Manuscript Number:

Title: An agent-based genetic algorithm for hybrid flowshops with sequence dependent setup times to minimise makespan

Article Type: Full Length Article

Keywords: Hydrid flowshop, sequence dependent setup times, agent, genetic algorithm, makespan.

Corresponding Author: Mr Pedro Gomez-Gasquet, Ph.D.

Corresponding Author's Institution: Universidad Politécnica de Valencia

First Author: Pedro Gomez-Gasquet, Ph.D.

Order of Authors: Pedro Gomez-Gasquet, Ph.D.; Pedro Gomez-Gasquet, Ph.D.; Carlos Andrés, Ph.D.; Francisco Cruz Lario-Esteban, Ph.D.

## Highlights

> *This paper deals with a variant of flowshop scheduling, namely, the hybrid or flexible flowshop with sequence dependent setup times.* > *An improved genetic algorithm (GA) based on software agent design to minimise the makespan is presented.* > *Computational experiments are conducted on a well-known benchmark problem dataset.*

# An agent-based genetic algorithm for hybrid flowshops with sequence dependent setup times to minimise makespan

*Pedro Gómez-Gasquet*[a*], *Carlos Andrés*[b], *Francisco-Cruz Lario*[a]

[a] Centro de Investigación de Gestión e Ingeniería de la Producción, Universitat Politècnica de València, Cno. de Vera s/n, Valencia, 46022, Spain. {pgomez, fclario}@cigip.upv.es

[b] Research Group in Reengineering , Operations management, Group work and Logisitics excellence, Universitat Politècnica de València, Cno. de Vera s/n, Valencia, 46022, Spain. candres@doe.upv.es

*Corresponding author. Address: Centro de Investigación de Gestión e Ingeniería de la Producción, Universitat Politècnica de València, Cno. de Vera s/n, Valencia, 46022, Spain. Tel.: +34 963 879 680; fax: +34 963 879 682

E-mail address: pgomez@cigip.upv.es

Abstract

*This paper deals with a variant of flowshop scheduling, namely, the hybrid or flexible flowshop with sequence dependent setup times. This type of flowshop is frequently used in the batch production industry and helps reduce the gap between research and operational use. This scheduling problem is NP-hard and solutions for large problems are based on non-exact methods. An improved genetic algorithm (GA) based on software agent design to minimise the makespan is presented. The paper proposes using an inherent characteristic of software agents to create a new perspective in GA design. To verify the developed metaheuristic, computational experiments are conducted on a well-known benchmark problem dataset. The experimental results show that the proposed metaheuristic outperforms some of the well-known methods and the state-of-art algorithms on the same benchmark problem dataset.*

*Keywords:* Hydrid flowshop, sequence dependent setup times, agent, genetic algorithm, makespan.

## 1. Introduction

Among the production scheduling systems, the flowshop scheduling (FS) problem is one of the most distinguished environments Gupta and Stafford (2006). In FS, all jobs follow the same operational order (processing route) and need to be processed at every stage. This problem could be considered as the foundation of several other interesting formulations.

Since the publication of the Arthanari and Ramamurthy (1971) seminal paper until the recent review of Ribas et al. (2010), the hybrid flowshop scheduling (HFS) problem has been of continuing interest for researchers and practitioners. When using several unrelated machines in parallel during HFS production stages it is important not to confuse the hybrid flowshop with the flowshop with multiple processors (FSMP), or the flexible flow line (FFL) problems. In these two latter problems, the machines available at each stage are identical. In all shop scheduling problems, the purpose is to discover a production sequence for the jobs on the machines such that a certain criteria, or a set of selected criteria, are optimised.

In HFS scheduling, the task of creating a feasible schedule is quite complicated. The search space of feasible schedules grows exponentially, as there are certain increases in the number of different jobs that must be processed and the number of facilities that can perform the process of each product. The HFS with two stages (with one machine in the first stage, and several machines in the second stage) is NP-hard Gupta (1988). Moreover, Hoogeveen et al. (1996) demonstrated that this problem remained NP-hard even if pre-emption is allowed.

The majority of papers assume that setup time is negligible, or part of the job processing time. But explicit setup times must be included in scheduling decisions in order to address a more realistic variant of hybrid flowshop scheduling problems. This inclusion avoids the adverse effect on the solution quality of using multiple scheduling applications. Several explicit setup times can be considered; however, the issue of sequence dependent setup time (SDST) is gaining increasing attention among researchers. The resultant problem is called the sequence dependent setup time (SDST) hybrid (flexible) flowshop; and with the makespan criterion it is denoted as FF/STsd/Cmax Allahverdi et al. (2008). An explicit mathematical model for the problem is developed in Andrés, (2001).

This problem is more complex than HFS and belongs to the NP-hard set of problems.

This paper presents an innovative solution based on the genetic algorithm (GA) method, yet designed and developed under the agent software paradigm. Both approaches have provided successful proposals for similar problems. Both are now combined to create a new GA with features not seen in the traditional GA.

This paper is structured as follows: Section 2 reviews the SDST hybrid flowshop literature. Section 3 discusses the proposed agent-based genetic algorithm and we present the new genetic algorithm in detail. In Section 4, an experimental analysis is carried out, along with a complete calibration using the design of experiments (DoE) approach. A comparative analogy between our algorithm and other effective algorithms in the literature is presented. Finally, Section 5 concludes the paper and introduces some directions for future work.

## 2. Literature review of HFS problem with makespan criterion

To focus the literature review we analyse the most relevant contributions related to HFS and minimising makespan. Firstly, we discuss the cases that do not consider SDST, and secondly, those that consider SDST. A comprehensive analysis of general HFS problems for this criterion is found in Hejazi and Saghafian (2005).

The term 'hybrid flowshop' was first used by Gupta (1988) in a shop with two stages, being multiprocessor-only in the first stage, and where it was demonstrated to be NP-hard. Hundreds of contributions have subsequently analysed the progress of the HFS problem state.

During the last year, several interesting literature reviews and surveys have analysed and classified various proposals regarding the HFS problem. The HFS problem is commented in the works of Vignier et al. (1995) and Vignier et al. (1999). In Vignier et al. (1999) the work is split into two parts. The first part is focused on two-stage flowshop problems, and the second stage on the general k-stage problem. The authors do not identify any solved problems related to makespan and average flow time Kis and Pesch (2005) and provide an extended literature review about exact methods in HFS. The authors focus on branch and bound (B&B) and constraint propagation techniques. An interesting classification of resolution methods is offered in Quadt and Kuhn (2007). Ribas et al. (2010) recently classified papers according to HFS characteristics and production limitations. This represents a new approach to the classification of papers in the HFS environment. Papers are also classified according to the proposed solution approach.

Heuristic and metaheuristic solution approaches are dominant, but some exact methods are often used for simple cases. Branch and bound (B&B) and dynamic programming techniques are the main actors on the stage of the exact techniques. In Gupta and Tunc (1991), Brah and Hunsucker (1991), Rajedran and Chaudhuri (1992) or Lee et al. (1993) various B&B solutions for minimising makespan are defined. Some authors develop dynamic programming algorithms to solve the problem optimally with makespan criterion. A two-machine flowshop scheduling problem where machines are not always available is studied in Lee (1997) and extended in Lee (1999). A two-stage HFS problem with one machine in the first stage, and two different machines in parallel in the second stage, is discussed in Riane et al. (2002).

When problems grow in complexity or data volume, authors usually propose approximate methods. For a basic review of HFS under the makespan criterion, several useful papers can be found. Early papers by Shen and Chen (1972) and Sriskandarajah and Sethi (1989) present two heuristics based on the Johnson algorithm; while Gupta (1988) introduces a new heuristic based on the longest processing time index. Over the last decade, metaheuristic and evolutionary approaches have been proposed that are used alone, or in combination with traditional heuristics. The authors of Haouari et al. (1997) propose two approximate methods developed in two phases. The first solution is generated using a longest remaining work rule. This schedule is improved using techniques based on simulated annealing and tabu search. Both solutions are encoded with the list used in the first phase. In Portmann et al. (1998) an improvement on the B&B proposed by Brah et al. (1991) is analysed. From the definition of improved levels, the authors present a metaheuristic using a GA in the B&B procedure to improve the values of the upper bound in certain stages. In Nowicki et al. (1998) a tabu search with graph representation is introduced. In Riane et al. (1998), a problem of scheduling n jobs on a three-stage HFS of a particular structure (one machine in the first and third stages, and two dedicated machines in stage two) is discussed. The objective is to minimise the makespan. The authors propose two heuristic procedures to cope with realistic problems. More recently, bio-inspired methods have gained relevance. GAs are proposed in Serifoglu and Ulusoy (2004) and Gao et al. (2006).

A particular case within HFS problems is reached when SDST constraint is included. This constraint corresponds to quite common situations in industry, and reduces the gap between theoretical and realistic models by considering events as common as changing colours, formats, etc., in production processes. This may be one of the least treated aspects historically, yet considerable interest has been generated in recent years. It should be noted that the proposals submitted by different authors are usually based on approximate methods.

During the last century some innovative works began to address the problem of HFS with SDST. In Yoshida and Hitomi (1979) flowshop problems were developed to consider setup times separate from processing times. Gupta and Darrow (1986) present several heuristics for the case of SDST two-machine flowshops. Gupta and Tunc (1994) discuss the two-stage HFS scheduling problem where the setup and removal times for each job at each stage are separated from the processing times. Four heuristic algorithms are developed for the case where there is one machine at stage one and the number of identical parallel machines at the second stage is less than the total number of jobs. In Aghezzaf et al. (1995a) and Aghezzaf et al. (1995b) a two-step heuristic algorithm is proposed for minimising makespan in the textile industry. A two-stage flowshop is solved in Andrés et al. (1998) using a genetic algorithm. The proposed GA defines the sequence in the first step and allocates jobs to machines in the second step. During these early years, the problem was addressed by relatively simple instances. Yang and Liao (1999) review static scheduling research in which setup time, or cost, is of main concern in the problem studied, and FHS is one of the reviewed cases.

During the first years of this century the complexity of the problem has grown and the methods have tended towards the use of pure or hybrid metaheuristics. Some relevant examples that focus on minimising the makespan are presented below. Allahverdi et al. (2008) present an extended survey regarding setup time consideration, with and without sequence dependency. The authors also analyse HSF problems.

Kurz and Askin (2004) highlight the difficulty in solving integer programming directly, and several heuristics are developed, based on greedy methods, flow line methods, and insertion heuristics for the travelling salesman problem, as well as the random keys genetic algorithm. Zandieh et al. (2006) propose an immune algorithm, and show that their algorithm outperforms the RKGA of Kurz and Askin (2004). Group scheduling within the context of a problem is introduced in Logendran et al. (2006). A search algorithm that uses short-term memory is recommended for problems of all sizes and levels of flexibility.

Ruiz and Maroto (2006) analyse a gap between theory and practice in the context of HFS. The authors introduce a new GA considering SDST and machine eligibility issues that are usual in the ceramic tile industry. New iterated greedy (IG) algorithms are proposed by Ruiz and Stutzle (2008) for minimising makespan and minimising total weighted tardiness. The first IG algorithm is a straightforward adaption of the IG principle, while the second incorporates a simple descent local search. Ruiz et al. (2008) propose a formulation along with a mixed integer modelisation and several heuristics for scheduling jobs in stages, where at each stage, there is a known number of unrelated machines. The authors also consider anticipatory and non-anticipatory SDST along with machine lag, release dates for machines, machine eligibility, and precedence relationships among jobs. Yaurima et al. (2009) present a GA for the HFS with unrelated machines, SDST, and availability constraints. The proposed GA is a modified and extended version of the algorithm for a problem without limited buffers. The GA takes into account additional limited buffer constraints and uses a new crossover operator and stopping criteria. In Behnamian et al. (2009) the authors consider problems with the objectives of minimising the makespan and sum of the earliness and tardiness of jobs, and present a multi-phase method. In the first phase, the population is decomposed using a GA into several sub-populations to obtain a good approximation of the Pareto front. In the second phase, non-dominant solutions are unified as one large population base for a local search. Finally, in phase three, the gaps between the non-dominated solutions and the improved Pareto front are covered using a hybrid metaheuristic. In work carried out by Naderi et al. (2010), the authors propose two advanced algorithms that specifically deal with parallel machines and setup characteristics of the addressed problem. The first algorithm is a dynamic dispatching rule heuristic, and the second is an iterated local search metaheuristic.

In the above literature there is almost no work on hybrid flowshops considering SDST. We can consider that even today it is possible to improve on current results. For this reason, we have developed a GA for this complex problem.

## 3. MAGSA algorithm

This section describes the multi-agent genetic scheduling algorithm (MAGSA). Section 2 shows that as problems become more complicated, metaheuristic solutions are more often used. In particular, metaheuristic solutions are dominant for bio-inspired problems, one of the most common being genetic algorithms. However, the task of improving the results obtained with genetic algorithms is difficult using the traditional scheme. The idea of revising the methodology based on new and interesting approaches is an attractive idea. The implementation framework and results obtained from the software agent paradigm suggest that this is a promising line of work.

Proposals based on multi-agent systems come mainly from the theories of distributed artificial intelligence (DAI) and have produced some interesting results Shen et al. (2006), Toptal and Sanbucuoglu (2010). Researchers who have developed proposals for the sequencing problem have done so primarily under the consideration that contributions would be made within the presented agent system architecture, and using the tools that the proposed agent 'society' system would use for sequencing tasks Kutanoglu and Wu (1999), Ng et al. (2006)). Other authors have used the characteristics of these systems to tackle more complex issues such as the integration of planning and sequencing Lim and Zhang (2004), Sanjay and Young (2008). However, few authors have tried to apply the very essence of the agent (autonomy, sociability, responsiveness, initiative, and rationality) to the design of methods already available (AG, Ant Systems, etc.). In other words, the development of agent-based systems, rather than multi-agent systems.

With the aim of creating a genetic algorithm consisting of individuals with features that are richer than traditional individuals, we start from the structure presented by Zhong et al. (2004), which is a proposal for a 'multiagent genetic algorithm for a global numerical optimisation'. This proposal performs an adaptation of the representation and the genetic operators in order to enable it to address a problem that is as radically different as sequencing.

As a starting point, and with the general idea of providing genetic algorithms with a greater affinity to the behaviour of natural systems, and taking the agent concept as a reference, we propose to enhance MAGSA:

- The generation of new individuals based on **local competition** as occurs in nature, and not global competition as proposed in most genetic algorithms.
- Strengthen the learning ability so that the **dynamic adjustment of certain parameters** can be made based on the circumstances of a changing environment.
- Encourage the differentiation of individuals with a **customised application of genetic operators**.

To achieve this, we modified the traditional functional layout of genetic algorithms with the proposal of Figure 1 that introduces a genetic learning stage, a grid-shaped structure for the population, and an application of genetic operators in the characteristic way. This is explained in more detail in the section below.

Figure 1. Functional structure of MAGSA

### a. Population structure, encoding, and initialisation

A genetic algorithm works on individuals with chromosomes, which are a representation or codification of the solutions to the problem. In this case, we have chosen an ordinal genetic representation. As shown in Figure 2, the individuals are identified by sequences so each element of the sequence is associated with a numeric identifier that represents a particular job.

Figure 2. Ordinal representation of a chromosome

From a sequence it is possible to calculate a hybrid flowshop using a simple rule. In this case, jobs are selected in the sequence order, and each job is assigned to a machine before the end of each its operations. Another task is not assigned until all previous operations have been assigned.

A proposal by Zhong et al. (2004) has been followed for the population and a square lattice has been defined that is made of individuals who can only communicate with their neighbours. These neighbours have been designed as agents, which we will term *agent-solutions*. Each agent-solution includes self-interest and a logic of action to achieve its interests. Additionally, we have designed an *agent-manager* that will act as a controller and ensure the rules of the algorithms are respected.

Figure 3. Model of the agent lattice

This lattice, along with the permitted connection types, reduces the communication of each agent-solution to a small area (contiguous neighbours), so that their relationships can be considered as local. The population maintains a constant size with a 6x6 lattice, and preliminary studies have shown this to offer a good balance between evolution (convergence and pressure) and computation time.

The generation of the population consists of establishing the lattice by creating the individuals. Each individual must contain a sequence or chromosome from which a solution that characterises it can be obtained. The lives of the individuals have the same duration as the algorithm, meaning that the agent-solution does not die until the algorithm ends. However, the content of the chromosome can change often.

The sequence value is generated independently by each individual using a greedy algorithm that aims to reduce the accumulated setup time. To simplify this process, the algorithm is calculated considering only one randomly chosen operation. From the goodness of the solutions, the agents learn which is the most interesting operation with regard to setup time. If the *bottleneck switch* (BS) is activated, agents can take advantage of the experience of each individual and share information with each other to facilitate the selection of the stage as a benchmark.

### b. Genetic operators

#### i. Crossover

The MAGSA algorithm uses two operators, selected from those that are considered able to offer the best results, and has again left it to the agents to use their acquired knowledge to select which operator to apply. Future work may increase the number of operators.

One of the selected crossover operators is called 'similar block 2-point order crossover (SB2OX)', which was used by Ruiz and Maroto (2006) with great success. The other operator is an adaptation of the 'neighbourhood competition operator (NCO)' used Zhong et al. (2004), whose implementation has not been tested for the problem of the hybrid flowshop with sequence dependent setup times.

The NCO operator needs only one parent (S2BOX needs two parents), which is selected by local competition between five individuals, the individual on which it is operating and four neighbours. The winning individual has the chromosome that achieves the smallest makespan solution. Once selected:

1. Two crossover points are chosen at random as shown in Figure **4.** Two values are obtained using a uniform distribution between 1 and the size of the sequence.

Figure 4. NCO crossover operator - step one

2. All the genetic information that is not between cut-off point 1 and cut-off point 2, is transferred from father to son.

3. The genetic information found between the two cut-offs constitutes a partial sequence that is transferred from father to son, in such a way that the position of the genes in the child corresponds to the reverse of their position in the partial sequence of the father's genes. Figure 5 shows the final step.

*Figure 5. NCO crossover operator - step three*

There are two steps to a crossover operation:

- Determine if an agent is to be crossed.
- Decide which of the two operators to apply.

Agents can be crossed only for a given percentage of occasions. This probability is related to the crossover factor that each agent-solution maintains with an individualised value. There are four possible initial values (0.2, 0.4, 0.6, and 0.8) that are randomly assigned among the individuals at the moment of their creation. The agents can vary this value if the parameter *crossover factor switch* (CFS) is enabled, otherwise it will keep the initial value. If the factor can evolve, each time an agent-solution is considered for a cross, the crossover factor is reduced by 0.05 units regardless of whether the cross is actually performed – and so reducing the possibilities that this operator is applied. If the chromosome value is changed, the crossover factor is reinitiated. To avoid values that are too low, a parameter called the *minimum crossover factor* (MCF) has been defined.

The selection of a crossover operator is based on a variable with uniform distribution, controlled by each agent-solution and termed the *crossover operator distribution*. This variable is initially fixed with a 50% distribution, which is the likelihood of using one of the two operators. Each individual modifies the probability distribution after evaluating the solutions obtained in a set of crossover operations. Increases or decreases by a value of 0.01 can be made in the value of the crossover operator distribution to improve the probability of the operator that has achieved best results in the makespan. In any case, the value of the crossover operator distribution ranges from a minimum of 0.1 to a maximum of 0.9. It is possible to prevent the agent-solution changing the value of the variable specified by inhibiting the learning process with the *crossover operator switch* (COS) parameter (enable, disable).

### ii. Mutation

The mutation operator is usually much simpler than the crossover operator, and normally achieves its purpose with a simple operation. The proposed mutation operator is based on an exchange of positions. It is a proximity-based mutation operator which given the position of a chromosome or gene whose location in the sequence is 'i' selects a gene 'j' that is located randomly between [i +1, i +3]. In calculating the location of the gene j, it is assumed that the sequence is cyclical, and if there are n jobs then the position n +1 of the sequence is position 1. An example can be seen in Figure 6.

*Figure 6. Mutation operator*

After applying the crossover operator to an individual there is an opportunity to apply the mutation operator on the individual according to a global random factor, meaning a factor whose value is unique and shared by all the solution-agents. This is termed the *mutation factor* (MF). If an individual is selected to apply the mutation operator, it tries to perform the operation with all of its genes. Each gene has a probability 1/n of being chosen.

#### iii. Genetic learning

Genetic learning is the name given to the proposed stage of the genetic algorithm during which an exploratory analysis is made in the vicinity of a given solution. This activity is developed in each generation for each of the agent-solutions that make up the matrix base. The activity consists in generating an initial population from the sequence of a given agent-solution, as shown in Figure 7, and applying the MAGSA simplified algorithm.

*Figure 7. Base matrix with learning matrix*

The objective is to run one genetic algorithm inside another, both algorithms being very similar. Figure 8 summarises the stages into which genetic learning, or the simplified MAGSA process, is divided.

*Figure 8. Genetic learning schema*

The approach is based on a short run and considerable freedom of action. This approach implies the removal of some of the constraints imposed in the main process.

### c. Generational schemes and restart

Once the agent-solutions have been created and arranged in a square 6x6 lattice, each is characterised by the genetic information associated with the given chromosome, and the population evolves in line with the schema shown in Figure **1**. All of the agent-solutions in the lattice are selected sequentially until the round is completed. On each agent an attempt is made to apply a crossover operator, then a mutation operator, and finally the genetic learning process. At the end of each round it is verified if the value of the makespan of the best solution found in the population is an improvement on the best value reached in the previous round. If a certain number of rounds are completed (determined by the factor termed ‘not improvement bound’ (NIB)) without any improvements being produced, then a global regeneration of the population occurs.

The regeneration of the population means exploring each of the agent-solutions and if:

1. The makespan of the solution of the agent-solution is greater than the value of the "makespan_bound" then a new sequence is always generated as a substitution.

2. If the above constraint is not satisfied, then a draw is made in which the agent-solution has a 70% chance of winning. If the agent-solution wins, then a new sequence is generated to substitute the current sequence. Otherwise, no change is made to the current agent-solution.

If a new chromosome is generated, then the same algorithm that was applied to generate the initial population is used.

Finally, it is worth noting that although any of the genetic operators, including the genetic learning process, can generate a new chromosome. However, no automatic replacement of the old chromosome is made. In general, a new chromosome only substitutes the old chromosome under the following conditions:

1. A verification is made as to whether the makespan obtained with the sequence of the new solution generated with the genetic learning process of the current agent-solution differs from all its neighbours. If the value of the makespan is found to be repeated, then the found solution is discounted, and no generational change is made.

2. If the filter mentioned in the previous point has been passed, verification is made that the value of newly generated solution is less or equal to the ‘makespan_bound’. The value of the ‘makespan_bound’ is obtained by multiplying the value of the makespan by the ‘*range factor*’ (RF).

3. If the previous constraint has been satisfied, then a verification is made as to whether the value of the makespan of the new solution is repeated in the historic set of the best values obtained with the population. To achieve this, a list containing the 500 values nearest the best current makespan is used to indicate the values previously obtained (dark colour) in order to avoid repetitions.

*Figure 9. List of the best 500 values*

### d. Algorithm implementation

To implement the MAGSA algorithm it is necessary to combine two fundamental elements: a programming language and an agent platform.

The MAGSA algorithm has been implemented using the JAVA programming language for its development (specifically, the open source development environment ECLIPSE version 3.4, available at http:\www.eclipse.org\platform. The Java programming language was selected because most of the agent platforms that we have found are being developed in Java. This choice opens the possibility of future changes in the agent platform with less time investment.

The selected agent platform was JADE, version 3.5. This platform is one of the most complete in terms of functionality, and meets international standards for developing agent applications.

## 4. Experimental analysis

### a. Explained variance

In any process of configuration and analysis of an algorithm it is necessary to establish explained variance, or the endogenousity of the model. In this case, we used as a comparative measurement the percentage increase over the optimal, or the lowest known level of the average result (IPSOVEP) of a given problem or instance. This measurement can be expressed as:

$$IPSOVEP = \frac{Current\_result - The\_best\_result}{The\_best\_result} * 100$$

The variable 'Current_result' is the value of the makespan obtained with a given instance of the algorithm under evaluation. The variable 'The_best_result' represents the value of the best known makespan for this instance. Therefore, positive values for IPSOVEP imply that the algorithm has a makespan that is worse than the benchmark used, and negative values for IPSOVEP imply that the model has been improved.

### b. Data

In this work the use of a standard dataset has been seen as fundamental. Its function is to help verify the quality of the results produced by the MAGSA algorithm with a set of instances. We propose the use of a database originally published in Vallada et al. (2003), and which is an adaptation of the dataset used in Taillard (1993). This dataset, adapted for the flowshop or hybrid flowshop with sequence dependent setup times, was subsequently made available to the scientific community so that researchers could test various proposals and offer improved results.

For the experiment we used a subset of the selected databank and which has been classified into 16 experimental sets, and although the nomenclature is explained in the original reference, it is based on the combination of three characteristics: (A) four types of sequence dependent setup times were considered, corresponding to 10%, 50%, 100%, and 125% of the average process time (termed SSD10, SSD50, SSD100 and SSD125); (B) consideration of two cases in relation with the numerical distribution of the machines per stage (P13 - randomly distributed between 1 and 3 machines per stage; and P3 - a constant number of three machines per stage); and (C) combination of two load levels in the workshop (20 to 50 pieces). In relation with these characteristics, we have denominated the following: P13_SSD10_20, P13_SSD10_50, P3_SSD10_20, P3_SSD10_50, P13_SSD50_20, P13_SSD50_50, P3_SSD50_20, etc. Given that each set consists of 15 instances, we then have a total of 240 instances (or instances).

Following a subdivision made in the experimental set, the IPSOVEPT variable will take into account the average IPSOVEP for a single complete experimental set.

### c. Factors and parametric calibration

In this section we discuss experiments carried out to correctly calibrate the MAGSA algorithm using the design of experiments (DoE) approach. Two steps have been defined to explore two behaviours. In the first step, learning processes have been disabled and the algorithm in this state is termed MAGSA-1. The objective in implementing this first step is to create an environment where agents do not learn, and where agents cannot develop a differentiated behaviour. This is achieved by disabling some parameters ('bottleneck switch' (BS), 'crossover factor switch' (CFS) and 'crossover operator switch' (COS)). A full factorial experimental design has been achieved for MAGSA-1 where all possible combinations of the following factors have been tested:

- Not improvement bound (NIB): 2 levels (50 and 200).
- Mutation factor (MF): 2 levels (0.1, and 0.3).
- Range factor (RF): 2 levels (1.05, and 1.2).

All the cited factors result in a total of 23=8 different combinations. For each combination we aim to solve a full set of 240 problems with two replicas (with three running) for a total of 5760 runs.

We will now comment on the results for the SSD50_P3_50 experiment where we have 15 instances with three machines per stage, setup times that are 50% of processing times, and 50 jobs per order. In this case, all simple factors are significant, but any double interaction is relevant. To choose the best levels for the studied factors we can use value plots to graphically see which level is best for the genetic algorithm. The averages for the three factors are plotted in Figure 10. Due to minimum values being established, 50, 0.3, and 1.2 are selected for NIB, MF, and RF respectively.

*Figure 10. Values for simple factors for the SSD50_P3_50 experiment*

The remaining graphics for all other experimental sets are not shown here. After obtaining the best values for the parameters for all 16 experimental sets, the results shown in Table 1 are displayed. Only the values highlighted have been identified as statistically significant, the other values have been freely selected.

| Experiment | NIB | MF | RF | Experiment | NIB | MF | RF |
|---|---|---|---|---|---|---|---|
| SSD10_P13_20 | 50 | 0.1 | 1.2 | SSD10_P3_20 | 50 | 0.1 | 1.2 |
| SSD10_P13_50 | 50 | 0.1 | 1.2 | SSD10_P3_50 | 50 | 0.1 | 1.2 |
| SSD50_P13_20 | 200 | 0.3 | 1.05 | SSD50_P3_20 | 50 | 0.3 | 1.2 |
| SSD50_P13_50 | 200 | 0.3 | 1.05 | SSD50_P3_50 | 50 | 0.3 | 1.2 |
| SSD100_P13_20 | 50 | 0.1 | 1.2 | SSD100_P3_20 | 50 | 0.3 | 1.2 |
| SSD100_P13_50 | 50 | 0.1 | 1.2 | SSD100_P3_50 | 50 | 0.3 | 1.2 |
| SSD125_P13_20 | 200 | 0.1 | 1.2 | SSD125_P3_20 | 50 | 0.1 | 1.2 |
| SSD125_P13_50 | 200 | 0.1 | 1.2 | SSD125_P3_50 | 50 | 0.1 | 1.2 |

*Table 1. MAGSA-1 algorithm calibration for 16 experimental sets*

The resulting algorithms for all 16 experimental sets differ considerably. More precisely, when NIB is significant, some 50 iterations without improved makespans are suggested, in other words, frequent re-starts are better. However, this factor is only significant for P3 cases. The range factor (RF) is probably the most relevant factor and this fact suggests that the algorithm works better when RF is 1.2, that is, the algorithm works better when the generational schemes are easily changed (relaxing the 2nd condition due to a high value of the makespan_bound parameter). The mutation factor (MF) is not usually relevant, only 4 out of 16 times, and it must be considered for future proposals.

In the second step, the parameters associated with the learning processes are enabled and the factor minimum crossover factor (MCF) is calibrated. This algorithm has been termed MAGSA-2. In this way, a test regarding the agent contribution, represented by MAGSA-1 and MAGSA-2, is easily carried out.

In case of MAGSA-2, the established values of the MAGSA-1 factors remain the same and only the 'minimum crossover factor' (MCF) factor with two levels (0.2 and 0.4) has been analysed after enabling the 'bottleneck switch' (BS), 'crossover factor switch' (CFS), and 'crossover operator switch' (COS) parameters. We aim to solve a full set of 240 problems with two replicas (total three) and a two-level factor for a total of 1440 runs. The results of the statistical analysis show that the MCF factor is only significant for P3 cases where 0.4 is the best value.

All experiments were performed in a cluster of four PC computers with Intel Core 2 2.66 GHz processors and two GB of main memory. The resulting experiments were analysed using a multifactor analysis of variance (ANOVA) technique. With regards to the suitability of ANOVA models for the data it can be said that all three hypotheses (normality, homogeneity of variance, and independence of the residuals) were accepted in all experiments. All the experiments were carried out at a 95% confidence level.

### d. Comparative analysis

To make a comparative analysis that enables an assessment of the goodness of the proposal we have expanded the work presented in Ruiz and Maroto (2006). In this way, we compare the proposal through implementations of MAGSA-1 and-MAGSA-2 with ten other methods. The condition of termination in all cases is 5000 iterations.

The first included method is the genetic algorithm, termed GAH, which was introduced in Ruiz and Maroto (2006), for the same type of problem as MAGSA. The other methods – which have been adapted to fit the problem constraints – follow below. The simulated annealing procedure Osman and Potts (1989) which was adapted by replacing just the makespan calculation has been termed SAOPH. The initialisation of the algorithm based on tabu search Wildmer and Hertz (1989) was modified by adapting the NEH heuristic to this problem (NEHH); in the same way, the evaluation of the solution for each step of the algorithm was performed with the calculation functions of the adapted makespan. The adaptation of this algorithm has been termed SpiritH and the original NEH heuristic of Nawaz et al. (1983) was also adapted. The change was made by addressing the allocation and the makespan calculation for the problem was amplified rather then being handled in a standard flowshop. The NEH heuristic adapted for the problem has been termed NEHH. The genetic algorithm Reeves (1995) was adapted by modifying the evaluation function, as well as the initialisation (now handled by the NEHH heuristic instead of the standard NEH heuristic), and has been termed GAReevH. In a similar way, we modified the genetic algorithms of Chen et al. (1995), Murata et al. (1996) and Ponnanbalam et al. (2001) and these are referred to as GAChenH, GAMITH, and GAPACH, respectively, and for which we simply changed the individual evaluation functions. The algorithms based on ant colonies, M-MMASH and PACOH Rajendran and Ziegler (2004), were also used.

Tables Table 2 and Table 3 show the average results obtained for all the experimental sets for each of the 12 implemented algorithms. Tables Table 2 show the results for the case of the hybrid flowshop with sequence dependent setup times with one and three machines per stage (case P13), and Table 3 shows where the workshop always has three machines available per stage (case P3). In both tables, a grey background highlights the best result, and the dotted background indicates the second best result.

| P13 Case | GAH | SOAPH | SpiritH | GAReevH | NEHH | GAChenH | GAPACH | GAMITH | M-MMASH | PACOH | MAGSA-1 | MAGSA-2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SSD10_P13_20 | 0.147 | 1.093 | 2.853 | 0.697 | 2.590 | 3.030 | 8.297 | 2.710 | 2.067 | 1.577 | ***0.021*** | ***0.042*** |
| SSD10_P13_50 | ***0.220*** | 1.607 | 2.797 | 0.730 | 2.633 | 3.790 | 10.393 | 5.050 | 2.097 | 1.653 | 0.851 | ***0.443*** |
| SSD50_P13_20 | ***0.593*** | 3.690 | 6.510 | 2.527 | 6.023 | 6.717 | 17.230 | 7.003 | 5.457 | 5.110 | ***0.830*** | 0.876 |
| SSD50_P13_50 | ***0.793*** | 3.637 | 6.110 | 2.293 | 18.283 | 8.910 | 21.423 | 10.527 | 4.317 | 3.737 | ***0.920*** | 0.996 |
| SSD100_P13_20 | ***1.190*** | 6.170 | 9.957 | 4.557 | 8.893 | 11.063 | 25.743 | 9.907 | 8.403 | 8.267 | ***1.215*** | 1.275 |
| SSD100_P13_50 | 0.823 | 5.377 | 8.023 | 3.303 | 6.873 | 13.907 | 31.877 | 15.087 | 6.277 | 6.283 | ***0.726*** | ***0.728*** |
| SSD125_P13_20 | ***1.333*** | 7.427 | 11.043 | 4.853 | 9.610 | 12.117 | 28.593 | 11.753 | 9.270 | 8.473 | 1.377 | ***1.171*** |
| SSD125_P13_50 | ***0.713*** | 5.877 | 9.177 | 3.600 | 7.877 | 15.977 | 35.563 | 15.920 | 7.460 | 7.120 | 0.760 | ***0.506*** |

*Table 2. ISOVEPT for evaluated methods in case P13*

Three aspects are striking. Firstly, the relative differences between the algorithms are very similar in all the experimental sets. However, the NEHH algorithm shows a sharp drop in SDD50_P13_50, and so becomes the second worst algorithm after GAMITH. On some occasions, algorithms with very similar results change positions in the ranking, such as

GAChenH and GAMITH. The second noteworthy aspect in the experimental sets with long setup times (SSD100 and SSD125) is that the most competitive algorithms (i.e. all except GAPACH, GAChenH, and GAMITH) show better results in cases of 50 pieces than in cases of 20 pieces. The third noteworthy aspect is that the GAH, MAGSA-1, and MAGSA-2 algorithms are found to be among the three best algorithms in all cases, except for the SSD10_P13_50 experimental set, in which the GAReeVH algorithm is in third position, and MAGSA-1 is in fourth position. This domination clearly differentiates these algorithms from the other algorithms.

From the point of view of implementing the GAH, MAGSA-1, and MAGSA-2 algorithms for the calculation of production programs it can be said that none offers a clear advantage and that they form a more or less homogeneous set with respect to the quality of their production programs.

*Figure 11. IPSOVEPT values for the algorithms for the P13 case*

| P3 Case | GAH | SOAPH | SpiritH | GAReevH | NEHH | GAChenH | GAPACH | GAMITH | M-MMASH | PACOH | MAGSA-1 | MAGSA-2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SSD10_P3_20 | 2.143 | 6.810 | 10.943 | 3.197 | 7.643 | 7.023 | 17.113 | 12.747 | 7.027 | 6.963 | ***1.774*** | ***1.398*** |
| SSD10_P3_50 | 1.720 | 10.487 | 11.200 | 2.700 | 5.413 | 4.730 | 18.293 | 11.603 | 4.817 | 4.957 | ***0.128*** | ***0.170*** |
| SSD50_P3_20 | 3.103 | 9.587 | 12.353 | 5.100 | 10.367 | 8.247 | 18.513 | 10.897 | 9.807 | 9.600 | ***2.824*** | ***2.439*** |
| SSD50_P3_50 | 1.900 | 10.270 | 10.170 | 3.263 | 6.250 | 4.383 | 17.007 | 11.203 | 5.743 | 5.650 | ***1.777*** | ***1.721*** |
| SSD100_P3_20 | 3.300 | 11.883 | 12.987 | 6.200 | 12.673 | 8.703 | 20.133 | 12.073 | 12.673 | 12.647 | ***2.665*** | ***2.584*** |
| SSD100_P3_50 | 2.460 | 10.113 | 10.227 | 4.103 | 7.093 | 4.933 | 17.097 | 11.573 | 7.093 | 7.093 | ***2.226*** | ***1.977*** |
| SSD125_P3_20 | 3.670 | 12.313 | 14.110 | 7.187 | 14.263 | 8.960 | 21.030 | 13.387 | 13.747 | 13.447 | ***3.263*** | ***2.633*** |
| SSD125_P3_50 | 2.913 | 10.367 | 10.773 | 4.707 | 7.817 | 5.813 | 17.790 | 11.990 | 7.193 | 7.213 | ***2.547*** | ***2.179*** |

*Table 3. ISOVEPT for evaluated methods in case P3*

In general, there are three notable aspects. Firstly, as in the case of P13, the relative differences between the algorithms are very similar in all the experimental sets. However, the NEHH algorithm shows a sudden worsening for sets with 20 pieces, which significantly increases its IPSOVEPT value and causes it to lose positions in the algorithm rankings – although it escapes last position. On some occasions, algorithms with very similar results have changed positions in the ranking – although not to the same degree as in the P13 case. Now only SpiritH and GAMITH change the positions. The second notable aspect is that in the SSD100 and SSD125 cases, all of the algorithms have a better IPSOVEPT for 50 pieces than 20 pieces. The SSD50 case maintains the trend except for the algorithms SOAPH and GAMITH. Moreover, SSD50 also maintains this trend, except for the SOAPH, SprithH, and GAPACH algorithms. The third noteworthy aspect is that algorithms GAH, MAGSA-1, and MAGSA-2 are found among the three best algorithms, usually with a clear difference with respect to the others. The GAPACH algorithm was, in all cases, by far the worst.

In this case, the MAGSA-1 and MAGSA-2 algorithms are always the best two performers. Therefore, we can confirm that for the analysed experimental sets, it is always best to use the MAGSA-1 or MAGSA-2 algorithms to produce new predictive production programs. However, for the P3 case, the domination of MAGSA-2 over MAGSA-1 is always significant, as it is the best in seven of the eight experimental sets.

*Figure 12. Value of IPSOVEPT of the algorithms for the P3 case*

In the P3 type of problem, a multiagent system that incorporates the features from the proposal implemented with MAGSA-2 supposes an advantage that enables a better performance than the best known GAH algorithm in all cases. Although the type P13 and P3 problems are both of NP-complete complexity, it is worth noting that that when scheduling a workshop in which one or more stages are bottlenecks and there is only one machine (case P13), the task is more easily achieved than in a better balanced workshop (P3 case).

It should be emphasised that in the execution of algorithms, MAGSA-1 improved on 57 occasions the best known value for the makespan. For its part, MAGSA-2 improved the best value on 37 occasions. Of the 120 instances used in the P3-type problem, MAGSA-1 achieved the best makespan for 46 instances, and MAGSA-2 for 25 instances. The best values obtained by running MAGSA-1 and MAGSA-2, and the corresponding Cmax value associated with the instances, are shown in the Annex I.

## 5. Conclusions

This paper proposes a method for the problem of the hybrid flowshop with sequence dependent setup times. After establishing the framework of the problem, a new genetic algorithm has been designed to provide a solution to this problem that is based on software agents.

To identify the possible contribution of the software agents, the process has been separated into two stages, which although working almost simultaneously, have provided two different algorithms, MAGSA-1 and MAGSA-2. The MAGSA-1 algorithm is based on multiagent technology, but does not incorporate all the features of agents in 'society' and so does not take advantage of 'teamwork'. However, MAGSA-2 does incorporate features that enable the advantages of teamwork in a society to be exploited.

After designing and implementing the algorithms, a thorough experimental analysis was made in two phases. In the first phase, an adjustment of all the parameters was made (where necessary) in both algorithms. The same values were always and deliberately used for MAGSA-2 as MAGSA-1, except for those values that are peculiar to MAGSA-2 and have been specifically configured. In the second phase, the most competitive versions of MAGSA-1 and MAGSA-2 were compared with some of the best algorithms found in the literature for this type of problem.

In the parametric adjustment phase, despite the fact that the experimental plan was simple, the number of runs was very high. It is worth highlighting that in this stage, MAGSA-1 established a new minimum makespan value for 106 instances, and MAGSA-2 for 41 instances, from a dataset total of 240.

In the comparative analysis phase, the algorithms GAH, MAGSA-1, and MAGSA-2 were identified as the most competitive for the P13 and P3 type problems. Although in the case of P13 type problems, there was no clear predominance for any of the three algorithms in any of the analysed experimental sets. In the case of P3-type problems, the MAGSA-2 algorithm was predominant in all the experimental sets. At this stage, the MAGSA-2 algorithm, and to a lesser extent the MAGSA-1 algorithm, proved to be highly competitive. Moreover, the MAGSA-2 algorithm achieved the highest average result in 9 of the 16 experimental sets, and always for the most complex cases, while the MAGSA-1 algorithm achieved the best result on three occasions.

As a final conclusion, the results obtained for MAGSA-1 and MAGSA-2 have made an interesting contribution to predictive algorithms for production scheduling in hybrid flowshops with sequence dependent setup times.

## Acknowledgements

This work has been carried out as part of the project DPI2008-06788-C02-01 (PERMACASI) funded by the Spanish Ministry of Science and Innovation as part of the project PAID-06-10-2396 (NegoSol-MAS) funded by the Vicerrectorado de Investigación of the Universidad Politécnica de Valencia.

The translation of this paper was funded by Universidad Politécnica de Valencia, Spain.

## References

Aghezzaf, E. A., Artiba, A., & Elmaghraby, S. E. (1995a). Hybrid FlowShop: an LP based heuristic for planning level problems. ETFA Proceedings, 551-559.

Aghezzaf, E. A., Artiba, A., Moursli, O., & Tahon, C. (1995b). Hybrid Flowshops problems, a decomposition based heuristic approach. International Conference on Industrial Engineering and Prodcution Managemanent (IEPM´95). FUCAM/IFIP/INRIA Proceeding, 43-56.

Allahverdi, A.,Ng, C.T., Cheng, T.C.E., & Kovalyov, M. Y. (2008). A survey of scheduling problems with setup times or cost. European Journal of Operational Research, 187, 985–1032.

Andres, C. (2001). Programación de la Producción en Talleres de Flujo Híbrido con Tiempos de Cambio de Partida dependientes de la Secuencia. Phd dissertation, presented at Universidad Politécnica de Valencia (Spain).

Andres, C., Abad, R., Ros, L. & Vicens, E. (1998). A Genetic Algorithm for Production Scheduling in a two stage Hybrid flowshop with sequence dependent setup times. 16th European Conference on Operation Research , Belgium.

Arthanari, T. S., & Ramamurthy, K. G. (1971). An extension of two machines sequencing problem. Operations Research 8, 10–22.

Behnamian, J., Ghomi, S. M. T. F., & Zandieh, M. (2009). A multi-phase covering Pareto-optimal front method to multi-objective scheduling in a realistic hybrid flowshop using a hybrid metaheurístic. Expert Systems With Applications, 36(8), 11057-11069.

Behnamian, J., Ghomi, S. M. T. F., & Zandieh, M. (2010). Development of a hybrid metaheuristic to minimise earliness and tardiness in a hybrid flowshop with sequence-dependent setup times. International Journal of Production Research, 48(5), 1415-1438.

Brah, S. A. & Hunsucker, J. L. (1991). Branch and Bound Algorithm for the Flow-Shop with Multiple Processors. European Journal of Operational Research, 51(1), 88-99.

Bang, J. Y. & Kim, Y. D. (2011). Scheduling algorithms for a semiconductor probing facility. Computers & Operations Research, 38(3), 666-673.

Brah, S. A. (1996). A comparative analysis of due date based job sequencing rules in a flow shop with multiple processors. Production Planning & Control, 7(4), 362-373.

Chen, C. L., Vempati, V. S. & Aljaber, N. (1995). An Application of Genetic Algorithms for Flow-Shop Problems. European Journal of Operational Research, 80(2), 389-396.

Davoudpour, H. & Ashrafi, M. (2009). Solving multi-objective SDST flexible flow shop using GRASP algorithm. International Journal of Advanced Manufacturing Technology, 44(7-8), 737-747.

Gao, J., Gen, M., & Sun, L. Y. (2006). Scheduling jobs and maintenances in flexible job shop with a hybrid genetic algorithm. Journal of Intelligent Manufacturing, 17(4), 493-507.

Gupta, J.N.D. & Darrow, W.P. (1986). The 2-machine sequence dependent flowshop scheduling problem. European Journal of Operational Research, 24(3), 439-446.

Gupta, J.N.D. (1988)Two stage hybrid flow shop scheduling problem. Journal of Operational Research Society, 39, 359–364.

Gupta, J.N.D. & Stafford Jr., E. F. (2006). Flowshop Scheduling research after five decades. European Journal of Operational Research, 169, 699–711.

Gupta, J. N. D. & Tunc, E. A. (1994). Scheduling A 2-Stage Hybrid Flowshop with Separable Setup and Removal Times. European Journal of Operational Research, 77(3), 415-428.

Gupta, J. N. D. & Tunc, E. A. (1991). Schedules for A 2-Stage Hybrid Flowshop with Parallel Machines at the 2Nd Stage. International Journal of Production Research, 29(7), 1489-1502.

Haouari, M. & M'Hallah, R. (1997). Heuristic algorithms for the two-stage hybrid flowshop problema. Operations Research Letters, 21(1), 43-53.

Hejazi, S. R. & Saghafia, S. (2005). Flowshop scheduling problem with makespan criterion: a review. International Journal of Production Research, 43(14), 2895-2929.

Hoogeveen, J.A., Lenstra, J.K. & Veltman, B. (1996). Preemptive scheduling in a two-stage multiprocessor flow shop is NP-hard. European Journal of Operational Research, 89, 172–175.

Hunsucker, J. L. & Shah, J. R. (1994). Comparative Performance Analysis of Priority Rules in A Constrained Flow-Shop with Multiple Processors Environment. European Journal of Operational Research, 72(1), 102-114.

Kia, H. R., Davoudpour, H. & Zandieh, M. (2010). Scheduling a dynamic flexible flow line with sequence-dependent setup times: a simulation analysis. International Journal of Production Research, 48(14), 4019-4042.

Kis, T. & Pesch, E. (2005). A review of exact solution methods for the non-preemptive multiprocessor flowshop problem. European Journal of Operational Research, 164(3), 592-608.

Kutanoglu, E. & Wu, S. D. (1999). On combinatorial auction and Lagrangean relaxation for distributed resource scheduling. IIE Transactions, 31(9), 813-826.

Kurz, M. E. & Askin, R. G. (2004). Scheduling flexible flow lines with sequence-dependent setup times. European Journal of Operational Research, 159(1), 66-82.

Lee, C. Y. (1999). Two-machine flowshop scheduling with availability constraints. European Journal of Operational Research, 114(2), 420-429.

Lee, C. Y. (1997). Minimizing the makespan in the two-machine flowshop scheduling problem with an availability constraint. Operations Research Letters, 20(3), 129-139.

Lee, C. Y., Cheng, T. C. E., & Lin, B. M. T. (1993). Minimizing the Makespan in the 3-Machine Assembly-Type Flowshop Scheduling Problem. Management Science, 39(5), 616-625.

Lim, M.K. & Zhang, Z. (2004). An integrated agent-based approach for responsive control of manufacturing resources. The 27th Int. Conf. on Computers and Industrial Engineering, 46 (2), 221-232.

Logendran, R., deSzoeke, P. & Barnard, F. (2006). Sequence-dependent group scheduling problems in flexible flow shops. International Journal of Production Economics, 102(1), 66-86.

Murata, T., Ishibuchi, H., & Tanaka, H. (1996). Genetic algorithms for FlowShop Scheduling Problem. Computers and Industrial Engineering, 30(4), 1061-1071.

Naderi, B., Ruiz, R. & Zandieh, M. (2010). Algorithms for a realistic variant of flowshop scheduling. Computers & Operations Research, 37(2), 236-246.

Naderi, B., Zandieh, M., Balagh, A. K. G. & Roshanaei, V. (2009). An improved simulated annealing for hybrid flowshops with sequence-dependent setup and transportation times to minimize total completion time and total tardiness. Expert Systems With Applications, 36(6), 9625-9633.

Nawaz, M., Enscore, E. E. & Ham, I. (1983). A Heuristic Algorithm for the M-Machine, N-Job Flowshop Sequencing Problem. Omega-International Journal of Management Science, 11(1), 91-95.

Ng, C. T., Cheng, T. C. E., & Yuan, J. J. (2006). A note on the complexity of the problem of two-agent scheduling on a single machine. Journal of Combinatorial Optimization, 12(4), 386-393.

Nowicki, E. & Smutnicki, C. (1998). The flow shop with parallel machines: A tabu search approach. European Journal of Operational Research, 106(2-3), 226-253.

Osman, I. H. & Potts, C. N. (1989). Simulated Annealing for Permutation Flowshop Scheduling. Omega-International Journal of Management Science, 17(6), 551-557.

Ponnanbalam, S. G., Aravindan, P. & Chandrasekaran, S. (2001). Constructive and Improvement Flow Shop Scheduling Heuristics: An extensive Evaluation. Production Planning and Control, 12(4), 335-344.

Portmann, M. C., Vignier, A., Dardilhac, D., & Dezalay, D. (1998). Branch and bound crossed with GA to solve hybrid flowshops. European Journal of Operational Research, 107(2), 389-400.

Quadt, D. & Kuhn, H. (2007). A taxonomy of flexible flow line scheduling procedures. European Journal of Operational Research, 178(3), 686-698.

Rajendran, C. & Chaudhuri, D. (1992). Scheduling in Normal-Job, Meta-Stage Flowshop with Parallel Processors to Minimize Makespan. International Journal of Production Economics, 27(2), 137-143.

Rajendran, C. & Ziegler, H. (2004). Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. European Journal of Operational Research 155(2), 426–438.

Reeves, C. R. (1995). A Genetic Algorithm for Flowshop Sequencing. Computers and Operations Research, 22(1), 5-13.

Riane, F., Artiba, A., & Elmaghraby, S. E. (2002). Sequencing a hybrid two-stage flowshop with dedicated machines. European Journal of Operational Research, 109(2), 321-329.

Riane, F., Artiba, A., & Elmaghraby, S. E. (1998). A hybrid three-stage flowshop problem: efficient heurístics to minimize makespan. International Journal of Production Research, 40(17), 4353-4380.

Ribas, I, Leisten, R. & Framiñan, J.M. (2010). Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective). Computers and Operations research, 37 (8), 1439-1454.

Ruiz, R., Serifoglu, F. S., & Urlings, T. (2008). Modeling realistic hybrid flexible flowshop scheduling problems. Computers & Operations Research, 35(4), 1151-1175.

Ruiz, R. & Stutzle, T. (2008). An Iterated Greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. European Journal of Operational Research, 187(3), 1143-1159.

Ruiz, R. & Maroto, C. (2006). A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. European Journal of Operational Research, 169, 781–800.

Sanjay, K.S. & Young, J.S. (2008). Bidding-based multi-agent system for integrated process planning and scheduling: a data-mining and hybrid tabu-SA algorithm-oriented approach. International Journal of Advanced Manufacturing Technology, 38(1), 163–175.

Serifoglu, F. S. & Ulusoy, G. (2004). Multiprocessor task scheduling in multistage hybrid flow-shops: a genetic algorithm approach. Journal of the Operational Research Society, 55(5), 504-512.

Shen, V. Y. & Chen, Y. E. (1972). A scheduling strategy for the flowshop problem in a system with two classes of processors. Conference on Information and Systems Science. Proceedings, 645-649.

Shen, W., Wang, L. & Hao, Q. (2006). Agent-based distributed manufacturing process planning and scheduling: a state-of-art survey. IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and review, 36 (4).

Sriskandarajah, C. and Sethi, S. P., 1989, “Scheduling Algorithms for Flexible Flowshops - Worst and Average Case Performance”, European Journal of Operational Research, 43(2), pp. 143-160.

Taillard, E. (1993). Benchmarks for Basic Scheduling Problems. European Journal of Operational Research, 64(2), 278-285

Toptal, A. & Sanbucuoglu, I. (2010). Distributed scheduling: a review of concepts and applications. International Journal of Production Research, 48(18), 5235-5265.

Vallada, E, Ruiz, R., & Maroto, C. (2003). Synthetic and Real Benchmarks for Complex Flow-shops Problems. Technical Report, Grupo de Investigación Operativa (GIO), Universitat Politécnica de València.

Vignier, A., Billaut, J. C. & Proust, C. (1999). Hybrid flowshop scheduling problems: State of the art. Rairo-Recherche Operationnelle-Operations Research, 33(2), 117-183

Vignier A., Billaut, J. C., & Proust, C. (1996). Solving k stage hybrid flowshop scheduling problems. Multiconference of Computational Engieneering in Systems Applications (CESA´96), 250-258. Lille (France).

Vignier A., Billaut, J. C. & Proust, C. (1995). Les Problemes d´ordennacement de type flow shop hybride. Etat de l´art. Journées d´Etude: affectacion et ordennancement. CNRS / GdR Automatique, GT3, 7-47, Tours.

Widmer, M. & Hertz, A. (1989). A New Heuristic Method for the Flow-Shop Sequencing Problem. European Journal of Operational Research, 41(2), 186-193.

Yang, W.H. and Liao, C.J. (1999). Survey of Scheduling research involving setup times. International Journal of Systems Science, 30(2), 143-155.

Yaurima, V., Burtseva, L., & Tchernykh, A. (2009). Hybrid flowshop with unrelated machines, sequence-dependent setup time, availability constraints and limited buffers. Computers & Industrial Engineering, 56(4), 1452-1463.

Yoshida, T. & Hotomi, K. (1979). Optimal 2-stage production scheduling with setup times separated. AIIE transactions, 11 (3), 261 -263.

Zandieh, M., Ghomi, S. M. T. F., & Husseini, S. M. M. (2006). An immune algorithm approach to hybrid flow shops scheduling with sequence-dependent setup times. Applied Mathematics and Computation, 180(1), 111-127.

Zhong, W., Liu, J., Xue, M. & Jiao, L. (2004). A Multiagent Genetic Algorithm for Global Numerical Optimization. IEEE Tranc. On Systems, Man And Cybernetics-Part B: Cybernetics, 34 (2), 1128-1141.

## ANNEX I

In the following tables are shown the best know values of Cmax corresponding to the instances used in the experimental phase. When the proposed algorithms overcome the previous known Cmax value both previous, in first position, and new Cmax value are provided. In other case, only the previous known value is included. Cells with gray background have a value of Cmax that has been overcome by using the algorithm SMAGA-2.

| MAGSA-1 | P13 | | | | P3 | | | |
|---|---|---|---|---|---|---|---|---|
| **instances** | **SSD10_P13_20** | **SSD50_P13_20** | **SSD100_P13_20** | **SSD125_P13_20** | **SSD10_P3_20** | **SSD50_P3_20** | **SSD100_P3_20** | **SSD125_P3_20** |
| **ta002** | 1029 | 1197 | 1410 | 1497 | 351 | 459 / 458 | 564 | 609 / 599 |
| **ta004** | 1178 | 1247 | 1340 | 1396 / 1389 | 352 | 461 / 455 | 571 | 610 |
| **ta006** | 1314 | 1392 | 1557 | 1635 | 348 | 440 | 538 / 535 | 588 |
| **ta008** | 1074 / 1068 | 1146 | 1232 / 1230 | 1294 / 1281 | 332 | 429 | 531 | 586 |
| **ta010** | 1038 / 1034 | 1177 | 1335 / 1332 | 1409 | 322 | 412 | 509 / 502 | 555 |
| **ta012** | 1439 | 1567 | 1818 | 1983 | 541 / 539 | 652 / 647 | 798 / 794 | 858 |
| **ta014** | 1326 | 1508 | 1704 | 1790 | 461 | 580 | 717 | 782 / 755 |
| **ta016** | 1347 / 1346 | 1515 | 1804 | 1945 | 509 | 637 | 769 | 827 |
| **ta018** | 1426 / 1417 | 1654 | 1967 | 2128 | 513 | 621 / 617 | 760 | 831 |
| **ta020** | 1316 / 1315 | 1477 | 1677 | 1795 | 533 | 646 | 794 | 861 / 856 |
| **ta022** | 1897 | 2199 | 2571 | 2750 | 786 / 784 | 928 | 1113 / 1108 | 1206 / 1202 |
| **ta024** | 1890 / 1876 | 2097 | 2471 | 2652 | 782 | 926 | 1112 | 1209 |
| **ta026** | 1849 / 1848 | 2115 | 2486 / 2474 | 2668 | 859 | 999 | 1184 / 1175 | 1264 / 1257 |
| **ta028** | 1886 / 1876 | 2138 | 2490 | 2699 | 869 | 1020 | 1204 | 1293 |
| **ta030** | 1724 / 1712 | 1992 / 1989 | 2353 | 2521 | 878 | 1012 | 1191 | 1269 |

*Table 4. The new best know values of Cmax for instances with 20-jobs (P13 and P3 cases) obtained after running MAGSA-1*

| MAGSA-1 | P13 | | | | P3 | | | |
|---|---|---|---|---|---|---|---|---|
| **instances** | **SSD10_P13_50** | **SSD50_P13_50** | **SSD100_P13_50** | **SSD125_P13_50** | **SSD10_P3_50** | **SSD50_P3_50** | **SSD100_P3_50** | **SSD125_P3_50** |
| **ta032** | 2593 | 2887 | 3274 | 3447 | 678 | 955 | 1256 | 1362 |
| **ta034** | 2726 / 2721 | 3049 / 3040 | 3420 / 3406 | 3693 / 3673 | 680 | 977 / 973 | 1275 / 1239 | 1368 |
| **ta036** | 2772 | 3316 / 3299 | 3925 / 3874 | 4312 / 4277 | 700 / 695 | 951 | 1248 / 1226 | 1372 |
| **ta038** | 2692 | 2977 | 3410 / 3406 | 3647 / 3611 | 656 / 645 | 932 / 922 | 1248 / 1233 | 1387 / 1384 |
| **ta040** | 2710 / 2700 | 3026 | 3480 / 3476 | 3722 / 3693 | 684 | 965 / 957 | 1248 | 1370 / 1362 |
| **ta042** | 3002 | 3570 / 3528 | 4334 / 4280 | 4612 / 4593 | 867 | 1215 / 1209 | 1555 | 1732 |
| **ta044** | 2992 / 2971 | 3636 / 3628 | 4400 / 4334 | 4784 / 4697 | 837 | 1174 / 1171 | 1570 | 1732 |
| **ta046** | 2924 | 3469 / 3465 | 4224 / 4141 | 4489 / 4473 | 940 | 1288 | 1658 | 1801 |
| **ta048** | 3002 | 3535 / 3526 | 4248 / 4213 | 4627 / 4598 | 947 | 1284 / 1276 | 1614 | 1778 |
| **ta050** | 2998 / 2994 | 3474 | 4062 / 4057 | 4354 / 4314 | 926 | 1261 / 1254 | 1612 | 1760 |
| **ta052** | 3419 / 3376 | 4030 | 4910 | 5379 / 5337 | 2168 / 1314 | 1658 | 2106 | 2331 / 2324 |
| **ta054** | 3201 | 3895 / 3889 | 4726 / 4718 | 5248 / 5231 | 1324 | 1718 | 2145 / 2135 | 2352 |
| **ta056** | 3383 | 4092 | 5122 / 5103 | 5534 / 5486 | 1325 / 1320 | 1713 | 2172 / 2157 | 2373 |
| **ta058** | 3421 | 4101 | 5118 / 5092 | 5512 | 1284 | 1678 / 1670 | 2100 / 2085 | 2307 |
| **ta060** | 3444 | 4164 | 5049 / 5046 | 5594 / 5516 | 1316 | 1704 | 2159 / 2157 | 2370 |

*Table 5. The new best know values of Cmax for instances with 50-jobs (P13 and P3 cases) obtained after running MAGSA-1*

| MAGSA-2 | P13 | | | | P3 | | | |
|---|---|---|---|---|---|---|---|---|
| **instances** | **SSD10_P13_20** | **SSD50_P13_20** | **SSD100_P13_20** | **SSD125_P13_20** | **SSD10_P3_20** | **SSD50_P3_20** | **SSD100_P3_20** | **SSD125_P3_20** |
| **ta002** | 1029 / 1028 | 1197 | 1410 | 1497 | 351 | 459 / 455 | 564 / 556 | 609 |
| **ta004** | 1178 | 1247 / 1245 | 1340 / 1338 | 1396 / 1386 | 352 | 461 | 571 / 565 | 610 |
| **ta006** | 1314 | 1392 | 1557 | 1635 | 348 | 440 | 538 / 537 | 588 / 569 |
| **ta008** | 1074 / 1067 | 1146 | 1232 | 1294 / 1281 | 332 | 429 | 531 | 586 |
| **ta010** | 1038 / 1034 | 1177 / 1175 | 1335 / 1332 | 1409 | 322 / 319 | 412 | 509 / 503 | 555 / 549 |
| **ta012** | 1439 | 1567 | 1818 | 1983 | 541 | 652 / 649 | 798 | 858 |
| **ta014** | 1326 / 1325 | 1508 | 1704 | 1790 | 461 | 580 | 717 | 782 |
| **ta016** | 1347 / 1344 | 1515 | 1804 | 1945 | 509 | 637 / 636 | 769 | 827 |
| **ta018** | 1426 / 1417 | 1654 | 1967 | 2128 | 513 / 512 | 621 / 617 | 760 / 757 | 831 |
| **ta020** | 1316 | 1477 | 1677 | 1795 | 533 | 646 | 794 | 861 / 849 |
| **ta022** | 1897 / 1889 | 2199 / 2193 | 2571 | 2750 | 786 / 784 | 928 | 1113 / 1110 | 1206 |
| **ta024** | 1890 / 1877 | 2097 / 2088 | 2471 | 2652 | 782 | 926 | 1112 | 1209 / 1201 |
| **ta026** | 1849 / 1845 | 2115 | 2486 / 2481 | 2668 / 2665 | 859 | 999 | 1184 / 1168 | 1264 |
| **ta028** | 1886 / 1865 | 2138 / 2137 | 2490 | 2699 | 869 | 1020 | 1204 / 1202 | 1293 / 1289 |
| **ta030** | 1724 / 1707 | 1992 / 1977 | 2353 | 2521 | 878 | 1012 | 1191 / 1185 | 1269 / 1262 |

*Table 6. The new best know values of Cmax for instances with 20-jobs (P13 and P3 cases) obtained after running MAGSA-2*

| MAGSA-2 | P13 | | | | P3 | | | |
|---|---|---|---|---|---|---|---|---|
| **instances** | **SSD10_P13_50** | **SSD50_P13_50** | **SSD100_P13_50** | **SSD125_P13_50** | **SSD10_P3_50** | **SSD50_P3_50** | **SSD100_P3_50** | **SSD125_P3_50** |
| **ta032** | 2593 | 2887 / 2859 | 3274 / 3247 | 3447 / 3445 | 678 / 674 | 955 / 953 | 1256 / 1226 | 1362 / 1339 |
| **ta034** | 2726 / 2718 | 3049 / 3020 | 3420 | 3693 / 3608 | 680 | 977 / 970 | 1275 / 1240 | 1368 / 1363 |
| **ta036** | 2772 | 3316 / 3298 | 3925 / 3916 | 4312 / 4261 | 700 / 693 | 951 / 939 | 1248 / 1232 | 1372 / 1330 |
| **ta038** | 2692 | 2977 / 2969 | 3410 / 3394 | 3647 / 3629 | 656 / 648 | 932 / 926 | 1248 / 1218 | 1387 / 1355 |
| **ta040** | 2710 / 2703 | 3026 | 3480 / 3418 | 3722 / 3657 | 684 | 965 / 952 | 1248 | 1370 |
| **ta042** | 3002 / 3001 | 3570 / 3527 | 4334 / 4218 | 4612 / 4474 | 867 | 1215 / 1205 | 1555 | 1732 / 1708 |
| **ta044** | 2992 / 2962 | 3636 / 3595 | 4400 / 4352 | 4784 / 4737 | 837 | 1174 | 1570 / 1544 | 1732 / 1722 |
| **ta046** | 2924 / 2911 | 3469 / 3463 | 4224 / 4130 | 4489 / 4446 | 940 | 1288 / 1280 | 1658 / 1644 | 1801 |
| **ta048** | 3002 | 3535 / 3525 | 4248 / 4126 | 4627 / 4508 | 947 / 943 | 1284 / 1280 | 1614 | 1778 |
| **ta050** | 2998 / 2997 | 3474 / 3440 | 4062 / 3985 | 4354 / 4291 | 926 | 1261 / 1259 | 1612 / 1592 | 1760 |
| **ta052** | 3419 / 3371 | 4030 | 4910 / 4894 | 5379 / 5329 | 2168 / 1313 | 1658 / 1657 | 2106 / 2105 | 2331 / 2298 |
| **ta054** | 3201 | 3895 / 3890 | 4726 | 5248 / 5228 | 1324 / 1321 | 1718 / 1713 | 2145 | 2352 |
| **ta056** | 3383 / 3378 | 4092 | 5122 / 5046 | 5534 / 5483 | 1325 | 1713 | 2172 / 2154 | 2373 / 2362 |
| **ta058** | 3421 | 4101 / 4094 | 5118 / 5051 | 5512 | 1284 / 1278 | 1678 / 1669 | 2100 | 2307 |
| **ta060** | 3444 | 4164 | 5049 / 5041 | 5594 / 5474 | 1316 | 1704 | 2159 | 2370 |

*Table 7. The new best know values of Cmax for instances with 50-jobs (P13 and P3 cases) obtained after running MAGSA-2*

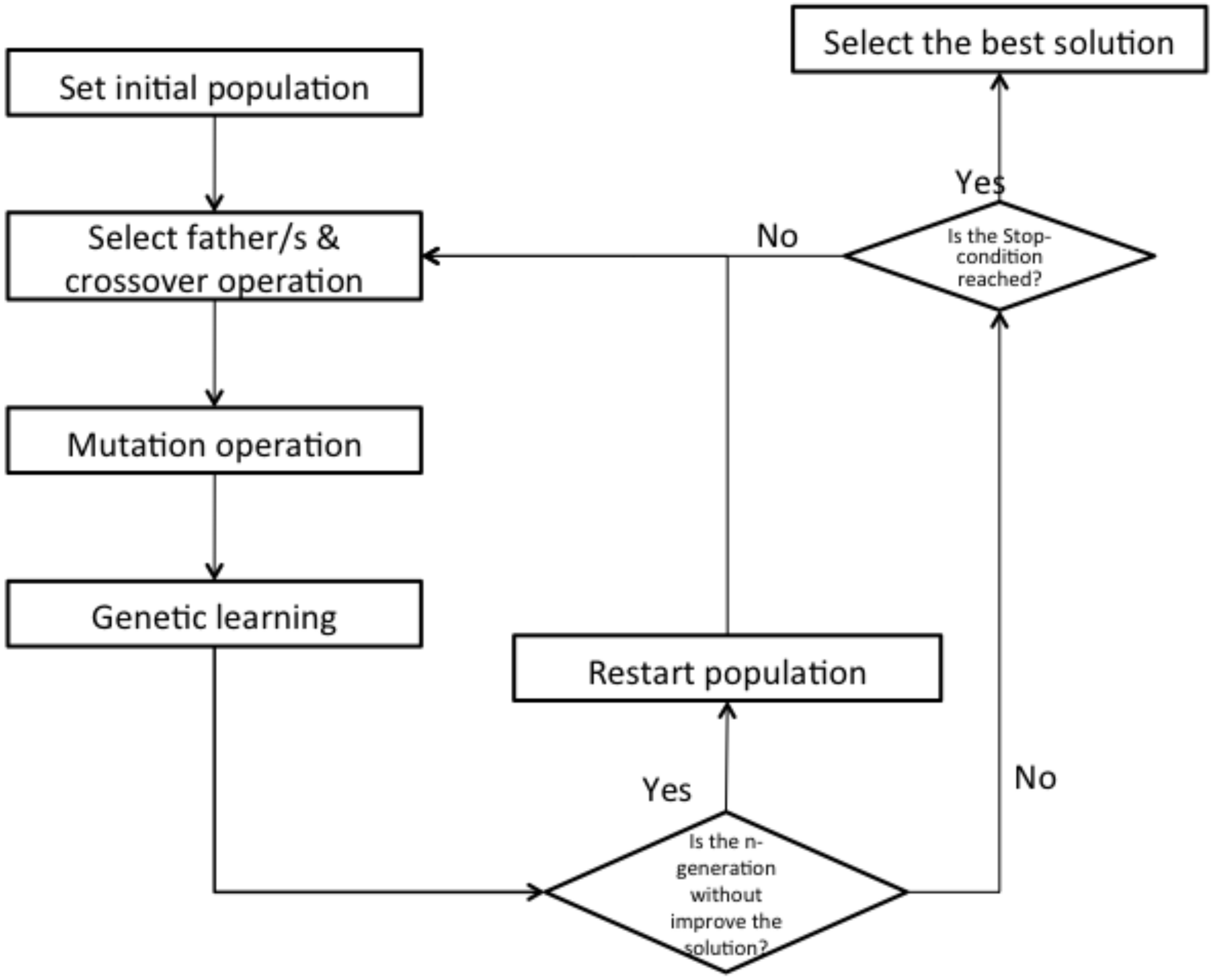

Set initial population
Select father/s & crossover operation
Mutation operation
Genetic learning
Select the best solution
Yes
No
Is the Stop-condition reached?
Restart population
Yes
No
Is the n-generation without improve the solution?

Figure 1. Functional structure of MAGSA

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 14 | 3 | 9 | 7 | 8 | 19 | 11 | 6 | 13 | 16 | 15 | 10 | 5 | 18 | 17 | 2 | 1 | 4 | 0 | 12 |

**Figure 2. Ordinal representation of a chromosome**

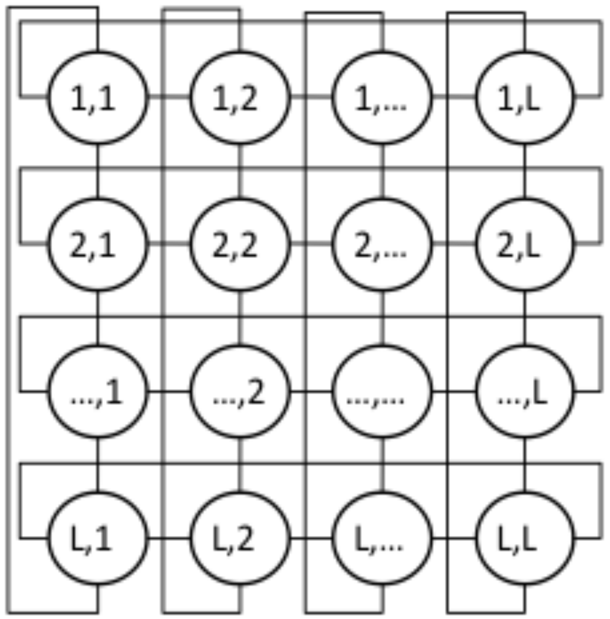


Figure 3. Model of the agent lattice

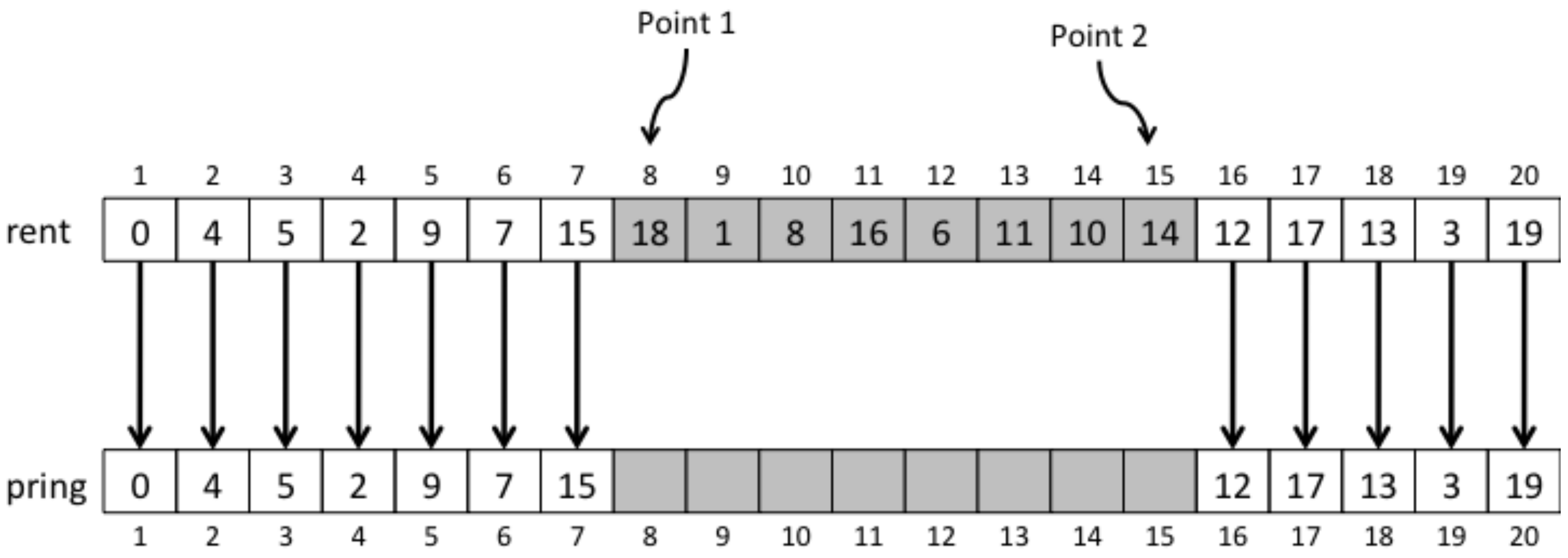


Figure 4. NCO crossover operator - step one

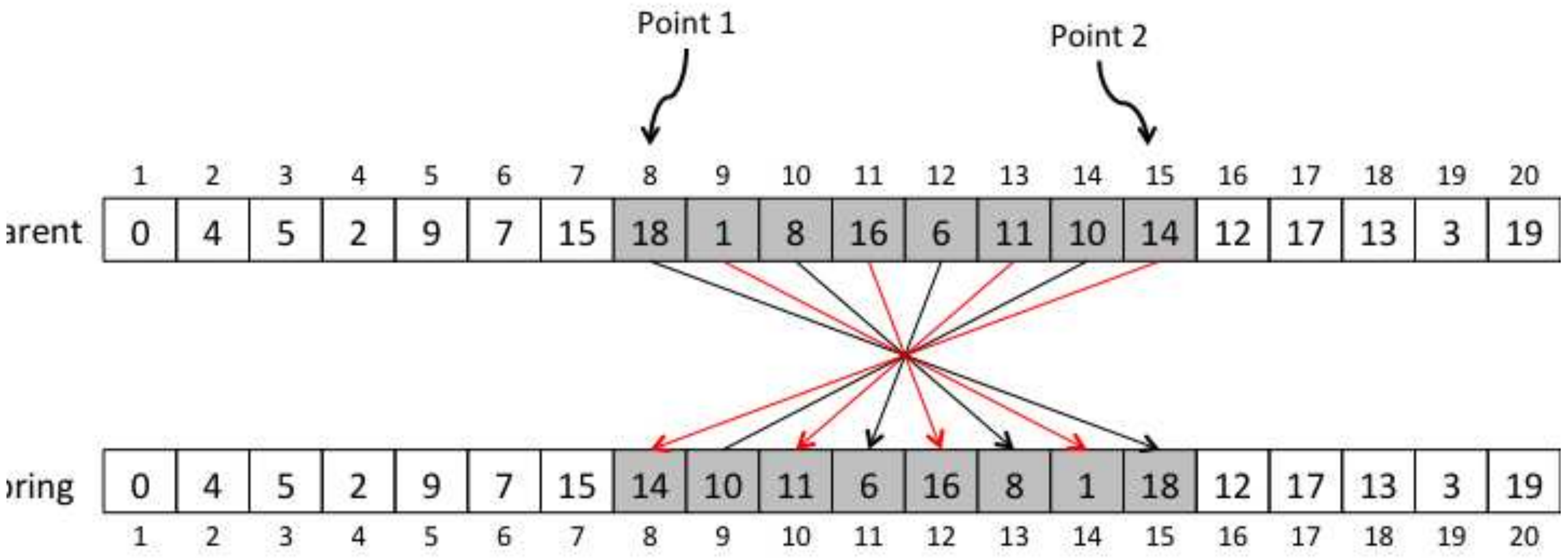


*Figure 5. NCO crossover operator - step three*

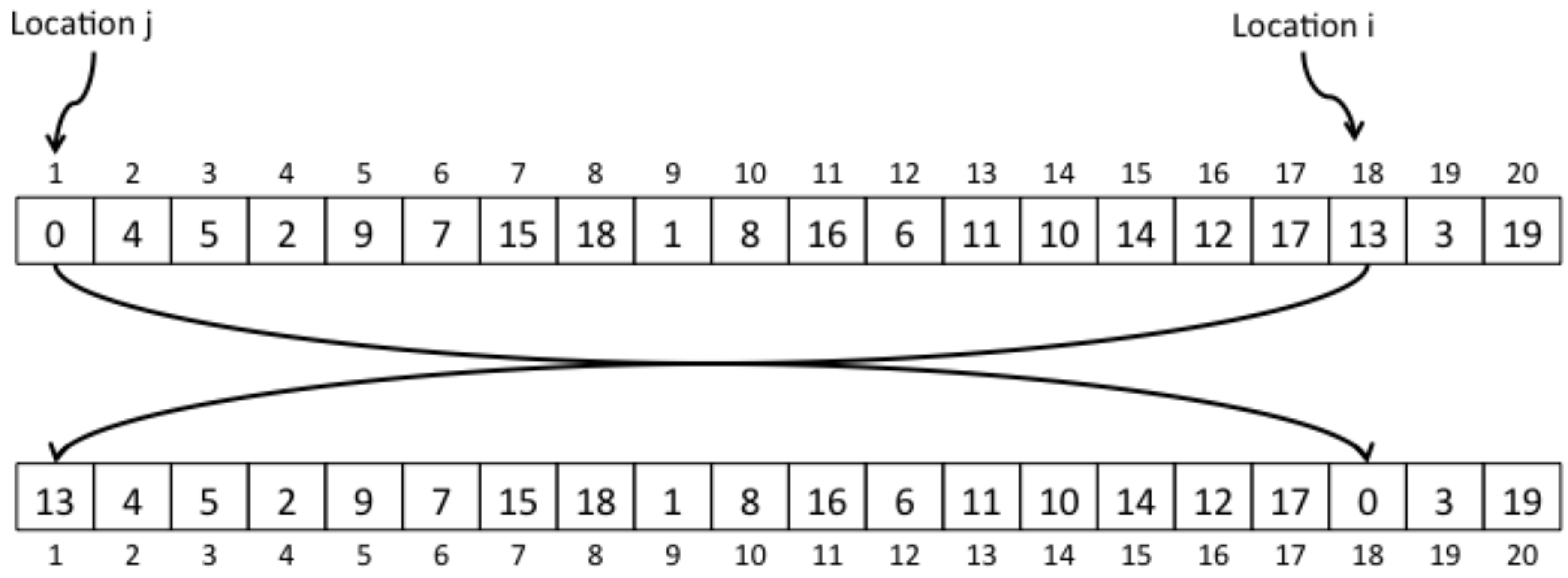


*Figure 6. Mutation operator*

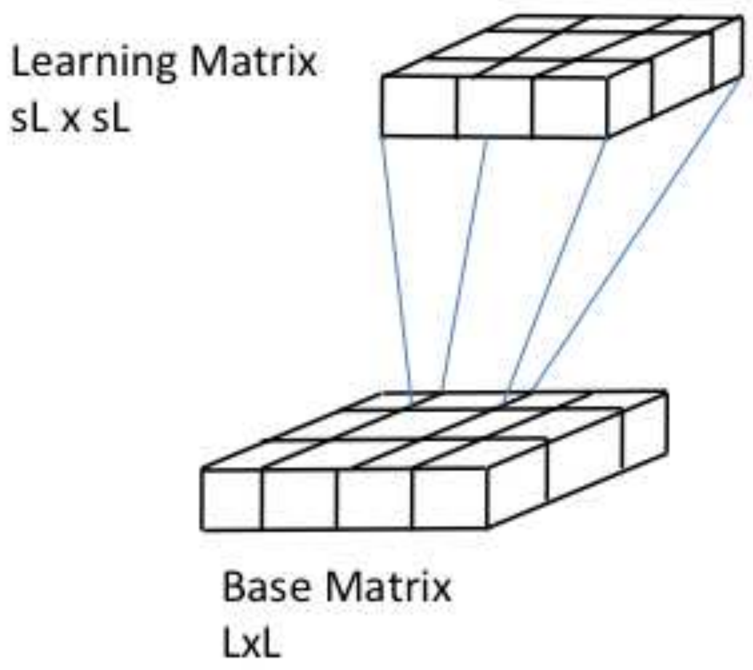


*Figure 7. Base matrix with learning matrix*

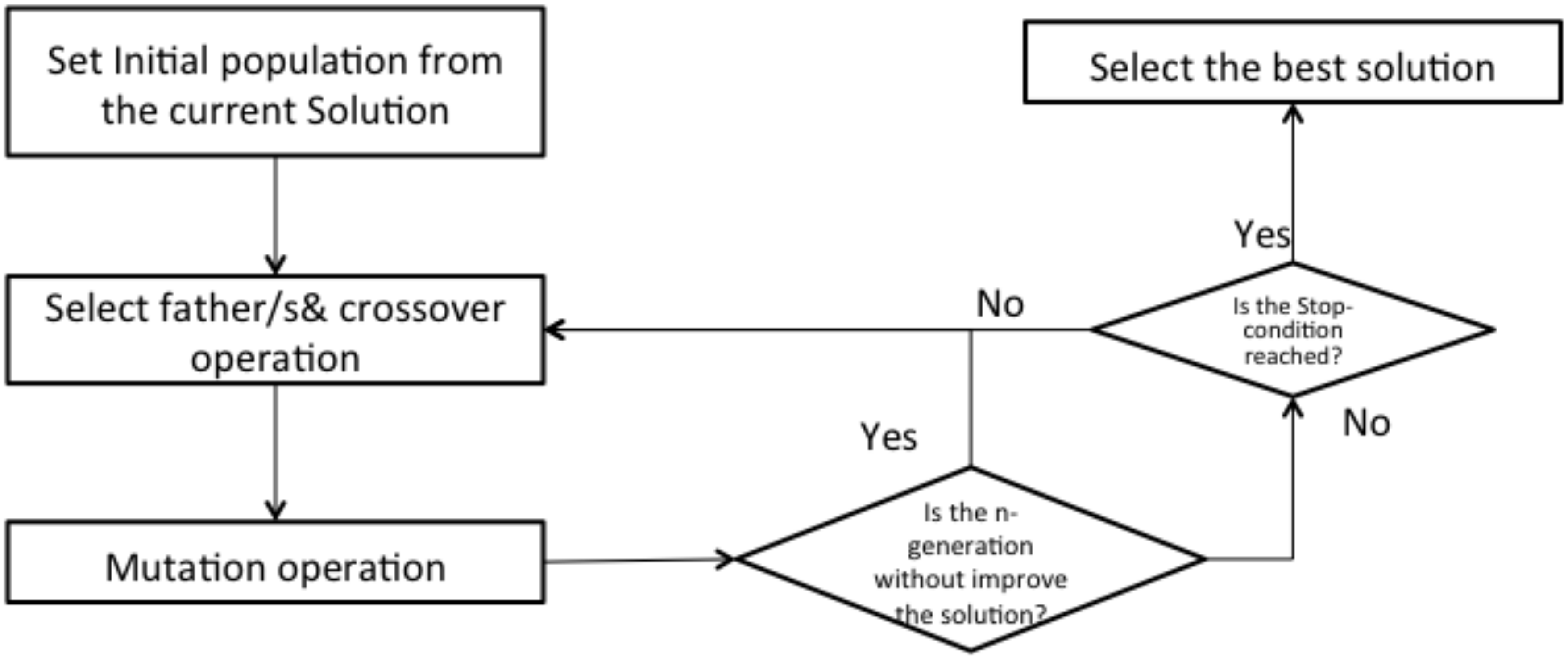

Set Initial population from the current Solution
Select father/s& crossover operation
Mutation operation
Is the n-generation without improve the solution?
Yes
No
Is the Stop-condition reached?
No
Yes
Select the best solution

*Figure 8. Genetic learning schema*

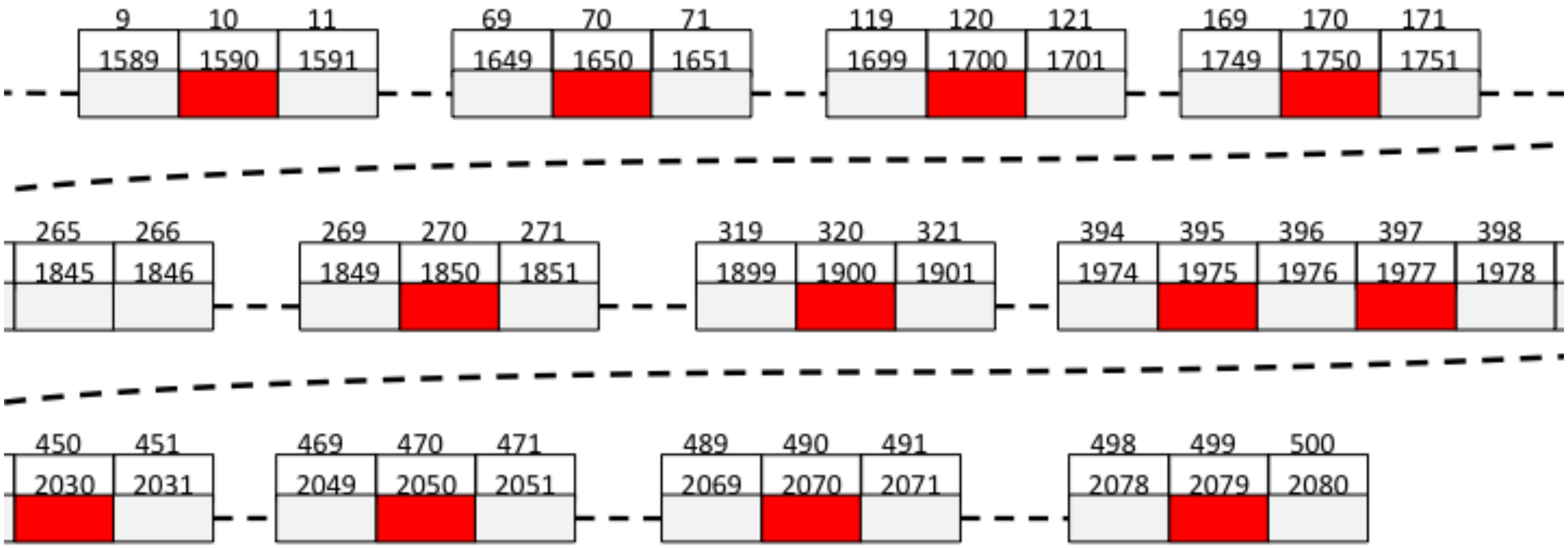


*Figure 9. List of the best 500 values*

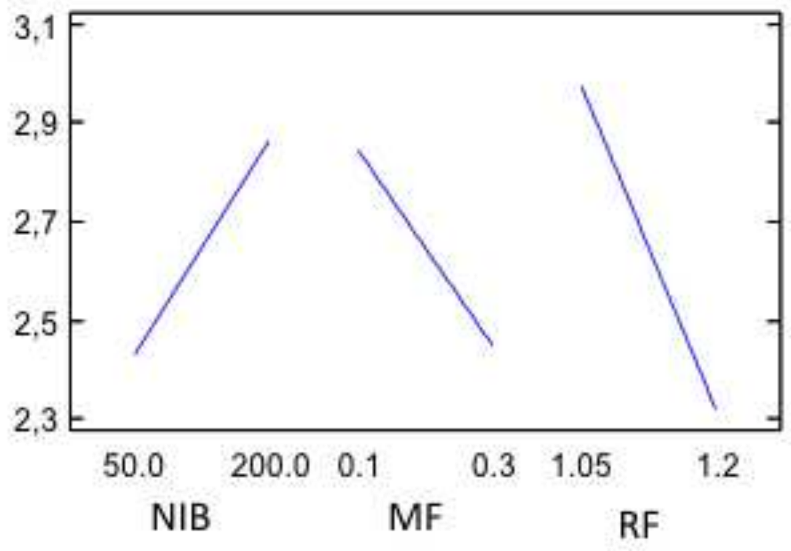


*Figure 10. Values for simple factors for the SSD50_P3_50 experiment*

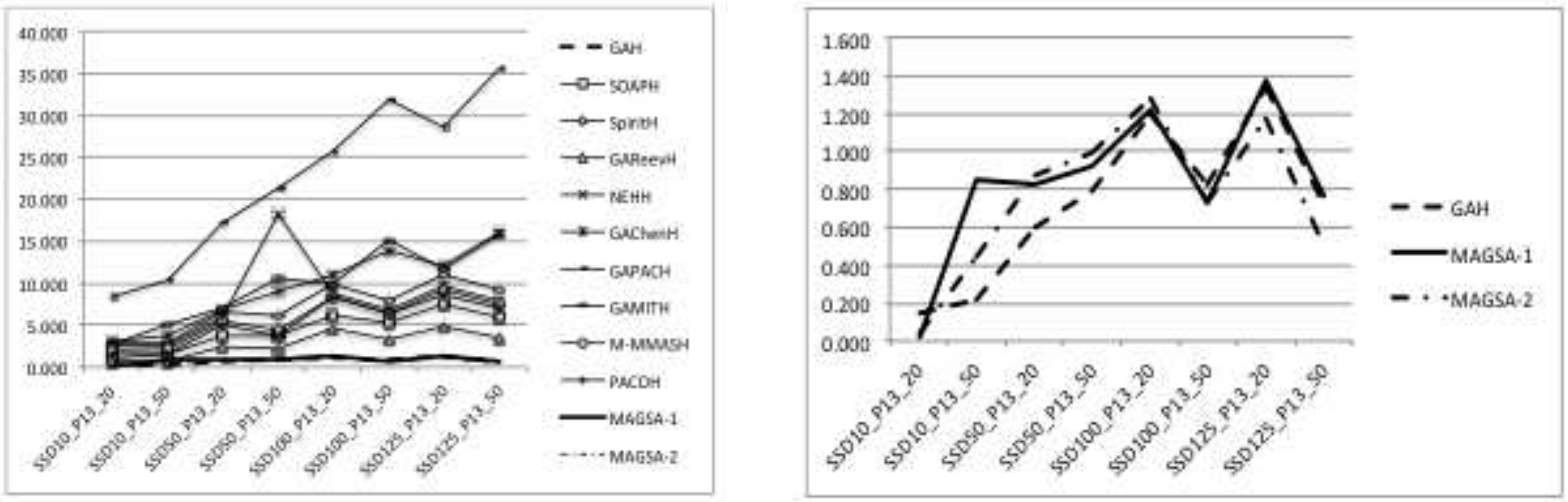
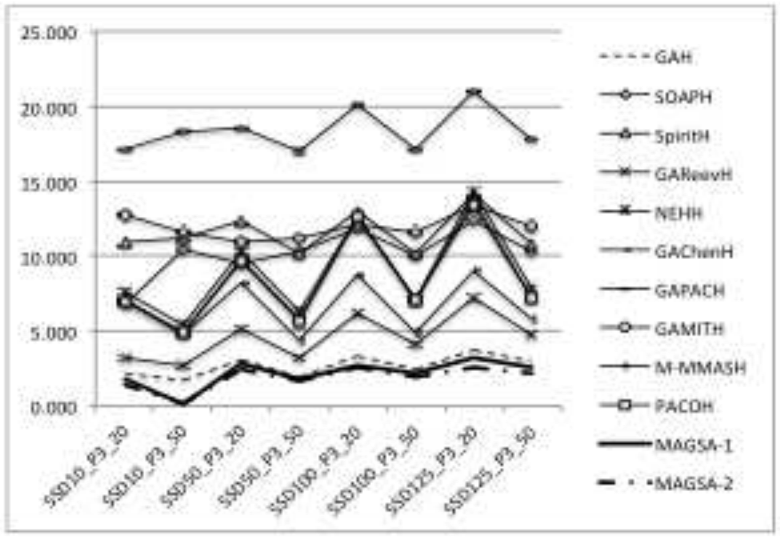


*Figure 11. IPSOVEPT values for the algorithms for the P13 case*



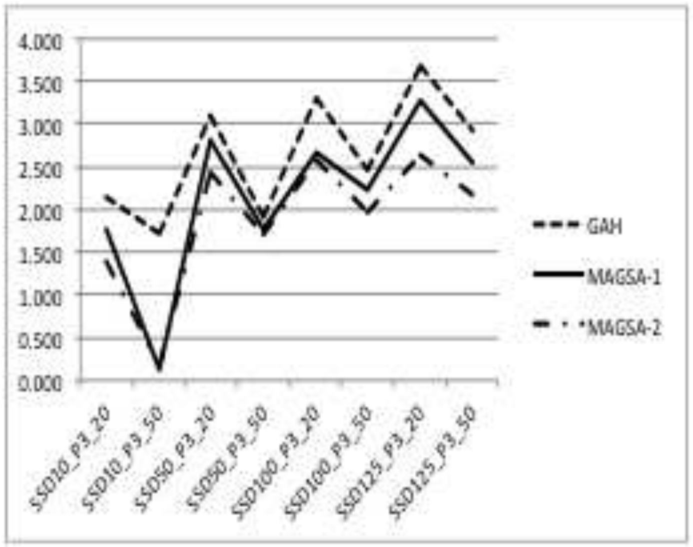


*Figure 12. Value of IPSOVEPT of the algorithms for the P3 case*