Facultad de Ingeniería

INGENIERÍA INDUSTRIAL

Trabajo de Grado – Primer Semestre de 2017

Capstone Final Project

A Decentralized Metaheuristic Approach Applied to the FMS Scheduling Problem

Juan Mauricio Moreno[ac], Gabriel Zambrano Rey[bc]

[a]*Undergraduate students, Industrial Engineering,*
[b]*Associate Professor, Project Director*
[c]*Pontificia Universidad Javeriana, Bogotá, Colombia*

Abstract

FMS scheduling has been one of the most popular topics for researchers. A number of approaches have been delivered to schedule FMSs including simulation techniques and analytical methods. Decentralized metaheuristics can be seen as a way where the population is divided into several subpopulations, aiming to reduce the run time and the numbers of evaluation, due to the separation of the search space. Decentralization is a prominent research path in scheduling so the computing cost can be reduced and solutions can be found faster, without penalizing the objective function. In this project, a decentralized metaheuristic is proposed in the context of a flexible manufacturing system scheduling problem. The main contribution of this project is to analyze other types of search space division, particularly those associated with the physical layout of the FMS. The performance of the decentralized approach will be validated with FMS scheduling benchmarks.

Keywords: FMS, scheduling, decentralization, artificial intelligence, metaheurístics, multithreading.

1. Justification and problem statement

Highly specialized transfer lines dedicated to mass production have evolved towards flexible manufacturing systems (FMS), with heterogeneous components and different layouts. Flexible manufacturing systems were introduced in the 1970s aiming for greater responsiveness to market changes, rapid turnaround, high quality, low inventory costs, and low labor costs (Basnet and Mize, 1994; ElMaraghy, 2006). Browne et al., (1984) defined a FMS as "*an integrated, computer controlled complex of automated material handling devices and computing numerically controlled (CNC) machine tools that can simultaneously process medium-sized volumes of a variety of product types*". Generally speaking, a FMS is expected to combine the productivity efficiency of transfer lines and the flexibility of job shops to attain mid-volume, mid-variety needs (Chan and Chan, 2004).

In the most general sense, FMS control is mainly composed of three activities: scheduling, product dispatching and on-line process monitoring, as seen in Figure 1 (Caramia and Dell'Olmo, 2006; Leitão, 2004). Scheduling, also termed as detailed or short-term scheduling due to the short decision horizon, defines the product flow though FMS resources, deciding "what" must be manufactured, at what time and during which time periods (i.e., "when"); and by using which FMS resources (i.e., "where") (Van Dyke Parunak, 1991). A usual industrial practice is to generate short-terms schedules and then proceed to product dispatching to execute those schedules taking into account current status of FMS resources (Caramia and Dell'Olmo, 2006). Last, on-line monitoring revises possible deviations caused by unexpected events and machine perturbations. Then, those perturbations are reported to the scheduling module so this latter can react and adapt properly, ensuring the continuing operation of the FMS.

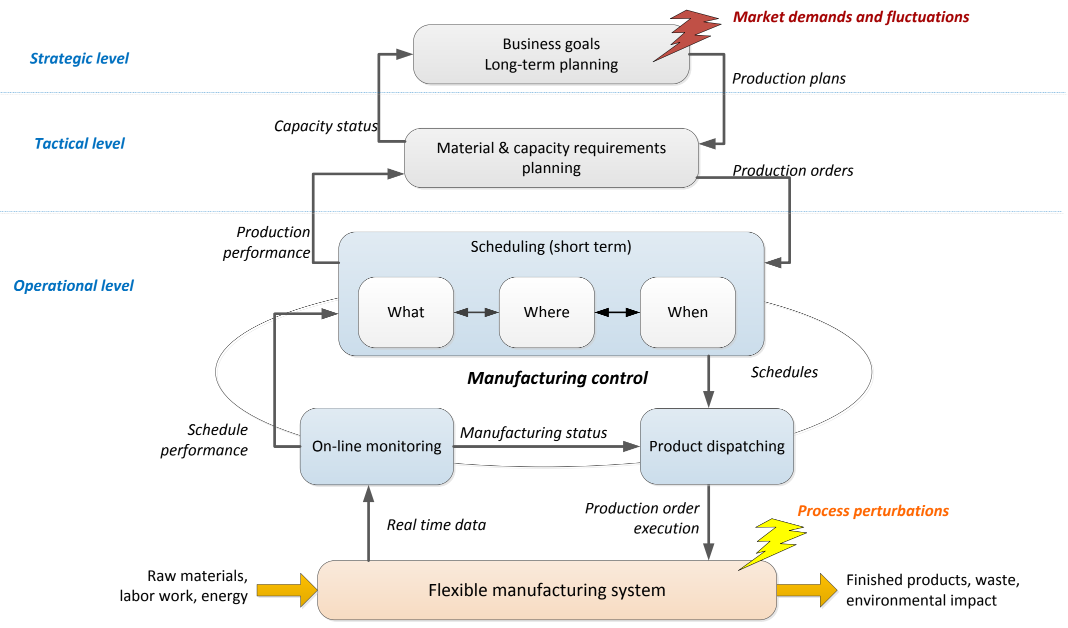


Figure 1 FMS control loop, taken from Zambrano Rey, (2014)

Industries using flexible manufacturing systems are facing several challenges issued from rapid technological innovations, globalized and customer-driven (i.e., mass customization) markets. Mass customization results in rapidly changing customer requirements, accelerated innovation and shorter product lifecycles (Christo and Cardeira, 2007). As a consequence, FMS scheduling has to deal with lower sized batches, even one-of-a-kind products, and smaller delivery times. Therefore, to accommodate such variations in product quantity, quality and specification types, FMS scheduling requires to be flexible, scalable, and easily reconfigurable (McFarlane and Bussmann, 2003). In addition, FMS scheduling needs to be reactive and adapt rapidly to external changes, fault tolerant to detect and gracefully recover from system failures and minimize their consequences; as well as modular and easy to interoperate to manage efficiently recent manufacturing technologies and legacy systems (Chituc and Restivo, 2009; Colombo and Karnouskos, 2009).

In addition, new managerial philosophies such as Just-in-time production (JIT) impose additional requirements to eliminate or reduce several sources of waste.

From an Operations Research perspective, scheduling in FMS is a more complex version of the classical flexible job-shop scheduling problem, which is known to be NP-hard (Conway et al., 2012). Therefore, it is frequently observed in literature that the FMS scheduling problem is addressed with hard assumptions (e.g., neglecting transport times, unlimited buffer capacity), constraint relaxations, and regular and single performance criteria (Shnits et al., 2004; Shirazi et al., 2011). Since there are different types of algorithms, scheduling algorithms can be classified in four basic approaches depending on the quality of solutions and algorithm complexity: optimal and near-optimal, artificial intelligence, heuristics and dispatching rules (denoted from a) to d) in) as reported by Zambrano Rey, (2014).

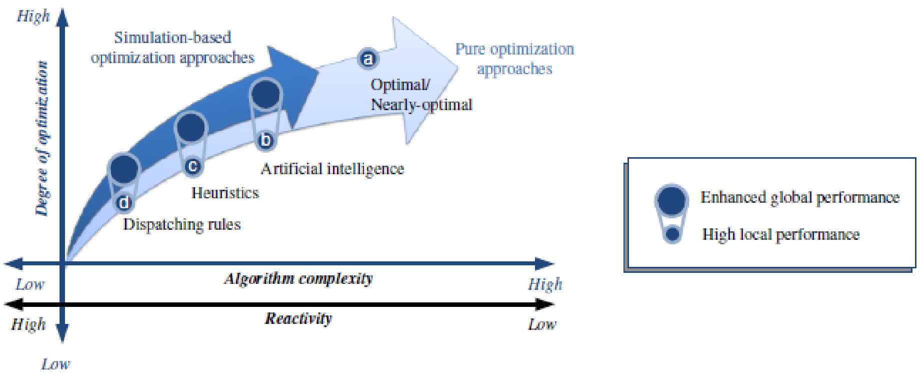


Figure 2: FMS scheduling techniques, taken from Zambrano Rey, (2014)

Since using optimal/nearly-optimal methods to solve the realistic FMS problem is computationally intensive, and sometimes no solutions can be found; artificial intelligence algorithms have become popular to solve such problems. Artificial intelligence try to mimic the human brain or biological systems in an attempt to find good solutions (possibly the optimal) to high complex problems, with less calculation effort thus favoring reactivity (Steels and Brooks, 1995).

A metaheuristic is an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high-quality solutions (Talbi, 2009). It may manipulate a complete (or incomplete) single solution or a collection of solutions at each iteration. The subordinate heuristics may be high (or low) level procedures, a simple local search, or just a construction method. Metaheuristics have been applied to solve realistic industrial problems such as vehicle routing (Kovacs et al., 2015), warehouse location and placement ("A Parallel Genetic Algorithm for the Multilevel Unconstrained Lot-Sizing Problem," 2008; Almeida-Luz et al., 2009; Byun et al., 2009; Homberger and Gehring, 2008), steel industry (Zhao et al., 2011), packing (Leon et al., 2009; Segura et al., 2011), and assembly line balancing (Ozbakir et al., 2011; Alba et al., 2013). In scheduling, metaheuristics haven been popular to solve single-machine (Bożejko et al., 2014), flow-shop (Dai et al., 2013), job-shop (Kuo and Cheng, 2013) and flexible-job shop problems (Mekni and Chaâr, 2014).

Metaheuristics can be divided into two groups: population-based or single-based metaheuristics. Population-based metaheuristics can be understood as an iterative improvement of a population of solutions. Some well-known population-based metaheuristics are genetic algorithms, evolutionary strategies, particle swarm optimization, and ant colony optimization (Talbi, 2009). Single-based metaheuristics can be interpreted as “walks” through neighborhoods or search trajectories through the search space of the problem at hand. Simulated annealing, tabu search, simple evolutionary strategies, trajectory or local search methods are some examples of single-based metaheuristics (Talbi, 2009). Contrary to single-based algorithms that focus on improving and maintaining one candidate solution; population-based metaheuristics focus on generate a multiple candidate solutions, applying the principle of evolution: survival of the fittest.

Given the challenges faced by FMS, population-based metaheuristics seem to offer promising results for FMS scheduling. However, a sequential and centralized execution of metaheuristics have a polynomial temporal complexity, thus they might not be suitable for industrial problems. Hence, decentralizing metaheuristic execution seems an interesting way not only to reduce the search time but also to improve the quality of the solutions provided (Alba et al., 2013; Cahon et al., 2004a).

2. Literature Review

2.1. FMS scheduling using metaheuristics

Scheduling is an important element of production systems because it serves as an overall plan on which many other shop activities are based. There are two key elements in any scheduling system: schedule generation and revisions (monitoring and updating the schedule). The first element which acts as a predictive mechanism determines planned start and completion times of operations of the jobs. The second element which is viewed as the reactive part of the system monitors the execution of the schedule and copes with unexpected events (i.e., machine breakdowns, tool failures, order cancelation, due date changes, etc) (Sabuncuoglu and Bayız, 2000).

Metaheuristics have been actively used for FMS scheduling and re-scheduling given their advantages. In fact some metaheuristics, such as genetic algorithms, are already available in commercial software for scheduling in industrial applications (Nie et al., 2013). Several centralized FMS scheduling approaches based on metaheuristics have been proposed. For instance, Zambrano Rey et al., (2015) used genetic algorithms and particle swarm optimization for solving the flexible job-shop scheduling problem. Such approach was implemented and validated with a realistic FMS. Souier et al., (2012) presented the results of a simulation study of a typical flexible manufacturing system, in which the routing scheduling problem was addressed using simulated annealing, tabu search, ant colony optimization, genetic algorithms and particle swarm optimization.

2.1. Decentralized metaheuristics

Population-based metaheuristics are stochastic search techniques that have been successfully applied in many real and complex applications (epistatic, multimodal, multiobjective, and highly constrained problems) that are involved in the FMS scheduling problem. A population-based algorithm is an iterative technique that applies stochastic operators on a pool of individuals (the population). Every individual in the population is the encoded version of a tentative solution. An evaluation function associates a fitness value to every individual indicating its suitability to the problem. Iteratively, the probabilistic application of “variation operators” on selected individuals guides the population to tentative solutions of higher quality (Alba et al., 2013).

Decentralized metaheuristics have shown to be useful in practice to solve a large number of applications. Many real-life problems may need days or weeks of computing time to be solved on serial machines. This is the usual scenario when considering difficult problems such as complex combinatorial problems with large search spaces, multi-objective problems with many hard-to-evaluate objective functions, or dynamic optimization problems, such as the dynamic FMS scheduling problem (Alba et al., 2013). In terms of decentralization of the population evolution Alba et al., (2013) reports the parallel models shown in Figure 3: Classical parallel models for population based meta-heuristics: (a) master-slave, (b) distributed and (c) cellular models. In (a) the master centralizes the population and manages the selection and the replacement steps. It sends sub-populations to the workers that execute recombination and evaluation tasks. The workers return back newly evaluated solutions to the master. In (b) the model divides the entire population into several sub-populations distributed among different processors and each one is responsible for the evolution of one sub-population. It executes all the steps of the meta-heuristic, and occasionally individuals migrate among islands. In (c) each processor is responsible for either a single individual or at most a small number. The difference with respect to the island paradigm is that the diffusion-based scheme requires a neighborhood structure of processors to perform the recombination and selection (Baños et al., 2006).

Another study focused on parallel metaheuristics was proposed by Bożejko et al., (2010). The proposed approach solves the flexible job shop scheduling problem by solving the classic job shop problem where operations have to be executed on one machine from a set of dedicated machines. The parallel metaheuristic implemented was based on two modules, the machine selection module and the operation scheduling module. The machine selection module was implemented with a tabu search (1st approach) and a population-based metaheuristics (2nd approach) working sequentially. This module was configured to help an operation to select one of the parallel machine from the set of machine types to process it. In turn, the operation scheduling module was used to schedule the sequence and timing of all operations assigned to each machine from the center. In other words, this module had to solve the classic job shop problem after having assigned operations to machines.

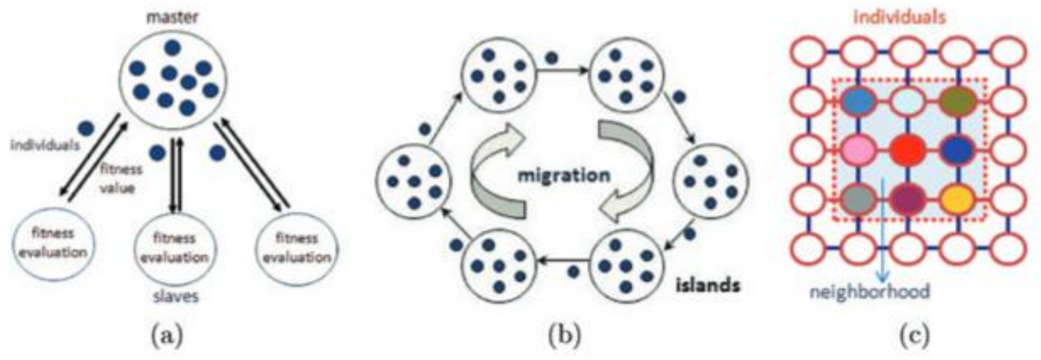


Figure 3: Classical parallel models for population based meta-heuristics: (a) master-slave, (b) distributed and (c) cellular models

Cahon et al., (2004) proposed a ParadisEO that is an extension of the evolving objects (EO) framework devoted to the design of serial evolutionary algorithms (EA). The parallel/distributed population evaluation is always stored on a single processor. However, the assignment of the individuals to the processors for evaluation depends on the hardware architecture. On shared memory multi-processors threads pick up individuals from the global list in a concurrent way. On distributed memory processors the population is divided into sub-populations according to the user defined granularity parameter.

3. Objectives

Taking into account that FMS reactive scheduling is of the outmost importance to face unexpected events, and that decentralization helps to improve scheduling reactivity and performance, the following questions arise: *What would be the criteria to take into account when designing a decentralized metaheuristic, in terms of problem encoding? How can the FMS scheduling problem be decomposed in order to decentralized the metaheuristic used to solve it? What type of behavior should be defined for determining the relationship among the different entities dealing with each part of the problem?*

Then, the main objective of this capstone final project is to propose a decentralized metaheuristic approach applied to the FMS scheduling problem. The specific objectives derived from this main objective are:

- Analyze the FMS scheduling encoding schemes that can be applied in decentralized approaches
- Propose a decentralized approach for the FMS scheduling problem
- Validate the proposed approach against a centralized approach from related literature using design of experiments

3.1. Design requirements

The main requirements for this project are:

- Reduced response times: one of the main goals of decentralizing a metaheuristic is to reduce the search time. This helps designing real-time and reactive optimization methods. Therefore, the proposed methodology should help to face one of the main challenges for FMS scheduling, which is to be reactive in order to cope with internal and external perturbations (Talbi, 2009).
- Improving robustness: As the optimization applications are often time-consuming the performance issue is crucial. The metaheuristic should be able to perform well on a large variety of instances and problems using the same parameters. The parameters of the metaheuristic may be over fitted using the training set of instances and less efficient for other instances (Talbi, 2009). Moreover, the execution of the algorithms must be robust to guarantee the reliability and the quality of the results. Hence, the proposed methodology should be suitable to tackle variations of the FMS scheduling problems, including different objective functions and/or multiple objective functions.
- Scalability: The proposed approach must show an improvement of the computing cost when dealing with large instances when compared with a centralized approach.

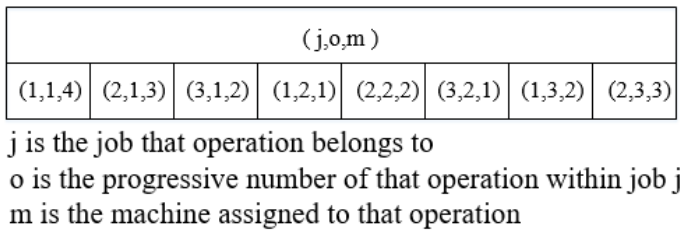
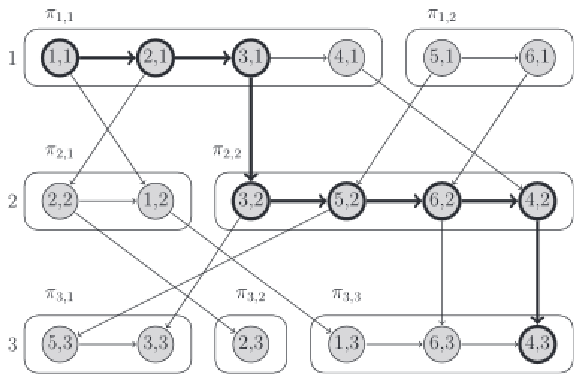
4. Methodology

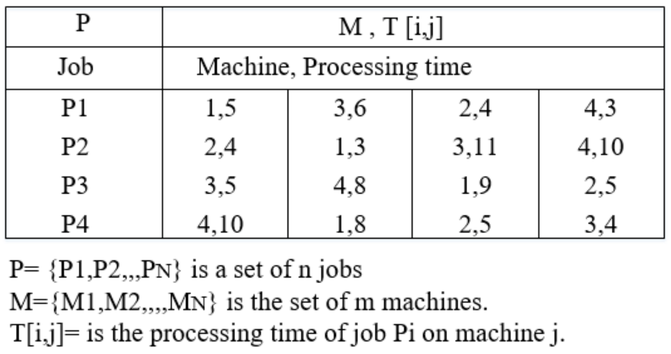
As presented in Section 2, there are various options to decentralize a population-based metaheuristic. However, all the aforementioned methods are similar in the sense that decentralized entities are used to evaluate an individual of the population or they have in charge the evolution of a subset of individuals. Therefore, the entire problem is evaluated, once or multiple times. This section first presents an analysis of encoding schemes, aiming to find one that ease decentralization. In addition, in the same Section, the strategy of Decentralization, focused on the machine sequence encoding scheme is explained. Section 4.2 describes the implementation to decentralize the FMS scheduling problem based on the chosen encoding scheme, independently from the metaheuristic. Last, Section 4.3 explains the programming strategy implemented in Java for two behaviors.

4.1. FMS encoding schemes

An interesting option to have an actual decentralization is to have multiple entities working on parts of the problem, hence it is important to find an encoding scheme that allows to break the problem into several problems. Table 1 reports the different encoding schemes found in FMS scheduling articles.

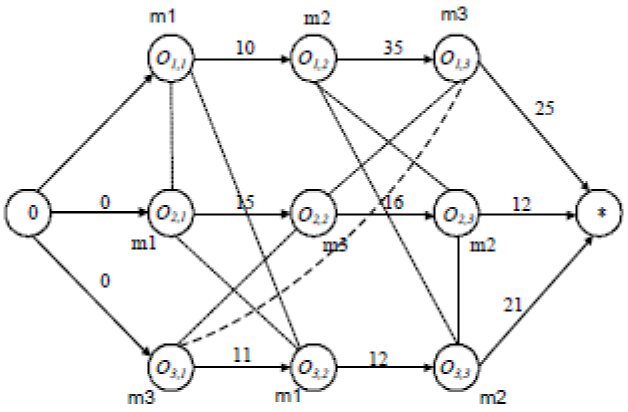
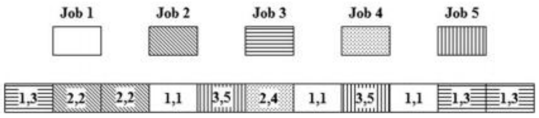
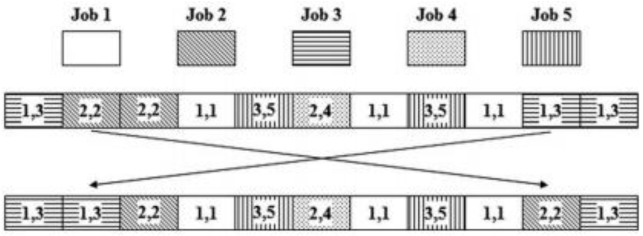
Table 1. Different encoding schemes

Authors	**strategy**	**Encoding**
Defersha and Chen, 2010	Island model	 (j,o,m) (1,1,4) (2,1,3) (3,1,2) (1,2,1) (2,2,2) (3,2,1) (1,3,2) (2,3,3) j is the job that operation belongs to o is the progressive number of that operation within job j m is the machine assigned to that operation In this representation, a chromosome is composed of serval genes, one for each operation, and each gene is formed by a triplet (j, o,m), where m can assume the index of an alternative machine on which an operation o of job j can be assigned. The sequence of the genes in the chromosome represents the sequence of the jobs in the machines.
Bożejko et al., 2013	Neigborhood model	 $\pi_{1,1}$ $\pi_{1,2}$ 1 1,1 2,1 3,1 4,1 5,1 6,1 $\pi_{2,1}$ $\pi_{2,2}$ 2 2,2 1,2 3,2 5,2 6,2 4,2 $\pi_{3,1}$ $\pi_{3,2}$ $\pi_{3,3}$ 3 5,3 3,3 2,3 1,3 6,3 4,3

		In this coding system the system with which the evaluation was carried out consists of having machine centers, that is, its machine sequence will be linked to a center where each center is equipped with identical machines, therefore at the moment to perform the evaluation of the coding, first they have to evaluate the processing of the works in each center and then they must evaluate the assignment in each of the centers, therefore this coding can not be evaluated in our problem because it is not It has machine centers, if not all machines have the same properties.
Leila Asadzadeh∗ , Kamran Zamanifar	Island model	In this model, was used an island model, the initial population is divided into sub-populations, and each sub-population is evolved separately. Communication between sub-populations is restricted to the migration of chromosomes. The evaluation and decision not to use this method was due to the fact that the communication between agents is only allowed at the moment of making a migration in the genetic algorithm, because the decision of the work was to do the analysis with a PSO algorithm, the communication within the algorithm because the sample space of the machine selection and the sequence of operations can undergo changes at the moment that the instance suffers some kind of disturbance this would generate a change within the division of the machine sequences to each agent and with this coding there would not be any kind of immediate reaction. 
Chiu and Liu, 2011	Master-slave model	In this method, the structure of master-slave model is adopted to achieve higher diversity of population and escape from being trapped in local optima, in addition to realize parallel processing. In the master-slave model, one computing node is designated as the master while the others are designated as slaves, which are separated from each other and evolve independently.

P	M , T [i,j]			
Job	Machine, Processing time			
P1	1,5	3,6	2,4	4,3
P2	2,4	1,3	3,11	4,10
P3	3,5	4,8	1,9	2,5
P4	4,10	1,8	2,5	3,4

P= {P1,P2,,,PN} is a set of n jobs
M={M1,M2,,,,MN} is the set of m machines.
T[i,j]= is the processing time of job Pi on machine j.

		 m1 m2 m3 $O_{1,1}$ $O_{1,2}$ $O_{1,3}$ 10 35 25 0 $O_{2,1}$ $O_{2,2}$ $O_{2,3}$ * 15 16 12 m1 m3 m2 0 21 $O_{3,1}$ $O_{3,2}$ $O_{3,3}$ 11 12 m3 m1 m2
L. De Giovanni, F. Pezzella	An Improved Genetic Algorithm	 Job 1 Job 2 Job 3 Job 4 Job 5 1,3 2,2 2,2 1,1 3,5 2,4 1,1 3,5 1,1 1,3 1,3  Job 1 Job 2 Job 3 Job 4 Job 5 1,3 2,2 2,2 1,1 3,5 2,4 1,1 3,5 1,1 1,3 1,3 1,3 1,3 2,2 1,1 3,5 2,4 1,1 3,5 1,1 2,2 1,3 The proposed Algorithm is obtained by a straightforward extension of the chromosome encoding used for classic job-shop problems, where no information on operation routing is required: each gene specifies a job and a related factory, while the sequence of operations on machines is determined by the decoding procedure, according to the order in which the genes appear.

The encoding schemes presented before combine the machine selection with the operation sequencing decisions, then metaheuristics have to explore both solution spaces at the same time. The main hypothesis of this project is based on the possibility of dividing the physical FMS configuration, so the metaheuristic can be decentralized to focus on parts on the system. This idea gives the possibility to assign parts of the system to an entity so such entity can intensively explore the reduced solution space, increasing the possibility to find better solutions. Consequently, in the following subsection, the encoding scheme proposed by (Zambrano Rey et al., 2015) is described because it allows to form subsets of individuals based on subsets of machine sequences.

4.1.1. Strategy of Decentralization-Machine sequence encoding scheme

Due to the different nature of the machine selection and job sequencing sub-problems, Zambrano Rey et al., (2015) propose an encoding technique that handles the two problems separately, through two-vector encoding. This chromosome structure has two main advantages. First, two-vector encoding allows generic problem modeling, making it possible to use the same structure for other kinds of problems (e.g., the job-shop problem and flow-shop like problems). Second, if both vectors are independent, various genetic operations can be applied, allowing more diversity and easier chromosome manipulation, thus improving the efficiency of the algorithm. In order to restrain the chromosome size, the sequencing vector is a permutation of the integers from 1 to *n* (where *n* is the number of jobs) and represents a feasible solution for the scheduling sub-problem. As shown in Figure 4, ej is the position of job *j* in the release sequence.

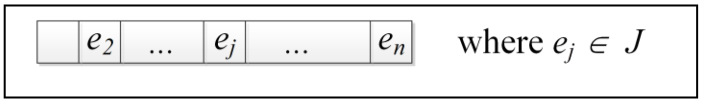


Figure 4: Sequencing vector

For the machine selection vector, an indirect encoding technique is used to explore the different machine sequences for each job. Since the size of the second vector must comply with the size of the first one, the genes in the second vector must represent an entire machine sequence for their corresponding jobs in the first vector. Given that different types of jobs may have a different number of available machine sequences, a bounded real coding scheme is adopted. As described in Figure 5, each gene g_j of the machine routing vector is generated by a uniform distribution. By multiplying the value of g_j with the maximum number of available machine sequences of each job *j* (MR_j), the index of one feasible machine sequence, denoted by mr_j, is obtained for each job *j*. The ceiling function (Figure 5) is used to obtain a meaningful value within the set of available machine sequences, i.e., the smallest upper integer.

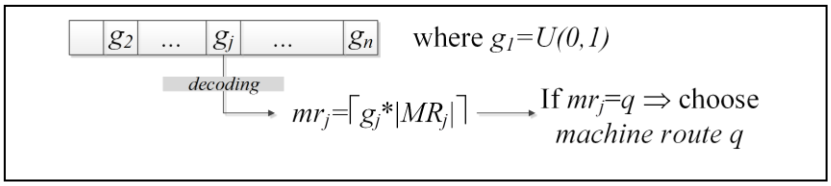
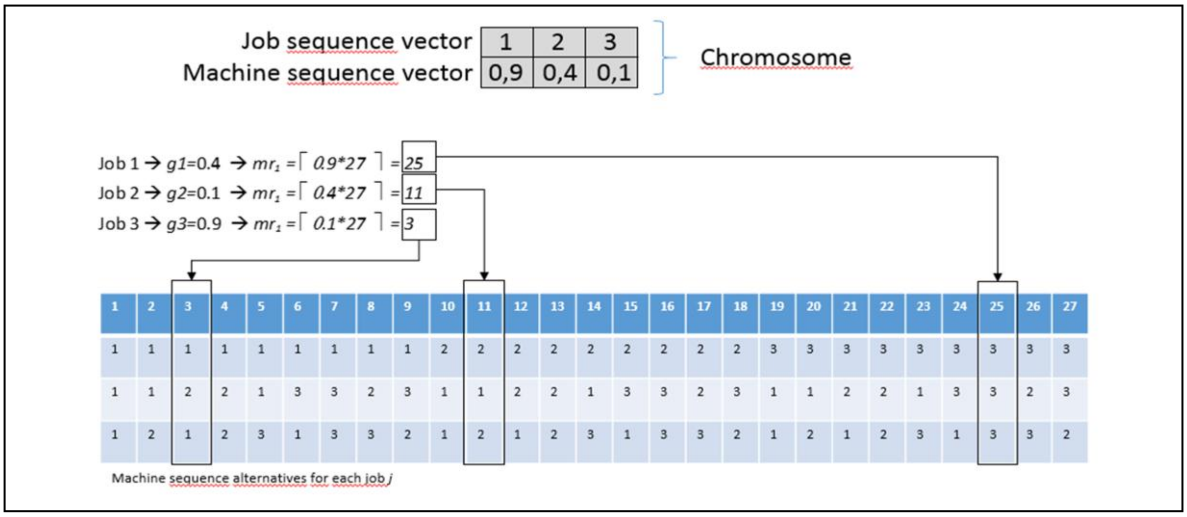

g_2
…
g_j
…
g_n
where $g_1=U(0,1)$
decoding
$mr_j=\lceil g_j * |MR_j| \rceil$
If $mr_j=q \Rightarrow$ choose machine route q

Figure 5: Machine selection vector

Let's consider a case where there are 3 jobs and a FMS system with 3 machines. The FMS has full machine flexibility meaning that all machines can execute all operations. Figure 6 shows the decoding process to obtain a machine sequence for each job. In this case, jobs will be dispatch in order [1 2 3] and the machine sequences for each job by the decoding process, obtaining machine sequences [3 11 25]. It is important to say that the machine sequence matrix is built at the beginning of the process, on the basis of the active machines; if a machine breaks down, then the machine sequence matrix should be replace to leave only the available sequences.

The interesting part of this encoding scheme is the capability of focusing of certain sequences, isolating for instance certain machines. Thus it is possible to create groups of machine sequences depending on the part of the system that we would like to analyze. For instance, in Figure 7, there are 5 machines and the area marked with red dotted lines contains all the sequences that use machines 1, 2 and 3. Consequently, the decentralized approach proposed herein is based on this division of the physical system, meaning that decentralized processing units take care of different areas of the system, and compete to offer the best scheduling performance. It is then expected that, the evaluation of the whole solution space is better achieved by multiple entities focusing on multiple physical layouts.


Job sequence vector | 1 | 2 | 3
Machine sequence vector | 0,9 | 0,4 | 0,1
Chromosome
Job 1 → $g1$=0.4 → mr_1 = ⌈ 0.9*27 ⌉ = 25
Job 2 → $g2$=0.1 → mr_1 = ⌈ 0.4*27 ⌉ = 11
Job 3 → $g3$=0.9 → mr_1 = ⌈ 0.1*27 ⌉ = 3

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	3	3	3	3	3	3	3	3	3
1	1	2	2	1	3	3	2	3	1	1	2	2	1	3	3	2	3	1	1	2	2	1	3	3	2	3
1	2	1	2	3	1	3	3	2	1	2	1	2	3	1	3	3	2	1	2	1	2	3	1	3	3	2

Machine sequence alternatives for each job *j*

Figure 6: Decoding process

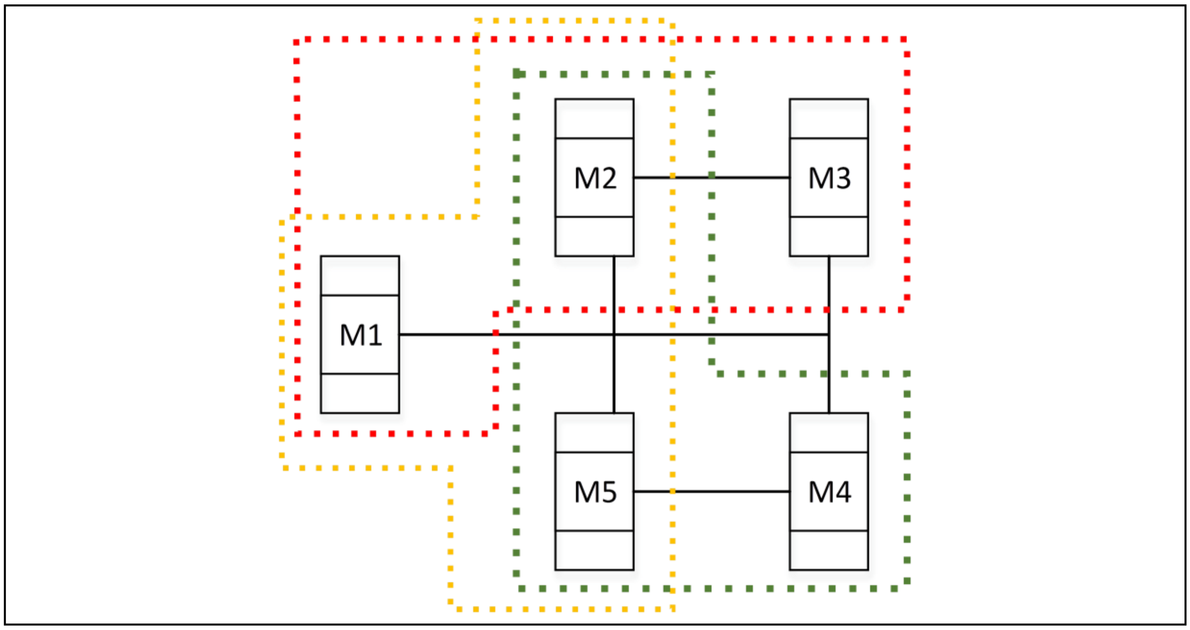


Figure 7:Different machine sequences

4.2. Implementation a decentralized metaheuristic for the FMS scheduling problem

4.2.1. Multi-agent systems

Multi-agent systems can be defined as a collection of, possibly heterogeneous, computational entities, having their own problem-solving capabilities and which are able to interact in order to reach an overall goal (Oliveira et al., 1999). The multi-agent systems (MAS) are a society organized, constituted by semi-autonomous agents, which interact with others, aiming to resolve collaboratively some problems, or to achieve some individuals or collectives goals. The agents may be homogeneous or heterogeneous and have common goals or not, but still maintain a degree of communication between them (Bennane, 2010). A multi-agent system is usually composed by a set of autonomous agents that have to coordinate their activities in order to accomplish a global objective. While the autonomy of the agents is essential for a MAS, it may cause the loss of the global coherence. The organization of a MAS is used to manage these intra- and inter-agent properties (Bordini et al., 2007). In the following, different organizations are described:

Hierarchies: Hierarchies could be draw as a tree structure, it worked dividing the real problem into sub-problems where these sub-problems are easily to be solved. The agent as more level has in the tree, it has more clarity of the problem; on the other hand it has strong disadvantages, such as bottlenecks and lags.

Holarchies: A holarchy is a hierarchy of self-regulating holons which function (a) as autonomous wholes in supra-ordination to their parts, (b) as dependant parts in sub-ordination to controls on higher levels, and (c) in co-ordination with their local environment (Botti and Giret, 2008). Holarchies systems are incredible useful when the problem can be divided and assigned to different holons as a result of some advantages, as stability, autonomy and cooperation capacity.

Coalitions: A coalition is a subset of agents formed for a purpose (goal directed) and dissolved when that need no longer exists (dynamic). There are some extended and difficult task that cannot be solved by a single agent because his capabilities, limited resources; or because his performance is too slow. Then, a group of agents formed a coalition with the purpose of solve the task o problem by cooperation. Its drawback is the additional complexity incurred if the partitioning of agents is not disjoint (Horling and Lesser, 2004).

Teams: A team is set of cooperative agents which have agreed to work together towards a common goal. In comparison to coalitions, teams attempt to maximize their utility (Anderson et al., 2004). This high-level goal remains relatively consistent and agents can change their roles over time in order to accomplish it.

Congregations: are groups of agents who have bounded together into a typically flat organization in order to derive additional benefits. Unlike other paradigms, congregations are assumed to be long-lived and are not formed to achieve a single specific goal. A congregation allows an agent to localize its search for other agents; rather than considering the entire population as being equally likely to contain agents worth interacting with, an agent can bias its search towards members of its congregation.

Societies: Agents can have different goals, varied levels of rationality and heterogeneous capabilities, and they may also differ in their available information; the societal construct provides a common domain through which they can act and communicate. A set of constraints (social laws or norms) is imposed on the behavior of the agents (Arcos et al., 2005). There are two aspects to the society formation problem. The first is to define the roles, protocols and social laws which form the foundation of the society. Given such a definition, the second problem is to implement the more literal formation of the society; that is, determining how agents may join and leave it.

Federations: consist in groups of agents which have ceded some amount of autonomy to a single delegate, which represents the group (called facilitator, mediator, translator or broker). Group members interact only with this agent, which acts as an intermediary between the group and the outside world (Jun Sawamoto et al., 2005). An agent federation is a small multi-agent system composed of one service agent and several function agents. The service agent constitutes the service management of agents, and it is responsible for coordinating the internal members and interacting with the intermediary and other agent federations on behalf of its own agent federation. The function agents are responsible for the implementation and execution of the specific functions in the agent federation.

Markets: are collections of agents with two main roles: buyers and sellers (shown in Fig. 2h as red and white respectively). Buyers send bids for a common set of items (e.g., resources, tasks, services, goods), which are received by sellers that process them and determine the winner. In these structures, agents are self-interested, driven completely by their own goals. Interaction in markets occurs through communication and negotiation (Gimpel et al., 2008; Nevmyvaka et al., 2003).

The organizational structure considered here in is based on a ***coalition of agents***, because the primary agent decides to call a group of agents to help with a scheduling task, giving its complexity. The primary agent divides de physical layout and assign to other agents in the coalition, smaller search spaces by giving only certain machine sequences to explore.

4.2.2. Multithreading

A thread is a sequence of instructions executed within the context of a process, and less overhead is associated with multiple threads (running under a process) than with multiple processes. The smaller memory and synchronization overhead of a thread, relative to a process, creates the potential for writing more efficient parallel programs with multiple threads. For example, a parallel application may run on a symmetric multiprocessor computer with one process running for each processor on a node, an approach which requires storage on each node of the entire overhead of all processes on that node, and which likely requires storage of redundant data. (Nielsen and Janssen, 2000). Multi-threaded processor architectures are capable of concurrently executing multiple threads using a shared execution resource. Multi-threaded processors have two major advantages over other types of processor: 1) they offer the ability to hide latency within a thread (e.g., memory or execution latency) and 2) they achieve high efficiency. A possible third advantage is that they are potentially less complex than alternative high performance processors. (Chalmers et al., n.d.). Multithreading is also used for hiding long memory latency in uniprocessors and multiprocessor computer systems and aims at increasing system efficiency(Vlassov and Ayani, 2000). Multi-threading provides multiple execution contexts within a single process. A process comprises a list of executable instructions, all the runtime data needed by the program, open file descriptors, a main thread of execution, and, possibly, other threads of execution. All threads in the process have access to all the data and file descriptors, and, additionally, there is a small amount of data unique to each thread. (Nielsen and Janssen, 2000).

In order to handle decentralization, Java libraries (Jade) were used in the first place because it allows the creation of multi-agents capable of performing the division of the scheduling problem and handles the coding of metaheuristic approaches. Also, Java allows the intelligent evaluation of the response that acquires by other agents. Given the limited knowledge in programming multi-agents systems in Java, another option to simulate decentralization was considered, aiming to obtain results in less amount of time, but guaranteeing to preserve the focus and objectives of the project. Besides, Java provides parallel programming support in the core of the language. This feature enables programmers to write code that exploits multithreading without the need to use any external library, as for instance multi-agent libraries (Eiras-Franco et al., 2016). Therefore, in this project, the multithreading concept was be used to simulate the coalition of agents, in which each thread simulates and agent. In the following section, the design methodology is explained.

4.3. Programming strategy

The strategy used to evaluate and physically divide the system is based on the creation of two types of agents (or decisional entities), a main agent and several secondary agents. Section 5.2 explains the functions and architecture of each of the agents. The main agent orchestrates the scheduling problem and leads secondary agents. These secondary agents evaluate a number of machine sequences thus physically evaluate only part of the problem (FMS system), according to the allocation done by the main agent. Secondary agents will work in parallel, solving the algorithm and at the end all agents send their response to the main agent. Then the main agent orders them according to the lower objective function (OF). In this project we propose two behaviors, one without and another with competition. Competition is established between secondary agents where at the end of the evaluations and after the main agent orders, a percentage of secondary agents are eliminated due to their poor performance. In the following, two pseudocodes are explained for the two behaviors.

Alg. 1 Decisional Entity (DE) pseudocode without competition

The objective is to achieve a solution between several launches of the main program without some type of competition between each one of the solutions.

```
    Begin
01. Set DE parameters:
        Benchmark (Benchmark), Number of Jobs (NbJobs), Number of machines (NbMacs), Number of
    threads (n), Combinations to evaluate (mat[I][NbJobs]), Machine Availability (MacAvailability),
    Batch Description (BatchDescrip), Processing Times (ProcTimes), Machine Routing Flexibility
    (MRF), Common Due Date (CommonDD).

02. Generation of the combinations to evaluate
            While (generation<I) do
            Create mat[generation] combination;
            EndWhile

03. Generation of the threads
            For i=1 to n
            Create Thread I;
            Send parameters to Thread i (i, MacAvailability, Benchmark, NbJobs, BatchDescrip,
            ProcTimes, JobNumber, JobDescrip, NbMacs, MRF, CommonDD, mat[i]);
            Endfor

04. Wait to all threads execution finished
```

```
            Wait the notification of all threads;

05. Sort threads in ascending order depending on their objective function
            Choose the Thread with the smallest objective function;
            End
```

Alg. 2 Decisional Entity(DE) pseudocode with competition
The objective is to reduce the computational cost through the competition between the different solutions.

```
Begin
01. Set DE parameters:
        Benchmark (Benchmark), Number of Jobs (NbJobs), Number of machines (NbMacs), Number of
    threads (n), Combinations to evaluate (mat[I][NbJobs]), Machine Availability (MacAvailability),
    Batch Description (BatchDescrip), Processing Times (ProcTimes), Machine Routing Flexibility
    (MRF), Common Due Date (CommonDD), Kill percent (KillDE), Threads Counter (x), Thread
    Validation (Validate[I])

02. Generation of the combinations to evaluate
            While (generation<I) do
            Create mat[generation] combination;
            EndWhile

03. Generation of the threads
            For i=1 to n
            Create Thread I;
            Send parameters to Thread i (i, MacAvailability, Benchmark, NbJobs, BatchDescrip,
            ProcTimes, JobNumber, JobDescrip, NbMacs, MRF, CommonDD, mat[i]);
            Endfor

04. Select the best Thread solution based on their objective function
            While (x >1) do
            //Check for termination of the iteration k is reached
            Wait the notification of all threads
            Sort threads in ascending order depending on their objective function;
            For i=n to (n-(x*KillDE))
            Assign value of 0 to the current thread objective function;
            Validate [Thread i] = false;
            Notify Thread to Stop;
            EndFor
            Calculate the Number of threads alive (new x);
            EndWhile
            End
```

5. Engineering design

5.1. Design statement

This project focuses on designing a decentralized approach to deal with the FMS scheduling problem. In this particular case, the coalition organizational paradigm is taken as reference and the architectural structure (decisional entities and their relationships) and dynamic behavior (competition or not) of the decentralized approach are defined. As a functional feature, a competition behavior between entities is also proposed in order to improve the computing cost of the proposal.

5.2. Design process

In order to design the proposed approach, the algorithm used by Zambrano Rey et al., 2015 was modified. In this section, the parameters, functions, and the entities capable of solving the FMS scheduling problem by decentralized metaheuristics is explained. It is important to state that this approach is independent from the metaheuristic, making it a generic approach.

The design process is explained in two phases. First, the actual architecture explaining the decisional entities (agents), the relationships between them, and their roles is shown in Figure 8.

- Decisional Entity ED: this is the main decisional entity (or main agent) that controls the entire process.
    - Role: Define the division of the problem, starts the process and gather all the solutions of the entities and make the decision of which entity follows and which entity dies.
    - Parameters: FMS Scheduling problem instance (based on literature benchmark), #Jobs, #Machines, #Threads, Competition/NoCompetition
    - Data send: Machine sequence, Stop Criteria
    - Data received: objective function, Completition times and computational cost.
- Decisional Entity TH#: these are the working entities (or secondary agents), which are assigned with a set of machine sequence combinations to test using a metaheuristic.
    - Role: Entity in charge of evaluating assigned part of the problem and returning the best solution to the main entity. This entity hosts the metaheuristic approach.
    - Parameters: Machine sequence iteration, objective function.
    - Data send: Completion time, objective function, Computational Cost i.
    - Data received: Stop criteria, Machine Sequence.

Parameters, Functions and Variables used in the decentralized strategy.

It is important to mention that because of previous works; the instantiation of this approach is done by choosing the PSO (particle swarm optimization) as the metaheuristic approach and the MSD (mean square deviation) as the objective function, since large deviations in job completion times are undesirable, the scheduling objective for just-in-time production is translated into the minimization of the mean-square due date deviation (MSD), according to the definition of MSD, the objective is to reduce the error between the estimator and what is estimated. However, the proposed approach does not constraint neither the selection of the metaheuristic, nor the selection of the objective function.

Number of Jobs: Number of Jobs in the FMS, it is a number depending of the type of problem associate
Number of Machines: Number of Machines available used in the FMS
Common due date: Common due-date to all jobs
Partial Flexibility: Boolean variable that identifies if the instances is partial flexibility or not

Iter Max: Maximum number of iterations where there is no improvement of <0.01 in the MSD
Iterations: Number of total iterations
Number of Threads: Number Threads working in different machine sequences
Thread Competition: Boolean variable that identifies if there is competition
Type Competition: Variable that identifies what type of competence will evaluate
Perc Kill Competition: Percentage of entities that will be eliminated after *n* iterations.
Number Iteration to Kill: Number of iterations reached where the best response of each thread will be evaluated
Machine sequence combinations(): Is the set of machine routes available for job j.
A machine route is defined as a sequence of machine that handle job's operation sequence.
Completion times: Variable time depending on the time result where the FMS Scheduling ended.
MSD_i: Variable of the MSD obtained from each of the entities
Computational cost i: Variable computational cost obtained after each iteration of the entities
PSO( ): Metaheuristic approach used to solve the FMS, the PSO has different parameters explained below.

Set PSO parameters: Swarm size, inertia weight (w in Eq. 1), acceleration constants (c1 and c2 in Eq. 1), maxnumber of generations, improvement parameter, Initialization, Initial population evaluation, Generation of the next swarm, Calculate the new velocity (Vk in Eq. 1), Calculate the new position (Xk in Eq. 2), Evaluation of the new swarm, Evaluate each particle's MSD, Replaced individual best solution (pbest) by the new particle, Sort particles in ascending order depending on their MSD Set global best solution(gbest) with the particle with the smallest MSD; Choose the particle with the smallest MSD

$$V_K(t+1) = w * V_k(t) + c_1 * r_1 * \left(pbest_k(t) - X_i(t)\right) + c_2 * r_2 * \left(gbest(t) - X_k(t)\right)) \quad \text{(Eq. 1)}$$

$$X_K(t+1) = X_k(t) + V_k(t+1) \quad \text{(Eq. 2)}$$

Where ω is the inertia weight, and c1 and c2 are the acceleration constants.
Return(Completion time, MSD, Computational Cost): When the current thread finished will communicate the current Completion time, the best MSD and the Computational Cost to the main.

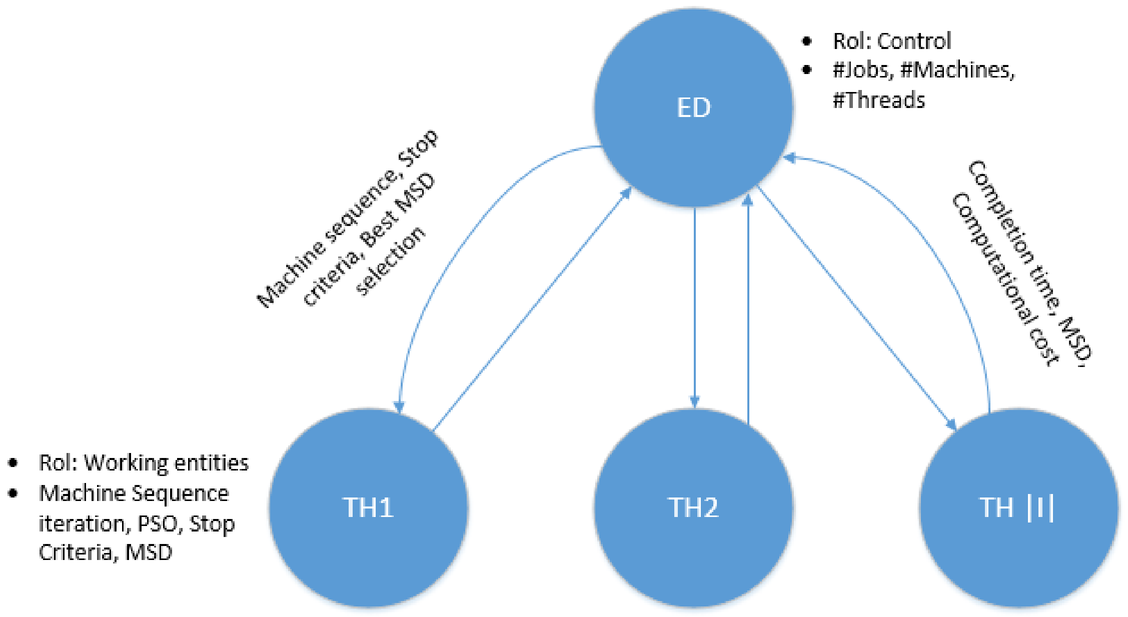

ED
• Rol: Control
• #Jobs, #Machines, #Threads
Machine sequence, Stop criteria, Best MSD selection
Completion time, MSD, Computational cost
• Rol: Working entities
• Machine Sequence iteration, PSO, Stop Criteria, MSD
TH1
TH2
TH |I|

Figure 8. Strategy Communication

The way of communication of the chosen strategy, including the roles and behavior, where the main thread or Decisional entity generates the combinations and loads the necessary data for the realization of the FMS scheduling problem (number of Jobs, number of machines, which machine will perform each job), after 10 iterations the main thread will evaluate the current solutions of the secondary thread, the current solution of each secondary thread is the best currently, and the Secondary threads will evaluate the PSO algorithm, creating an initial population and evaluating the position of each particle to obtain the MSD which is the solution of each iteration, all secondary threads are synchronized to send the current solution to the main thread.

Secondly, de dynamic strategy shown in Figure 9 is explained with a UML diagram where we show the different behaviors of the entities when a competition exists or where it not, also the Figure 10 and 11 describe the different parameters, function and variables that the strategy used.
The UML sequence diagram shows the behavior of the entities, how they communicate with each other, the handling of the information of parameters, variables and functions, as well as the order in which the whole process is carried out. As we can see, a cycle only ends when the main entity has eliminated all entities, for this all entities in each of the iterations send their responses (MSD) to the main entity so that it evaluates and decides and re-initiates the process. Where the parameters send by the main entity to the entities in the PSO are (Number of Jobs, Number of Machines, Machine sequence, MSD i), all the entities works with these parameters to be able to return to main the best MSD founded.

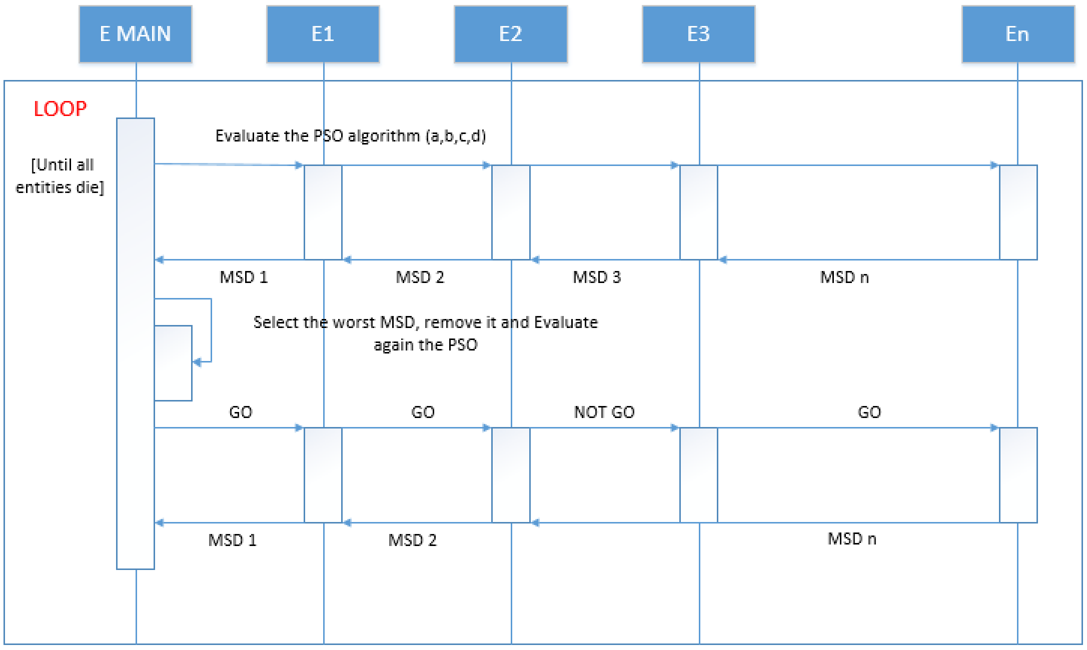

E MAIN
E1
E2
E3
En
LOOP
[Until all entities die]
Evaluate the PSO algorithm (a,b,c,d)
MSD 1
MSD 2
MSD 3
MSD n
Select the worst MSD, remove it and Evaluate again the PSO
GO
GO
NOT GO
GO
MSD 1
MSD 2
MSD n

Figure 9. UML Diagram

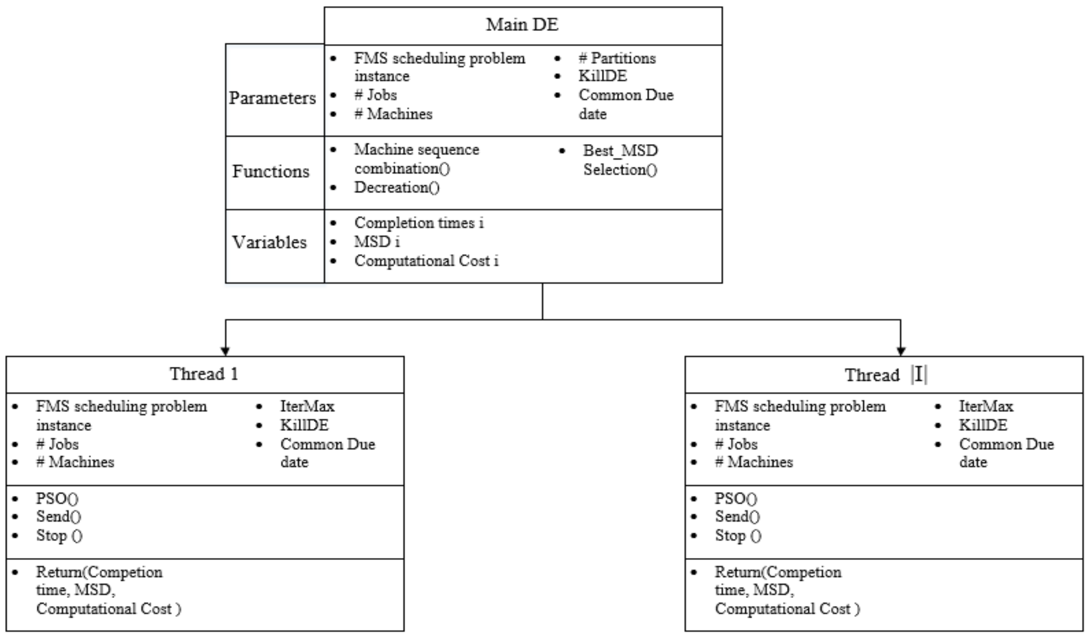


Figure 10. Decentralized Strategy Structure-Without competition

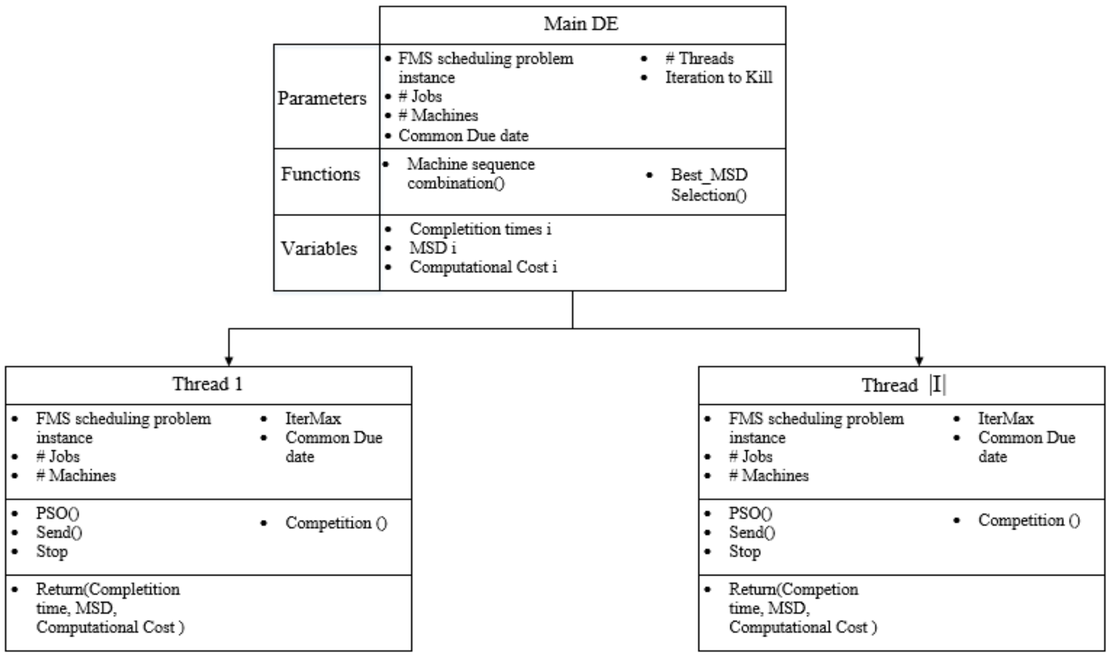


Figure 11. Decentralized Strategy Structure-With competition

The decentralized strategy structure demonstrates the different variables, parameters and functions that each type of entity must have for the correct operation of the strategy, as well as the different architecture for when there is and there is no competition. This project was based on showing the behavior when physical division of the system is possible, so it is very important to take into account the difference when there is and there is no competition and how the behavior of entities can influence the response.

Two types of competition between entities were carried out in this project, the first type of competition is, after n iterations, the current best MSD is evaluated for each of the threads and the thread with the worst response is eliminated; The second type of competition is that after iterations the best MSD is evaluated for each of the threads and x percentage of the current threads is eliminated.

5.3. Performance Requirements

The computational tool must provide a feasible solution to the FMS scheduling problem. It must consider all the proposed constraints. In addition, the proposed approach must provide the user to assign the value of the parameters corresponding to the problem that he/she wants to evaluate. Last, the proposed approach must provide a computational cost, MSD, the machine sequence and the completion times. In this particular case, the performance requirements are evaluation using a PSO optimization model. Thus, they may vary depending on the metaheuristic approach and objective functions used.

5.4. Performance Test

Existence of parallelism: In order to guarantee the decentralization and the execution of the whole system in parallel, the multithreading method was used in JAVA, which allows the parallel evaluation of the entities that work with only part of the problem.

5.5. Design restrictions

The FMS under study has been configured to assemble a set of products, each product with a different operation sequence and component requirements. The proposed methodology designed based on the following parameters, assumptions and constraints:

<u>Static parameters</u>: the following parameters are considered as static because they are defined on the basis of FMS components, configuration and technical features, which remain unchanged for long periods of time. Parameters related to product design are also considered static and included herein. Static parameters taken into consideration in this project are: the types of products, set of machines, machines' buffer capacity, product operation sequences, set of operations supported by the FMS, set of manufacturing operations supported by each machine, the distance between machines and transport device parameters (i.e. nominal speed, minimum distance, number).

<u>Dynamic parameters</u>: The following parameters were considered dynamic because they depend on FMS current status and production order data. Dynamic parameters considered herein are: the set of products to be manufactured, set of active redundant machines for each operation, the number of machine sequences available for each product and the set of available transport routes and transportation times.

<u>Assumptions and constraints</u>: the following assumptions and technical and managerial constraints are mostly taken into consideration in FMS studies with realistic implementations (Luh, 1998; Caumond et al., 2009; Herrero-Perez and Martinez-Barbera, 2010; Berger et al., 2010):

- A machine can process one operation at a time
- An operation is performed by only one machine
- Precedence constraints may exist between operations of the same product
- Machine's input buffer capacity is limited
- A product, once it has finished on a machine, is transferred directly to an available machine's input buffer
- Product re-circulation is allowed
- The number of simultaneous products in the FMS is limited
- Within one batch, setup times can be neglected.
- Preemptions are not accepted, then an operation has to be completed without interruption once it has started.

The mathematical model for the FMS taken into consideration, with the aforementioned parameters, assumptions and constraints, can be found in Zambrano Rey, (2014).

5.6. Norms and standards

Regarding the norms and standards for this project, the methodology was based on ISO 13053- 1 standard. DMAIC is an acronym for the five phases of the methodology which stands for: Define, Measure, Analyze, Improve and Control. Firstly, in the Define phase not only the importance of the problem to solve is determined but also the research question is established (see "Justification and Problem Statement" section). Additionally, this phase is related to the design of the decentralized approach, parameters, variables and funtions (see "Methodology" section). Considering the importance of evaluating the design of the model and its functionality, different FMS scheduling problems were analyzed for the validation of the effectiveness of the decentralized approach. Referring to the Measure and Analyze stages, the obtained results were analyzed using one way ANOVA, parameters were interpreted and adjusted. In the "Performance tests" section, the analysis and their respective relevance are described. In addition, in the "Results" section, a more detailed analysis of the obtained results is shown.

Having solved the model, the Improve phase responds to the purpose of analyzing the impact of the proposed model in comparison with the centralized approach. In this stage, the impact of the model was studied using the computational cost and the solution established in the design statement. This analysis is presented in the "Performance tests" and "Results" sections. Finally, the Control phase, seen as a segment of the implementation of the model in a real situation, was not a part of the scope of this project. However, it was included from a theoretical point of view, reviewing the results of the model and the computational tool functionality (See "Results" and "Conclusions" sections).

6. Results

The following experiments were carried out, to evaluate if the MSD varies according to the number of Threads created, also there is a competition between the Threads in 2 cases, 1. At the (10,20 and 50) Iterations there will be a comparison and the thread that has the worst solution is eliminated also another experiment where competition is entered every 20 iterations and 10,20, or 30% is eliminated ; in addition, the results are compared with the results obtained by Zambrano Rey, (2014) , where it evaluates a centralized structure through the metaheuristics Genetic Algorithm and PSO. Appendix 1 will show the results of all the experiments

To run these tests was used a virtual machine of Microsoft Standard_D15_v2 with the following specifications: 20 vCPUs, 140GB storage, 64GB RAM, this in order to use the multithreading methodology so that each Thread to work separately.

6.1. Case study: FMS Benchmarks

The proposed approach will be validated using the theoretical flexible job-shop scheduling benchmark proposed by Brandimarte, (1993). In such benchmark, there are 15 problem instances, denoted mk1 to mk15. In this project only, instances mk1 to mk10 were evaluated. The number of jobs varies from 10 to 30, the number of machines from 6 to 15 and the number of operations from 58 to 232. Two of the problems are fully flexible (mk2 and mk7) and the other were designed as partially flexible.

6.2. Analysis of results

Table 2. Results for Brandimarte's benchmark

Problem instance		DECENTRALIZED PSO			CENTRALIZED GA			CENTRALIZED PSO		
Problem ID	Due Date (s)	Avg MSD	{std}	Avg. CT(S)	Avg MSD	{std}	Avg. CT(S)	Avg MSD	{std}	Avg. CT(S)
Mk1	10	21.47	{0.42}	193.55	17.45	{0.31}	107	18.57	{0.83}	139
Mk2	10	16.84	{0.89}	174.25	10.67	{1.63}	153	10.04	{1.86}	189
Mk3	10	201.67	{4.82}	710.8	176.4	{3.41}	401	169.76	{20.89}	843
Mk4	30	55.51	{1.69}	545.4	34.06	{1.03}	356	28.61	{1.16}	854
Mk5	40	128.04	{2.56}	586.35	93.02	{1.37}	585	85.11	{1.41}	1028
Mk6	70	89.38719	{1.75}	219.6	25.6	{3.24}	625	13.45	{2.61}	878
Mk7	70	123.7105	{2.93}	1240.75	9.25	{1.95}	678	31.35	{7.89}	3561
Mk8	100	195.5325	{3.63}	1511.5	116.82	{2.09}	718	86.62	{3.11}	2693
Mk9	100	306.3114	{3.87}	1732.85	200.33	{6.94}	849	336.15	{81.81}	3254
Mk10	120	236.7173	{4.40}	1625.5	126.3	{2.90}	1118	186.23	{9.77}	2983

From the results we can conclude that the decentralized methodology is not improving the solution (MSD) obtained by Zambrano Rey, (2014) , this may be for several reasons such as the decoding scheme, or because the PSO algorithm does not have additional techniques to improve efficiency , however for the Standard deviation(STD) we observe that we do not obtain such high data as in some cases the centralized GA and PSO, this means that the sample space that is evaluating the algorithm does not present high peaks that may affect the final solution

For the following experiments, only the Mk2 test was taken into account, because it is completely flexible, it obtains greater complexity due to the greater number of sequences.

Experiment 1. Evaluate if MSD variates with the different number of threads

In this experiment, we will evaluate if the number of threads differs in the value of the MSD, in order to establish if it is possible to work with the minimum number of threads and thus decrease the computational cost.

$$H_O = The\ number\ of\ threads\ does\ not\ differ\ significantly\ in\ the\ value\ of\ the\ MSD$$
$$H_a = The\ number\ of\ threads\ differ\ in\ the\ value\ of\ the\ MSD$$

ANOVA					
MSD					
	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	4,971	2	2,486	,960	,393
Within Groups	82,811	32	2,588		
Total	87,782	34			

After performing the analysis of a factor, the result is that the number of threads not differ significantly at the level α= 0.05 in the calculated MSD. This means that for the experiment to evaluate the MSD with the different threads, it is possible to decrease the total number of threads to the minimum worked in this project (5), so all the simulations will handle with the minimum threads to minimize the computational cost.

Experiment 2. Evaluate the maximum number of iterations without competition

In this experiment, we will evaluate if the number of iterations(200,500,1000) differs in the value of the MSD, to establish if it is possible to work with the minimum number of iterations and thus decrease the computational cost.

$$H_O = The\ number\ of\ iterations\ does\ not\ differ\ significantly\ in\ the\ value\ of\ the\ MSD$$
$$H_a = The\ number\ of\ iterations\ differ\ in\ the\ value\ of\ the\ MSD$$

ANOVA					
MSD_Without Competition					
	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	,805	2	,402	,550	,580
Within Groups	41,698	57	,732		
Total	42,503	59			

After performing the analysis of a factor, the result is that the number of iterations no differ significantly at the level α= 0.05 in the calculated MSD. This means that for the experiment to evaluate the MSD when the number of iterations changes, it is possible to decrease the number of iterations to the minimum worked in this project(200), so all the simulations will handle with the minimum number of iterations and will minimize the computational cost.

Experiment 3. Evaluate if there is a difference in the MSD if change the number of iterations to enter competition

In this experiment we will evaluate if modifying the number of iterations to enter competition (10,20,50), affects the result of the MSD

$H_O = The\ number\ of\ iterations\ does\ not\ differ\ significantly\ in\ the\ value\ of\ the\ MSD\ with\ a\ experiment\ with\ competition$

$H_a = The\ number\ of\ iterations\ differ\ in\ the\ value\ of\ the\ MSD\ with\ a\ experiment\ with\ competition$

ANOVA					
MSD_With Competition					
	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	,549	2	,274	,183	,833
Within Groups	85,613	57	1,502		
Total	86,162	59			

After performing the analysis of a factor, the result is that the number of iterations between competition no differ significantly at the level α= 0.05 in the calculated MSD. This means that for the experiment to evaluate the MSD when the number of iterations changes, it is possible to decrease the number of iterations between competition, that could help with the solution because the sample space will be reduce with each dead thread.

6.3. Impact of the model

Aiming to analyze the impact of the model, due to the work has a theoretical approach it is very difficult to perform an economic impact analysis, taking into account that the case of the study are tests found in the literature that address especially the FMS, however if we take into account that our results obtained an improvement in computational time, we can say that because the strategy found a solution before this can be implemented previously earning time and saving money in the business environment.

7. Conclusions and recommendations

7.1 Conclusions

This paper proposed a decentralized Metaheuristic for the FMS scheduling problem aimed at minimizing the computational cost and improve the robustness in JIT context. The meta-heuristic approaches proposed are innovative because of their particle encodings, sharing a bi-level structure and a real-code-based encoding scheme. An important aspect of the encoding technique proposed is that it always offers legal and feasible solutions, thus avoiding repair mechanisms.

Taking into account that a completely flexible system was used for this project the DoE of this project, it should be established that for a problem that is partially flexible, this should work the same because partially flexible system is less complicated instance because it would have fewer combinations of machines possible for each of the iterations.

Due to the physical evaluation of the system and using the codification mentioned in section 4.3 it is possible to determine that the problem is adjusted to any type of disturbances in the system, and that it is not necessary to change the type of codification to the different problems and disturbances that Can be presented in an FMS.

Owing to the parallelization, each decisional entity has a smaller sample space if compared to a non-parallelized strategy, thanks to which the time in which the solutions are stabilized is smaller.

Due to the use of Threads, we see in Table 2 that the computational cost of the decentralized proposal is less than the centralized one, this is because the sample space is divided therefore for each iteration the agent only searches within the sample space assigned.

Through the designs of the proposed experiments, the results show that for this strategy it does not significantly affect the number of total iterations when improving the MSD, this tells us that we can base the minimum number of iterations worked on.

After analyzing the results obtained, it is important to highlight that it was possible to carry out the decentralization of the system, by the type of coding worked, however we found that the results are not what was expected according to what was evaluated and compared with a centralized instance.

Through the analysis of the different coding schemes used previously, it is found that in none of the schemes was a PSO algorithm used, this gives us the hypothesis that the PSO algorithm when used in a decentralized environment does not find a better solution if we compare it with the centralized, this may be due to the size of the population that is created at the moment of making the change from particle to particle.

It is found that within the code because multiple divisions of the sample space are generated this also generates multiple garbage data that generate slowness when performing an iteration and at the same time delays when delivering a final solution.

7.2 Recommendations

Future work on PSO will focus on techniques that allows to parallelize the solution, for PSO, it will be interesting to implement diverse parallelism methods as a multi agent systems, that interact between them and one principal agent take decisions to improve the results.

Future work could establish another type of competence when removing agents and therefore eliminate decisional sample space, this could improve the solution or computational times.

It is important to say that the PSO algorithm may not be optimized to work with the multithreading methodology, possibly with an improvement in the algorithm it may be possible to reach a better solution due to the decrease of the sample space.

As future work will could perform the analysis of what the results would be by means of a genetic algorithm, using the type of coding worked on in this project.

8. Glossary

Flexible Manufacturing System(FMS): A flexible manufacturing system (FMS)is a computer-controlled manufacturing system. It can produce multiple types of parts using shared resources such as robots, machines, etc. Parts are manufactured according to a defined sequence of operations. Some operations of parts may be processed by more than one machine (routing flexibility), and some resources may be used to process more than one part type (resource flexibility). When an operation of a part is completed, the part may be moved to the next machine through material handling systems or transferred to an intermediate buffer.(Lei et al., 2017)

Scheduling: Scheduling is an important element of production systems because it serves as an overall plan on which many other shop activities are based. There are two key elements in any scheduling system: schedule generation and revisions (monitoring and updating the schedule). The first element which acts as a predictive mechanism determines planned start and completion times of operations of the jobs. The second element which is viewed as the reactive part of the system monitors the execution of the schedule and copes with unexpected events (i.e., machine breakdowns, tool failures, order cancelation, due date changes, etc) (Sabuncuoglu and Bayız, 2000)

Decentralization: Decentralized metaheuristics have shown to be useful in practice to solve a large number of applications. Many real-life problems may need days or weeks of computing time to be solved on serial machines. This is the usual scenario when considering difficult problems such as complex combinatorial problems with large search spaces, multi-objective problems with many hard-to-evaluate objective functions, or dynamic optimization problems, such as the dynamic FMS scheduling problem (Alba et al., 2013).

Artificial intelligence: Artificial Intelligence (AI) is a general term that implies the use of a computer to model intelligent behavior with minimal human intervention. AI is generally accepted as having started with the invention of robots. The term derives from the Czech word robota, meaning biosynthetic machines used as forced labor. (Hamet and Tremblay, 2017)

Metaheuristics: Metaheuristic algorithms have provided efficient tools to engineering designers by which it became possible to determine the optimum solutions of engineering design optimization problems encountered in every day practice. Generally metaheuristics are based on metaphors that are taken from nature or some other processes.(Saka et al., 2016)

Multithreading: Multithreading allows users to take advantage of multicore systems without timposing the overhead of creating multiple processes and providing direct access to a common address space. However the creation and management of threads introduces a computational overhead that makes the use of threads suboptimal when the tasks parallelized have low complexity.(Eiras-Franco et al., 2016)

9. Table of annexes

Num Annex	Name	Development	Type of document	Short link (https://goo.gl)	Relevance of the document
1	FMSSP	OWN	JAVA	https://drive.google.com/open?id=1hGbE6XRiVK-W3gT8E1yCCd-UL6ThsRjj	5
2	Results	OWN	Excel	https://drive.google.com/open?id=1JLQxHU1lxFDY-4WpOEekus-1DIrDJmo_	5

References

A Parallel Genetic Algorithm for the Multilevel Unconstrained Lot-Sizing Problem, 2008. . Inf. J. Comput. 20, 124–132. doi:10.1287/ijoc.1070.0224

Alba, E., Luque, G., Nesmachnow, S., 2013a. Parallel metaheuristics: recent advances and new trends. Int. Trans. Oper. Res. 20, 1–48. doi:10.1111/j.1475-3995.2012.00862.x

Alba, E., Luque, G., Nesmachnow, S., 2013b. Parallel metaheuristics: recent advances and new trends. Int. Trans. Oper. Res. 20, 1–48. doi:10.1111/j.1475-3995.2012.00862.x

Almeida-Luz, S.M., Rodríguez-Hermoso, M.M., Vega-Rodríguez, M.A., Gómez-Pulido, J.A., Sánchez-Pérez, J.M., 2009. GRASP and grid computing to solve the location area problem, in: World Congress on Nature Biologically Inspired Computing, 2009. NaBIC 2009. Presented at the World Congress on Nature Biologically Inspired Computing, 2009. NaBIC 2009, pp. 164–169. doi:10.1109/NABIC.2009.5393648

Anderson, J., Tanner, B., Baltes, J., 2004. Dynamic coalition formation in robotic soccer, in: Proceedings of the AAAI-04 Workshop on Forming and Maintaining Coalitions and Teams in Adaptive Multiagent

Systems. pp. 1–10.
Arcos, J.L., Esteva, M., Noriega, P., Rodríguez-Aguilar, J.A., Sierra, C., 2005. Engineering open environments with electronic institutions. Eng. Appl. Artif. Intell. 18, 191–204. doi:https://doi.org/10.1016/j.engappai.2004.11.019
Baños, R., Gil, C., Paechter, B., Ortega, J., 2006. Parallelization of population-based multi-objective meta-heuristics: An empirical study. Appl. Math. Model. 30, 578–592. doi:10.1016/j.apm.2005.05.021
Basnet, C., Mize, J.H., 1994. Scheduling and control of flexible manufacturing systems: a critical review. Int. J. Comput. Integr. Manuf. 7, 340–355.
Bennane, A., 2010. Tutoring and Multi-Agent Systems: Modeling from Experiences. Inform. Educ. 9, 171.
Berger, T., Sallez, Y., Valli, B., Gibaud, A., Trentesaux, D., 2010. Semi-Heterarchical allocation and routing processes in FMS control: a stigmergic approach. J. Intell. Robot. Syst. 58, 17–45.
Bordini, R.H., Hübner, J.F., Wooldridge, M.J., 2007. Programming multi-agent systems in AgentSpeak using Jason, Wiley series in agent technology. Wiley, Chichester.
Botti, V., Giret, A., 2008. ANEMONA: A Multi-agent Methodology for Holonic Manufacturing Systems. Springer Science & Business Media.
Bożejko, W., Pempera, J., Smutnicki, C., 2013. Parallel tabu search algorithm for the hybrid flow shop problem. Comput. Ind. Eng. 65, 466–474. doi:10.1016/j.cie.2013.04.007
Bożejko, W., Rajba, P., Wodecki, M., 2014. Scheduling Problem with Uncertain Parameters in Just in Time System, in: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (Eds.), Artificial Intelligence and Soft Computing, Lecture Notes in Computer Science. Springer International Publishing, pp. 456–467.
Bożejko, W., Uchroński, M., Wodecki, M., 2010. Parallel hybrid metaheuristics for the flexible job shop problem. Comput. Ind. Eng. 59, 323–333. doi:10.1016/j.cie.2010.05.004
Brandimarte, P., 1993. Routing and scheduling in a flexible job shop by tabu search. Ann. Oper. Res. 41, 157–183.
Browne, J., Dubois, D., Rathmill, K., Sethi, S.P., Stecke, K.E., others, 1984. Classification of flexible manufacturing systems. FMS Mag. 2, 114–117.
Byun, J.H., Datta, K., Ravindran, A., Mukherjee, A., Joshi, B., 2009. Performance analysis of coarse-grained parallel genetic algorithms on the multi-core sun UltraSPARC T1, in: IEEE Southeastcon, 2009. SOUTHEASTCON ’09. Presented at the IEEE Southeastcon, 2009. SOUTHEASTCON ’09, pp. 301–306. doi:10.1109/SECON.2009.5174094
Cahon, S., Melab, N., Talbi, E.-G., 2004a. ParadisEO: A Framework for the Reusable Design of Parallel and Distributed Metaheuristics. J. Heuristics 10, 357–380. doi:10.1023/B:HEUR.0000026900.92269.ec
Cahon, S., Melab, N., Talbi, E.-G., 2004b. ParadisEO: A Framework for the Reusable Design of Parallel and Distributed Metaheuristics. J. Heuristics 10, 357–380. doi:10.1023/B:HEUR.0000026900.92269.ec
Caramia, M., Dell’Olmo, P., 2006. Effective resource management in manufacturing systems: optimization algorithms for production planning, Springer series in advanced manufacturing. Springer, London ; New York.
Caumond, A., Lacomme, P., Moukrim, A., Tchernev, N., 2009. An MILP for scheduling problems in an FMS with one vehicle. Eur. J. Oper. Res. 199, 706–722.
Chalmers, A., Mirmehdi, M., Muller, H., n.d. The “Uniform Heterogeneous Multi-threaded”Processor Architecture.
Chan, F.T.S., Chan, H.K., 2004. A comprehensive survey and future trend of simulation study on FMS scheduling. J. Intell. Manuf. 15, 87–102. doi:10.1023/B:JIMS.0000010077.27141.be
Chituc, C., Restivo, F., 2009. Challenges and Trends in Distributed Manufacturing Systems: Are wise engineering engineering systems the ultimate answer? Second Int. Symp. Eng. Syst. MIT Camb. Mass. 1–15.
Chiu, K.M., Liu, J.S., 2011. Robot routing using clustering-based parallel genetic algorithm with migration,

in: 2011 IEEE Workshop On Merging Fields Of Computational Intelligence And Sensor Technology. Presented at the 2011 IEEE Workshop On Merging Fields Of Computational Intelligence And Sensor Technology, pp. 42–49. doi:10.1109/MFCIST.2011.5949511

Christo, C., Cardeira, C., 2007. Trends in intelligent manufacturing systems. Presented at the IEEE International Symposium on Industrial Electronics, 2007. ISIE 2007., IEEE, pp. 3209–3214.

Colombo, A.W., Karnouskos, S., 2009. Towards the factory of the future: A service-oriented cross-layer infrastructure. ICT Shap. World Sci. View Eur. Telecommun. Stand. Inst. ETSI John Wiley Sons 65, 81.

Conway, R.W., Maxwell, W.L., Miller, L.W., 2012. Theory of scheduling. Courier Corporation.

Dai, M., Tang, D., Giret, A., Salido, M.A., Li, W.D., 2013. Energy-efficient scheduling for a flexible flow shop using an improved genetic-simulated annealing algorithm. Robot. Comput.-Integr. Manuf. 29, 418–429. doi:10.1016/j.rcim.2013.04.001

Defersha, F.M., Chen, M., 2010. A parallel genetic algorithm for a flexible job-shop scheduling problem with sequence dependent setups. Int. J. Adv. Manuf. Technol. 49, 263–279. doi:10.1007/s00170-009-2388-x

Eiras-Franco, C., Bolón-Canedo, V., Ramos, S., González-Domínguez, J., Alonso-Betanzos, A., Touriño, J., 2016. Multithreaded and Spark parallelization of feature selection filters. J. Comput. Sci. 17, 609–619. doi:10.1016/j.jocs.2016.07.002

ElMaraghy, H.A., 2006. Flexible and reconfigurable manufacturing systems paradigms. Int. J. Flex. Manuf. Syst. 17, 261–276. doi:10.1007/s10696-006-9028-7

Gimpel, H., Jennings, N.R., Kersten, G.E., Ockenfels, A., Weinhardt, C., 2008. Negotiation, Auctions, and Market Engineering: International Seminar, Dagstuhl Castle, Germany, November 12-17, 2006, Revised Selected Papers, Lecture Notes in Business Information Processing. Springer Berlin Heidelberg.

Hamet, P., Tremblay, J., 2017. Artificial intelligence in medicine. Metabolism 69, S36–S40. doi:10.1016/j.metabol.2017.01.011

Herrero-Perez, D., Martinez-Barbera, H., 2010. Modeling Distributed Transportation Systems Composed of Flexible Automated Guided Vehicles in Flexible Manufacturing Systems. IEEE Trans. Ind. Inform. 6, 166–180.

Homberger, J., Gehring, H., 2008. A Two-Level Parallel Genetic Algorithm for the Uncapacitated Warehouse #x0A0; Location Problem, in: Hawaii International Conference on System Sciences, Proceedings of the 41st Annual. Presented at the Hawaii International Conference on System Sciences, Proceedings of the 41st Annual, pp. 67–67. doi:10.1109/HICSS.2008.42

Horling, B., Lesser, V., 2004. A Survey of Multi-agent Organizational Paradigms. Knowl Eng Rev 19, 281–316. doi:10.1017/S0269888905000317

Jun Sawamoto, Tsuji, H., Tilwaldi, D., Koizumi, H., 2005. A Proposal of Multi-Agent System Development Framework for Cooperative Problem Solving and Its Experimental Evaluation. IEEE, pp. 47–52. doi:10.1109/AINA.2005.51

Kovacs, A.A., Parragh, S.N., Hartl, R.F., 2015. The multi-objective generalized consistent vehicle routing problem. Eur. J. Oper. Res. 247, 441–458.

Kuo, R.J., Cheng, W.C., 2013. Hybrid meta-heuristic algorithm for job shop scheduling with due date time window and release time. Int. J. Adv. Manuf. Technol. 67, 59–71. doi:10.1007/s00170-013-4753-z

Lei, H., Xing, K., Han, L., Gao, Z., 2017. Hybrid heuristic search approach for deadlock-free scheduling of flexible manufacturing systems using Petri nets. Appl. Soft Comput. 55, 413–423. doi:10.1016/j.asoc.2017.01.045

Leitão, P.J.P., 2004. An agile and adaptive holonic architecture for manufacturing control. University of Porto.

Leon, C., Miranda, G., Segura, C., 2009. A memetic algorithm and a parallel hyperheuristic island-based model for a 2D packing problem, in: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation. ACM, pp. 1371–1378.

Luh, P.B., 1998. Scheduling of flexible manufacturing systems, in: Siciliano, P.B., Valavanis, P.K.P. (Eds.), Control Problems in Robotics and Automation, Lecture Notes in Control and Information Sciences. Springer Berlin Heidelberg, pp. 227–243.

McFarlane, D.C., Bussmann, S., 2003. Holonic Manufacturing Control: Rationales, Developments and Open Issues, in: Agent Based Manufacturing: Advances in the Holonic Approach. pp. 303–326.

Mekni, S., Chaâr, B.F., 2014. An Enhanced PSO-based Algorithm for Multiobjective Flexible Job Shop Scheduling Problems(FJSSPs), in: Proceedings of the 6th International Conference on Management of Emergent Digital EcoSystems, MEDES '14. ACM, New York, NY, USA, p. 30:178–30:182. doi:10.1145/2668260.2668289

Nevmyvaka, Y., Sycara, K., Seppi, D., 2003. Electronic market making: initial investigation, in: The Proceedings of Third International Workshop on Computational Intelligence in Economics and Finance.

Nie, L., Gao, L., Li, P., Li, X., 2013. A GEP-based reactive scheduling policies constructing approach for dynamic flexible job shop scheduling problem with job release dates. J. Intell. Manuf. 24, 763–774.

Nielsen, I.M., Janssen, C.L., 2000. Multi-threading: A new dimension to massively parallel scientific computation. Comput. Phys. Commun. 128, 238–244.

Oliveira, E., Fischer, K., Stepankova, O., 1999. Multi-agent systems: which research for which applications. Robot. Auton. Syst. 27, 91–106. doi:http://dx.doi.org/10.1016/S0921-8890(98)00085-2

Ozbakir, L., Baykasoglu, A., Gorkemli, B., Gorkemli, L., 2011. Multiple-colony ant algorithm for parallel assembly line balancing problem. Appl. Soft Comput. 11, 3186–3198.

Sabuncuoglu, I., Bayız, M., 2000. Analysis of reactive scheduling problems in a job shop environment. Eur. J. Oper. Res. 126, 567–586.

Saka, M.P., Hasançebi, O., Geem, Z.W., 2016. Metaheuristics in structural optimization and discussions on harmony search algorithm. Swarm Evol. Comput. 28, 88–97. doi:10.1016/j.swevo.2016.01.005

Segura, C., Segredo, E., León, C., 2011. Parallel island-based multiobjectivised memetic algorithms for a 2D packing problem, in: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation. ACM, pp. 1611–1618.

Shirazi, B., Mahdavi, I., Mahdavi-Amiri, N., 2011. iCoSim-FMS: An intelligent co-simulator for the adaptive control of complex flexible manufacturing systems. Simul. Model. Pract. Theory 19, 1668–1688. doi:10.1016/j.simpat.2011.04.003

Shnits, B., Rubinovitz *, J., Sinreich, D., 2004. Multicriteria dynamic scheduling methodology for controlling a flexible manufacturing system. Int. J. Prod. Res. 42, 3457–3472. doi:10.1080/00207540410001699444

Souier, M., Sari, Z., Hassam, A., 2012. Real-time rescheduling metaheuristic algorithms applied to FMS with routing flexibility. Int. J. Adv. Manuf. Technol. 64, 145–164. doi:10.1007/s00170-012-4001-y

Steels, L., Brooks, R. (Eds.), 1995. The Artificial Life Route to Artificial Intelligence: Building Embodied, Situated Agents. L. Erlbaum Associates Inc.

Talbi, E.-G., 2009. Metaheuristics: from design to implementation. John Wiley & Sons.

Van Dyke Parunak, H., 1991. Characterizing the manufacturing scheduling problem. J. Manuf. Syst. 10, 241–259. doi:10.1016/0278-6125(91)90037-3

Vlassov, V., Ayani, R., 2000. Analytical modeling of multithreaded architectures. J. Syst. Archit. 46, 1205–1230. doi:https://doi.org/10.1016/S1383-7621(00)00021-7

Zambrano Rey, G., 2014. Reducing Myopic Behavior in FMS Control: A Semi-Heterarchical SimulationOptimization Approach. Université de Valenciennes et du Hainaut-Cambresis.

Zambrano Rey, G., Bekrar, A., Trentesaux, D., Zhou, B.-H., 2015. Solving the flexible job-shop just-in-time scheduling problem with quadratic earliness and tardiness costs. Int. J. Adv. Manuf. Technol. 1–21.

doi:10.1007/s00170-015-7347-0

Zhao, J., Liu, Q., Wang, W., Wei, Z., Shi, P., 2011. A parallel immune algorithm for traveling salesman problem and its application on cold rolling scheduling. Inf. Sci. 181, 1212–1223.