added to A. In the second step, $A = \{O_{1,3}, O_{2,2}, O_{3,1}, O_{4,1}\}$ and the algorithm selects $O_{2,2}$ as the operation o with the earliest completion time. Then it looks for operations processed by the same machine M_2 with the shortest processing time. Thus, step 2 defines the set $Q = \{O_{1,3}, O_{2,2}\}$ and randomly selects $O_{1,3}$. The algorithm will keep running until all the operations of A have been scheduled (step 13).

4.3. Crossover

Crossover only will be applied to one of the parts of the composite chromosome in each iteration of the evolutionary algorithm. A random number is used to decide which part will be crossed.

Two crossover operators have been adapted to machine assignments. Both operators cross two randomly selected chromosomes to generate a new schedule although row crossover affects jobs whereas column crossover operations:

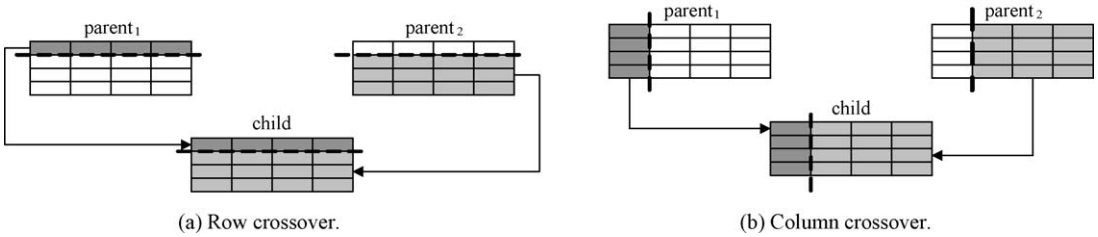
- Row crossover (Fig. 9(a)): the child chromosome consists of the $1, \ldots, k$ jobs of $parent_1$ and the $k+1, \ldots, n$ jobs of $parent_2$.
- Column crossover (Fig. 9(b)): the child chromosome is made up of the $1, \ldots, k$ operations of $parent_1$ and the $k+1, \ldots, n$ operations of $parent_2$.

where k is a randomly selected cross point.

A two point crossover has been applied to the priority dispatching rules encoding, as it lets us to interchange a random number of rules between individuals in any position of the chromosome.

4.4. Mutation

This operator uses the same strategy as crossover for selecting which of the parts of the chromosome to mutate. For machine



(a) Row crossover. (b) Column crossover.

Fig. 9. Two crossover operators for the parallel job encoding. (a) Row crossover and (b) column crossover.

assignments, only machine mutation is used since parallel job encoding does not allow to interchange jobs or operations in the chromosome. Taking this constraint into account, two mutation operators have been used, however, only one of them is applied per mutation:

- Random mutation: a different machine is randomly selected to perform operation $O_{i,j}$.
- Load balancing mutation: a different machine is selected to perform operation $O_{i,j}$. The selection is based on the load of the machine in the schedule so this operator looks for balancing the machine loads.

where ith row and jth column are randomly selected. Note that both mutation operators always generate feasible schedules.

These two mutation operators are complementary. The first one assumes a uniform probability for all the machines. The second one tries to improve the load balancing taking into account a probability of selection that is inversely proportional to the loan of the machine.

For priority dispatching rules, a uniform mutation is applied as there is no a priori information of the advantages of the selection of one rule over the others.

4.5. Objective functions

Many time performance measures exist for JSSP, such as makespan, total flow-time, total lateness, and total tardiness. Our aim is to minimize the following two objectives:

- C_{max}: Makespan. This measure returns the maximum completion time of the jobs:

$$C_{max} = max(C_i) \tag{2}$$

where $i \in \{1, \ldots, n\}$ and the completion time of a job i is:

$$C_i = max\{s_{k,i,j} + p_{k,i,j}\} \tag{3}$$

for $j \in \{1, \ldots, o\}$, and $k \in \{1, \ldots, m\}$.
- T_Σ: Total tardiness. The tardiness measures how much later than the deadline the job finishes. If the job finished earlier than d_i, it is assigned a negative lateness:

$$T_\Sigma = \sum_{i=1}^{n} max(0, C_i - d_i) \tag{4}$$

Makespan and *total tardiness* are the most important objectives when evaluating a schedule in the context in which this work has been developed. Cost is also relevant but it can be considered a time-dependent variable and like another objectives, such as the communication capacity between two manufacturing steps, has little influence in the selection of the most suitable schedule.

For the schedule example depicted in Fig. 1(a), the makespan is $C_{max} = max\{10, 6, 8, 11\} = 8$ and total tardiness of the $T_\Sigma = 1 + 0 + 0 + 0 = 1$. Note that both objective functions depend on the processing time of machines. The estimation of this time is an important task in any manufacturing industry [44] and has a huge impact on the quality of the work plans produced by the scheduling process.

5. Neural network approach for processing times estimation

The approach presented in this section looks for a high accuracy regression model for processing times estimation. Processing times of a machine can be described as polynomials of several input variables where variables can be combined in many different ways:

$$\sum_i \alpha_i \cdot \prod_{j=1}^{na} x_j^{\delta_{i,j}} \tag{5}$$

where α_i are the coefficients, x_j are the input variables for $j = 1, \ldots, na$, and $\delta_{i,j}$ is an indicator variable defined by

$$\delta_{i,j} = \begin{cases} 1 & \text{if } x_j \in i\text{th term of the polynomial} \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Note that for a given machine, there can coexist different polynomials, each one representing the estimation of the processing time of a class of the input variables. For example, in an specific machine, processing times of pieces with a thickness over a threshold could be estimated with a polynomial, and under that threshold with another polynomial.

Summarizing, the learning model must have the capacity to approximate non-linear functions, to capture complex relationships in the data, and to identify the regression function to apply for each class of input variables for a machine and operation. Furthermore, the learning model must be reactive to changes in the supply chain configuration. That is, it may learn new polynomials for each updating of machinery or new operation a machine can perform. Taking into account the previous conditions, a neural network approach has been used. Neural networks are universal approximators and can easily be trained to map multidimensional nonlinear functions because of their parallel architecture.

5.1. Multilayer perceptron

We approach each time estimation through a multilayer perceptron (MLP). The computations performed by the network for a single hidden layer can be written as

$$\mathbf{y} = f(\mathbf{x}) = \mathbf{B}\varphi(\mathbf{A}\,\mathbf{S} + \mathbf{a}) + \mathbf{b} \tag{7}$$

where $\mathbf{S}$ is a vector of inputs, $\mathbf{x}$ a vector of outputs, $\mathbf{A}$ and $\mathbf{a}$ are the weight matrix and the bias vectors of the first layer whereas $\mathbf{B}$ and $\mathbf{b}$ are the weight matrix and the bias vectors of the second layer. The function φ denotes a sigmoidal function $\varphi(z) = 1/(1 + \exp^{-z})$.

The standard backpropagation (BP) learning algorithm was adopted in the learning process of time estimations [54]. Here the output values are compared with the production time to compute the value of the error function. The error is then fed back through

Table 2
Summary of the results obtained for the considered methods.

Benchmark	Size	Method	C_{max}		T_Σ		Method	C_{max}		T_Σ	
			Best	Avg.	Best	Avg.		Best	Avg.	Best	Avg.
abz8	10 × 10	FastPGA	803	852.00	866	1158.00	GDE3	826	890.14	1212	1648.14
		NSGA-II	**784**	851.75	**780**	1022.00	PAES	820	877.22	1000	1409.67
		PESA-II	794	823.33	841	993.00	SPEA2	786	831.40	828	1070.80
abz9	10 × 10	FastPGA	795	800.00	721	802.00	GDE3	815	859.25	1070	1550.50
		NSGA-II	**763**	810.14	619	1109.86	PAES	808	880.25	976	1271.25
		PESA-II	777	857.00	618	969.67	SPEA2	766	823.60	**609**	859.80
car5	10 × 6	FastPGA	5373	5588.70	2979	4695.82	GDE3	6030	7032.75	6461	9909.12
		NSGA-II	5369	5849.17	**2040**	5025.93	PAES	5431	5750.00	4101	4101.00
		PESA-II	**5324**	5966.70	2586	6072.22	SPEA2	5413	5707.62	2554	3396.46
car6	8 × 9	FastPGA	**5486**	5941.00	**299**	299.00	GDE3	6519	6986.50	**299**	1988.33
		NSGA-II	5554	5819.00	**299**	299.00	PAES	5759	6362.76	**299**	304.78
		PESA-II	**5486**	5708.00	**299**	299.00	SPEA2	**5486**	5706.62	**299**	326.72
la06	15 × 5	FastPGA	**837**	1009.18	**3117**	4145.41	GDE3	888	929.20	3961	4825.80
		NSGA-II	**837**	895.34	3294	4053.81	PAES	862	927.85	3547	4066.98
		PESA-II	842	898.71	3550	3987.35	SPEA2	846	894.92	3306	3857.52
la11	20 × 5	FastPGA	1119	1229.64	8025	8725.36	GDE3	1184	1325.25	8458	9482.75
		NSGA-II	**1109**	1136.48	6946	7941.38	PAES	1112	1177.00	8384	9307.33
		PESA-II	1113	1175.38	7985	9539.12	SPEA2	**1109**	1187.33	**6881**	8483.49
la16	10 × 10	FastPGA	762	811.00	**49**	49.00	GDE3	918	976.00	119	338.60
		NSGA-II	**734**	849.03	**49**	68.78	PAES	789	955.26	**49**	90.10
		PESA-II	777	903.12	**49**	235.16	SPEA2	756	799.77	**49**	101.59
la24	15 × 10	FastPGA	1076	1123.91	577	1393.64	GDE3	1167	1291.60	1242	1917.50
		NSGA-II	**982**	1103.33	**236**	981.25	PAES	1090	1163.80	810	1596.50
		PESA-II	1015	1209.91	707	1257.36	SPEA2	988	1064.14	338	815.29
la29	20 × 10	FastPGA	1308	1459.40	4484	5525.80	GDE3	1400	1491.86	5212	6280.71
		NSGA-II	1295	1405.88	**3565**	4846.38	PAES	1321	1465.60	4102	6466.20
		PESA-II	1327	1385.75	4440	5455.25	SPEA2	**1287**	1378.38	3841	4935.00
la34	30 × 10	FastPGA	1923	1976.80	18,098	20715.80	GDE3	1985	2092.67	19,030	20106.67
		NSGA-II	**1879**	2007.00	17,420	18523.40	PAES	1940	2027.20	18,686	20933.40
		PESA-II	1897	2008.25	17,310	19976.62	SPEA2	1891	2008.71	**16,951**	18645.14
la35	30 × 10	FastPGA	1944	2082.17	17,706	20085.83	GDE3	2028	2186.25	18,729	19837.00
		NSGA-II	1912	2105.80	16,976	17887.00	PAES	2002	2090.70	18,711	20250.10
		PESA-II	1934	2021.38	16,918	19516.50	SPEA2	**1889**	2002.67	**16,796**	18612.83
la39	15 × 15	FastPGA	1381	1472.50	103	566.38	GDE3	1417	1514.00	362	990.89
		NSGA-II	**1280**	1441.86	**46**	364.14	PAES	1379	1554.86	125	702.86
		PESA-II	1343	1474.00	51	391.60	SPEA2	1320	1443.45	64	297.09
la40	15 × 15	FastPGA	1363	1473.33	42	458.00	GDE3	1413	1523.22	228	751.56
		NSGA-II	1290	1388.44	**20**	187.44	PAES	1397	1508.60	195	688.60
		PESA-II	1340	1518.25	24	228.12	SPEA2	**1287**	1458.83	35	176.67
mt10	10 × 10	FastPGA	744	826.92	**31**	72.20	GDE3	840	927.00	69	363.60
		NSGA-II	731	840.07	**31**	127.05	PAES	772	823.90	**31**	142.71
		PESA-II	**717**	795.99	**31**	43.60	SPEA2	742	743.00	**31**	31.00
mt20	20 × 5	FastPGA	1074	1182.30	6904	8276.90	GDE3	1118	1206.90	8127	9640.30
		NSGA-II	**1049**	1112.14	6450	7880.14	PAES	1086	1244.80	7670	8334.60
		PESA-II	1068	1104.19	6985	9331.56	SPEA2	1057	1114.87	**6236**	8508.13
orb8	10 × 10	FastPGA	664	710.14	**27**	57.86	GDE3	762	822.67	31	267.00
		NSGA-II	637	723.58	**27**	111.94	PAES	694	768.79	**27**	78.72
		PESA-II	672	746.18	**27**	63.67	SPEA2	**621**	695.68	**27**	46.47
orb9	10 × 10	FastPGA	764	824.51	**51**	112.70	GDE3	869	971.29	82	413.29
		NSGA-II	**726**	803.24	**51**	88.03	PAES	769	907.74	**51**	73.06
		PESA-II	750	777.23	**51**	131.14	SPEA2	730	852.64	**51**	136.28

the network and the algorithm adjusts the weights of each connection in order to reduce the value of the error function by some small amount. After repeating this process for a sufficiently large number of training cycles the network will usually converge to some state where the error of the calculations is small. To adjust weights we use a method for non-linear optimization that is called conjugate gradient. It works by iteratively computing search directions, along with a search line procedure that minimize the function, producing a new approximation to the (local) minimum of the objective function.

However, MLP networks trained with BP suffer some disadvantages: (i) they are easily trapped into local minima, (ii) they have slow convergence, and (iii) network topology must be determined by trial and error. To get around the first problem, the learning algorithm simply trained multiple nets and pick the best. The second and third problems are mitigated with a network optimization scheme. Specifically, the training method implements a network growing approach: the layout starts with a small network with only a single hidden unit. The network is trained until the improvement in the error over one epoch falls below some threshold. Then

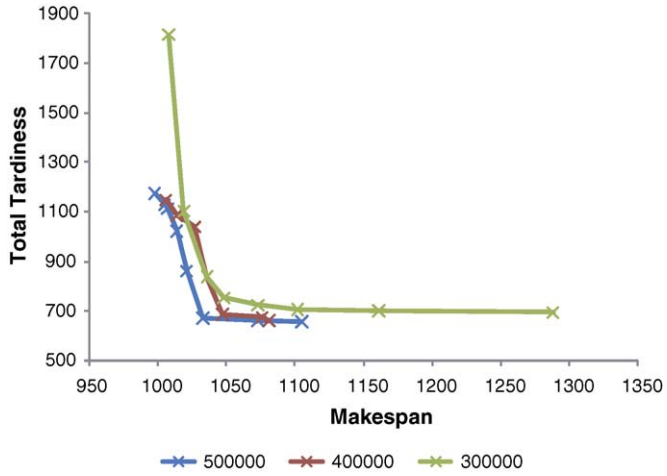


Fig. 10. Evolution of the NSGA-II Pareto front for the test problem *la24*.

the method adds an additional hidden unit, with weights from inputs and to outputs (weights have been randomly initialized) and resume training. The process continues until no significant gain is achieved by adding an extra unit.

6. Results

6.1. Machine scheduling

Our NSGA-II approach for scheduling has been validated with several benchmark problems of the classical JSSP. More specifically, we used the test suites created by Hurink et al. [55] that adapt original JSSP problems to flexible JSSP. In these problems, each operation has assigned a set of machines and not only the machine M_k used in the original problem. The problem data are available in [56]. The due dates of the benchmark problems have been defined according to [57]: jobs 2, 3 and 11 have a due date 1.5 times the corresponding processing time of the job; job n, where n is the number of jobs of the problem, has a due date equal to its processing time; and the remaining jobs have a due date 2 times the corresponding processing time.

We have compared our NSGA-II approach with other MOEAs.[1] Table 2[2] shows the best and average makespan and the best and average total tardiness for 10 runs of each approach. Each execution consisted of 500,000 iterations of a population of 200 individuals, with crossover and mutation rates set to 0.95 and 0.05, respectively.[3] In each test problem, the best makespan and best total tardiness of all the approaches is marked in boldface. Note that the number of iterations has been established taking the Pareto-front evolution into account. An example of this evolution for the test problem *la24* is depicted in Fig. 10. In this case we can see how the improvement is less than 1% between the fronts of the iteration 400,000 and 500,000.

[1] See Appendix A for a short description of the different MOEAs that have been compared with our approach.

[2] Results have been obtained using the software jMetal [58] which stands for meta-heuristic algorithms in Java, and it is an object-oriented Java-based framework aimed at the development, experimentation, and study of meta-heuristics for solving multi-objective optimization problems.

[3] The different values of the parameters (crossover rate, mutation rate, etc.) have been selected using standard common parameters that work well in most cases, instead of searching for very specific values to apply the GA to our specific problem. Moreover, we have set a large number of generations to allow the algorithm to achieve an appropriate convergence. No significant changes were achieved by increasing that number of generations or by reasonably changing these parameters.

Table 3
Comparison results between B =NSGA-II and the other methods.

Benchmark	Measure	A				
		FastPGA	GDE3	PAES	PESA-II	SPEA2
abz8	$\tilde{C}(A,B)$	0.26	0.00	0.00	0.70	0.95
	$\tilde{C}(B,A)$	1.00	1.00	1.00	0.97	0.85
abz9	$\tilde{C}(A,B)$	0.49	0.00	0.03	0.69	0.99
	$\tilde{C}(B,A)$	0.98	1.00	1.00	0.97	0.59
car5	$\tilde{C}(A,B)$	0.82	0.00	0.40	0.64	0.79
	$\tilde{C}(B,A)$	0.77	1.00	1.00	0.92	0.90
car6	$\tilde{C}(A,B)$	0.22	0.00	0.19	0.22	0.22
	$\tilde{C}(B,A)$	0.00	0.86	0.19	0.00	0.16
la06	$\tilde{C}(A,B)$	0.50	0.00	0.07	0.53	0.96
	$\tilde{C}(B,A)$	0.79	1.00	1.00	0.97	0.73
la11	$\tilde{C}(A,B)$	0.02	0.00	0.02	0.01	0.73
	$\tilde{C}(B,A)$	1.00	1.00	0.99	0.99	0.91
la16	$\tilde{C}(A,B)$	0.60	0.00	0.50	0.50	0.60
	$\tilde{C}(B,A)$	0.52	1.00	0.83	0.80	0.58
la24	$\tilde{C}(A,B)$	0.16	0.00	0.05	0.55	0.83
	$\tilde{C}(B,A)$	1.00	1.00	1.00	0.88	0.90
la29	$\tilde{C}(A,B)$	0.41	0.00	0.24	0.23	0.79
	$\tilde{C}(B,A)$	1.00	1.00	1.00	1.00	0.98
la34	$\tilde{C}(A,B)$	0.36	0.00	0.09	0.64	0.89
	$\tilde{C}(B,A)$	1.00	1.00	1.00	0.97	0.90
la35	$\tilde{C}(A,B)$	0.51	0.00	0.00	0.79	0.87
	$\tilde{C}(B,A)$	1.00	1.00	1.00	0.97	0.90
la39	$\tilde{C}(A,B)$	0.25	0.00	0.13	0.52	0.61
	$\tilde{C}(B,A)$	1.00	1.00	1.00	0.99	1.00
la40	$\tilde{C}(A,B)$	0.00	0.00	0.00	0.55	0.86
	$\tilde{C}(B,A)$	1.00	1.00	1.00	1.00	0.98
mt10	$\tilde{C}(A,B)$	0.47	0.00	0.22	0.57	0.47
	$\tilde{C}(B,A)$	0.40	1.00	0.41	0.64	0.58
mt20	$\tilde{C}(A,B)$	0.22	0.00	0.02	0.65	0.97
	$\tilde{C}(B,A)$	1.00	1.00	1.00	0.99	0.68
orb8	$\tilde{C}(A,B)$	0.18	0.00	0.06	0.18	0.30
	$\tilde{C}(B,A)$	0.72	1.00	0.75	0.68	0.60
orb9	$\tilde{C}(A,B)$	0.58	0.00	0.46	0.75	0.75
	$\tilde{C}(B,A)$	0.73	1.00	0.74	0.75	0.61

As shown in Table 2, NSGA-II obtains the best makespan and the best total tardiness on 11 of the 17 problems.

Table 3 summarizes the comparison results between NSGA-II and FastPGA, GDE3, PAES, PESA-II and SPEA2. The $\tilde{C}$ metric [57] is used to compare the approximate Pareto-optimal set between the NSGA-II and each of the MOEAs. More specifically, the $\tilde{C}(A,B)$ measures the fraction of members of the set B that are dominated by the members of the set A:

$$\tilde{C}(A,B) = \frac{|\{b \in B : \exists a \in A, a \succ b\}|}{|B|} \quad (8)$$

As shown in Table 3 NSGA-II outperforms most of other MOEAs in most of the situations. NSGA-II obtains lower $\tilde{C}$ metric than FastPGA on 13 problems, than GDE3 and PAES on all the problems, than PESA-II on 15 problems, and than SPEA2 on 10 of the 17 problems. We must remark that SPEA2, the second ranked algorithm, only outperforms clearly NSGA-II in the *abz9*, *la06*, *mt20*, and *orb9* cases. In the other three problems for which SPEA2 gets a better $\tilde{C}$ metric, the results of NSGA-II and SPEA2 are similar. To illustrate the convergence and diversity of the solutions, the non-dominated solutions of the final generation produced by NSGA-II and SPEA2 for the problem *la24* are presented in Fig. 11. Solutions of both algorithms are well spread and converged. However, NSGA-

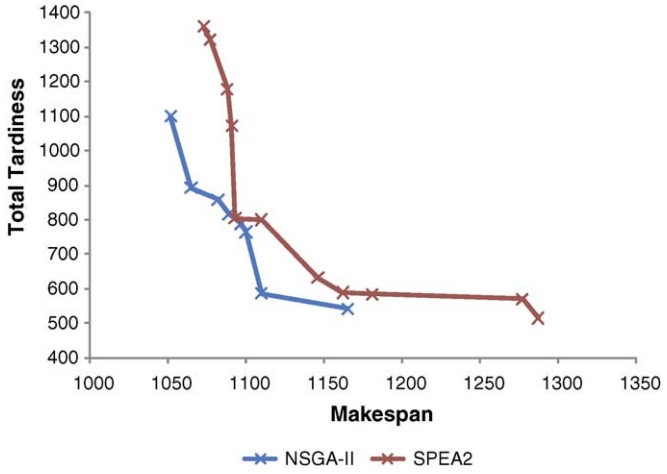


Fig. 11. Final Pareto fronts of NSGA-II and SPEA2 for the test problem *la24*.

II produces more non-dominated solutions for most of the test problems.

6.1.1. Discussion

Table 4 summarizes the characteristics of the different MOEAs. The second column indicates the way in which the fitness of each individual is calculated [59]. From the dominance point of view, the algorithms can be classified in three groups: *depth* (the front the individual belongs to), *rank* (number of individuals that dominate the individual), and *count* (number of individuals dominated by the individual). From the diversity point of view, the algorithms use two preservation mechanisms: crowding and fitness sharing. Also, fitness sharing is based on two criteria: nearest neighbor and histogram. Finally, the last column indicates the evolutionary operators that are used: crossover (c) and mutation (m).

Taking into account the characteristics of the algorithms and the results shown in Tables 2 and 3 for the benchmarks of JSSP, we can extract the following conclusions on the use of MOEAs for JSSP:

- The best MOEAs are NSGA-II and SPEA2, which agrees with previous knowledge on the performance of MOEAs in different types of problems.
- The MOEAs that use any dominance criteria (NSGA-II, SPEA2 and FastPGA) are superior to those that rely only on the dominance between pairs of individuals (PAES, PESA-II and GDE3).
- The dominance depth (NSGA-II), which only relies on the quality of the individual, works better (in most of the problems) than other dominance approaches (SPEA2 and FastPGA), which also take into account the solutions dominated and that dominate the individual. It seems that better results are obtained when the population has a high number of non-dominated individuals, although diversity could be low in the best fronts. This could be due to the huge size of the search space in JSSP, and the fact that the best individuals are obtained from small changes in former best individuals. This advantage of NSGA-II in most of the problems becomes a disadvantage in some of them, where more diversity in the best Pareto fronts should be necessary.

Table 4
Characteristics of the different MOEAs.

Algorithm	Fitness	Operators
NSGA-II [5]	Dominance depth and crowding	c + m
SPEA2 [60]	Dominance rank and count, and fitness sharing (nearest neighbors)	c + m
PAES [61]	(1+1), Pareto and fitness sharing (histogram)	m
PESA-II [62]	Pareto and fitness sharing (histogram)	c + m
GDE3 [63]	(1+1), Dominance rank and crowding	c + m
FastPGA [64]	Dominance rank and count, and crowding	c + m

Table 5
Results of the fivefold cross-validation for machine CSLB.

Method	MSE_{tra}		MSE_{tst}	
	$\bar{x}$	σ	$\bar{x}$	σ
MOGUL-TSK	5683.54	588.31	7330.85	1183.87
M5	2737.21	276.89	3252.85	735.89
QLMS	**2510.29**	124.59	**2549.84**	496.28
NN-MPCG	2762.84	146.48	2923.59	579.85

Table 6
Results of the fivefold cross-validation for machine RS-II.

Method	MSE_{tra}		MSE_{tst}	
	$\bar{x}$	σ	$\bar{x}$	σ
MOGUL-TSK	1497.49	50.56	1774.93	189.06
M5	**888.69**	37.72	1086.61	82.06
QLMS	1271.61	41.99	1301.65	173.51
NN-MPCG	950.42	96.80	**977.82**	90.40

Taking this into account, as future work it would be interesting to modify the fitness to include not only the dominance depth, but also other criteria to introduce more diversity in the best Pareto fronts. This is already done in the NSGA-II using the crowding distance, but it is only used when individuals of the last Pareto front have to be selected for the next population. The solution should be a midway between NSGA-II and SPEA2 fitness approaches.

6.2. Time estimation

Our neural network approach for processing time estimation has been validated with a subset of the machines that are currently being used in the production plans of a wood furniture industry. These machines are: rip saw for edging and ripping I (RS-I), abrasive calibrating machine (ACM), veneer slicers (VS), rip saw for edging and ripping II (RS-II), and commissioning system for large boards (CSLB). Data of 1,500 different custom pieces of furniture, that have been built in the factory along several years, have been used to generate the examples sets. The dimensions of each of the pieces was obtained and then, for each of the machines, the processing time was measured. These times are very noisy because many of the operations require some kind of manipulation by a human operator and, also, because this operator measures the times manually.

In each example the following variables are considered for each piece: length, width and thickness of the piece of furniture, as well as the measured processing time for that piece in the machine. The neural network has to minimize the error in time estimations. Experiments were performed with a fivefold cross-validation for each of the examples sets. Each set was divided in five subsets of equal size, and the learning process was run five times, using as training set four of the subsets, and as test set the remaining one. The test set was different in each of the runs.

We have compared our neural network approach (NN-MPCG [6]) with other regression techniques.[4] In the experiments, our approach was run with one hidden layer with 30 neurons. Tables 5–9 show, for each of the machines (or data sets), the mean square error of training (MSE_{tra}), and test (MSE_{tst}) of the fivefold

[4] The methodologies that have been compared with our approach are described in short in Appendix B.

Table 7
Results of the fivefold cross-validation for machine RS-I.

Method	MSE_{tra}		MSE_{tst}	
	$\bar{x}$	σ	$\bar{x}$	σ
MOGUL-TSK	6179.25	315.76	7433.03	1172.69
M5	4234.22	68.69	4512.69	354.77
QLMS	**4009.73**	116.56	**4066.40**	427.05
NN-MPCG	4215.76	152.32	4352.03	530.21

Table 8
Results of the fivefold cross-validation for machine VS.

Method	MSE_{tra}		MSE_{tst}	
	$\bar{x}$	σ	$\bar{x}$	σ
MOGUL-TSK	1080.58	50.50	1336.62	170.22
M5	953.66	23.26	**959.71**	94.12
QLMS	**952.19**	23.72	966.05	97.19
NN-MPCG	983.89	13.77	1012.28	120.13

Table 9
Results of the fivefold cross-validation for machine ACM.

Method	MSE_{tra}		MSE_{tst}	
	$\bar{x}$	σ	$\bar{x}$	σ
MOGUL-TSK	961.92	114.35	1277.09	217.76
M5	703.94	34.96	893.68	149.63
QLMS	1401.70	32.61	1439.56	127.17
NN-MPCG	**655.60**	21.54	**688.39**	212.08

cross-validation experiments for each technique.[5] In each table, the lower average values for MSE_{tra} and MSE_{tst} are marked in boldface.

The methods can be ranked in increasing order of accuracy: MOGUL-TSK, M5, QLMS, and NN-MPCG. If we compare the average values of MSE_{tst} for all the machines, M5 is the best method for machine VS, QLMS overcomes the other algorithms in two machines (RS-I and CSLB), and our NN-MPCG regression method is the best in another two machines (RS-II and ACM). Comparing our approach with the other proposals, the following discussion can be made: the error of MOGUL-TSK, taking into account MSE_{tst}, ranges between a 151% (CSLB, Table 5) and a 32% (VS, Table 8) higher than NN-MPCG. Also, for the M5 algorithm, the error ranges between a 30% higher (machine ACM, Table 9) and a 5% lower (VS, Table 8).

Finally, QLMS is better than our proposal in three of the machines, ranging from a 5% (VS, Table 8) to a 13% (CSLB, Table 5) lower error. However, in the other two machines the accuracy of the QLMS algorithm is very poor when compared with the NN-MPCG approach: the error is a 33% (RS-II, Table 6) and a 109% (ACM, Table 9) higher. In summary, our proposal (NN-MPCG) obtains a high time estimation accuracy with a good regularity over all the machines. This means that, although other approaches have a lower error in some of the machines, in the worst case the improvement is low. On the contrary, when NN-MPCG overcomes other methods the improvement is much higher.

7. Conclusions

A solution for multi-objective machine scheduling in the custom furniture industry has been presented. The solution is based on a multi-objective evolutionary approach together with a neural network. The system uses the NSGA-II algorithm to produce reliable schedules which are optimized on the basis of their makespan and total tardiness. Moreover, processing time estimations have been obtained through a multilayer perceptron neural network.

The system has been extensively tested and compared with other approaches. Our NSGA-II-based algorithm has been compared with another five MOEAs on 17 different classical JSSP, outperforming the other algorithms in most of the problems. Also, processing time estimations through the multilayer perceptron neural network has shown a much better accuracy than the other regression techniques that have been analyzed. Moreover, tests also demonstrated a time efficient response both to a huge work load and to changes in the environment, such as the definition of new client orders or changes in production due dates.

Acknowledgments

Authors wish to thank Reza Sadigh Balay for his help, support, and valuable hints. This work was supported in part by the Spanish Ministry of Science and Innovation under grants TIN2008-00040 and TSI2007-65677C02-1. Manuel Mucientes is supported by the *Ramón y Cajal* program of the Spanish Ministry of Science and Innovation.

Appendix A. Description of other MOEAs

The MOEAs that have been compared with our NSGA-II-based approach are:

- FastPGA [64]: the Fast Pareto Genetic Algorithm uses a new fitness assignment and ranking strategy where each solution evaluation is relatively computationally expensive. This is often the case when there are time or resource constraints involved in finding a solution. A population regulation operator is introduced to dynamically adapt the population size as needed up to a user-specified maximum population size.
- GDE3 [63]: Generalized Differential Evolution 3 is an extension of Differential Evolution (DE) for global optimization with an arbitrary number of objectives and constraints. GDE3 improves earlier GDE versions in the case of multi-objective problems by giving a better distributed solution.
- PAES [61]: the Pareto Archived Evolution Strategy consists of a 1 + 1 evolution strategy (i.e., a single parent that generates a single offspring) in combination with a historical archive that records the non-dominated solutions previously found. This archive is used as a reference set against which each mutated individual is being compared. Such a historical archive is the elitist mechanism adopted in PAES.
- PESA-II [62]: improved variant of PAES in which the unit of selection is a hyperbox in objective space. In this technique, instead of assigning a selective fitness to an individual, selective fitness is assigned to the hyperboxes in objective space which are currently occupied by at least one individual in the current approximation to the Pareto frontier. A hyperbox is thereby selected, and the resulting selected individual is randomly chosen from this hyperbox.
- SPEA2 [60]: the Strength Pareto Evolutionary Algorithm II assigns a fitness value to each individual that is the sum of its strength raw fitness and a density estimation. The algorithm applies the selection, crossover, and mutation operators to fill an archive of individuals; then, the non-dominated individuals of both the original population and the archive are copied into a new population. If the number of non-dominated individuals is greater than the population size, a truncation operator based on calculating the distances to the kth nearest neighbor is used. This way, the individuals having the minimum distance to any other individual are chosen.

[5] Results have been obtained using the software KEEL [65].

Appendix B. Description of other regression techniques

The processing time estimation techniques that have been compared with our approach are:

- MOGUL-TSK [66]: a two-stage evolutionary algorithm based on MOGUL (a methodology to obtain Genetic Fuzzy Rule-Based Systems under the Iterative Rule Learning approach). The first stage performs a local identification of prototypes to obtain a set of initial local semantics-based TSK rules, following the Iterative Rule Learning approach and based on an evolutionary generation process within MOGUL. Then, a postprocessing stage is applied. It consists in a genetic niching-based selection process to remove redundant rules and a genetic tuning process to refine the fuzzy model parameters. The method was run with the standard parameter values, and the initial partition of each variable was of five labels.
- M5 [67,68]: a technique for regression using model trees (decision trees for regression). In the first stage, a decision tree induction algorithm is used. The splitting criterion tries to minimize the intra-subset variation in the class values down each branch. The second stage is pruning, and consideration is given to replacing a node by a regression plane instead of a constant value. Attributes that define that regression are those that participate in decisions in nodes subordinate to the current one. The method was run with a pruning factor of 2.
- QLMS [69]: an statistical model for regression using a quadratic combination of the features. The weights of such combination are fitted as a quadratic discriminant using least mean squares.

References

[1] M.R. Garey, D.S. Johnson, R. Sethi, The complexity of flowshop and jobshop scheduling, Mathematics of Operational Research 1 (2) (1976) 117–129.
[2] J. Andersson, Multiobjective optimization in engineering design, PhD dissertation, Linköpings University, Linköping, Sweden (2001).
[3] C.A. Coello Coello, Recent trends in evolutionary multiobjective optimization, in: Evolutionary Multiobjective Optimization: Theoretical Advances And Applications, Springer-Verlag, 2005, pp. 7–32.
[4] C.A. Brizuela Rodríguez, Genetic algorithms for shop scheduling problems: partial enumeration and stochastic heuristics, PhD dissertation, Kyoto Institute of Technology, Japan (2000).
[5] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Transactions on Evolutionary Computation 6 (2) (2002) 182–197.
[6] F. Moller, A scaled conjugate gradient algorithm for fast supervised learning, Neural Networks 6 (1990) 525–533.
[7] R. Vaessens, Generalized job shop scheduling: complexity and local search, Ph.D. dissertation, Eindhoven University of Technology, Netherlands (1995).
[8] P. Brucker, B. Jurish, B. Sieners, A branch and bound algorithm for the job-shop scheduling problem, Discrete and Applied Mathematics 49 (1–3) (1994) 107–127.
[9] J. Carlier, E. Pinson, An algorithm for solving the job-shop scheduling problem, Management Science 35 (2) (1989) 164–176.
[10] A.S. Jain, S. Meeran, Deterministic job-shop scheduling: past, present and future, European Journal of Operational Research 113 (1999) 390–434.
[11] M. Ballicu, A. Giua, C. Seatzu, Job-shop scheduling models with set-up times, in: Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, IEEE Service Center, 2002, pp. 95–100.
[12] E. Balas, N. Simonetti, A. Vazacopoulos, Job shop scheduling with setup times, deadlines and precedence constraints, in: Proceedings of the 2ndd Multidisciplinary International Conference on Scheduling: Theory and Application, 2005, pp. 520–532.
[13] M. Zweben, M.S. Fox, Intelligent Scheduling, Morgan Kaufmann, 1994.
[14] V. Subramaniam, T. Ramesh, G.K. Lee, Y.S. Wong, G.S. Hong, Job shop scheduling with dynamic fuzzy selection of dispatching rules, International Journal of Advanced Manufacturing Technology 16 (10) (2000) 759–764.
[15] S. Feng, L. Li, L. Cen, J. Huang, Using mlp networks to design a production scheduling system, Computers and Operations Research 30 (6) (2003) 821–832.
[16] J.-P. Watson, J.C. Beck, A.E. Howe, L.D. Whitley, Problem difficulty for tabu search in job-shop scheduling, Artificial Intelligence 143 (2) (2003) 189–217.
[17] E. Nowicki, C. Smutnicki, A fast taboo search algorithm for the job-shop scheduling problem, Management Science 42 (6) (1996) 797–813.
[18] P.J.M. van Laarhoven, E.H.L. Aarts, J.K. Lenstra, Job shop scheduling by simulated annealing, Operations Research 40 (1) (1992) 113–125.
[19] E.H.L. Aarts, P.J.M. van Laarhoven, J.K. Lenstra, N.L.J. Ulder, A computational study of local search algorithms for job shop scheduling, ORSA Journal on Computing 6 (2) (1994) 118–125.
[20] C.Y. Zhang, P. Li, Y. Rao, Z. Guan, A very fast ts/sa algorithm for the job shop scheduling problem, Computers & Operations Research 35 (1) (2008) 282–294.
[21] J. Adams, E. Balas, D. Zawack, The shifting bottleneck procedure for job shop scheduling, Management Science 34 (3) (1988) 391–401.
[22] E. Hart, P. Ross, D. Corne, Evolutionary scheduling: a review, Genetic Programming and Evolvable Machines 6 (2) (2005) 191–220.
[23] S.G. Ponnambalam, P. Aravindan, P. Rao, Comparative evaluation of genetic algorithms for job-shop scheduling, Production Planning and Control 12 (6) (2001) 560–574.
[24] S. Uckum, S. Bagachi, K. Kawamura, Managing genetic search in job shop scheduling, IEEE Expert: Intelligent Systems and Their Applications 8 (5) (1993) 15–24.
[25] K. Hamada, T. Baba, K. Sato, M. Yufu, Hybridizing a genetic algorithm with rule-based reasoning for production planning, IEEE Expert: Intelligent Systems and Their Applications 10 (5) (1995) 60–67.
[26] J. Branke, D.C. Mattfeld, Anticipation in dynamic optimization: the scheduling case, in: Proceedings of the 6th International Conference on Parallel Problem Solving from Nature, Springer-Verlag, London, UK, 2000, pp. 253–262.
[27] V.J. Leon, S.D. Wu, R.H. Storer, Robustness measures and robust scheduling for job shops, IIE Transactions 26 (5) (1994) 32–43.
[28] P.-C. Chang, J.-C. Hsieh, C.-Y. Wang, Adaptive multi-objective genetic algorithms for scheduling of drilling operation in printed circuit board industry, Applied Soft Computing 7 (3) (2007) 800–806.
[29] H. Tamaki, E. Nishino, S. Abe, A genetic algorithm approach to multi-objective scheduling problems with earliness and tardiness penalties, in: Proceedings of the 1999 Congress on Evolutionary Computation, IEEE Service Center, Washington D.C., USA, 1999, pp. 839–848.
[30] P. Sridhar, C. Rajendran, Scheduling in flowshop and cellular manufacturing systems with multiple objectives—a genetic algorithmic approach, Production Planning and Control 7 (4) (1996) 374–382.
[31] S.G. Ponnambalam, V. Ramkumar, N. Jawahar, A multiobjective genetic algorithm for job shop scheduling, Production Planning and Control 12 (8) (2001) 764–774.
[32] L.-N. Xing, Y.-W. Chen, K.-W. Yang, Multi-objective flexible job shop schedule: design and evaluation by simulation modelling, Applied Soft Computing 9 (1) (2009) 362–376.
[33] C.M. Fonseca, P.J. Fleming, Genetic algorithms for multiobjective optimization: formulation, discussion and generalization, in: Proceedings of the Fifth International Conference on Genetic Algorithms, IEEE Service Center, 1993, pp. 416–423.
[34] J. Horn, N. Nafpliotis, D.E. Goldberg, A niched pareto genetic algorithm for multiobjective optimization, in: Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence, IEEE Service Center, 1994, pp. 82–87.
[35] D. Corne, J.D. Knowles, M.J. Oates, The pareto envelope-based selection algorithm for multi-objective optimisation, in: Proceedings of the 6th International Conference on Parallel Problem Solving from Nature, Springer-Verlag, London, UK, 2000, pp. 839–848.
[36] E. Zitzler, L. Thiele, Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach, IEEE Transactions on Evolutionary Computation 3 (4) (1999) 257–271.
[37] S. Binato, W.J. Hery, D.M. Loewenstern, M.G.C. Resende, A GRASP for job shop scheduling, in: Essays and Surveys on Metaheuristics, Kluwer Academic Publishers, 2001, pp. 59–79.
[38] N. Liouane, I. Saad, Ant systems & local search optimization for flexible job shop scheduling production, International Journal of Computers, Communications & Control 2 (2) (2007) 174–184.
[39] C.S. Chong, A.I. Sivakumar, M.Y.H. Low, K.L. Gay, A bee colony optimization algorithm to job shop scheduling, in: WSC'06: Proceedings of the 38th Conference on Winter simulation, Winter Simulation Conference, 2006, pp. 1954–1961.
[40] K. Shah, N. Ripon, An evolutionary approach for solving the multi-objective job-shop scheduling problem, in: Evolutionary Scheduling, Springer, 2007, pp. 165–195.
[41] J. Lee, K. Lee, H. Park, J. Hong, J. Lee, Developing scheduling systems for Daewoo Shipbuilding: DAS project, European Journal of Operational Research 97 (2) (1997) 380–395.
[42] M. Wilhelm, A. Smith, B. Bidanda, Integrating an expert system and a neural network for process planning, Engineering Design and Automation 1 (4) (1995) 259–269.
[43] M. Mucientes, J.C. Vidal, A. Bugarín, M. Lama, Processing time estimations by variable structure TSK rules learned through genetic programming, Soft Computing 13 (5) (2009) 497–509.
[44] J.W. Herrmann, M.M. Chincholkar, Reducing throughput time during product design, Journal of Manufacturing Systems 20 (6) (2001) 416–428.
[45] A. Kusiak, W. He, Design of components for schedulability, European Journal of Operational Research 164 (1999) 185–194.
[46] S.K. Gupta, D.S. Nau, A systematic approach for analyzing the manufacturability of machined parts, Computer Aided Design 27 (5) (1995) 323–342.
[47] I. Minis, J.W. Herrmann, G. Lam, E. Lin, A generative approach for concurrent manufacturability evaluation and subcontractor selection, Journal of Manufacturing Systems 18 (6) (1999) 383–395.

[48] A. Allahverdi, T.C.E. Ng, C.T. Cheng, M.Y. Kovalyov, A survey of scheduling problems with setup times or costs, European Journal of Operational Research 187(3) (2008) 985–1032.
[49] T.C.E. Cheng, Q. Ding, B.M.T. Lin, A concise survey of scheduling with time-dependent processing times, European Journal of Operational Research 152 (2004) 1–13.
[50] D. Shabtay, G. Steiner, A survey of scheduling with controllable processing times, European Journal of Operational Research 155 (2007) 1643–1666.
[51] G. Boothroyd, P. Dewhurst, W. Knight, Product Design for Manufacture and Assembly, Marcel Dekker, 1994.
[52] K. Mesghouni, S. Hammadi, P. Borne, Evolutionary algorithms for job-shop scheduling, International Journal of Applied Mathematics and Computer Science 14 (1) (2004) 93–103.
[53] B. Giffler, G.L. Thompson, Algorithms for solving production scheduling problems, Operations Research 8 (4) (1960) 487–503.
[54] S.I. Gallant, Perceptron-based learning algorithms, IEEE Transactions on Neural Networks 1 (2) (1990) 179–191.
[55] E. Hurink, B. Jurish, M. Thole, Tabu search for the job shop scheduling problem with multi-purpose machine, Operations Research Spektrum 15 (1994) 205–215.
[56] M. Mastrolilli, L.M. Gambardella, Effective neighborhood functions for the flexible job shop problem, Journal of Scheduling 3 (1) (1998) 3–20.
[57] D. Lei, Z. Wu, Crowding-measure-based multiobjective evolutionary algorithm for job shop scheduling, International Journal of Advanced Manufacturing Technology 30 (1–2) (2006) 112–117.
[58] J.J. Durillo, A.J. Nebro, F. Luna, B. Dorronsoro, E. Alba, jMetal: a java framework for developing multi-objective optimization metaheuristics, Tech. Rep. ITI-2006–10, Departamento de Lenguajes y Ciencias de la Computación, University of Málaga, E.T.S.I. Informática, Campus de Teatinos, Available at http://jmetal.sourceforge.net/index.html (December 2006).
[59] C. Coello, G. Lamont, D. Van Veldhuizen, Evolutionary algorithms for solving multi-objective problems, Springer-Verlag, 2007.
[60] E. Zitzler, M. Laumanns, L. Thiele, Spea2: improving the strength pareto evolutionary algorithm, Tech. Rep. 103, Swiss Federal Institute of Technology (ETH) (2001).
[61] J.D. Knowles, D.W. Corne, Approximating the nondominated front using the pareto archived evolution strategy, Evolutionary Computation 8 (2) (2000) 149–172, doi: http://dx.doi.org/10.1162/106365600568167.
[62] D.W. Corne, N.R. Jerram, J.D. Knowles, M.J. Oates, PESA-II: region-based selection in evolutionary multiobjective optimization, in: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001), Morgan Kaufmann, San Francisco, California, USA, 2001, pp. 283–290.
[63] S. Kukkonen, J. Lampinen, Gde3: the third evolution step of generalized differential evolution, in: Proceedings of the 2005 Congress on Evolutionary Computation (CEC 2005), IEEE Service Center, Edinburgh, Scotland, 2005, pp. 443–450.
[64] H. Eskandari, C.D. Geiger, A fast pareto genetic algorithm approach for solving expensive multiobjective optimization problems, Journal of Heuristics 14 (3) (2008) 203–241, doi: http://dx.doi.org/10.1007/s10732–007-9037-z.
[65] J. Alcalá-Fdez, L. Sánchez, S. García, M. del Jesus, S. Ventura, J. Garrell, J. Otero, C. Romero, J. Bacardit, V. Rivas, J. Fernández, F. Herrera, Keel: a software tool to assess evolutionary algorithms to data mining problems, Soft Computing 13 (3) (2009) 307–318.
[66] R. Alcalá, J. Alcalá-Fdez, J. Casillas, O. Cordón, F. Herrera, Local identification of prototypes for genetic learning of accurate TSK fuzzy rule-based systems, International Journal of Intelligent Systems 22 (2007) 909–941.
[67] J. Quinlan, Learning with continuous classes, in: Proceedings of the 5th Australian Joint Conference on Artificial Intelligence, Singapore, 1992, pp. 343–348.
[68] I. Wang, I. Witten, Induction of model trees for predicting continuous classes, in: Proceedings of the 9th European Conference on Machine Learning, Prague (Czech Republic), 1997, pp. 128–137.
[69] J. Rustagi, Optimization Techniques in Statistics, Academic Press, 1994.