Undefined 1 (2015) 1–5
IOS Press

Search spaces and genetic algorithms for the scheduling problem with skilled operators

Raúl Mencía [a], María R. Sierra [a], Carlos Mencía [b], Ramiro Varela [a,*]
[a] *Department of Computer Science, University of Oviedo, Spain*
[b] *CASL, University College Dublin, Ireland.*

Abstract. Scheduling problems involving physical machines and human resources are frequent in real production environments. In this paper, we tackle a problem of this class that was recently proposed and is motivated by a real handicraft company. In this problem, a set of tasks must be performed on a set of machines under the assistance of human operators, subject to some constraints such as precedence relations on the tasks, limited capacity of machines and operators, and skills of the operators to assist the processing of tasks. To solve this problem, we first defined a disjunctive model to represent problem instances and solutions. This model was then used to devise and analyze the properties of a generic schedule builder that may be particularized to build schedules in different search spaces. The schedule builder was finally exploited as a decoder in a genetic algorithm which also incorporates other problem-dependent elements such as an encoding schema and genetic operators. All the proposals were evaluated on a benchmark set with instances of different characteristics. The experimental study revealed useful insights of practical interest and showed substantial improvements of the genetic algorithm over existing methods in the literature.

Keywords: scheduling, genetic algorithms, heuristics, schedule generation schemes, constraint satisfaction and optimization

1. Introduction

Scheduling problems arise routinely in a growing number of domains, including engineering, management science or distributed and parallel computing, to mention a few. The practical significance of scheduling, and the fact that many of the most representative scheduling problems are computationally intractable [1], have attracted a large body of research since the early 60s, especially from areas such as operational research and artificial intelligence. As a result, ever more efficient methods capable of solving ever more challenging scheduling problems have been proposed in the literature [2,3,4].

In the last decade, a considerable effort has been made to extend scheduling models and algorithms in order to capture and deal with complex constraints and features present in real-life environments. Notable examples of this are scheduling problems considering setup times in the use of resources [5,6], uncertainty in the duration of the tasks [7,8], flexibility in machines [9,10] or constraints derived from the presence of human operators [11,12,13,14,15,16,17].

This paper is concerned with the last case. Arguably, human operators play a fundamental role in numerous production environments, and so their presence imposes a number of constraints that need to be taken into account. For example, it may be the case that an organization counts on only a few operators to assist the processing of tasks on (many) different machines. This situation may have a significant impact in the production process, as, in principle, at most as many machines as the number of operators can be working in parallel. This assumption was considered in [11,18], where an extension of the classic job-shop scheduling problem (JSP), termed job-shop scheduling with operators (JSO), was proposed. While the JSO problem is general enough to model a variety of real-life situations and particular enough to allow for the development of efficient algorithms, it does not make any distinction

[*]Corresponding author: Ramiro Varela, Department of Computer Science, University of Oviedo, 33204 Gijón, Spain, E-mail: ramiro@uniovi.es.

between the operators involved in the process. As recently noted in [12], this limitation represents an important drawback in environments where the processing of tasks require expert workers.

As a solution, the scheduling problem with arbitrary precedence relations and skilled operators (SPSO) was defined in [12]. In this problem, a set of tasks must be performed on a set of physical machines and the processing of a particular task on a machine has to be assisted by an operator skilled to assist that task. The number of operators may be scarce w.r.t. the number of tasks and machines, and each operator is skilled to assist only a subset of tasks. There are also capacity constrains of the operators and the machines by which an operator and a machine must be allocated to a task over its whole processing time and cannot be allocated to more than one task simultaneously. In addition, there are arbitrary precedence relations on the processing order of tasks. The problem is NP-*hard*, as it generalizes the classic job-shop scheduling problem (JSP).

In [12], the SPSO was motivated by a real-life handicraft company where the operators have different expertise. For example, an apprentice may be able to perform only a limited subset of single assembly tasks while the most difficult operations may require more experienced workers. Therefore, finding good solutions for the SPSO is an issue of major interest for human resources management as it would allow the company to make the best use of its employees and to plan training or recruitment actions for future projects.

To solve the SPSO, two heuristic algorithms were proposed in [12], which were evaluated across a benchmark set defined in accordance with the characteristics of the problems of the handicraft company. These algorithms are termed STA-OMSB and MSB-DOS respectively, and both of them exploit the well-known shifting bottleneck heuristic [19]. The results of these algorithms were compared with those obtained by Cplex MILP solver from a formulation proposed in the same work.

In this paper we make several contributions towards understanding and solving the SPSO problem. Firstly, we define a disjunctive graph model, which makes it easier to reason about the problem. Then, we explore different ways of obtaining solution spaces. Concretely, we propose a general schedule builder, termed $SOG\&T$, and particularize it in different ways giving rise to a number of dominant and non-dominant search spaces. The hardness of the SPSO motivates the use of approximate algorithms. Accordingly, we propose a genetic algorithm that exploits the $SOG\&T$ and incorporates a novel problem-dependent coding scheme. The results from an experimental study indicate that our approach is very effective and outperforms previous methods in the state of the art for this problem.

The remaining of the paper is organized as follows. In the next section, we give the formal definition of the SPSO. Then, in Section 3 we propose a disjunctive graph model to represent problem instances and solutions. In Section 4 we describe and formalize the general schedule builder termed $SOG\&T$, and show how it can be particularized to build schedules in different search spaces. Section 5 describes the proposed genetic algorithm. The results of the experimental study are reported in Section 6. Finally, in Section 7, we summarize the main conclusions of the paper and outline some ideas for future work.

2. Problem formulation

In the scheduling problem with skilled operators and arbitrary precedence relations, we are given

- $\mathcal{M}$, a set of q machines.
- $\mathcal{O}$, a set of p operators.
- $\mathcal{T}$, a set of n tasks or operations.
- The processing time p_u (integer) for each task $u \in \mathcal{T}$.
- The machine $m_u \in \mathcal{M}$ on which the task u must be processed.
- $\mathcal{O}_u \subseteq \mathcal{O}$, the set of operators that are skilled to assist the task u.
- $(\mathcal{T}, E)$, the *task graph* that expresses precedence relations on the processing of tasks.

The objective is to allocate a starting time st_u and an operator o_u to assist the processing of each task $u \in \mathcal{T}$, such that the makespan, defined as

$$C_{max} = \max\{C_u; u \in \mathcal{T}\} \qquad (1)$$

is minimized, where C_u denotes the completion time of the operation u, and the following constraints are satisfied:

- Unary constraints. For every $u \in \mathcal{T}$:
  - $i.$ It must be assisted by a skilled operator; i.e., $o_u \in \mathcal{O}_u$, and the operator is allocated to the task over its whole processing time.
  - $ii.$ It cannot be preempted; i.e., $C_u = st_u + p_u$.
- Binary constraints. For every two tasks $u, v \in \mathcal{T}$:

$iii.$ They must be processed following the order expressed by the task graph; i.e., if $(u, v) \in E$ then $st_u + p_u \leq st_v$.

$iv.$ If they are assisted by the same operator or processed on the same machine they cannot overlap; i.e., if $(o_u = o_v) \vee (m_u = m_v)$ then $(st_u + p_u \leq st_v) \vee (st_v + p_v \leq st_u)$.

This problem was firstly defined in [12] and denoted JSSO (Job Shop Scheduling with Skilled Operators). We prefer to use the nomenclature SPSP (Scheduling Problem with Skilled Operators) as the term Job Shop is often used in the literature to refer to a particular task graph structure where the tasks are organized into sequences that are called *jobs*. As mentioned earlier, the SPSO problem generalizes the JSP as well as the Job Shop Scheduling with Operators (JSO) defined in [11], and at the same time it particularizes more general models, such as the Multi-mode Resource Constrained Project Scheduling Problem (MRCPSP).

3. A disjunctive graph model

Scheduling problems are usually represented by means of a disjunctive graph model [20]. In this kind of modeling two types of arcs are commonly used. Conjunctive arcs between every two consecutive tasks in the task graph, and also disjunctive arcs between every two tasks that require or may require the same resource. Disjunctive graphs enable reasoning about scheduling problems in terms of graph-theoretical notions, which usually turns out to be very useful in devising efficient solving methods. For example, they allow for solving scheduling problems by deciding about the relative order among tasks requiring the same resource, instead of considering for every task all its possible starting times. We propose to use the following model for the SPSO, which extends the one used used in [21] for the JSO.

A problem instance is represented by a directed graph $G = (V, E \cup D \cup O \cup I)$ where:

- Each node in V represents either a task in $\mathcal{T}$, or one of the fictitious tasks with null processing time; namely, starting tasks for each operator $o \in \mathcal{O}$, and the dummy operations *start* and *end*.
- E is the set of arcs of the task graph also called *conjunctive arcs*. $P(v)$ and $S(v)$ will denote the sets of tasks which are predecessors and successors of v respectively in the task graph.
- D is the set of *disjunctive arcs* which represent capacity constraints of the machines. D is partitioned into subsets D_j with $D = \cup_{j=1,\dots,q} D_j$, D_j including an arc (v, w) for each pair of operations requiring machine M_j.
- O is the set of *operator arcs* and includes two types of arcs: one arc (u, v) for each pair of operations of the problem such that $\mathcal{O}_u \cap \mathcal{O}_v \neq \emptyset$, and arcs (o, u) for each operator node o and task u such that $o \in \mathcal{O}_u$.
- The set I includes arcs connecting node *start* to each node $o \in O$ and arcs connecting each task without successors in the task graph to node *end*.

Each arcs is weighted with the processing time of the task at the outgoing node.

From this representation, building a schedule can be seen as a process of fixing disjunctive and operator arcs. A disjunctive arc between operations u and v gets fixed when either (u, v) or (v, u) is selected and so the other one discarded. If the operator arc (o, u) is fixed, the task u is assisted by o, and consequently all arcs (o', u) for o' other than o are discarded. Also, if the operator arc (u, v) is fixed then u and v are assisted by the same operator, u before v. In this case, the operator arc (v, u) and the remaining operator arcs connecting u or v to operations assisted by operators other than o are discarded. So, discarding both arcs (u, v) and (v, u) means that u and v will be assisted by different operators.

A feasible schedule S is represented by an acyclic subgraph of G, of the form $G_S = (V, E \cup F \cup Q \cup I)$, where $F \subset D$ expresses the processing order of tasks on the machines and $Q \subset O$ expresses the sequences of tasks that are assisted by each operator. In other words:

- $F = \cup_{j=1,\dots,q} F_j$, $F_j \subset D_j$ such that $(u, v) \in F_j$ if and only if $m_u = m_v$ and u is processed before v in S. So, F_j represents the machine sequence or processing order of tasks on machine M_j.
- $Q = \cup_{o=1,\dots,p} Q_o$, where Q_o represents the operator sequence of o, and includes the arcs (o, u), (o, v) and (u, v) for each pair of operations assisted by the operator o such that u is processed before v. Each operation u must be included in one and only one operator sequence.

A *critical path* is a longest cost path in G_S from node *start* to node *end*. The *head* r_v of an operation v is the cost of the largest cost path from node *start* to node v and defines a lower bound for st_v. For

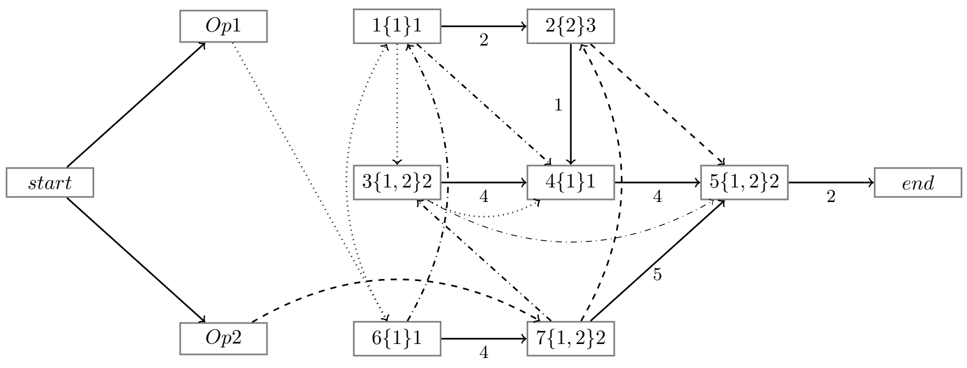


Fig. 1. A solution graph for a problem instance with 7 tasks, 3 machines and 2 operators. Each task node includes the task number, the set of skilled operators and the machine using the notation $u\mathcal{O}_u m_u$. Dotted and dashed arcs represent operator sequences starting in an operator node, while dotted-dashed arcs represent machine sequences. For the sake of clarity, only the arcs between consecutive nodes in the relations are represented and only the cost of the arcs in the task graph are displayed. Every arc should be labeled with the processing time of the task at the outgoing node. The makespan is 19 and corresponds to the cost of the critical path ($start$ Op1 6 7 3 4 5 end).

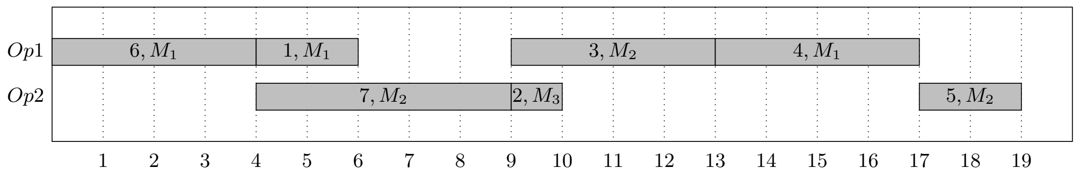


Fig. 2. A Gantt chart that represents the same schedule as that in Fig. 1.

most regular objective functions, in particular for the makespan, taking $st_v = r_v$ produces the optimal value restricted to the processing ordering defined by the solution graph. In this case, the cost of a critical path is the makespan of the schedule S denoted $C_{max}(S)$.

Figure 1 shows a solution graph for a problem instance with 7 tasks, 3 machines and 2 operators. The same schedule is also represented by means of Gantt chart in Figure 2.

A partial schedule is given by a subgraph of G where some of the disjunctive and operator arcs have not been fixed yet. In such a schedule $PM(v)$ denotes the disjunctive predecessors of v; $w \in PM(v)$ means that $m_w = m_v$ and that the disjunctive arc (w, v) was fixed (analogously, $SM(v)$ denotes the disjunctive successors of v). $PO(v)$ denotes the operator predecessors of v, i.e., $w \in PO(v)$ if the operator arc (w, v) was fixed (analogously, $SO(v)$ are the operator successors of v).

4. Schedule generation scheme and search spaces

In this section we propose a new schedule generation scheme for the SPSO which is a generalization of the well-known $G\&T$ algorithm proposed in [22] for the classic JSP and also of the $OG\&T$ algorithm proposed in [21] for the JSO. The new schedule builder is denoted $SOG\&T$ (Skilled Operators $G\&T$). Besides, we will see how the $SOG\&T$ may be particularized to search over different search spaces. We will consider and analyze two kinds of them: dominant and non-dominant search spaces.

4.1. The SOG&T schedule builder

This algorithm iterates over n steps and in each iteration one of the tasks is scheduled following an order compatible with the partial ordering defined by the task graph. So, by the time the task u is scheduled, all tasks $v \in P(u)$ have already been scheduled. At this time, u is allocated an operator $o_u \in \mathcal{O}_u$ and a starting time st_u such that it is scheduled after all the scheduled tasks that require the same machine m_u or that were allocated the same operator o_u, i.e., it is an *appending* scheme.

Let SC be the set of scheduled operations before the current iteration and let $G_{SC} = (V, E \cup H \cup R \cup I)$ be the partial solution graph built so far. For all $u, v \in SC$ such that $m_u = m_v$, either (u, v) or (v, u) are in H. Also, for all $u \in SC$, (o, u) is in R for some $o \in \mathcal{O}$, and if (o, v) is also in R then either (u, v) or (v, u) are in R as well.

The *set of eligible operations* in the current iteration is defined as:

$$A = \{v; v \notin SC, P(v) \subseteq SC\} \tag{2}$$

A includes the first unscheduled tasks in the task graph. In principle, each task in A is a candidate to be scheduled next and it can be assisted by any of the operators skilled to do it. So, the *set of scheduling options* is defined as:

$$\mathcal{A} = \{(u, o); u \in A, o \in \mathcal{O}_u\} \tag{3}$$

If the option $(u, o) \in \mathcal{A}$ is selected in the current iteration, then the earliest starting time of u is given by

$$st_u(o) = \max\{st_x + p_x, st_y + p_y, st_z + p_z\} \tag{4}$$

where

- x is the task in $P(u)$ with the largest completion time; i.e.,

$$x = argmax\{st_v + p_v; v \in P(u)\}, \tag{5}$$

- y is the last scheduled task on the machine required by u and
- z is the last scheduled task assisted by the operator o.

The option (u, o) establishes the time interval $[st_u(o), st_u(o) + p_u)$ for processing u under the assistance of the operator o. Therefore, u is scheduled in such way that it satisfies all four constraints w.r.t. all previous scheduled operations and so we can establish the following result that guarantees the soundness of the $SOG\&T$ schedule builder.

Proposition 1. *After each iteration of the SOG&T algorithm, for all tasks in $SC \subseteq \mathcal{T}$, constraints i., ii., iii., and iv. are satisfied.*

It becomes clear that the set $\mathcal{A}$ may be considerably large, in particular for large instances. So, we may consider reducing the options to some subset $\mathcal{X} \subseteq \mathcal{A}$. To do this, we may observe that taking the option (u, o), u is given a starting time $st_u \in [T_{\mathcal{A}}, C)$ where

$$T_{\mathcal{A}} = \min\{st_{u'}(o'); (u', o') \in \mathcal{A}\}. \tag{6}$$

and

$$C = \max\{st_{u'}(o'); (u', o') \in \mathcal{A}\} \tag{7}$$

So, we may consider taking subsets of options $\mathcal{X} \subseteq \mathcal{A}$ that restrict the starting time of the operations to some subinterval of $[T_{\mathcal{A}}, C)$. We will study this possibility in the next sections.

Algorithm 1 shows the general structure of the schedule builder $SOG\&T$. It starts from a set A containing the tasks with no predecessors in the task graph and iterates along n steps. In each iteration an option (u, o) is taken non deterministically from a subset $\mathcal{X}$ of the set of scheduling options $\mathcal{A}$. The task u is scheduled at the time $st_u(o)$ given by exp. (4) and assisted by the operator o. Then the partial schedule built so far, G_{SC}, and the set A are updated accordingly for the next iteration. Finally, after n iterations, G_{SC} represents a feasible schedule.

Algorithm 1 Schedule builder $SOG\&T$. It builds a feasible schedule in n steps.

```
Data: A JSSO problem instance $\mathcal{P}$
Result: A feasible schedule for $\mathcal{P}$
$A = \{u \in \mathcal{T}; \neg\exists (w, u) \in E\}$;
for i=1 to n do
    $\mathcal{A} = \{(u, o); u \in A, o \in \mathcal{O}_u\}$;
    $\mathcal{X}$ = a subset of $\mathcal{A}$;
    choose $(u, o) \in \mathcal{X}$ non deterministically;
    set $st_u = st_u(o)$ and $o_u = o$;
    add $u$ to $SC$ and update $G_{SC}$;
    $A = \{v; v \notin SC, P(v) \subseteq SC\}$;
end
return the built schedule $G_{SC}$;
```

Proposition 1 guarantees that any schedule built by Algorithm 1 is feasible. On the other hand, depending on the subset of options considered, the algorithm generates schedules in different search spaces. We will consider here two types of them. Firstly, spaces containing at least one optimal schedule under the objective function considered, in our case the makespan, which are usually known as *dominant* search spaces;

and then non-dominant search spaces, i.e. spaces that for some instances may not contain optimal solutions.

4.2. Dominant search spaces

The simplest way to get a dominant search space is taking $\mathcal{X} = \mathcal{A}$ in Algorithm 1.

In this case, as all possible orderings among tasks and operator assignments are considered, it is clear that the set of schedules defined by the $SOG\&T$ algorithm is dominant. However, it is also clear that it may be very large. It is possible to reduce this search space, without loss of dominance, as we will see in the following.

Let (v^*, o^*) be the option in $\mathcal{A}$ with the earliest completion time; i.e.

$$(v^*, o^*) = \arg\min\{st_u(o) + p_u; (u, o) \in \mathcal{A}\} \quad (8)$$

and let $C^* = st_{v^*}(o^*) + p_{v^*}$. We define the sets of options $\mathcal{A}'$ and $\mathcal{B}$ as follows.

$$\mathcal{A}' = \{(u, o) \in \mathcal{A}; st_u(o) < C^*\} \quad (9)$$

$$\mathcal{B} = \{(u, o) \in \mathcal{A}'; (m_u = m_{v^*} \vee o = o^*)\} \quad (10)$$

The set $\mathcal{A}'$ reduces the scheduling options to the options in $\mathcal{A}$ having a starting time lower than C^*. So, each option in $\mathcal{A}'$ establishes a starting time for a task in the interval $[T_{\mathcal{A}}, C^*)$.

Then, $\mathcal{B}$ further restricts the number of options by filtering those involving a machine other than m_{v^*} and an operator other than o^* at the same time. So, the set $\mathcal{B}$ contains the options for tasks in A that would require the machine m_{v^*} or the operator o^* at some time in the interval $[T_{\mathcal{B}}, C^*)$ if they were chosen in the current iteration, where

$$T_{\mathcal{B}} = \min\{st_u(o); (u, o) \in \mathcal{B}\}, \quad (11)$$

being $T_{\mathcal{B}} \geq T_{\mathcal{A}}$, as the option that establishes the value of $T_{\mathcal{A}}$ in expression (6) may or may not be included in $\mathcal{B}$.

The following result establishes that if we take $\mathcal{X} = \mathcal{B}$ in each iteration, then the search space is dominant.

Proposition 2. *In at least one the best schedules that can be eventually reached from the current iteration, one of the operations in A is scheduled in accordance with and option in $\mathcal{B}$.*

Proof. Let S be one schedule that can be eventually built from the current iteration such that none of the tasks in A is scheduled in accordance with an option in $\mathcal{B}$. In S, m_{v^*} and o^* are idle over the interval $[st_{v^*}(o^*), C^*)$. Then v^* could be rescheduled in accordance with the option (v^*, o^*); i.e., starting at $st_{v^*}(o^*)$ and assisted by o^*. As none of the remaining operations has to be delayed and $st_{v^*}(o^*) \leq st_{v^*}$, then $C_{max}(S') \leq C_{max}(S)$. So, if S is one of the best schedules, then S' is also one of the best schedules. □

Corollary 1. *There is a sequence of non deterministic choices of options, each one from the set $\mathcal{B}$ calculated in each iteration, that allows Algorithm 1 to reach and optimal schedule. In other words, the search space generated by Algorithm 1 when the set of options $\mathcal{X}$ is $\mathcal{B}$ in each iteration is dominant.*

Proof. It follows from Proposition 2 considering the initial iteration in which none of the tasks is scheduled. □

Corollary 2. *The search space generated by Algorithm 1 when the set of options $\mathcal{X}$ is $\mathcal{A}'$ in each iteration is dominant.*

Proof. It follows from the fact that $\mathcal{B} \subseteq \mathcal{A}'$. □

Remark 1. *In principle, it may seem reasonable consider $\mathcal{X} = \mathcal{B}$ in each iteration of the SOG&T algorithm as in this way we will get the smallest of the search spaces considered that guarantees dominance. However, from a thorough analysis of the sets $\mathcal{A}$, $\mathcal{A}'$ and $\mathcal{B}$ we can draw the following observations. As we have mentioned, if in a given iteration the chosen option (u, o) is selected from $\mathcal{A}'$, then $st_u \in [T_{\mathcal{A}}, C^*)$, while if it is selected from $\mathcal{B}$, then $st_u \in [T_{\mathcal{B}}, C^*)$, with $T_{\mathcal{A}} \leq T_{\mathcal{B}}$. This fact may have consequences on the idle times of the machines and operators and so on the average makespan of the schedules. So, as a consequence of the potentially larger mean idle times of the schedules generated from $\mathcal{B}$, due to the difference $T_{\mathcal{B}} - T_{\mathcal{A}}$, the average of these schedules may be larger than the average makespan of the schedules generated from $\mathcal{A}'$.*

Also, if the option (u, o) is selected from $\mathcal{A}$, then $st_u \in [T_{\mathcal{A}}, C)$, with $C \geq C^*$, so the average makespan of the schedules generated from $\mathcal{A}$ is expected to be larger than that of the schedules generated from $\mathcal{A}'$ or $\mathcal{B}$.

Furthermore, as $C - C^*$ is likely to be greater than $T_{\mathcal{B}} - T_{\mathcal{A}}$, the difference between the average makespan *of the schedules generated from $\mathcal{A}$ and those generated from $\mathcal{A}'$ may be expected to be larger than the difference in the average makespan of the schedules generated from $\mathcal{A}'$ and $\mathcal{B}$ respectively.*

From this observations, we can expect that, for example, for a genetic algorithm that searches over the aforementioned spaces, the space generated considering the set of options $\mathcal{A}'$ may be the best one. However, this has to be analyzed experimentally.

4.3. Non-dominant search spaces

The dominance of the spaces defined in the previous section relies on the fact that the respective sets of options contain the set $\mathcal{B}$. In this section, we will consider sets of options for which is not guaranteed to contain the set $\mathcal{B}$ and consequently, they may give rise to non-dominant search spaces. These spaces may be interesting if they have small size and at the same time they contain a large fraction of high-quality solutions. Clearly, from the point of view of a genetic algorithm searching over spaces having these characteristics, the fact that they may not contain an optimal schedule should not be considered as a real problem due to the fact that a genetic algorithm cannot guarantee to reach the best solution contained in the search space taking finite time.

To generate non-dominant search spaces, we may start just considering the sets of options $\mathcal{A}, \mathcal{A}'$ and $\mathcal{B}$ and in all three cases discard some of the options that may give rise to the largest idle times of operators and machines. Bearing this in mind, we propose a number of parameterized sets of options, whose rationale is quite similar to that behind the so called hybrid schedules for dynamic job shop scheduling problems used in [23].

The first parameterized set of options is defined as:

$$\mathcal{A}(\delta) = \{(u,o) \in \mathcal{A}; st_u(o) < T_{\mathcal{A}} + \delta(C - T_{\mathcal{A}})\} \quad (12)$$

where $\delta \in [0,1]$ is a parameter. So, $\mathcal{A}(\delta)$ is a reduction of the set $\mathcal{A}$ in which the options that may give rise to idle times larger than $T_{\mathcal{A}} + \delta(C - T_{\mathcal{A}})$ are removed. It is clear that for a sufficiently small value of δ, the set $\mathcal{A}(\delta)$ may not contain the set $\mathcal{B}$ and so the generated search space may not be dominant.

Similarly, we define reductions of the sets $\mathcal{A}'$ and $\mathcal{B}$ as

$$\mathcal{A}'(\delta) = \{(u,o) \in \mathcal{A}'; st_u(o) < T_{\mathcal{A}} + \delta(C^* - T_{\mathcal{A}})\} \quad (13)$$

and

$$\mathcal{B}(\delta) = \{(u,o) \in \mathcal{B}; st_u(o) < T_{\mathcal{B}} + \delta(C^* - T_{\mathcal{B}})\} \quad (14)$$

In the extreme cases of $\delta = 1$ and $\delta = 0$, $\mathcal{A}(1) = \mathcal{A}$, $\mathcal{A}'(1) = \mathcal{A}'$, $\mathcal{B}(1) = \mathcal{B}$ and $\mathcal{A}(0) = \mathcal{A}'(0) = \mathcal{B}(0) = \emptyset$, so the significant cases are those with $\delta \in (0,1]$. Regarding the relationship among these set of options, for all δ, $\mathcal{B}(\delta) \subseteq \mathcal{A}'(\delta) \subseteq \mathcal{A}(\delta)$; and for all δ', δ'' such that $\delta' < \delta''$ and $\mathcal{X} \in \{\mathcal{A}, \mathcal{A}', \mathcal{B}\}$, $\mathcal{X}(\delta') \subseteq \mathcal{X}(\delta'')$. Also, $\mathcal{A}(\delta_1) = \mathcal{A}'$ being

$$\delta_1 = (C^* - T_{\mathcal{A}})/(C - T_{\mathcal{A}}) \quad (15)$$

In all three cases, δ establishes a bound on the maximum length of the time intervals a machine and an operator remain idle while there is a task that can be processed on the machine and assisted by the operator; the lower δ the lower the bound.

4.4. Summary of search spaces

In previous sections, we have defined six search spaces for the SPSO, three of them are dominant, while the other three are non-dominant. Each space is characterized by the set of options considered in each iteration of the $SOG\&T$ algorithm (see Algorithm 1), some of these sets are in fact subsets of others and this induces a "subspace of" relation among the search spaces. Abusing language each space will be identified by the same symbol as the corresponding set of options, namely $\mathcal{A}, \mathcal{A}', \mathcal{B}, \mathcal{A}(\delta), \mathcal{A}'(\delta)$ and $\mathcal{B}(\delta)$. $\mathcal{A}, \mathcal{A}', \mathcal{B}$ are dominant search spaces, while $\mathcal{A}(\delta), \mathcal{A}'(\delta)$ and $\mathcal{B}(\delta)$ are non-dominant for all $\delta \in (0,1)$ and become $\mathcal{A}, \mathcal{A}', \mathcal{B}$ respectively for $\delta = 1$.

Fig. 3 shows how these search spaces are related through the relation "subspace of". The symbol "*" in some of the arcs has to do with the idle periods allowed in the sets of options at the outgoing and incoming spaces.

5. Genetic algorithm for the SPSO

Genetic algorithms have been successfully applied to scheduling problems such as JSP [24,25,26] or JSO [27]. In these cases, the problems were defined by a set of jobs, what allowed the GAs to use permutations with repetition, and efficient coding schema proposed

$$\begin{tikzcd}
\mathcal{A} & \mathcal{A}' \arrow[l, "*"'] & \mathcal{B} \arrow[l] \\
\mathcal{A}(\delta) \arrow[u, "*"'] & \mathcal{A}'(\delta) \arrow[l, "*"'] \arrow[u, "*"'] & \mathcal{B}(\delta) \arrow[u, "*"']
\end{tikzcd}$$

Fig. 3. "Subspace of" relationships among the six search spaces defined. For an arc $(\mathcal{Y}, \mathcal{Z})$, the symbol "*" means that in every iteration, for all options (u, o) in $\mathcal{Z}$ that are not in $\mathcal{Y}$, $st_u(o) \geq st_{u'}(o')$ for all (u', o') in $\mathcal{Y}$, and therefore it may be expected that the average makespan of the schedules in the space $\mathcal{Y}$ to be lower than that in the space $\mathcal{Z}$

in [25]. In the SPSO, as it was defined here, this encoding cannot be used due to the arbitrary precedence relations. So, in principle, we opted to use a more conventional coding schema as single permutations of tasks.

At the same time, in JSP and JSO no information about operators needs to be included in the chromosome; in the first case no operators exist, while in the second all of them are skilled to assist any operation and so they may be in fact considered as a cumulative resource of capacity p. However, in the SPSO, it is reasonable to express operator preferences for the tasks in the chromosomes in order to get an appropriate coding schema.

In the following subsections, we detail the main components of the proposed GA for the SPSO; namely, the coding schema, the decoding algorithm, the genetic operators and the general structure of the GA used.

5.1. Coding schema

We propose here a coding schema for the SPSO where a chromosome consists of two permutations of symbols. The first one is the *task sequence* which is a conventional permutation of the numbers $1 \dots n$, while the second is the *operator sequence* and is given by a permutation with repetition of the symbols $1 \dots p$ of size n and represents operator preferences. Notice that some operators could appear several times in the chromosome and also some operator may be absent. For example the following two permutations represent a feasible chromosome for the instance considered in Figure 1.

$$\begin{array}{l}(6,7,5,4,1,2,3)\\(1,2,2,2,1,1,1)\end{array} \qquad (16)$$

This encoding should be understood in such a way that if task u appears before task v in the first permutation, then u should be preferably scheduled before v. At the same time, the second permutation represents priorities for the operators to be allocated to tasks. In order to avoid that all tasks see the same order of operators, the first operator for a task u in the position i, $1 \leq i \leq n$, in the task sequence will be that in position i in the operator sequence, and then the remaining ones are taken following this last sequence as a circular structure from the position i. An important observation here is that the task ordering and the operators' allocation in the schedule does not only depend on the chromosome, but also on the decoding algorithm, as we will see in the next section.

One of the most interesting properties of this encoding is that any solution can be encoded into a chromosome representing the same machine orderings and operator allocations. At the same time, it is simple and so it allows for designing efficient genetic operators for crossover and mutation. In principle, the only inconvenience comes from the fact that the number of replicas of a symbol in the operator sequence is variable and so an operator could disappear from the chromosome. However, this problem can be easily solved by means of some heuristic repairing as we will see.

5.2. Decoding algorithm

Decoding algorithms map chromosomes to feasible solutions. To this aim, we use the $SOG\&T$ algorithm presented above, exploiting the information encoded in the chromosomes to guide the search. This algorithm is issued and in each iteration the option $(u, o) \in \mathcal{X}$ that is the "leftmost" in the chromosome, i.e., that fulfills the following two conditions, is chosen

- u is the leftmost in the task sequence of the chromosome among all tasks appearing in some option in $\mathcal{X}$, and
- o is the first operator skilled to assist u in the preference list defined by the operator sequence, among the operators appearing in some option in $\mathcal{X}$ for the operation u. If the operator of none of the options for u in $\mathcal{X}$ is present in the operator sequence, then the operator o with $(u, o) \in \mathcal{X}$ that allows u to start earliest is selected.

In the experimental study we will consider all the possibilities above for the set of options $\mathcal{X}$; namely, the dominant spaces $\mathcal{A}$, $\mathcal{A}'$ and $\mathcal{B}$ and their non-dominant extensions $\mathcal{A}(\delta)$, $\mathcal{A}'(\delta)$ and $\mathcal{B}(\delta)$.

We illustrate how the decoding algorithm works by means of an example. Let us consider the chromosome given in expression (16). It contains the tentative orderings (6,4,1) and (7,5,3) for machines 1 and 2 respectively, which are inconsistent due to the fact that the task 4 must be processed after the task 1, and the task 5 must be processed after the tasks 7 and 3, in accordance with the precedences expressed by the task graph. In spite of that, the chromosome is feasible as the decoding algorithm will build a feasible schedule from it in which some of the tentative orderings and operator preferences will hold, while others will not. If we consider the set $\mathcal{X} = \mathcal{A}$, in the first iteration the scheduling options would be $\{(1,1),(3,1),(3,2),(6,1)\}$; in all four cases the starting time would be 0. So, according to the chromosome, the task scheduled in this step would be 6, as it is the leftmost one in the chromosome among 1, 3, and 6, and the operator assigned would be 1 (as this is the only option). In the next iteration, the options would be $\{(3,2)\}$ with starting time 0 and $\{(1,1),(3,1),(7,1),(7,2)\}$ with starting time 4. In accordance with the chromosome, the chosen option would be (7,2). In the third iteration the options would be $\{(3,1),(3,2)\}$ with starting time 9 and $\{(1,1)\}$ with starting time 4, and the chosen option would be (1,1). This way, the tentative ordering (4,1) in the chromosome is not kept in the schedule. Finally, we will have the schedule of Fig. 1. It is important to remark that with $\mathcal{X} = \mathcal{A}'$ or $\mathcal{X} = \mathcal{B}$, we could reach different schedules from the same chromosome.

5.3. Genetic operators

We build on two point crossover and single mutation operators, and start with an initial population of chromosomes generated at random following an uniform distribution.

We will use a double-chromosome variant of the classic order crossover (OX) which translates the subsequence of symbols between two cutting points from one parent to the offspring and the relative order of the remaining values from the second. OX may be a good option for the SPSO due to the fact that relevant characteristics, as processing order of tasks on the machines or priorities of operators for assisting the tasks, may be transmitted from parents to offsprings. To avoid a strong disruptive effect of the crossover, the cutting points will be the same in both sequences (tasks and operators).

Also, we will consider single mutation (SM) by swapping two consecutive positions at random. This seems to be appropriate as it may produce small changes. As before, the same positions will be swapped in both sequences. Moreover, to get different distributions of operators in the operator sequences, we will also use an operator mutation (OM) which will change the value in a location of the operator sequence at random. When a chromosome is mutated, one of SM or OM is chosen with probability 0.5.

Algorithm 2 Genetic Algorithm.

```
Data: A SPSO problem instance ($\mathcal{P}$) and set of parameters ($P_c$, $P_m$, $timeLimit$, $popSize$, $\mathcal{X}$, $codingBack$)
Result: A feasible schedule for $\mathcal{P}$
Generate and evaluate the initial population $P(0)$;
t = 0;
while $timeLimit$ is not reached do
    t = t+1;
    Selection: organize the chromosomes in $P(t-1)$ into pairs at random;
    Recombination: mate each pair of chromosomes and mutate the two offsprings in accordance with $P_c$ and $P_m$;
    Evaluation: evaluate the resulting chromosomes considering the search space $\mathcal{X}$ and code back the schedules into the chromosomes if $codingBack$ was chosen;
    Replacement: make a tournament selection from every two parents and their two offsprings to generate $P(t)$;
end
return the best schedule built so far;
```

Additionally, we consider coding back the structure of the schedule into the chromosome. If chosen, it means rebuilding the chromosome structure in such a way that the order of tasks in the task sequence is compatible with the partial ordering on the tasks in the schedule, and that for each task the first operator in the operator list is that assisting the task in the schedule. This way, the machine and operator sequences in the schedule are translated to the chromosome and so these relevant characteristics of the schedules may be better translated from parents to offsprings. This is a sort of Lamarckism as some of the characteristics learned over the breeding of the phenotype (the schedule) from the genotype (the chromosome) are coded back into the chromosome. As the breeding is just produced by decoding the chromosome, this operation is expected to introduce small changes in the chromosome structure and so we refer to the effect it produces in the genetic algorithm as weak Lamarckian evolution.

5.4. Genetic algorithm structure

We use here a genetic algorithm with generational replacement. In order to avoid premature convergence, we opted not to use the classic roulette wheel selection combined with unconditional replacement. Instead, in the selection phase all chromosomes are organized into pairs at random, then each pair undergoes crossover and mutation. After this, the offsprings are evaluated and the schedules coded back into the chromosomes if $codingBack$ was chosen. Finally, the new population is obtained by means of tournament selection keeping the best two individuals among every two parents and their two offsprings. The algorithm requires 5 parameters: crossover and mutation probabilities (P_c and P_m), time limit ($timeLimit$), population size ($popSize$), the search space considered $\mathcal{X}$ and the coding-back option ($codingBack$) Algorithm 2 shows the main steps of the genetic algorithm.

6. Experimental study

The purpose of this experimental study is to evaluate the characteristics of the proposed search spaces, analyze the performance of the genetic algorithm and compare this algorithm with the state of the art. To this end, we used a benchmark set with instances of different sizes and characteristics. To evaluate the search spaces, we first generated random solutions in each space and analyzed the quality and distribution of the makespan values. Then, we analyzed the evolution of the genetic algorithm when searching solutions in these spaces and in particular the influence that the coding-back option has on the convergence. The objective of these preliminary experiments was to establish reasonable conditions for running the genetic algorithm which, as will see, depend on the size and other characteristics of the instances, and to identify the most appropriate search spaces for each type of instances. In order to avoid a combinatorial explosion on the number of experiments, we opted to fix some of the parameters to rather standard values, in particular $popSize = 100$, $P_c = 1.0$ and $P_m = 0.1$. The values for other parameters as $timeLimit$, $\mathcal{X}$ or $codingBack$ will be established from the results of these preliminary experiments. To demonstrate the performance of the proposed genetic algorithm, we show the results across all instances of the benchmark set under the best conditions we have identified for each type of instances, and when possible we establish a fair comparison with other methods. The target machine was Intel Xeon 2.26 GHz. 24 GB RAM and the algorithms were coded in C++.

6.1. Benchmark sets

We considered three sets of instances[1]: firstly the set proposed in [12] which includes 50 instances organized in 5 groups with 10 instances each. Each group is defined by the values of n, p and q ranging in the sets $\{100, 150, 200\}$, $\{10, 15, 20\}$ and $\{15, 30, 50\}$ respectively. The processing times are uniformly distributed in $[1, 100]$. The topology of the task graph was inspired in real-life assembly trees and was distributed in a number of branches, between 3 and 6. The sets of operators skilled for each operation were generated at random and the workload of the machines was roughly balanced. We obtained detailed results from the algorithm MSB-DOS and the Cplex implementation proposed in [12] through personal communication from the authors. These results are summarized in Table 5.

The instances in the second set are derived from job-shop scheduling problem (JSP) instances. A JSP instance is characterized by a number of N jobs and a number of M machines. Each job is a sequence of M operations, each one requiring a different machine and so each job has to visit all the machines, but the order is in general different for every two different jobs. Therefore, the task graphs express sequences among operations in each job. In these instances $n = N \times M$ and $q = M$. The JSP instances considered are $FT10(10 \times 10)$, $LA21-25(15 \times 10)$, $LA26-30(20 \times 10)$, $LA31-35(30 \times 10)$, $LA36-40(15 \times 15)$, $ta11-20(20 \times 15)$, $ta21-30(20 \times 20)$ and $ta31-40(30 \times 15)$. For each of these 51 JSP instances, three different numbers of operators p were considered depending on the number of resources q and the skill matrix was created taking different probabilities Pr that an operator is skilled for a given task. Specifically, $p \in \{5, 7, 9\}$ for $q = 10$, $p \in \{7, 11, 14\}$ for $q = 15$ and $p \in \{10, 15, 19\}$ for $q = 20$. Also, we considered 3 different combinations of Pr: 0.2, 0.6 and 0.2 or 0.6 chosen at random for each task. We think these combinations may be significant because they show the cases

[1]The instances and more details of the experimental study are available at http://www.di.uniovi.es/iscop (Repository).

in which a task requires a high level of specialization, and therefore not many operators can assist the task (0.2), and the cases in which a task requires less specialization and so it may be assisted by more operators (0.6). So, this set contains 459 instances.

For the third set, we considered instances with general precedence relations. To this end, we looked at the precedence relations on tasks for 5 instances of the resource-constrained project scheduling problem (RCPSP): $j1201\_1$, $j1202\_1$, $j1203\_1$, $j1204\_1$ and $j1205\_1$, each one having 120 tasks; then we took $q = 10$ and assigned each task one machine at random. p and Pr taking values as in the instances having $q = 10$ in the second set. So, this set contains 45 instances in all.

The JSP and RCPSP instances considered are available in the OR-library [28].

6.2. *Analysis of the search spaces*

In order to get an initial idea about the quality of solutions in the different search spaces considered, we generated 1000 random schedules in each one of them and observed the diversity and quality of these solutions. We start considering three instances derived from FT10, the smallest instance in the second set, and the dominant search spaces $\mathcal{A}$, $\mathcal{A}'$ and $\mathcal{B}$. Figure 4 shows the makespan outcomes for each set of schedules. It is remarkable that for the three instances the best random solution generated in the set $\mathcal{A}$ is much worse than the worst solutions generated from the sets $\mathcal{A}'$ and $\mathcal{B}$. So, even though the diversity of the solutions from $\mathcal{A}$ is clearly larger than the diversity of the random schedules from $\mathcal{A}'$ and $\mathcal{B}$, the search space $\mathcal{A}$ does not seem to be a good choice. As mentioned earlier, the reason for the low quality of the solutions in this space is due to the fact that it contains many schedules having large periods of inactivity for machines and operators, and so it may be even difficult to reach a schedule within the spaces $\mathcal{A}'$ and $\mathcal{B}$ if the search is performed on the whole space $\mathcal{A}$. This can be observed, at least, in a random search. For this reason, we discard $\mathcal{A}$ from the remaining experiments. Regarding the spaces $\mathcal{A}'$ and $\mathcal{B}$, the average value of the random schedules is better in the first than it is in the second, while the diversity seems to be slightly better in the second than in the first.

We also analyzed random instances from some large JSP instances, in this case considering both dominant and non-dominant search spaces. Figure 5 shows results from the search spaces $\mathcal{A}(\delta)$, $\mathcal{A}'(\delta)$ and $\mathcal{B}(\delta)$ taking different values of δ. Remember that $\delta = 1$ gives rise to dominant search spaces; $\delta \sim 0$ means that the value of δ is sufficiently close to 0 for considering only the options with starting times $T_{\mathcal{A}}$ or $T_{\mathcal{B}}$ depending on the search space. Looking at Figs. 5 (a), (c) and (d), we can observe that the lower δ the lower both diversity and average makespan. Again, it seems clear that $\mathcal{A}(\delta)$ is not a good option as it shows the worst average makespan for all values of δ. In Fig. 5 (b) and (c) we can observe that the average values of the schedules from $\mathcal{A}(0.25)$ and $\mathcal{A}'(1)$ are rather similar, but $\mathcal{A}'(1)$ shows better diversity and it is dominant, while $\mathcal{A}(0.25)$ is not dominant. As before, $\mathcal{A}'(\delta)$ shows better average makespan than $\mathcal{B}(\delta)$ and at the same time similar diversity for all values of δ.

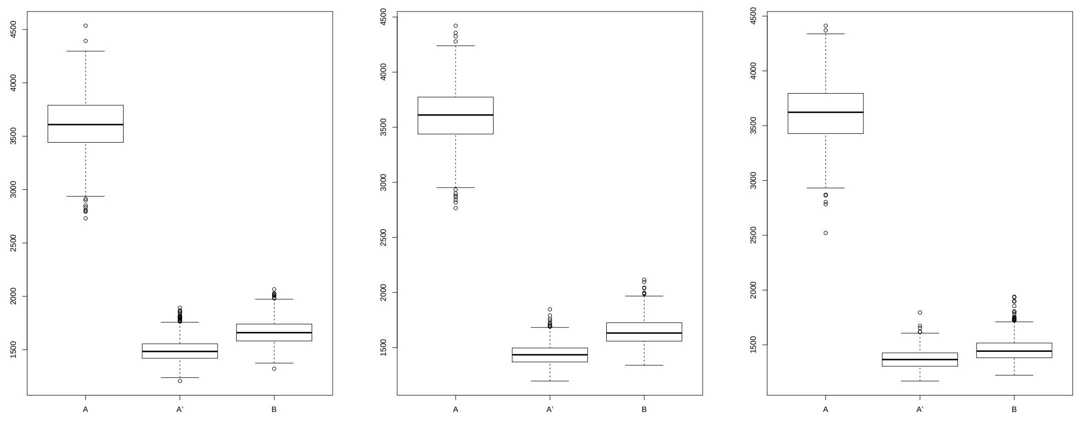
So, from these preliminary results, we discard the search space $\mathcal{A}(\delta)$, for all values of δ, due to the low average quality of the schedules and only consider $\mathcal{A}'(\delta)$ and $\mathcal{B}(\delta)$ in the remaining experiments. Even though $\mathcal{A}'(\delta)$ seems to be the best choice for the sake of successful evolution and convergence of the GA, this hypothesis must be assessed from an analysis of the convergence of the genetic algorithms searching in these spaces. This is the objective of the next section.

6.3. *Analysis of the convergence of the genetic algorithm*

It is well known that genetic algorithms are sensitive to mechanisms that introduce bias in the search towards regions of the solution spaces with above average solutions. In general these mechanisms may allow the genetic algorithms to reach better solutions but they may also give rise to the well-known phenomenon of premature convergence. In this paper, we introduced a number of elements of this nature as the coding-back option or the definition of different search spaces. It is therefore interesting to analyze how these elements may affect the convergence of the genetic algorithms.

6.3.1. *Coding-back option and search spaces $\mathcal{A}'$ and $\mathcal{B}$*

Let us start considering the effect of the coding-back option. Figure 6 shows the convergence patters of the average and best solutions, for three instances of the second benchmark set derived from LA31, over the spaces $\mathcal{A}'$ and $\mathcal{B}$ with and without choosing $codingBack$. In all experiments $timelimit = 1200s$ and the remaining parameters were set as indicated before. As we can observe, not only the average and best makespan in the initial populations are better for



(a) Instance 2_9_1 (b) Instance 2/6_9_1 (c) Instance 6_9_1

Fig. 4. Distribution of makespan values obtained from 1000 random schedules for the instances 2_9_1, 2/6_9_1 and 6_9_1 derived from the FT10 in the search spaces $\mathcal{A}$, $\mathcal{A}'$ and $\mathcal{B}$.

$\mathcal{A}'$ than they are for $\mathcal{B}$, but also the convergence is better when the coding-back option is chosen in both cases. This is quite clear looking at both the evolution of the average and best solutions. So, searching on $\mathcal{A}'$ and coding the schedules back into the chromosomes seems to be the best choice if we want to search over a dominant space.

Table 1

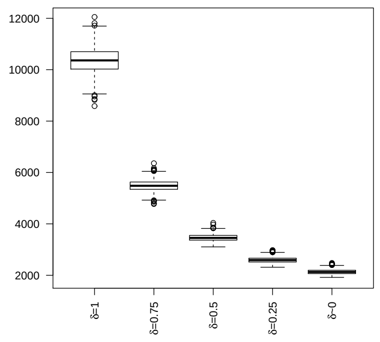
Summary of results from GA on the sets of instances derived from FT10, considering the search spaces $\mathcal{A}'$ and $\mathcal{B}$. The results are averaged for each subset of 5 instances with the same values of p and Pr.

Sets		$\mathcal{A}'$			$\mathcal{B}$		
Pr	p	%Err. Best.	%Err. Avg.	%CV	%Err. Best.	%Err. Avg.	%CV
	5	0.00	3.17	1.57	3.56	7.25	1.71
2	7	0.00	2.82	1.45	4.20	8.23	2.05
	9	0.00	2.96	1.79	3.99	8.90	2.15
	Avg.	0.00	2.98	1.60	3.92	8.13	1.97
	5	0.00	3.09	1.37	5.45	8.81	1.51
6	7	0.00	3.44	1.59	5.97	10.09	1.69
	9	0.00	3.41	1.53	3.24	7.44	1.78
	Avg.	0.00	3.32	1.49	4.89	8.78	1.66
	5	0.00	3.42	1.48	6.98	11.09	1.77
2/6	7	0.00	3.26	1.72	8.32	12.90	2.22
	9	0.00	3.05	1.72	7.95	11.66	1.70
	Avg.	0.00	3.24	1.64	7.75	11.89	1.89
T. Avg.		0.00	3.18	1.58	5.52	9.60	1.84

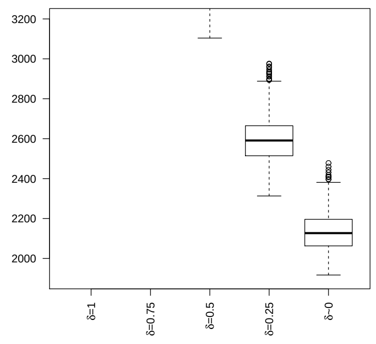
To further assess the differences between decoding in the subsets $\mathcal{A}'$ and B, we solved the 45 instances derived from FT10 with the coding-back option. For these experiments, we set $timeLimit = 120s$. Each instance was solved 30 times and the best and average of the solutions reached in all runs were recorded. We calculated the average error for the best and averages with respect to the best solution found for each instance in these runs. Also, in order to study the stability of the algorithm, we calculated the average Pearson coefficient of variation (CV). These results, averaged for each subset of 5 instances, are summarized in Table 1. As we can observe, searching on $\mathcal{A}'$ the genetic algorithm reaches the best solution found and achieves considerably better results in average in all cases. At the same time, it is more stable in average than it is when it searches on $\mathcal{B}$.

6.3.2. Search spaces $\mathcal{A}'(\delta)$ and $\mathcal{B}(\delta)$

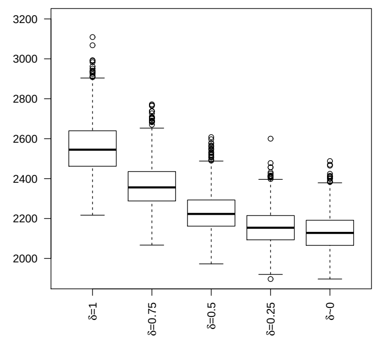
We now analyze the evolution of the genetic algorithms on the spaces $\mathcal{A}'(\delta)$ and $\mathcal{B}(\delta)$. To this end, we consider 5 different values for the parameter δ: 1, 0.75, 0.5, 0.25 and ~ 0, and instances of different sizes. Remember that $\delta = 1$ means that the search space is dominant while lower values of δ constrain the search to reduced search spaces with limited idle periods of resources that may not be dominant. So, one can expect that for small instances the best option could be $\delta = 1$



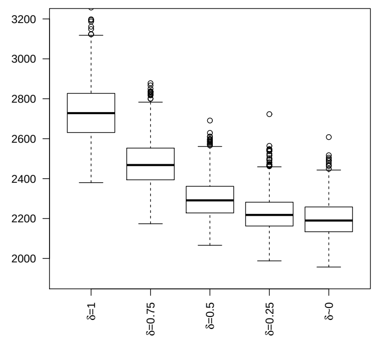
(a) $\mathcal{A}$



(b) $\mathcal{A}$



(c) $\mathcal{A}'$



(d) $\mathcal{B}$

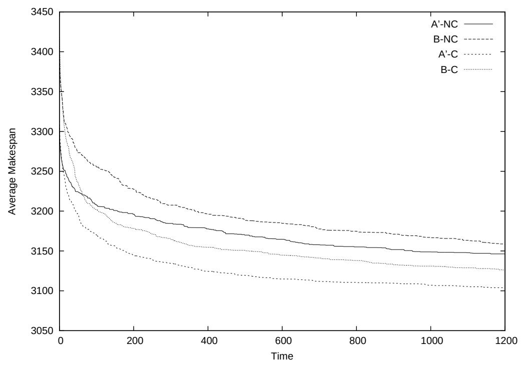
Fig. 5. Distribution of makespan values obtained from 1000 random schedules for the instance LA31_06_9op derived from the LA31 in the search spaces $\mathcal{A}$, $\mathcal{A}'$ and $\mathcal{B}$ with different δ values. Note that (a) and (b) represent the same results using different scales; (b), (c) and (d) are on the same scale.

as for these instances we have the chance of reaching the optimal solution. However, for the largest instances, as the search space grows exponentially with the problem size, the best choice is likely to search over a reduced space.

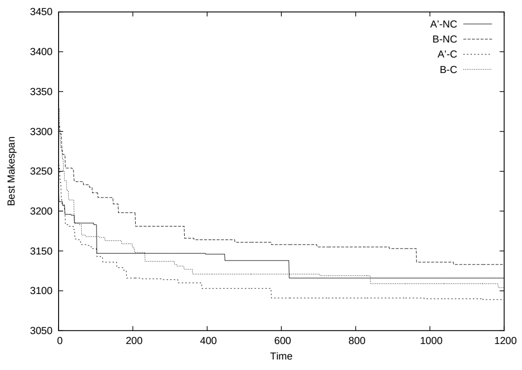
Let us start with the set of instances proposed in [12]. Figure 7 shows the convergence patterns of the genetic algorithm for one of these instances. As we can see, the worst option is clearly $\delta \sim 0$, followed by $\delta = 0.25$, while there are not too much differences among the remaining values. However, $\delta = 1$ is the best option in both cases $\mathcal{A}'$ and $\mathcal{B}$. It seems that in both cases the optimal solution is reached, and that for small values of δ the search space contains no optimal schedules.

Let us now consider the instances derived from FT10 and LA21-25, i.e., the smallest instances of the second set with 100 and 150 tasks respectively, and the instances derived fro RCPSP with 120 tasks. Figures 8, 9 and 10 show the convergence patterns of the genetic algorithm for one instance derived from FT10, one instance derived from the RCPSP, and one instance derived from LA21, respectively. In all three cases, it seems clear that $\delta = 1$ is the worst choice, but no sharp conclusions can be drawn for the remaining values. Besides, it seems clear that the convergence of the genetic algorithms on $\mathcal{A}'(\delta)$ is better that it is on $\mathcal{B}(\delta)$. Therefore, these results suggest that these instances are harder to solve than those in the first set, the dominant search spaces become so large that it is worth to restrict the search to some subset of schedules with limited idle times for resources even at the risk of leaving all optimal schedules out of the effective search space.

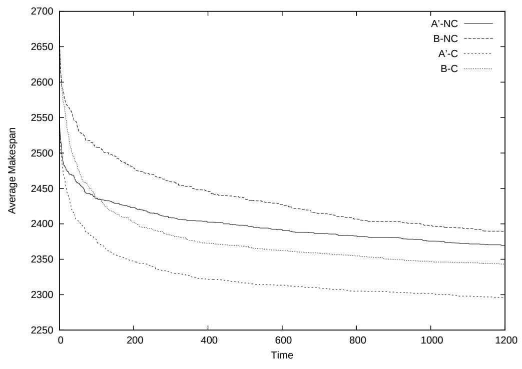
Let us finally consider the largest instances of the benchmark set, i.e., those derived from JSP instances other than FT10 and LA21-25. Figure 11 shows the



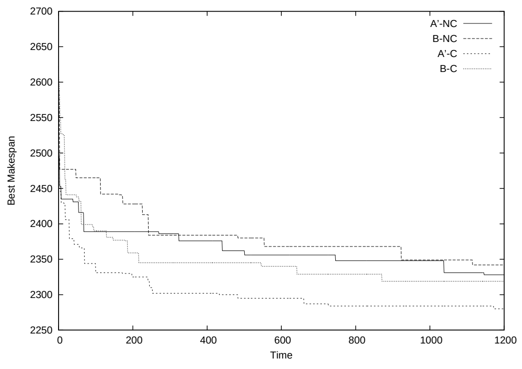
(a) Instance LA31_06_5op



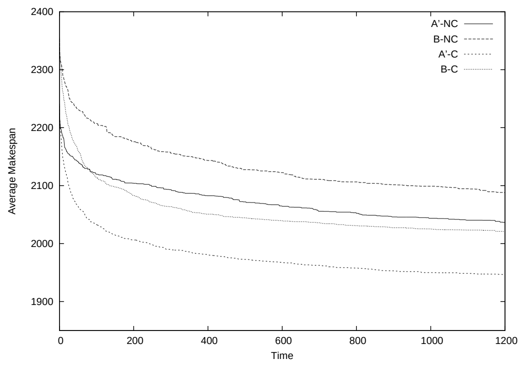
(b) Instance LA31_06_5op



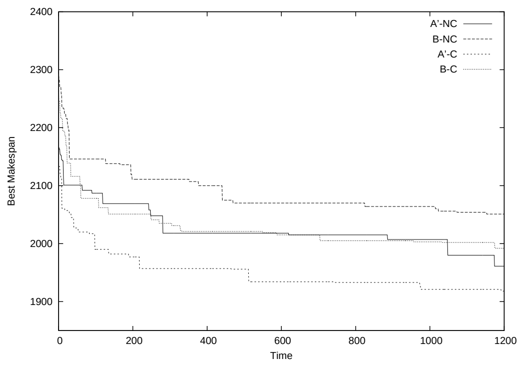
(c) Instance LA31_06_7op



(d) Instance LA31_06_7op



(e) Instance LA31_06_9op

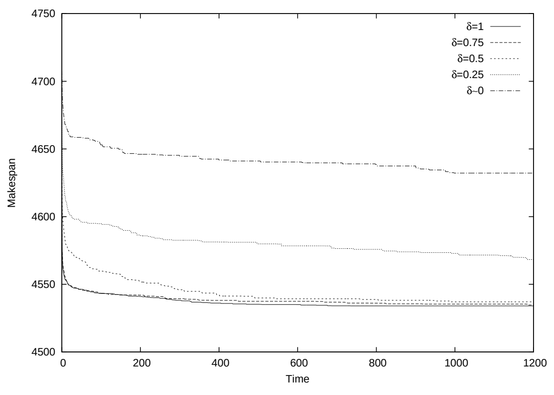


(f) Instance LA31_06_9op

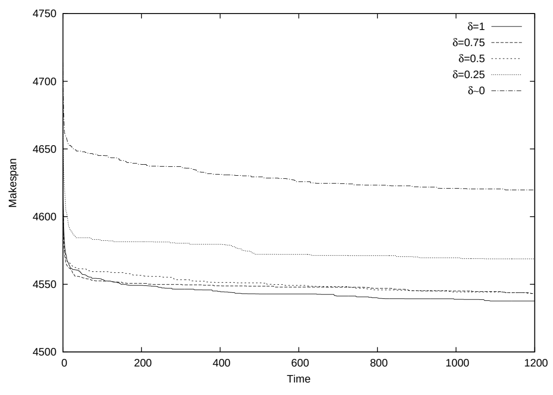
Fig. 6. Convergence of the GA for the instances LA31_06 derived from the LA31, combining the search spaces $\mathcal{A}'$ and $\mathcal{B}$ and $codingBack$ chosen (C) or not chosen (NC). Each plot represents the evolution of the mean/best makespan of the population averaged for 30 runs.

evolution of the genetic algorithm for one of them. At difference of the smallest and medium size instances, we can observe that the initial values and the evolution of the genetic algorithm are clearly better for the lowest values of δ. In particular $\delta = 1$ is also the worst choice and $\delta \sim 0$ is now the best one. So, for these instances, the search spaces are so huge that restricting the search to the schedules where a machine and an operator are never idle at the same time that there is some task available for them, seems to be the best choice. Also, $\mathcal{A}'(\delta)$ seems to be better than $\mathcal{B}(\delta)$ for all values of δ.

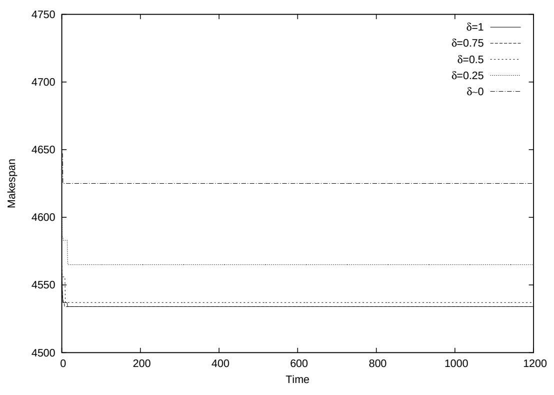
From the above experiments, we consider the space $\mathcal{A}'(\delta)$ as the best one and propose to choose $\delta = 1$ for the small instances, those in the set proposed [12], $\delta \in [0.25, ..., 0.5]$ for medium size instances, those derived



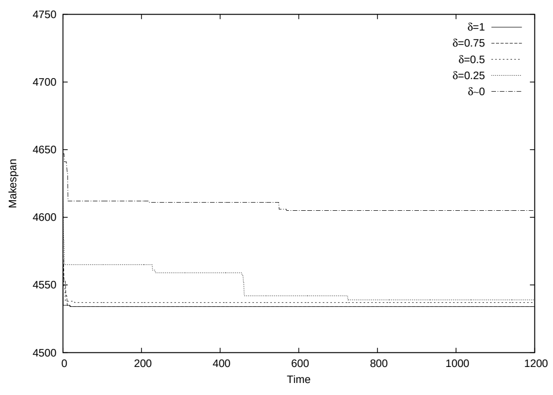
(a) $\mathcal{A}'(\delta)$ Average Makespan



(b) $\mathcal{B}(\delta)$ Average Makespan



(c) $\mathcal{A}'(\delta)$ Best Makespan



(d) $\mathcal{B}(\delta)$ Best Makespan

Fig. 7. Convergence of the GA for the instance 10_15_100_8 from the Agnetis Benchmark, combining the search spaces $\mathcal{A}'(\delta)$ and $\mathcal{B}(\delta)$ with different δ values and the coding-back (C) option. Each plot represents the evolution of the mean/best makespan of the population averaged for 30 runs.

from FT10, LA21-25 and RCPSP, and $\delta \sim 0$ for the large ones, the remaining ones in the set.

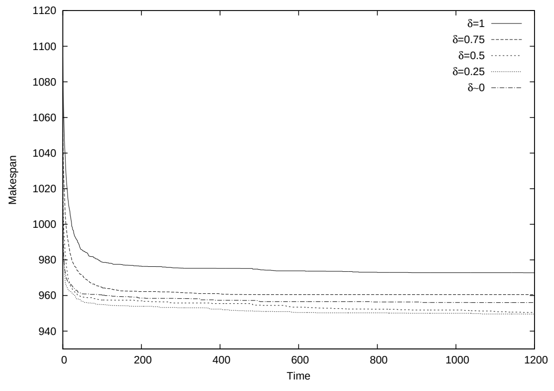
6.4. *Evaluation of the genetic algorithm*

We evaluated the proposed GA considering the search space $\mathcal{A}'$, the coding-back option and different δ values. The evaluation was done across the second and third sets of instances. For the instances derived from $FT10$ and the medium size instances we considered $\delta \in \{\sim 0, 0.25, 0.5, 0.75, 1\}$ and for the large instances we considered $\delta \sim 0$ and also $\delta = 1$ just for the purpose of comparison.

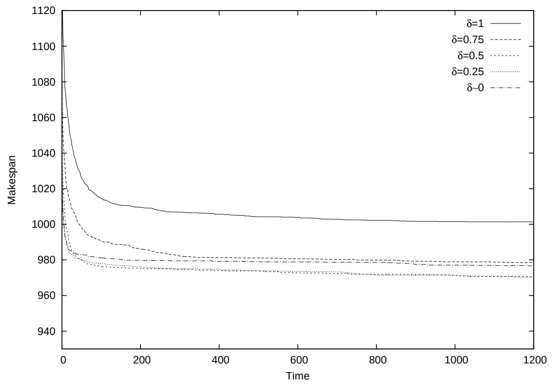
We have chosen a rather conventional parameter setting: $P_c = 1^2, P_m = 0.1, popSize = 100$. $timeLimit$ was different for each set of instances: 120 seconds for the instances derived from $FT10$ instance, 150 seconds for the medium size instances and 300 seconds for the large instances. Each instance was solved 30 times and the best and average of the solutions reached in all runs were recorded. Apart from the average error, in order to study the stability of the algorithm with each configuration, we calculated the average Pearson coefficient of variation in percentage terms of the solutions (CV) computed as the ratio of standard deviation and the average of the solutions across all the runs.

Table 2 shows the results obtained for the instances derived from FT10, averaged for each group of 5 instances with the same values of p and P_r. As we can see, in average $\delta = 0.5$ is the best option, $\delta = 0.25$ the second, $\delta = 0.75$ the third and $\delta \sim 0$ and $\delta = 1$ the worst options. We can also see that the algorithm is more stable with lower values of δ, which was predictable since the size of the search space is in inverse

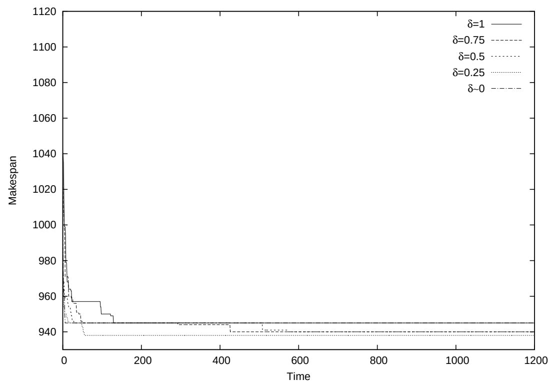
[2]Taking $P_c < 1$ makes that some pairs of chromosomes are not mated and in this case the offsprings are the same as their parents if they are not mutated, so the best parent is chosen twice in the replacement phase, what may contribute to premature convergence.



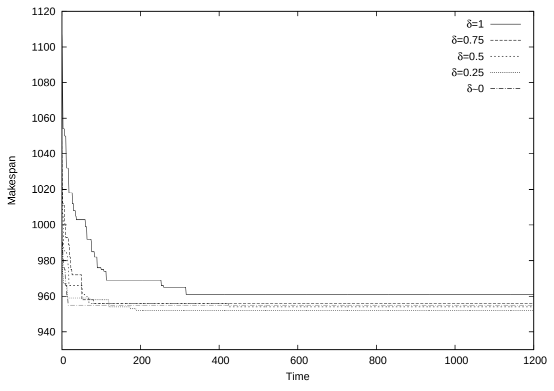
(a) $\mathcal{A}'(\delta)$ Average Makespan



(b) $\mathcal{B}(\delta)$ Average Makespan



(c) $\mathcal{A}'(\delta)$ Best Makespan



(d) $\mathcal{B}(\delta)$ Best Makespan

Fig. 8. Convergence of the GA for the instance FT10_06 derived from the FT10, combining the search spaces $\mathcal{A}'(\delta)$ and $\mathcal{B}(\delta)$ with different δ values and the coding-back (C) option. Each plot represents the evolution of the mean/best makespan of the population averaged for 30 runs.

ratio with δ. Looking at the results for each Pr, we see a similar behavior with some exceptions. Firstly, for instances with $Pr = 0.2$, the second best option is $\delta = 0.75$ instead of $\delta = 0.25$; which suggest that as these instances are more constrained, the search spaces may be smaller, and so reducing them may lead to lose high-quality solutions. On the other hand when $Pr = 0.6$, the best choice is $\delta = 0.25$, which may be explained by the fact that there are more options in these instances than in the remaining ones and so the GAs need to cope with larger search spaces.

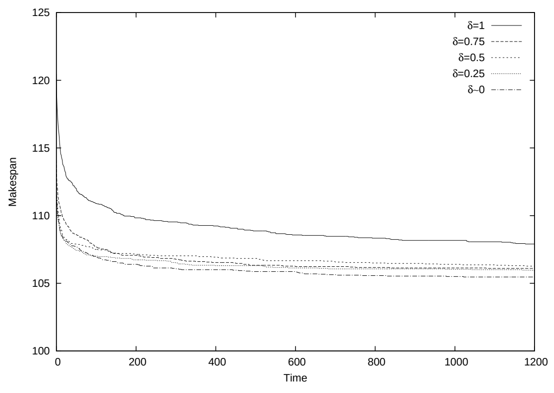
Table 3 shows the results the GA obtained with different configurations on the medium size instances. We can see that in average, the best options are those with $\delta \sim 0$ and $\delta = 0.25$, with $\delta \sim 0$ being better than $\delta = 0.25$ in the RCPSP instances, and $\delta = 0.25$ being the best one at the LA21-25 instances. Also, we can see that the larger δ, the worse results returned by the GA.

Finally, Table 4 shows the results the GA yielded for the large instances considering two values of δ. We can observe that in every case, the genetic algorithm searching in $\mathcal{A}'(\delta \sim 0)$ layouts much better results and shows more stability than searching in $\mathcal{A}'(\delta = 1)$.

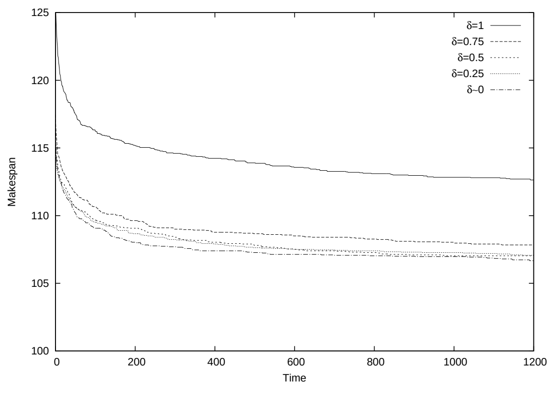
All things considered, we can conclude that the larger the instance is, the lower the value of the parameter δ should be. Also, we can point out that the algorithm is more stable the lower δ is.

6.5. Comparison with other methods

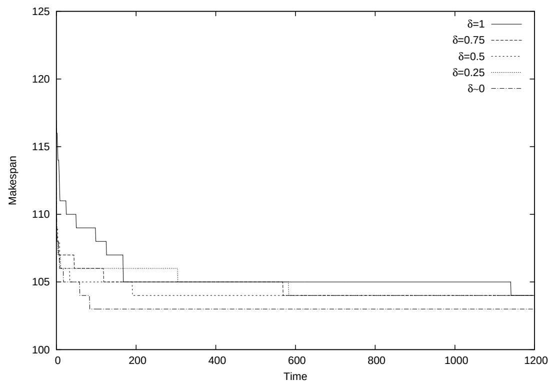
In [12] the authors proposed two heuristic methods, MSB-DOS and STA-OMSB, and experimented with Cplex 12.2 on a MILP formulation they proposed. They report results from this study averaged for each subset of instances showing clearly that MSB-DOS is the best of the two heuristics proposed. Their target machine was AMD Athlon II 2.70 GHz using Matlab 2009b. Cplex was given a time limit of 3600 seconds.



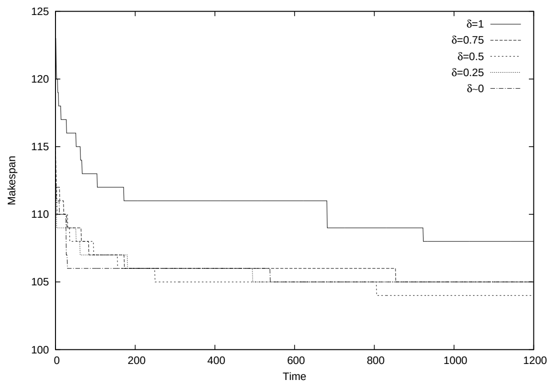
(a) $\mathcal{A}'(\delta)$ Average Makespan



(b) $\mathcal{B}(\delta)$ Average Makespan



(c) $\mathcal{A}'(\delta)$ Best Makespan



(d) $\mathcal{B}(\delta)$ Best Makespan

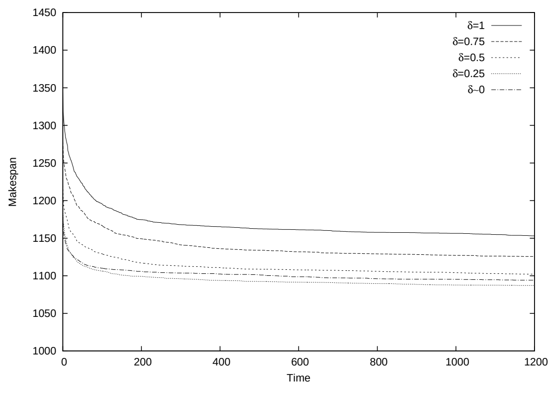
Fig. 9. Convergence of the GA for the instance j1205_1_06_9op derived from the RCPSP benchmark, combining the search spaces $\mathcal{A}'(\delta)$ and $\mathcal{B}(\delta)$ with different δ values and the coding-back (C) option. Each plot represents the evolution of the mean/best makespan of the population averaged for 30 runs.

As far as we know, these are the best result reported for the SPSO.

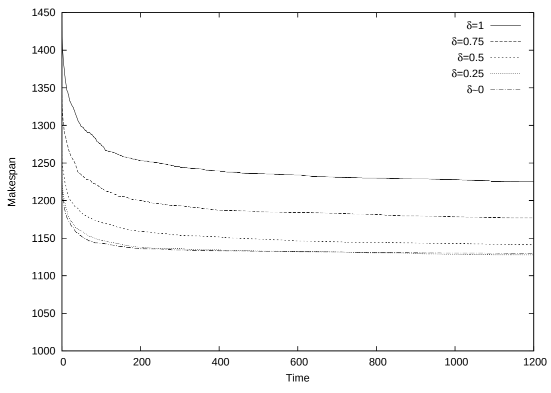
Table 5 summarizes the results produced by MSB-DOS and Cplex, reported in [12], together with the results of our genetic algorithm (GA). In accordance with previous results, we chosen $\mathcal{X} = \mathcal{A}'(\delta = 1)$, and for the purpose of comparison with the other methods $timeLimit = 5s$. GA was run 30 times for each instance. Considering average values, Cplex can reach optimal solutions for all the instances taking 419.04s. MSB-DOS takes much lower time, 34.67s, but obtains optimal solutions for only 6.4 instances in each subset with average gap of 0.37%. Regarding GA, it was able to obtain optimal solutions for every instance in at least one run, and obtained optimal solutions in all 30 runs for all instances of subsets 2 and 3, while it obtained optimal solutions in 86.33%, 95.67% and 98.67% of the runs in subsets 1, 4 and 5 respectively. The hardest instance for GA was the 8th one in set 1 which was optimally solved in only 2 runs, and the second hardest was the 6th instance of set 4 which was optimally solved 18 times. Overall, GA obtained optimal solutions in all 30 runs for 43 instances and in 96.13% of the 1500 runs. Therefore, despite of the difference on the target machines, we can conclude that GA is faster than the methods proposed in the literature, and finds much better solutions than any other approximate method.

7. Conclusions and future work

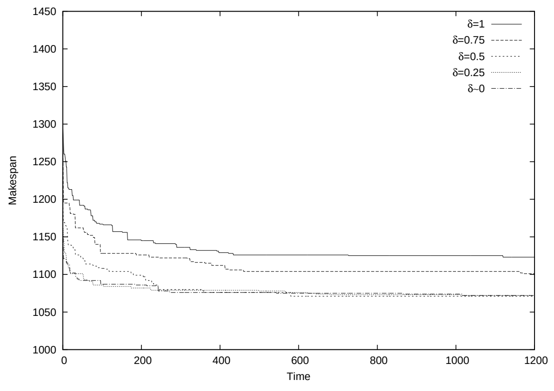
We have seen that the well-known $G\&T$ schedule builder proposed in [22] for the classic JSP problem can be extended to efficiently solve the SPSO proposed in [12], in a similar way as it was previously extended to other problems such as the $EG\&T$ for the JSP with Sequence Dependent Setup times [5], the



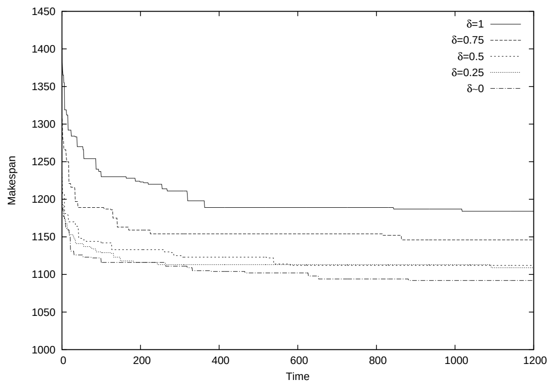
(a) $\mathcal{A}'(\delta)$ Average Makespan



(b) $\mathcal{B}(\delta)$ Average Makespan



(c) $\mathcal{A}'(\delta)$ Best Makespan



(d) $\mathcal{B}(\delta)$ Best Makespan

Fig. 10. Convergence of the GA for the instance LA21_06_9op derived from the JSP benchmark, combining the search spaces $\mathcal{A}'(\delta)$ and $\mathcal{B}(\delta)$ with different δ values and the coding-back (C) option. Each plot represents the evolution of the mean/best makespan of the population averaged for 30 runs.

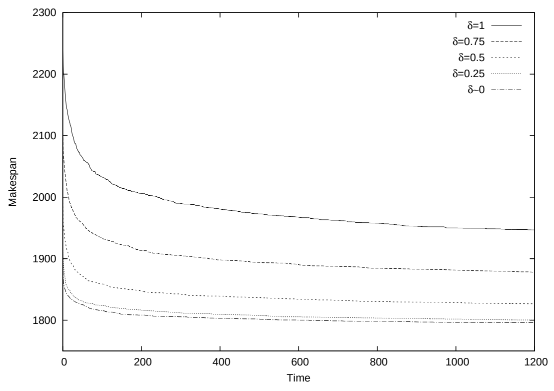
$OG\&T$ for the JSO [21] or the $fG\&TSGS$ for the JSP with fuzzy processing times [29,30]. The formalization of the $SOG\&T$ algorithm required a thorough analysis of the SPSO which is this paper was done with the aid of the proposed disjunctive model. From different particularizations of the $SOG\&T$, we devised a number of dominant and parameterized non-dominant search spaces for the SPSO and analyzed their suitability depending on the size of the problem instances.

Much in the same way as $G\&T$, $EG\&T$, $fG\&T$-SGS and $OG\&T$ were exploited in [24], [6], [29] and [27] respectively, $SOG\&T$ has been exploited here to devise a decoder which is the core of the genetic algorithm proposed to solve the SPSO. The success of this algorithm relies not only on the capability of $SOG\&T$ to generate schedules in different search spaces, but also in the proposed coding schema that encodes candidate solutions by means of a conventional permutation of tasks and a permutation with repetition of operators. We have demonstrated that for large instances, the best performance is obtained when the GA is restricted to search on a reduced non-dominant search space, while for small instances searching on a dominant unrestricted search space is preferable.

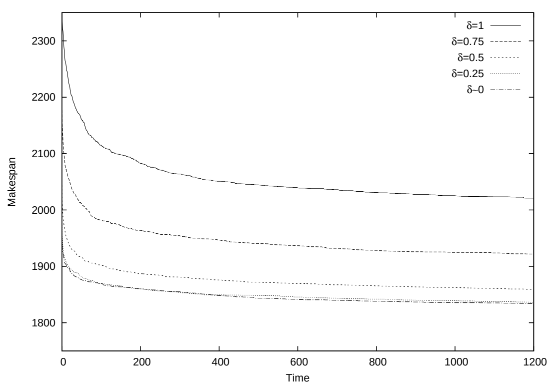
As future work, we plan to enhance the genetic algorithm, to exploit the $SOG\&T$ in other frameworks and also to consider new characteristics of the scheduling problems.

Our next step will be combining local search algorithms with the genetic algorithm. To this end, we will devise neighborhood structures for the SPSO from the proposed disjunctive model. In principle, we will try to extend the structures proposed in [31,32], but we will also consider new structures aiming at exploring changes in the assignment of operators.

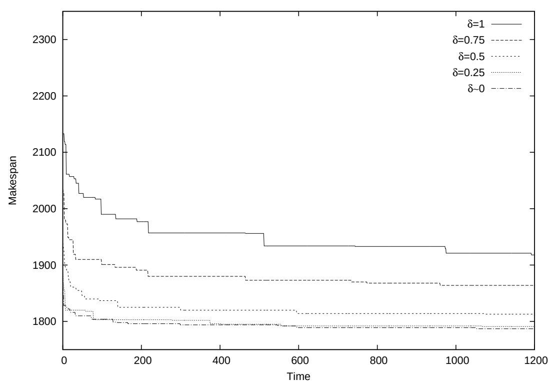
Furthermore, we will tackle the SPSO in the framework of heuristic search using $SOG\&T$ as branching scheme, in this case the set of options $\mathcal{B}$ will be likely



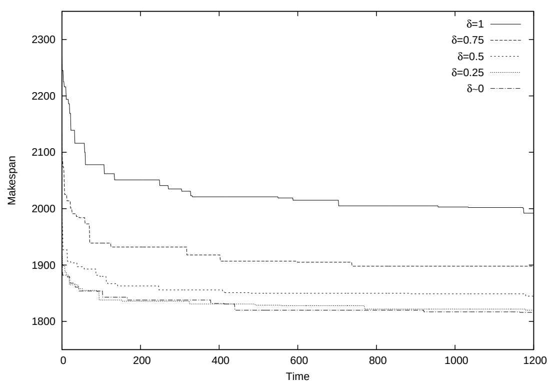
(a) $\mathcal{A}'(\delta)$ Average Makespan



(b) $\mathcal{B}(\delta)$ Average Makespan



(c) $\mathcal{A}'(\delta)$ Best Makespan



(d) $\mathcal{B}(\delta)$ Best Makespan

Fig. 11. Convergence of the GA for the instances LA31_06 derived from the LA31, combining the search spaces $\mathcal{A}'(\delta)$ and $\mathcal{B}(\delta)$ with different δ values and the coding-back (C) option. Each plot represents the evolution of the mean/best makespan of the population averaged for 30 runs.

the best option as it gives rise to the smallest dominant search space. In this setting, the disjunctive graph model could help to devise consistent heuristics and dominance rules. We hope this strategy will perform well as happened, for example, in [33] for the JSO. It will also be interesting to consider and evaluate constraint programming approaches on the SPSO, both general frameworks as CPLEX CP Optimizer [34,35], or more specific algorithms such as the successful Solution Guided Search [36].

With regards to adding new real-life characteristics of the production systems, we plan to consider uncertainty and multi-criteria optimization. These elements were already considered in other scheduling problems such as, for example, the open shop scheduling [37] with uncertainty on the processing time of the tasks, and the pipe spool fabrication problem tackled in [38]. In both cases, the authors used concepts of fuzzy sets theory to model uncertainty. Also, energy-aware scheduling is a recent topic of interest in theory and practice; it was considered, for example, in [39] and [40] and requires minimizing at the same time objective functions accounting for the use of energy and others such as makespan.

8. Acknowledgments

This research has been supported by the Spanish Government under research project TIN2013-46511-C2-2-P. Carlos Mencía is supported by SFI grant BEACON (09/IN.1/I2618).

References

[1] Garey MR, Johnson DS. Computers and Intractability; A Guide to the Theory of NP-Completeness. New York, NY, USA: W. H. Freeman & Co.; 1979.
[2] Brucker P. Scheduling algorithms (4. ed.). Springer; 2004.
[3] Brucker P, Knust S. Complex Scheduling. Springer; 2006.

Table 2

Summary of results for instances obtained from FT10; in all cases $n = 100$ and $q = 10$. For each instance, values of $\delta \in \{\sim 0, 0.25, 0.5, 0.75, 1\}$ were considered. The results are averaged for instances with the same values of p and Pr. $timeLimit = 120s$.

		$\mathcal{A}'(\delta \sim 0)$			$\mathcal{A}'(\delta = 0.25)$			$\mathcal{A}'(\delta = 0.5)$			$\mathcal{A}'(\delta = 0.75)$			$\mathcal{A}'(\delta = 1)$		
Pr	p	%Err. Best	%Err. Avg	%CV	%Err. Best	%Err. Avg	%CV	%Err. Best	%Err. Avg	%CV	%Err. Best	%Err. Avg	%CV	%Err. Best	%Err. Avg	%CV
2	5	1.62	3.71	0.95	1.11	2.53	0.92	0.43	2.53	1.03	0.48	3.01	1.40	1.16	4.37	1.57
	7	2.55	4.46	1.03	1.41	3.27	1.00	0.09	2.61	1.20	0.52	2.99	1.29	1.52	4.39	1.45
	9	1.87	4.26	1.03	1.28	3.66	1.00	0.57	2.88	1.12	0.00	2.72	1.30	0.84	3.83	1.79
Avg.		2.01	4.14	1.00	1.27	3.15	0.97	0.36	2.67	1.12	0.33	2.91	1.33	1.17	4.19	1.60
6	5	0.96	2.33	0.68	0.15	1.69	0.76	0.30	2.08	0.87	0.98	3.26	1.02	2.19	5.35	1.37
	7	0.67	2.33	0.85	0.25	2.00	0.98	0.46	2.38	1.10	0.55	3.43	1.45	2.13	5.63	1.59
	9	0.92	2.23	0.67	0.21	1.77	0.83	0.21	1.90	1.08	0.21	2.50	1.11	0.64	4.07	1.53
Avg.		0.85	2.30	0.74	0.20	1.82	0.86	0.32	2.12	1.01	0.58	3.06	1.19	1.65	5.02	1.49
2/6	5	1.39	3.25	0.88	0.14	2.81	1.07	0.75	2.62	0.97	1.10	3.41	1.13	2.29	5.79	1.48
	7	1.51	3.72	0.99	0.89	3.12	1.17	0.08	2.57	1.31	0.53	3.42	1.54	1.50	4.81	1.72
	9	1.09	3.22	1.14	0.86	2.68	1.12	0.21	2.56	1.20	0.10	2.78	1.40	1.32	4.41	1.72
Avg.		1.33	3.40	1.00	0.63	2.87	1.12	0.35	2.58	1.16	0.58	3.20	1.36	1.71	5.00	1.64
T. Avg.		1.40	3.28	0.91	0.70	2.61	0.98	0.34	2.46	1.10	0.50	3.06	1.29	1.51	4.74	1.58

Table 3

Summary of results for medium size instances with $\delta \in \{\sim 0, 0.25, 0.5, 0.75, 1\}$ averaged for each subset of instances with the same n. $timeLimit = 150s$.

		$\mathcal{A}'(\delta \sim 0)$			$\mathcal{A}'(\delta = 0.25)$			$\mathcal{A}'(\delta = 0.5)$			$\mathcal{A}'(\delta = 0.75)$			$\mathcal{A}'(\delta = 1)$		
Instances	n	%Err. Best	%Err. Avg	%CV	%Err. Best	%Err. Avg	%CV	%Err. Best	%Err. Avg	%CV	%Err. Best	%Err. Avg	%CV	%Err. Best	%Err. Avg	%CV
j120	120	0.75	2.28	0.71	0.85	2.32	0.70	0.90	2.36	0.72	0.75	2.48	0.84	2.31	4.52	1.04
LA21-25	150	0.44	1.91	0.69	0.30	1.86	0.73	0.76	2.50	0.81	1.77	3.79	0.94	3.43	6.15	1.22
T. Avg.		0.59	2.09	0.70	0.58	2.09	0.71	0.83	2.43	0.76	1.26	3.14	0.89	2.87	5.33	1.13

[4] Pinedo ML. Scheduling: Theory, Algorithms, and Systems. 3rd ed. Springer Publishing Company, Incorporated; 2008.

[5] Artigues C, Lopez P, Ayache PD. Schedule Generation Schemes for the Job Shop Problem with Sequence-Dependent Setup Times: Dominance Properties and Computational Analysis. Annals of Operations Research. 2005;138:21–52.

[6] Vela CR, Varela R, González MA. Local Search and Genetic Algorithm for the Job Shop Scheduling Problem with Sequence Dependent Setup Times. Journal of Heuristics. 2010;16(2):139–165.

[7] Fortemps P. Jobshop scheduling with imprecise durations: a fuzzy approach. Fuzzy Systems, IEEE Transactions on. 1997;5(4):557–569.

[8] González Rodríguez I, Vela CR, Puente J, Varela R. A new Local Search for the Job Shop Problem with uncertain Durations. In: Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS-2008). Sidney: AAAI Press; 2008. .

[9] Brucker P, Schlie R. Job-shop scheduling with multi-purpose machines. Computing. 1990;45(4):369–375.

[10] González MA, Vela CR, Varela R. Scatter search with path relinking for the flexible job shop scheduling problem. European Journal of Operational Research. 2015;245(1):35 – 45.

[11] Agnetis A, Flamini M, Nicosia G, Pacifici A. A job-shop

Table 4

Summary of results for the large instances with $\delta \sim 0$ and $\delta = 1$. The results are averaged for each set of instances with the same n. $timeLimit = 300s$.

		$\mathcal{A}'(\delta \sim 0)$			$\mathcal{A}'(\delta = 1)$		
Instances	n	%Err. Best	%Err. Avg	%CV	%Err. Best	%Err. Avg	%CV
LA26-30	200	0.00	1.20	0.56	4.47	6.67	1.02
LA31-35	300	0.00	0.89	0.39	6.41	8.01	0.67
LA36-40	225	0.00	1.68	0.77	3.72	6.33	1.23
Tai_20_15	300	0.00	1.64	0.74	6.76	9.00	0.97
Tai_20_20	400	0.00	1.66	0.75	7.71	9.96	0.92
Tai_30_15	450	0.00	1.22	0.53	8.46	10.22	0.69
	T. Avg.	0.00	1.42	0.64	6.72	8.82	0.90

Table 5

Summary of results from MSB-DOS, Cplex and GA averaged for the 10 instances in each of the 5 subsets of instances proposed in [12] . #Opt. is the number of instances in each subset which are optimally solved. Time is given in seconds. GA was run 30 times for each instance with $timeLimit = 5.00$ in each run, %Opt. is the percentage of runs GA reached an optimal solution in each subset.

Sets				MSB-DOS			Cplex		GA		
#	n	p	q	GAP%	Time	#Opt.	Time	#Opt.	% Opt	Time	#Opt.
1	100	10	15	0.78	106.44	5.00	106.21	10.00	86.33	5.00	10.00
2	150	15	30	0.20	8.28	6.00	190.54	10.00	100.00	5.00	10.00
3	150	15	50	0.03	9.58	9.00	129.28	10.00	100.00	5.00	10.00
4	200	15	30	0.42	25.62	6.00	755.85	10.00	95.67	5.00	10.00
5	200	20	30	0.41	23.43	6.00	913.32	10.00	98.67	5.00	10.00
Average				0.37	34.67	6.40	419.04	10.00	96.13	5.00	10.00

problem with one additional resource type. J Scheduling. 2011;14(3):225–237.

[12] Agnetis A, Murgia G, Sbrilli S. A job shop scheduling problem with human operators in handicraft production. International Journal of Production Research. 2014;52(13):3820–3831.

[13] Mencía R, Sierra MR, Mencía C, Varela R. Schedule Generation Schemes and Genetic Algorithm for the Scheduling Problem with Skilled Operators and Arbitrary Precedence Relations. In: Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015, Jerusalem, Israel, June 7-11, 2015.; 2015. p. 165–173.

[14] Mencía R, Sierra MR, Varela R. Genetic Algorithm for the Job-Shop Scheduling with Skilled Operators. In: Bioinspired Computation in Artificial Systems - International Work-Conference on the Interplay Between Natural and Artificial Computation, IWINAC 2015, Elche, Spain, June 1-5, 2015, Proceedings, Part II; 2015. p. 41–50.

[15] Artigues C, Gendreau M, Rousseau LM, Vergnaud A. Solving an integrated employee timetabling and job-shop scheduling problem via hybrid branch-and-bound. Computers & Operations Research. 2009;36(8):2330 – 2340.

[16] Guyon O, Lemaire P, Pinson E, Rivreau D. Solving an integrated job-shop problem with human resource constraints. Annals of Operations Research. 2014;213(1):147–171.

[17] Benavides AJ, Ritt M, Miralles C. Flow shop scheduling with heterogeneous workers. European Journal of Operational Research. 2014;237(2):713–720.

[18] Mencía R, Sierra MR, Mencía C, Varela R. Memetic algorithms for the job shop scheduling problem with operators. Appl Soft Comput. 2015;34:94–105.

[19] Adams J, Balas E, Zawack D. The Shifting Bottleneck Procedure for Job Shop Scheduling. Management Science. 1988;34(3):391–401.

[20] Roy B, Sussman B. Les problemes d'ordonnancements avec contraintes disjonctives. Notes DS no. 9 bis, SEMA, Paris; 1964. .

[21] Sierra MR, Mencía C, Varela R. New schedule generation schemes for the job-shop problem with operators. Journal of Intelligent Manufacturing. 2015;26(3):511–525.

[22] Giffler B, Thompson GL. Algorithms for Solving Production Scheduling Problems. Operations Research. 1960;8:487–503.

[23] Bierwirth C, Mattfeld DC. Production Scheduling and Rescheduling with Genetic Algorithms. Evol Comput. 1999 Mar;7(1):1–17.

[24] Mattfeld DC. Evolutionary Search and the Job Shop Investigations on Genetic Algorithms for Production Scheduling. Springer-Verlag; 1995.

[25] Bierwirth C. A generalized permutation approach to job shop scheduling with genetic algorithms. OR Spectrum. 1995;17:87–92.

[26] Gonçalves JF, Resende MGC. An extended Akers graphical method with a biased random-key genetic algorithm for job-shop scheduling. International Transactions in Operational Research. 2014;21:215–246.

[27] Mencía R, Sierra MR, Mencía C, Varela R. A genetic algorithm for job-shop scheduling with operators enhanced by weak Lamarckian evolution and search space narrowing. Natural Computing. 2014;13(2):179–192.

[28] Beasley JE. OR-Library: Distributing Test Problems by Electronic Mail. J Oper Res Soc. 1990;41(11):1069–1072.

[29] González Rodríguez I, Vela CR, Puente J. A Memetic Approach to Fuzzy Job Shop Based on Expectation Model. In: Proceedings of IEEE International Conference on Fuzzy Systems, FUZZ-IEEE2007. London: IEEE; 2007. p. 692–697.

[30] Palacios JJ, Vela CR, González-Rodríguez I, Puente J. Schedule Generation Schemes for Job Shop Problems with Fuzziness. In: ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic; 2014. p. 687–692.

[31] Dell' Amico M, Trubian M. Applying Tabu Search to the Job-shop Scheduling Problem. Annals of Operational Research. 1993;41:231–252.

[32] Laarhoven PJMv, Aarts EHL, Lenstra JK. Job Shop Scheduling by Simulated Annealing. Operations Research. 1992;40(1):pp. 113–125.

[33] Mencía C, Sierra MR, Varela R. An efficient hybrid search algorithm for job shop scheduling with operators. International Journal of Production Research. 2013;51(17):5221–5237.

[34] Laborie P. IBM CP Optimizer for Detailed Scheduling Illustrated on Three Problems. In: van Hoeve WJ, Hooker J, editors. Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. vol. 5547 of Lecture Notes in Computer Science. Springer Berlin Heidelberg; 2009. p. 148–162.

[35] Vilím P, Laborie P, Shaw P. Failure-directed Search for Constraint-based Scheduling. In: CPAIOR '15: Proceedings of the 12th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. Springer-Verlag; 2015. p. 437–453.

[36] Beck JC, Feng TK, Watson J. Combining Constraint Programming and Local Search for Job-Shop Scheduling. INFORMS Journal on Computing. 2011;23(1):1–14.

[37] Palacios JJ, González-Rodríguez I, Vela CR, Puente J. A particle swarm solution based on lexicographical goal programming for a multiobjective fuzzy open shop problem. AI Communications. 2015;28(2):239–257.

[38] Rokni S, Fayek AR. A Multi-criteria Optimization Framework for Industrial Shop Scheduling Using Fuzzy Set Theory. Integr Comput-Aided Eng. 2010 Aug;17(3):175–196.

[39] Li XX, Li WD, Cai XT, He FZ. A hybrid optimization approach for sustainable process planning and scheduling. Integr Comput-Aided Eng. 2015;Preprint(Preprint):1–16.

[40] Dai M, Tang D, Salido MA, Li WD, Li WD, Cai XT, et al. Energy-efficient scheduling for a flexible flow shop using an improved genetic-simulated annealing algorithm. Robotics and Computer-Integrated Manufacturing. 2013;29(5):418–429.