Scheduling of Flexible Job Shop Problem in Dynamic Environment

Mariana Bayão Horta Mesquita da Cunha
mariana.cunha@tecnico.ulisboa.pt
Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal
November, 2017

***Abstract*—Data collection is reaching unprecedented high levels and the ability to analyse it is turning industries into digital. With the right decisions supported by data, cost savings and plant efficiency have the potential to change the whole business structures. Within Industry 4.0, scheduling in dynamic environments is one of the major opportunities. Presently, real-time knowledge of the plant scheduling can be done as online activity, optimizing not only the expected process but adapting as well to real unforeseen events. In this thesis tools are provided to both plan in a static predicted environment and to react in case of disturbance.**

Two planning techniques were implemented: a deterministic mixed integer linear programming model, $MILP$, and a genetic algorithm with a variable neighbourhood search, $hGAJobS$. The proposed $MILP$ algorithm was unable to solve instances with 10 jobs, consisting of 10 operations each, and 10 machines. However, it was able to successfully solve less complex instances. The proposed genetic algorithm, $hGAJobs$, combines different initial population creation and offspring generation methods, with a coding system that only allows feasible solutions. Moreover, to this genetic algorithm a variable neighbourhood search is added. $hGAJobs$ was applied to various Flexible Job Shop problem performing well in comparison to other genetic algorithms in the literature.

To respond in a dynamic environment two ant inspired multi-agent algorithms are proposed. The first is an autonomous architecture, $AABS$. Because of the myopia associated with autonomous architectures, a novel approach is proposed in a mediator architecture, $MABS$.

***Index Terms*—Flexible job shop, Dynamic environment, Mixed integer linear programming, Genetic algorithm, Variable neighbourhood search, Multi-agent based systems.**

I. Introduction

Increasing data collection and storage together with new computational capabilities are leading to major changes in the industry. Just as steam engine powered factories in the XIX^{th} century, electrification enabled mass production in the XX^{th} century, and automation in the 70s were revolutionary in their time, now is the rise of digital industry known as Industry 4.0. Sensors, machines, parts and information systems will be connected forming cyberphysical systems. These systems will be able to analyse data, predict failures and adapt to changes. Hence processes will be much faster, flexible and efficient, producing high quality, customized and on-time goods at lower costs [1], [2].

Industry 4.0 will enable companies to cut on cost and save time on both services and production, through better planning and man-machine interfaces, better production control, raw material availability control, inventory levels, and energy consumption. With the right decision making in place customer and suppliers relations can be significantly improved.

It is within Industry 4.0 scope, and more specifically Pharma 4.0, that the scheduling and rescheduling problem arises. To improve plant productivity, and having data from demands and plant state, it is possible to make planning decision to optimize future outcome, corresponding to either minimize costs and/or maximize revenues. Furthermore, with a rescheduling strategy, not only the plan is optimized but it is updated according to the evolving current plant situation.

Determining the best schedule can be a trivial task or a very complex problem, depending on the environment, process constraints and the performance indicator [3]. Particularizing to the manufacturing scheduling problem one can find two main streams: the job shop problem, JSP; and the flexible job shop problem, FJSP. The former represents one of the most difficult problems, in which a set of operations, each one requiring a single machine, which can process uninterruptedly one operation at a time and the machines are continuously available, being a NP-Hard problem. The latter extends the former, in the way that operations are allowed to be processed by a machine on a set of available ones. The main goal of the JSP is to sequence the operations to maximize the performance measured by the chosen indicator. The FJSP introduces a new decision dimension, since it demands not only the sequence of operations but also the machine allocation, also referred as the job routes. Consequently, the JSP space of solutions is a subspace of the FJSP space of solutions, $S^{JSP} \subset S^{FJSP}$. The work on [4] demonstrates the NP-hardness and combinatorial characteristics of FJSP.

Manufacturing systems plants usually have an *a priori* plan for when activities should take place according to a predetermined objective. Nonetheless, real world manufacturing systems are dynamic systems with uncertainty which, typically, have two sources:

- **Resource Related -** machine breakdown, operator uncertain performance or illness, loading limits, unavailability or tool breakdown, materials shortage or arrival delay, not compliant materials, etc.
- **Jobs Related -** last minute high priority jobs, jobs cancellation, due dates alteration, unexpected jobs arrival, jobs processing time change, etc.

A dynamic environment prone to unpredictability and uncertainty, as the ones aforementioned, may result in a poor

manufacturing system performance of the previously scheduled activities. Inevitably, to reduce the deviation from reality a new type of scheduling techniques has been created, known as dynamic scheduling [5]. To account for real time events and to address its changes and minimize their effect on the shop floor, a great deal of effort has been spent on rescheduling, which is the process of updating an existing production scheduling when facing disruptive events [6]. According to this, in this work a predictive-reactive strategy is proposed. First a static schedule is generated followed by its dynamic updating in case of disruption.

The remaining of this paper is organized as follows. Section 2 presents a brief review of genetic algorithms, ant colony optimization and multi-agent based algorithms for FJS scheduling. Section 3 details the genetic algorithm implemented, $hGAJobs$. Section 4 presents the two multi-agent based systems applied for dynamic scheduling, $AABS$ and $MABS$. Results are presented and discussed in section 5. Section 6 concludes the paper and presents future research steps.

II. Literature Review

Most research in meta-heuristics applied to scheduling is focused on genetic algorithm (GA) and ant colony optimization (ACO) [7], [8]. In [9] GA was applied for scheduling and rescheduling. They concluded based on experimental results that while GA produces far better results than shortest processing time dispatching rule, the capabilities of GA vanish with an increasing problem size. To combat this GA deficiency, they limited the search using bounds, which may exclude optimal and even near optimal solutions. GA static integrated approaches for FJSP were introduced in [10], [11], [12], [13], [14]. [12] introduced a chromosome representation that integrates routing and sequencing, together with an approach by localization method to find relevant initial assignments, which are followed by dispatching rules. In [10] Chen split the chromosome representation into routing policy and sequence of operations on each machine. [11] presented an efficient GA, called GENACE, with incorporates cultural evolution, i.e. domain knowledge is passed to the next generation. When applying crossover and mutation operators uses information from previous generations of what kind of operations promoted good solutions. In [13] many of the choices of [12] were adopted. Nevertheless, different strategies for creating the initial population, selecting the individuals for reproduction and reproducing new individuals were created. The results suggest that introducing new strategies to the genetic framework leads to higher quality solutions. In [14] global and local selection were used to generate a higher quality combination of assignments and operations sequencing for the initial population, along with different strategies for crossovers and mutations.

In [15] a GA was presented with three objectives, minimize makespan, maximal machine workload and total workload. This work follows a hierarchical approach, using a two vectors representation, with advanced crossovers and mutations operators. The search ability was improved by a variable neighbourhood descent search, involving two local searches. The first tries to move one operation and the second tries to move two operations. It is suggested by the authors that the moving of two operations improves the algorithm performance.

The aforementioned GA applications were for static scheduling, yet GA has also been used for dynamic scheduling. In [16] GA was used for a manufacturing shop in the presence of machine breakdown and alternate job routine. When a dynamic event occurs the algorithm is used to provide a new schedule taking into account mean job tardiness and mean job cost as performance indicators, proving that GA significantly outperforms dispatching rules. [17] presented a GA that improves the solution given by dispatching rules, reducing its makespan, in the presence of events as the arrival of a new batch, unavailability of parts to manufacture and machine breakdowns. [18] compared the performance of genetic algorithms with local searches methods to generate robust schedules, demonstrating great improvements on the makespan and stability.

Traditionally scheduling systems for industrial environments were developed in a centralised, hierarchical perspective [19]. Centralised perspectives rely on central databases, usually giving a single computer the task to schedule, monitor, and dispatch corrective actions. Many drawbacks arise from this situation. First, the central computer becomes the bottleneck, limiting capacity of the shop, and moreover represents a failure point that can bring down the whole shop. Another is the flow of information, that by having concurrent messages, a single computer handling a lot of information may delay decision making [7]. Although centralised architectures may result in global better schedules when not facing many disruptions, they have been found to encounter great difficulties in real situations [20], [7].

To respond to market demands of high productivity and flexibility, agent-based approaches have shown great promise once they provide high fault tolerance, high efficiency and robustness, rapid response to real time systems, and coherent global performance by means of local decision-making [20], [21]. An agent-based system is made of autonomous agents that collaborate and cooperate, as a network, to satisfy local and global objectives [22], meaning that information flow and decision making is more efficient, and failure of one component will not halt the entire manufacturing system [21]. Performance of the whole system is highly dependant on coordination between agents. For that purpose multiple coordination mechanisms have been studied, such as Contract Net Protocol (CNP), Levelled Commitment Contracting Protocol (LCCP) and social insects coordination.

More recently multi-agent based systems research has been using insect inspired communication protocols [21]. Although individually insect based agent might be less-than-intelligent, collective intelligence can emerge from their interaction [23], [24], [25]. Two important social insect-inspired coordination mechanisms in multi-agent manufacturing system should be considered, Wasp-like agents [24] and Ant-like agents [26], [21].

III. Problem Formulation

The job shop problem is known to be a NP-hard problem. Being the FJSP not only a sequencing, as in the job shop problem, but also an assignment problem, it is at least NP-hard as well.

The FJSP can be stated as follows:

- A set of independent jobs $J = J_1, J_2, ..., J_n$;
- A set of machines $M = M_1, M_2, ..., M_m$;
- Each job is formed by a sequence of operations $O_{i_k} = O_{i_1}, O_{i_2}, ..., O_{i_h}$
- Each operation, O_{i_k}, can be processed in a subset group of machines such that $M_{ik} \subset M$. If $M_{ik} = M$ for all i and k, the problem becomes a complete flexible job shop problem;
- Operations need to be executed in order;
- A machine can only execute one operation at a time;
- The time it takes for operation O_{i_k} of job J_i to be performed in machine M_m is PT_{ikm}, without interruption. The processing time PT_{ikm} may be machine dependant or not.

Given the formulation, the problem can be summarized as in table I, where the rows correspond to the operations to be performed and the columns to machines. The entries are the correspondent processing times, being ∞ when an operation cannot be preformed on that machine.

TABLE I
Processing time of each operation O_i in each machine M

	M_1	M_2	M_3
O_{11}	5	4	3
O_{12}	7	5	∞
O_{13}	10	5	11
O_{21}	4	3	7
O_{22}	8	10	1
O_{31}	5	4	6
O_{32}	∞	7	2

IV. Mixed Integer Linear Programming

The FJSP can be stated and solved as a Mixed integer linear programming problem. In tables II the parameters used in the $MILP$ formulation and in table III the variables are introduced.

TABLE II
Parameters used in MILP formulation

J	Set of jobs $J = J_1, J_2, ..., J_n$
O_i	Set of operations $O_i = O_{i_1}, O_{i_2}, ..., O_{i_h}$
M	Set of machines $M = M_1, M_2, ..., M_m$
$PT(j,i,k)$	Processing time of operation i of job j in machine k
$a(j,i,k)$	$\begin{cases} 1, \text{if operation} i \text{ of job} j \text{can be in machine} k \\ 0, \text{otherwise} \end{cases}$
BN	Big number

The objective function is to minimize the makespan, equation (1). In order to define makespan in a linear formulation, a constraint needs to be enforced, so that C_{max} is at least equal to the last finished operation, equation (2). There are three classes of constraints to be enforced in this MILP formulation for the FSJP: precedence constraints, allocation constraints and link constraints. Precedence constraints ensure that the sequence of operations is not violated, equations (3) and (4). Allocation constraints need to be enforced to guarantee that a given operation is only allocated to one machine, equation (6); that at the same time only one operation is being allocated to a given machine, equation (5); and that operations are only being given to machines that can perform them, equation (7). Finally, it is important to force the starting time of an operation to be the same as the starting time of the machine that will preform it, equations (8) and (9).

TABLE III
Variables used in MILP formulation

$t(j,i)$	Starting time of operation i of job j
$Tm(k,l)$	Starting time of operation done at priority l in machine k
$X(k,j,i,l)$	$\begin{cases} 1, \text{if op.} i \text{ of job} j \text{is done in mach.} k \text{at priority} l \\ 0, \text{otherwise} \end{cases}$
C_{max}	Maximum makespan

$$\text{Minimize} C_{max} \tag{1}$$

Subject to:

$$C_{max} \geq t(j,i) + \sum_{k,l} PT(j,i,k)X(k,j,i,l) \tag{2}$$

$$t(j,i) + \sum_{k,l} PT(j,i,k)X(k,j,i,l) \leq t(j,i+1) \tag{3}$$

$$Tm(k,l) + \sum_{j,i} PT(j,i,k)X(k,j,i,l) \leq Tm(k,l+1) \tag{4}$$

$$\sum_{j,i} X(k,j,i,l) \leq 1 \tag{5}$$

$$\sum_{k,l} X(k,j,i,l) = 1 \tag{6}$$

$$\sum_{l} X(k,j,i,l) \leq a(j,i,k) \tag{7}$$

$$Tm(k,l) \leq t(j,i) + (1 - X(k,j,i,l)) \times BN \tag{8}$$

$$Tm(k,l) + (1 - X(k,j,i,l)) \times BN \geq t(j,i) \tag{9}$$

V. Genetic Algorithm

Here, will be presented a GA for planning purposes. It uses a hierarchical approach, along with a coding that guarantees feasible solutions. Different strategies were explored and combined to create the initial population, crossovers and mutations. The solution quality is increased through local search, using a variable neighbourhood search algorithm.

A GA starts from an initial population of solutions. Then the whole population is scored against the objective function, called fitness function. The individuals, based on their scoring, will be selected to either (1) stay elites, maintaining their chromosomes the same, (2) be parents for crossover, implying a mix of both parents to create an offspring, or (3) mutate, creating the new generation. The best solution in each generation is saved. The optimization stops when the stopping criterion is met, for instance, when the maximum number of generations is reached or no improvement of the best solution has been found for a certain number of generations.

The implemented genetic algorithm was based in [13]. Changes to the algorithm described in [13] were implemented in the coding, introducing a chromosome sequence configuration that makes every combination feasible. Latter on, a local search based on [15] work implemented. On the following sections the GA formulation is described, starting from coding, then initial population creation, the mutation operators and, finally, crossover operators. The fitness function is makespan, and the selection criterion is a roulette wheel selection.

A. Coding

As explained in the previous section, GA uses population chromosomes to evaluate the fitness function, in this case to score individuals based on makespan.

So that the sequence feasibility would not be an issue, a different coding system was used, based in [27]. The used coding representation is often referred as $operation - based$ representation, where the first half of the chromosome corresponds to the assignment of jobs to machines and the second to the sequencing.

The assignment part of the chromosome is done in a vector form, where the vector entry (i, k) corresponds to the machine m preforming operation k of job i. In table IV is an example of the assignment part of the chromosome, where operation O_{11} is assigned to machine 3 and operation O_{12} to machine 2.

TABLE IV
ASSIGNMENT HALF OF THE CHROMOSOME

O_{11}	O_{12}	O_{21}	O_{22}	O_{31}
3	**2**	**2**	**3**	**1**

The chromosome sequencing part is the second half of the chromosome and is based on the idea that the first operation will start as soon as possible, i.e. if all machines are available at time $t = 0$, then the first operation will start at that time. With this assumption, the time variable is taken out of the optimization, sequence becoming the only concern. The sequencing section has the same length as the assignment one. The entries are the job to be sequenced there, meaning that the first time entry 1 appears, it corresponds to operation O_{11}, and the second time to O_{12}. By doing this all sequence combinations will be feasible. An example of the sequencing chromosome section can be seen in table V.

The resulting chromosome is the combination of table IV and table V.

TABLE V
SEQUENCING HALF OF THE CHROMOSOME

1	**2**	**3**	**1**	**2**

B. Initial Population

The initial population is created in two stages. First, the assignment part of the chromosome is created; and then the sequencing part.

For the first stage, one of two methods are used, all based on a combination of both processing time and machine workload. The first method - smallest processing time method - follows the process described below:

1) Find in the processing time table the smallest processing time and fix that assignment;
2) Add to the rest of the operations in that column the processing time corresponding to the operation fixed;
3) Return to point 1 excluding the already fixed rows, until all operations are assigned.

The second method for assignment, the reordering method, is similar to the first, but this time first a shuffling of the table rows is done, maintaining the operation order, and then a search for the minimum is goes row by row starting at the beginning of the table. The steps are as follows:

1) Shuffling of job order in the processing time table;
2) By row find the minimum of processing time and fix that assignment;
3) Add to the rest of the operations in that column the processing time corresponding to the operation fixed.

While the previous assigning method has a small amount of different outcomes, this second assignment method is highly dependant on the randomly chosen job order.

After the assignment is done, the sequencing can be done with one of the following three rules:

1) Randomly select a job;
2) Job with most work yet to be done (sum of processing time in the assigned machines);
3) Job with most number of operations yet to be done.

C. Mutation Operators

After the selection of parents for mutation is completed, one of three different mutation operators can be applied:

1) Precedence Preserving Shift mutation (PPS) [28], [13];
2) Assignment mutation;
3) Intelligent mutation [13].

PPS mutation applies to the sequence part of the chromosome. This operator randomly selects a one operation and moves it to another position in chromosome. It is important to mention that, because of the way the coding is done, there is no concerns about the feasibility of the new solution.

Assignment mutation chooses a single operation assignment randomly and changes the assignment to another randomly chosen machine. If the operation chosen cannot be preformed on any other machine the process is restarted until either an operation can be done on another machine or there are no more operations to investigate.

Finally intelligent mutation consists on selecting an operation from the machine with maximum workload and assigning it to the one with minimum workload. If it is not compatible no mutation occurs.

D. Crossover Operators

The crossover operators uses two parents to generate an offspring. In this work two types of crossover operator were used. One to preform crossover in the assignment section of the chromosome, assignment crossover, and another in the sequential section, Precedence Preserving Order-based crossover (POX) [28].

The assignment crossover mixes the assignment from both parents. At first, a set of randomly selected operations are copied from parent one to the offspring. The remaining operations assignment is copied from the second parent. The sequencing is maintained from the first parent to the offspring.

POX operator selects form the first parent a job to copy the sequence. The selected job sequence is copied to the offspring, and the rest of the jobs are taken from the second parent in the order in which they appear. In this operator the assignment is maintained from the first parent to the offspring.

E. Variable Neighbourhood Search

The evolution convergence speed of standard GAs is extensively reported to be slow. A promising solution for its improvement is the use of local search algorithms along with GA. A hybridized GA with local search, envisions the introduction of a local optimizer that is applied to every children before its insertion into the population. Hence the space of solutions global exploration is in charge of GA, while the local search explores the chromosome features and may introduce problem specific knowledge in the form of rules or conditions to increase the solution quality and speed of convergence. To employ local search a Variable Neighbourhood Search (VNS) was introduced. VNS aims at exploring the neighbourhood space of solutions by trying to move an operation, randomly chosen, from the bottleneck path to a feasible slot in a neighbour. An important characteristic of VNS is that the solution quality never gets degraded, by only moving from one solution to another if the recently obtained is better.

The VNS is applied after the crossover and mutation operators first by moving one operation and then by moving two operations simultaneously. Both this operation moves are detailed in the remaining of this section.

a) Moving operations*:* A solution graph makespan is defined by the length of the critical paths, which is equivalent to say that the makespan cannot be reduced if the critical paths are the same. The local search goal is to successively identify and break the current critical paths to obtain a new schedule with smaller makespan.

The concept of critical operation, in simple words, is that if one removes an operation r and reallocates in another feasible position, and the original critical path machine latest completion time does not increase, then r is not a critical operation.

If an operation r from the critical path, i.e. an operation with a total slack of zero, is taken from a machine and inserted behind an operation with a total slack of at least the same as the processing time of r, this second solution has at the most the same makespan as the original. When moving an operation it is important to consider a few rules, taking r as the moved operation and v as the candidate operation for r to be assigned before:

1) The total slack of v needs to be at least equal to the processing time of r in the candidate machine, equation 10;
2) The latest possible starting time of operation v needs to be at least equal to the maximum between the earliest completion time of the job predecessor of v and r plus the processing time of r in the candidate machine, equation 11;
3) The latest starting time of the job successor of r needs to be at least equal to the maximum between the earliest completion time of the job predecessor of v and r plus the processing time of r in the candidate machine, equation 12.

Where TS corresponds to total slack, s^E to the earliest starting time, s^L to the latest starting time, c^E to the earliest completion time and PT to the processing time.

$$TS(v) \geq PT(r) \tag{10}$$

$$s^L(v) \geq \max\{c^E(r-1), c^E(v-1)\} + PT(r) \tag{11}$$

$$s^L(r+1) \geq \max\{c^E(r-1), c^E(v-1)\} + PT(r) \tag{12}$$

b) Moving two operations*:* Moving two operations was proven to be always better than moving one [29]. It consists of the trying to move two consecutive operations, and if one does not find an appropriate position, then a new combination operations is chosen.

VI. Dynamic Scheduling

While static scheduling techniques can guarantee very good results, in real life plants events are dynamic, generating gaps between predicted schedule outcome and the actual shop state. Dynamic scheduling techniques are not upset by disturbances once the schedule can be done quickly at almost real time. The drawback of many of this methods, however, is the lack of horizon driven from the local decision making, generating schedule solutions far from optimal.

For the dynamic scheduling part of this paper a multi-agent based system method will be presented, using ant based communication between agents. To combat the stated myopia associated with this type of systems, a prediction horizon was added and the agent based architecture changed from an autonomous to a mediator architecture. Both methods, the autonomous, $AABS$, and the mediator architecture, $MABS$, are described in the throughout this chapter.

A. Autonomous Ant-based System

A multi-agent based system inspired in ant behaviour was implemented based on [21] approach. In a this agent architecture, multiple agents are responsible for different parts of the system. The whole system success is highly dependent on their ability to communicate and coordinate tasks. All agents are in the same hierarchy level and there is no agent evaluating the global plant performance. This type of organization is described as an autonomous architecture.

When orders arrive interaction between job agents and machine agents starts. First, machine agents make a proposal to job agents based on their status, queue and processing time of that combination of operation and machine. Job agents evaluate machine proposal and determine which machine they should follow to. When the machine is determined, each job agent makes a proposal to machine based on due date, total processing time and processing time on that machine. Based on those proposals the machine agent decides which job in the queue will be processed next.

Agent Coordination*:* Both job agents and machine agents make decision, so both have associated pheromones, τ_{Mach} and τ_{Job}. Machine pheromones are related to how fast a job processing can be started on that machine, taking in consideration the rest of the jobs in line. This means that the longer the machine queue, the lower the pheromone intensity associated to the machine. However, the job associated pheromones are related to how much time it has left before due date subtracting the work yet to be done. Hence, even if a job has a due date very close, another job with more time until due date will have a higher pheromone value if it has more work left to be done.

The multi-agent based method presented works in two stages. First, it starts by assigning jobs to operations. Second, it sequences jobs on each machine.

A job is assigned to the machine with the highest bidding. The bidding of a machine for a job is given by equation 15. In this bidding, pheromone function τ_{Mach} and heuristic function η are weighted by parameter α and β, respectively. The machine pheromone intensity is given by equation 13. A machine has three different status:

1) *Idle*: There is no job assigned to that machine. Then the pheromone value will be 1
2) *Busy*: The pheromone value is as low as the sum of processing times of the jobs in queue, assigned to that machine. It is given by equation 13, where x_{Mach_i} is 1, because the machine is available, and q^i_{kl} is 1 if operation l of job k is in queue in machine i, and 0 otherwise. PT^i_{kl} corresponds to the processing time of operation l of job k in machine i.
3) *Down*: If a machine is in maintenance, its pheromone value is 0. Taking equation 13, it corresponds to setting parameter x_{Mach_i} to 0.

The heuristic function associated to each machine i is given by the inverse of the processing time of operation l of job k in machine i, 14. Parameters α and β are tunable for changing the relative weight of pheromones and heuristic when making an assignment decision.

$$\tau_{Mach_i}(t) = \frac{x_{Mach_i}}{\sum_k \sum_l q^i_{kl} PT^i_{kl}} \tag{13}$$

$$\eta^i_{kl} = \frac{1}{PT^i_{kl}} \tag{14}$$

$$p(MA_i) = \frac{(\tau_{Mach_i})^\alpha (\eta^i_{kl})^\beta}{\sum_j (\tau_{Mach_j})^\alpha (\eta^j_{kl})^\beta} \tag{15}$$

The second stage of the algorithm, the sequencing phase is through a competition of biddings between the jobs assigned to the same machine. Meaning that if there is more than one job assigned to the same machine, the machine agent needs to decide which job will be processed next. Hence, job agents will bid for their turn in the machine. The bidding function, equation 17, also works as well with a weighted combination of pheromones and heuristic function.

The heuristic function is the same as the one used for the machine bidding, 14, calculated through the inverse of the processing time of the job's operation in the associated machine.

The pheromone value is a function of how close their due date is and how much time they still need to be complete. It is given by equation 16, where term f_l is associated to operation l of job k and takes value 1 if that operation is not completed or 0 if otherwise. Once the pheromone function behaves as the known function $f(x) = \exp(-x)$, as time goes by and the job is approaching its deadline, if there is still a lot left to be down, the pheromone value will increase exponential.

$$\tau_{Job_k}(t) = \exp - \left(Due_{Job_k} - \left(t + \sum_l f_l PT_{kl} \right) \right) \tag{16}$$

$$p(Job_i) = \frac{(\tau_{Job_i})^\alpha (\eta^i_{il})^\beta}{\sum_k (\tau_{Job_k})^\alpha (\eta^k_{kl})^\beta} \tag{17}$$

B. Mediator Ant-based System

Although autonomous architectures have shown very good results, they lack predictability and optimality in complex problems. In order to accommodate global knowledge and maintain reactivity, mediator architectures are used. In this type of architectures decisions are still made locally by the agents negotiating with each other. However, in a mediator architecture, there is a mediator agent, higher in hierarchy, able to advice, impose or update decisions according to the global target.

The mediator architecture was applied to the previously described method, by adding to it the responsibility to approve the sequence decision based on a look-ahead point state prediction. To achieve that, changes to the previous method were done:

1) There are agents for both current job operation and agents for following job operation
2) The process of machine assignment and sequencing is done on both types of job agents separately.

3) If there is both a current job agent and a following job agent bidding for the same machine and sequenced as next, the mediator acts to solve the situation.

For the mediator to solve the situation, a pheromone contest between the current job agent and the following job agent is done. In a horizon corresponding to the current job agent processing time, pheromones are calculated and compared for both agents. Then, if in all time intervals the current job agent had a higher pheromone value, it is the current job agent sequenced for that machine. On the other hand, if in any of those time intervals the next job agent shows a higher pheromone intensity value, the machine waits for the operation corresponding to the following job agent to be assigned by a current job agent.

VII. Results

In this section the genetic algorithm, $hGAJobs$, is compared to known FJSP benchmarks. Then the $AABS$ algorithm is tested and compared to other agent-based algorithms. Finally, the two dynamic multi-ant-like architectures, $AABS$ and $MABS$, are compared in a dynamic environment. Results for the $MILP$ algorithm are not presented in this section since it could not solve instances as complex as the ones presented for $hGAJobs$. However in the thesis that originated this paper, results for the $MILP$ algorithm are presented.

A. Genetic Algorithm

The results obtained for the static scheduling using the genetic algorithm are presented here. The developed genetic algorithm was tested using Brandimarte 10 problem instances with medium flexibility [30]. The parameters of the instances are in table VI. It is possible to drive from table VI that the instances complexity increases both in number of jobs as in number of machines.

TABLE VI
Brandimarte dataset instances characteristics. Lower and upper bounds correspond to makespan.

Instance	No. Jobs	No. Machines	LB	UB
Mk01	10	6	36	39
Mk02	10	6	24	26
Mk03	15	8	204	204
Mk04	15	8	48	60
Mk05	15	4	168	172
Mk06	10	15	33	58
Mk07	20	5	133	139
Mk08	20	10	523	523
Mk09	20	10	299	307
Mk10	20	15	165	197

Based in these flexible job shop problem, the genetic algorithm was tuned in terms of initial population, population size and crossover fraction. The parameters that resulted from this tuning are stated in table VII. The population size was adjusted to the instance complexity, varying from a range of 500 to 3000 individuals.

Results were compared with three alternative genetic algorithms:

TABLE VII
Genetic algorithm parameters

Parameter	Value
Initial Population	
Assigned with smallest processing time method	0.4
Assigned with reordering method	0.6
Sequenced with random job selection	0.6
Sequenced with MWR selection	0.2
Sequenced with NOR method	0.2
Crossover Fraction	0.9
Crossover using assignment crossover	0.5
Crossover using POX crossover	0.5
Mutation Fraction	0.1
Mutation using PPS mutation	0.2
Mutation using assignment mutation	0.2
Mutation using intelligent mutation	0.6
Elite Fraction	0.01

- **Chen:** In [10] proposed an innovative genetic algorithm for solving the flexible job shop problem. The coding introduced in this paper split the chromosome in two sections, the first being routing policy and the second the sequence of operations in each machine.
- **NGA:** A recent approach proposed by [31]. The authors introduced a GA with a new chromosome representation.
- **hGA:** In [15] the authors proposed a a genetic algorithm with a variable neighbourhood descent search after every iteration. Results show very good performance establishing new upper bounds.
- **hGAJobS:** Approach proposed by the author.

In table VIII a comparison of the results for the hybrid genetic algorithm is shown.

TABLE VIII
Makespan for the Brandimarte dataset (* represents best known Lower Bound).

Instance	Chen[10]	NGA[31]	hGA[15]	hGAJobS
Mk01	40	*37	40	40
Mk02	29	*26	*26	27
Mk03	*204	*204	*204	*204
Mk04	63	*60	*60	*60
Mk05	181	173	*172	173
Mk06	60	67	*58	64
Mk07	148	148	*139	141
Mk08	*523	*523	*523	*523
Mk09	308	*307	*307	312
Mk10	212	212	*197	211

We can see that $hGAJobS$ is a much more competitive algorithm than $GAJobS$. Although it did not preform as well as hGA, it obtained very close results in seven of the ten instances. It is very likely that with more time and maybe better tuned parameters, $hGAJobS$ would reach the optimum solutions in more instances. $hGAJobs$ has clearly a similar performance as NGA, although NGA outperforms $hGAJobs$ in instances $Mk01$, $Mk02$ and $Mk09$, $hGAJobs$ presents better results for instances $Mk06$, $Mk07$ and $Mk10$. In comparison to $Chen$, $hGAJobs$ shows consistently better or similar results, excepting for $Mk06$ and $Mk09$. Once again, in both cases it is likely that more testing and parameter

adjustment would lead to better results. To conclude it is also important to state the consistency of the proposed algorithm. It consistently presents best or second to best results, while other algorithms perform well on some instances but fall very behind on others.

B. Dynamic Algorithm

In this section we will provide results of both dynamic algorithms testing against two study cases: (1)**GM truck painting:** In [24] adapted a dynamic real problem from [32] consisting on a General Motors truck painting study-case. All painting jobs are single operations and setup times are considered; (2)**Flexible Plant:** This study case does not consider setup time, but takes in account due dates. This study-case is used to compare the two agent-based architectures.

1) GM truck Painting*:* The GM tuck painting problem was presented originally by [32]. In the presented case, trucks role out of an assembly line at a determined rate, and each truck has to be painted with a specific colour. There are more colours than painting booths. A painting booth is dedicated to a colour. Whenever the booth needs to change painting colour, a setup process consisting of flushing the old paint and refilling with the new paint starts. In [32] the setup time associated with reconfiguring the painting booth was not defined, therefore [24] problem data is going to be adopted.

The system does not have any prior knowledge of which colour will be assigned. The colours are determined by a uniform distribution with 50% of all trucks being assigned to one colour, and the remaining 50% requiring a colour, drawn at random, from the other 13 colours. There are seven painting booths, which can process at most one truck. The queue associated to each painting booth can have a maximum of three trucks.

A truck takes 3 minutes to be painted. The setup time associated with changing a painting colour was considered in two alternatives, (1) Not very Significant, setup time of 1 minute; (2) Very Significant, setup time of 10 minutes.

The results are presented in table IX for the case with not very significant setup time, $t_{setup} = 1min$, and in table X for the case with very significant setup time, $t_{setup} = 10min$.

It is possible to see in table IX that the developed algorithm, $AABS$, is very competitive. It shows better throughput and cycle time results than $R{-}Wasps{-}D$ and $M3$, however it has a very high number of setups done. The results are related to the algorithm local decision making. $AABS$ algorithm decides between jobs, in this problem, mainly through pheromone comparison. Pheromones in the case where setup penalty is 1 minute may not defer much, since the problem has 7 painting booths and in the penalization time span of 1 minute only one extra truck arrives. By not penalizing much the setup time, the $AABS$ algorithm is able to have a higher throughput as well as lower cycle time.

In the higher penalizing setup problem, table X, the $AABS$ outperforms the $R{-}Wasps{-}D$ in throughput but stays behind when compared to $M3$. Moreover, it's cycle time and number of setups are also much higher than the other two methods. Comparing $AABS$ with $R-Wasps$ and keeping in mind the objective of maximizing throughput, it can be said that $AABS$ preforms better once it's decisions while worst in terms of setups and cycle time increase the throughput. However, when comparing the results with $M3$ algorithm, it is clear $AABS$ falls behind. The main reason for this may be the fact that $M3$ adapts its parameters according to the shop floor state, while $AABS$ parameters are static and predefined. This parameter adjustment can prevent a painting booth from taking a setup needed job if the main buffer is not too full.

TABLE IX
Comparison of results for GM truck painting problem. Not very significant setup penalty, $t_{setup} = 1min$. Results for 100 simulations.

	R-Wasps-D [24]	M3 [25]	AABS
Throughput	994.03 ± 0.36	997	997.48 ± 0.50
Cycle Time	7.16 ± 0.10	12.2	3.50 ± 0.04
Number of setups	287.61 ± 2.15	171	497.52 ± 44.73

TABLE X
Comparison of results for GM truck painting problem. Very significant setup penalty, $t_{setup} = 10min$. Results for 100 simulations.

	R-Wasps-D [24]	M3 [25]	AABS
Throughput	972.22 ± 2.11	982	976.41 ± 5.17
Cycle Time	26.72 ± 1.08	23	57.81 ± 5.85
Number of setups	265.33 ± 6.08	197	392.01 ± 17.26

2) Flexible Plant*:* To compare both multi-agent based algorithms, the autonomous architecture $AABS$ and the mediator architecture $MABS$, a study-case involving parallel machines and multiple operations jobs was elaborated. Also, this study-case considers due dates, allowing to compare the approaches on a lateness objective as well.

A factory composed of sixteen machines is divided into five work-centres. Each of the work-centre consists of 4,2,5,3 and 2 machines respectively. There are three part types being produced, with arrival probability of 0.3, 0.5 and 0.2 respectively. The processing time of an operation is a gamma random variable with a shape parameter of 2. In table XI mean processing time and workcenter routing for each part and operation are presented.

The arrival rate is given by a an exponential distribution with a mean of 15 jobs/time unit. Once the sample time was fixed at 0.05 time units, the mean becomes $15/20$ jobs/time unit.

The due date for each job e calculated considering the job arrival time and the time it takes to produce it, i.e. the job due date will correspond to its arrival date plus its total processing time multiplied by a due date tightness factor, equation 18.

$$Due_{Job_k} = t_{arrival_k} + k \times \sum_{l} PT_{kl} \tag{18}$$

TABLE XI
PART TYPES ROUTING, MEAN PROCESSING TIME AND PROBABILITY OF ARRIVING

Part type	Part routing in work-centres				
	op1	op2	op3	op4	op5
1	3	1	2	5	
Processing Time	0.25	0.15	0.10	0.3	
2	4	1	3		
Processing Time	0.15	0.2	0.3		
3	2	5	1	4	3
Processing Time	0.15	0.10	0.35	0.2	0.2

Simulations were run for different due date tightness factors: (1) regular due date, $k = 10$; (2) tight due date, $k = 5$; (3) very tight due date, $k = 3$. The objective of this study-case was to, as due dates are getting tighter, evaluate the performance of both $AABS$ and $MABS$. Both algorithms objective was to minimize maximum lateness. Simulation were run until 1000 orders were completed.

Results for the $MABS$ and $AABS$ methods when varying due date tightness are presented in tables XII, XIII and XIV, maximum lateness, mean lateness and throughput per 8 time units.

It is possible to observe that as $AABS$ has always a better mean lateness but, in terms of maximum lateness it is clearly outperformed by $MABS$. The big advantage of $MABS$ in comparison to $AABS$ is that it can compromise locally in order to benefit globally, i.e. in the mediator in $MABS$, contrary to $AABS$, prefers to compromise the mean lateness to prevent orders from finishing after due date.

The throughput is always higher for the $AABS$ algorithm than for the $MABS$, but its advantage vanishes as due date tightness is increased. It is important to mention once again that the objective was to minimize tardiness not maximize throughput. However it is always better to have as much throughput as possible.

TABLE XII
COMPARISON OF RESULTS BETWEEN $AABS$ AND $MABS$ FOR REGULAR DUE DATE, $k = 10$

	AABS	MABS
Max Lateness	-0.2	-0.55
Mean Lateness	-7.29	-7.03
Throughput per 8 time units	109.11	108.67

TABLE XIII
COMPARISON OF RESULTS BETWEEN $AABS$ AND $MABS$ FOR TIGHT DUE DATE, $k = 5$

	AABS	MABS
Max Lateness	1.25	-0.45
Mean Lateness	-3.15	-2.89
Throughput per 8 time units	109.11	108.78

TABLE XIV
COMPARISON OF RESULTS BETWEEN $AABS$ AND $MABS$ FOR VERY TIGHT DUE DATE, $k = 3$

	AABS	MABS
Max Lateness	1.85	0.85
Mean Lateness	-1.39	-1.24
Throughput per 8 time units	108.89	108.78

VIII. CONCLUSIONS

This paper presents a set of tools capable of creating a schedule for dynamic environments. To achieve this objective static scheduling algorithms were implemented for a planning phase, and dynamic reactive methods were implemented and created for a dynamic environment state. For the planning phase two static algorithms were successfully implemented, a mixed integer linear programming algorithm, $MILP$, and a genetic algorithm, $GAJobS$, which was then extended with a local search, $hGAJobS$. To respond to unpredicted or disruptive events, multi-agent based systems coordinated with ant inspired functions were chosen. A first autonomous architecture was used, $AABS$, and then it was extended to an innovative mediated architecture, $MABS$.

The first planning algorithm implemented was the $MILP$ in $Matlab\ 17a$. It was tested in simple benchmark problems and, although it was able to solve the simpler ones even when shop flexibility was increased, it was unable to generate solutions in cases with more variables, i.e. when the number of operations and machines was increased it was not able to find a solution. The difficulty to solve very complex problems was expected due to the combinatorial nature of the FJSP.

Because of the difficulties encountered with $MILP$ a metaheuristic approach was taken by implementing a genetic algorithm. The implemented genetic algorithm, $hGAJobS$, uses a combination strategies to generate the initial population and create offspring. Furthermore a coding scheme that prevents infeasible solutions was used. To try to make it more competitive, a local search using a variable neighbourhood algorithm was added. By adding the local search, the solution quality improved resulting in an algorithm that was able to find the optimal solution in three of the ten instances and in three other instances did not reach the optimum by only one time unit. Moreover, $hGAJobS$ proved to be very consistent when comparing to other algorithm, achieving in all ten instances the best or second to best solutions. It is likely that with more parameter adjustment and simulation time, the $hGAJobS$ will be able to find the optimum in the ten tested instances.

The multi-agent based systems were both successfully implemented. The autonomous architecture was tested against a well known benchmark case, the GM truck painting, maximizing throughput. It was able to reach very good results, only falling behind in the case where very high setup penalties were employed. However, it showed very promising results, and it is likely that with dynamic parameter adjustment it will be able to highly improve its results. Finally, $MABS$ was tested in a against $AABS$ in a study-case whose objective was to minimize maximum lateness. $MABS$ proved to be a

less nervous algorithm and showed results that, although in some metrics were worse than $AABS$, where much better in light of its objective.

To conclude two static and two dynamic scheduling algorithms were implemented. All algorithms were successful but they all showed some limitations that are only relevant considering the application. For instance, for a very small stable plant that has very high processing times, a $MILP$ might be the right choice. On the other hand, if the plant is very dynamic with constant new order arrival and cancellations, the multi-agent approach might be indicated. And maybe for a high demand plant that has sporadic critical disruptive events, the combination of a genetic algorithm, such as $hGAJobS$, and a dynamic reacting algorithm, $MABS$, is likely to be the best solution.

A. Future Work

While the demonstrated results were good some improvements still need to be made to achieve higher quality solutions and reach a state where the proposed tools can be implemented successfully in a real dynamic system.

The genetic algorithm, $hGAJobs$, although it showed good quality solutions, they could be improved by, in variable neighbourhood search moving three operations at a time. However, the drawback might be that for very large problem, it might be too computationally expensive.

The Multi-agent Based systems are very promising algorithms once they can react very fast and with reasonable solutions, specially $MABS$. It would be very interesting to see how $MABS$ would behave with dynamic parameter adjustment based in the current plant state. Results would most certainly improve, yet the computational burden of the parameter function might take one of the big advantages of the algorithm, its rapid response.

Finally, exploring the combination of $hGAJobS$ and $MABS$ in a hybrid predictive-reactive scheduling algorithm.

REFERENCES

[1] M. Rüßmann, M. Lorenz, P. Gerbert, M. Waldner, J. Justus, P. Engel, and M. Harnisch, "Industry 4.0: The future of productivity and growth in manufacturing industries," *Boston Consulting Group*, vol. 9, 2015.

[2] J. Lee, B. Bagheri, and H.-A. Kao, "A cyber-physical systems architecture for industry 4.0-based manufacturing systems," *Manufacturing Letters*, vol. 3, pp. 18–23, 2015.

[3] M. Pinedo, *Planning and scheduling in manufacturing and services*. Springer, 2005, vol. 24.

[4] M. R. Garey, D. S. Johnson, and R. Sethi, "The complexity of flowshop and jobshop scheduling," *Mathematics of operations research*, vol. 1, no. 2, pp. 117–129, 1976.

[5] P. Fattahi and A. Fallahi, "Dynamic scheduling in flexible job shop systems by considering simultaneously efficiency and stability," *CIRP Journal of Manufacturing Science and Technology*, vol. 2, no. 2, pp. 114–123, 2010.

[6] G. E. Vieira, J. W. Herrmann, and E. Lin, "Rescheduling manufacturing systems: a framework of strategies, policies, and methods," *Journal of scheduling*, vol. 6, no. 1, pp. 39–62, 2003.

[7] D. Ouelhadj and S. Petrovic, "A survey of dynamic scheduling in manufacturing systems," *Journal of scheduling*, vol. 12, no. 4, pp. 417–431, 2009.

[8] B. Çaliş and S. Bulkan, "A research survey: review of ai solution strategies of job shop scheduling problem," *Journal of Intelligent Manufacturing*, vol. 26, no. 5, pp. 961–973, 2015.

[9] C. Bierwirth and D. C. Mattfeld, "Production scheduling and rescheduling with genetic algorithms," *Evolutionary computation*, vol. 7, no. 1, pp. 1–17, 1999.

[10] H. Chen, J. Ihlow, and C. Lehmann, "A genetic algorithm for flexible job-shop scheduling," in *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, vol. 2. IEEE, 1999, pp. 1120–1125.

[11] N. B. Ho and J. C. Tay, "Genace: An efficient cultural algorithm for solving the flexible job-shop problem," in *Evolutionary Computation, 2004. CEC2004. Congress on*, vol. 2. IEEE, 2004, pp. 1759–1766.

[12] I. Kacem, S. Hammadi, and P. Borne, "Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 32, no. 1, pp. 1–13, 2002.

[13] F. Pezzella, G. Morganti, and G. Ciaschetti, "A genetic algorithm for the flexible job-shop scheduling problem," *Computers & Operations Research*, vol. 35, no. 10, pp. 3202–3212, 2008.

[14] G. Zhang, L. Gao, and Y. Shi, "An effective genetic algorithm for the flexible job-shop scheduling problem," *Expert Systems with Applications*, vol. 38, no. 4, pp. 3563–3573, 2011.

[15] J. Gao, L. Sun, and M. Gen, "A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems," *Computers & Operations Research*, vol. 35, no. 9, pp. 2892–2907, 2008.

[16] G. Chryssolouris and V. Subramaniam, "Dynamic scheduling of manufacturing job shops using genetic algorithms," *Journal of Intelligent Manufacturing*, vol. 12, no. 3, pp. 281–293, 2001.

[17] A. Rossi and G. Dini, "Dynamic scheduling of fms using a real-time genetic algorithm," *International Journal of Production Research*, vol. 38, no. 1, pp. 1–20, 2000.

[18] S. D. Wu, R. H. Storer, and P.-C. Chang, "A rescheduling procedure for manufacturing systems under random disruptions," in *New directions for operations research in manufacturing*. Springer, 1992, pp. 292–306.

[19] W. Shen, D. H. Norrie, and J.-P. Barthès, *Multi-agent systems for concurrent intelligent design and manufacturing*. CRC press, 2003.

[20] W. Shen, "Distributed manufacturing scheduling using intelligent agents," *IEEE intelligent systems*, vol. 17, no. 1, pp. 88–94, 2002.

[21] W. Xiang and H. P. Lee, "Ant colony intelligence in multi-agent dynamic manufacturing scheduling," *Engineering Applications of Artificial Intelligence*, vol. 21, no. 1, pp. 73–85, 2008.

[22] N. R. Jennings, "On agent-based software engineering," *Artificial intelligence*, vol. 117, no. 2, pp. 277–296, 2000.

[23] V. A. Cicirello and S. F. Smith, "Insect societies and manufacturing," in *The IJCAI-01 Workshop on Artificial Intelligence and Manufacturing: New AI Paradigms for Manufacturing*, 2001, pp. 33–38.

[24] ——, "Wasp-like agents for distributed factory coordination," *Autonomous Agents and Multi-agent systems*, vol. 8, no. 3, pp. 237–266, 2004.

[25] L. Meyyappan, C. Saygin, and C. H. Dagli, "Real-time routing in flexible flow shops: a self-adaptive swarm-based control model," *International Journal of Production Research*, vol. 45, no. 21, pp. 5157–5172, 2007.

[26] V. A. Cicirello and S. F. Smith, "Ant colony control for autonomous decentralized shop floor routing," in *Autonomous Decentralized Systems, 2001. Proceedings. 5th International Symposium on*. IEEE, 2001, pp. 383–390.

[27] F. Werner, "Genetic algorithms for shop scheduling problems: a survey," *Preprint*, vol. 11, p. 31, 2011.

[28] K.-M. Lee, T. Yamakawa, and K.-M. Lee, "A genetic algorithm for general machine scheduling problems," in *Knowledge-Based Intelligent Electronic Systems, 1998. Proceedings KES'98. 1998 Second International Conference on*, vol. 2. IEEE, 1998, pp. 60–66.

[29] L. Gambardella and M. Mastrolilli, "Effective neighborhood functions for the flexible job shop problem," *Journal of scheduling*, vol. 3, no. 3, pp. 3–20, 1996.

[30] P. Brandimarte, "Routing and scheduling in a flexible job shop by tabu search," *Annals of Operations research*, vol. 41, no. 3, pp. 157–183, 1993.

[31] I. Driss, K. N. Mouss, and A. Laggoun, "A new genetic algorithm for flexible job-shop scheduling problems," *Journal of mechanical science and technology*, vol. 29, no. 3, p. 1273, 2015.

[32] R. Morley, "Painting trucks at general motors: The effectiveness of a complexity-based approach," *Embracing Complexity: Exploring the application of complex adaptive systems to business*, pp. 53–58, 1996.