Dynamic Scheduling using Artificial Bee Colony Algorithm

Inês Carolina Azevedo Ferreira
ines.azevedo.ferreira@tecnico.ulisboa.pt

Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal

March 2019

Abstract

An industrial revolution is happening and smart factories using Industry 4.0 based on collaborative systems represent the future of industrial networks. Industry 4.0 is based on flexibility, factory visibility, and optimized decision-making. Consequently, scheduling in a dynamic environment plays an important role and an increasing amount of data is produced every day which requires the right tools to be processed. Through the right processing, solutions which are adapted to the disruptions and dynamic events are created, implying costs and resources saving and an increase in efficiency. In a first instance, a scheduling technique using the Artificial Bee Colony algorithm is implemented for a static predicted environment. The proposed algorithm combines different initial populations and generation of new food source methods, including a moving operations technique and a local search method increasing the variable neighbourhood search that, as a result, improves the solution quality. The algorithm is validated and its performance is tested in a static environment for Flexible Job Shop Problem. The work focuses on developing tools to process the information on the factory through the development of good solutions when facing disruptions and dynamic events. Three real-time events are considered on the dynamic environment: jobs cancellation, operations cancellation and new jobs arrival. Two scenarios are studied, for each real-time event, the first situation considers the minimization of the disruption between the previous schedule and the new one. The second situation generates a completely new schedule after the occurrence. Summarizing, six heuristics are created to solve dynamic environment scenarios and their performances are tested using the Flexible Job Shop Problem and the benchmarks of two case studies. Finally, the main task of the work is the creation of datasets for dynamic testing.
Keywords: Dynamic environment; New Jobs Arrival; Operations Cancellation; Jobs Cancellation; Flexible Job Shop Rescheduling;;

1. Introduction

Factories and governments are launching the fourth industrial revolution called Industry 4.0, due to the dynamic nature of the manufacturing environments and the growing of the virtual world. Industrial production will be highly flexible in production volume and suppliers, and above all sustainable and efficient [10]. Smart factories using Industry 4.0 based on collaborative systems represent the future of industrial networks.

According to a PWC survey from 2013 [4], 50% of German enterprises plan their new industrial network and 20% are already involved in smart factories. A survey by American Society for Quality (ASQ), from 2014 [1], states that 82% of organizations that implemented smart manufacturing experienced increased efficiency, 49% experienced fewer product defects and 45% experienced increased customer satisfaction [8]. Hence, companies can highly benefit from the implementation of Industry 4.0 concepts.

Industry 4.0 represents a smart manufacturing network concept where machines and products interact with each other without human intervention. Supply chains in such networks have dynamic structures which evolve over time. In these settings, short-term supply chain scheduling is challenged by sources of uncertainty. Manufacturing environments are dynamic by nature and there are several events which can occur and change the system status affecting the performance, known as real-time events. Exchanging data and information between different parties in real time is the key element of smart factories; such data could represent production status, energy consumption behaviour, material movements, customer orders and feedback, suppliers data, etc. The next generation of smart factories, therefore, will have to be able to adapt, almost in real time, to the continuously changing market demands, technology options and regulations [2].

2. Flexible Job Shop Scheduling Problem

The Flexible Job Shop Scheduling Problem (FJSSP) is a generalization of the classical Job Shop

Scheduling Problem (JSSP). The JSSP follows the idea that a set of jobs ($J = \{1, 2, ..., m\}$) is processed by a set of machines ($M_k = \{1, 2, ..., m\}$). Every job consists of a finite set of operations and the processing of the operation has to be performed on a preassigned machine. The $i - th$ operation of job j, denoted by O_{ji}, is processed on machine $k \in M$ and the JSSP solves the assignment of jobs to the machines. The order of operations of each job is fixed, meaning the JSSP doesn't need to solve the operation sequence. The aim of the classical static n-by-m JSSP is to find a schedule for processing n jobs on m machines fulfilling an objective function. The FJSSP has one more condition than the JSSP, it is imposed job variability which creates the need for an operation sequence solution, besides the assignment machine solution.

The FJSSP follows some ideas, rules and assumptions. No machine is able to process more than one job at the same time and no job may be processed by more than one machine. Each job and each operation must be processed exactly one time. There is independence between machines and jobs. The sequence of machines a job visits is specified, having a linear precedence structure. If the precedent operation is still being processed, the remaining operations commenced until the processing is completed. The processing time of the O_{ji} operation using a specific machine takes $PT_{jik} > 0$ time unities and is known. Machines must always be available at the usage time zero.

The ideas, rules and assumptions can be formulated as follows. There are m machines defined as $M_k = \{1, 2, ..., m\}$. There are a group of n jobs independent of each other defined as $J = \{1, 2, ..., n\}$. Each job has a set of h operations defined as $O_{ji} = \{O_{j1}, O_{j2}, ..., O_{jh}\}$. For each operation O_{ji}, there is a set of machines capable of performing it. The set is denoted by $M_{ji} \subset M_k$. If $M_{ji} = M_k$ for all i and k, the problem becomes a complete flexible job shop problem. The processing time of operations O_{ji} on machine k is stated as PT_{jik}. The start time for every operation O_{ji} on machine k is presented as ST_{jik}. The finishing time of operation O_{ji} on machine k is presented as FT_{jik}.

This problem has also some constraints. The technique constraint describes that the operation must be processed after all precedent operations have been processed. The operations should not be overlapped and the machine will be available to other operations, if the previous operations are completed. The resource constraint demands that one machine can only handle exactly one operation at a time. There is also a precedence constraint for operations of the same job.

The objective of the FJSSP is to determine a feasible schedule minimizing the makespan that is the maximum completion time of the jobs. In other words, the total elapsed time between the beginning of the first task and the completion of the last task.

3. Static Scheduling

The implementation of the ABC algorithm developed to solve the Flexible Job Shop Scheduling Problem for a static environment is described and it was based on the work [11].

3.1. Artificial Bee Colony Algorithm

The ABC Algorithm is inspired by the intelligent foraging behaviour of a honeybee swarm. The model that leads to the emergence of the collective intelligence of honey bee swarms consists of three essential components: food sources, employed foragers and unemployed foragers.

3.2. ABC Procedure

$f(FS_i)$ is the objective value of the solution representing the selected food source FS_i, t_{FS} is the total number of food sources and p_i is the probability of selecting a food source.

1. Initialize parameters: algorithm termination criteria, defined in subsection 3.8, (ter_{max} and gen_{max}), local search termination criteria ($move_{max}$), number of the employed bees (n_{eb}), number of onlooker bees (n_{ob}) and number of scout bees (n_{sb}).
2. Initialize a population of food sources (FS_i).
3. Calculate the objective value $f(FS_i)$ of each food source FS_i and then determine the best food source g_{best}.
4. Employed bees phase
   (a) For every employed bee generate the new food source FS_i^1 from FS_i.
   (b) Onlooker bees phase
      i. Calculate the probability of selecting the food source FS_i^1 according to equation $p_i = \frac{[f(FS_i)]}{\sum_{j=1}^{t_{FS}}[f(FS_i)]}$.
      ii. Calculate the number of onlooker bees to be sent to the food source FS_i^1 according to $NE_i = p_i \times n_{ob}$.
      iii. For every onlooker bee generate N_i new food sources FS_i^2 from FS_i^1 using local search according to its termination criteria.
      iv. Choose the best from all the N_i food sources generated and set it as FS_i^{best}
      v. If $f(FS_i^{best}) \leq f(FS_i^1)$, then $FS_i^1 = FS_i^{best}$

vi. For all the employed bees, if $f(FS_i^{best}) \leq f(gbest)$, then $gbest = FS_i$

5. Scout bees phase

   (a) Initialize scout bees with random solutions and update g_{best}, if possible.

   (b) Determine the worst employed bees and replace them with the best scout bees, if they are better.

6. If the stopping criteria is met the output is g_{best} and its objective value; otherwise, go back to point 4.

3.3. Solution Representation

The solutions are a combination of two vectors: machine assignment and operation sequence. The first one codes the assignment of operations to the machines and the second one codes the processing sequence of operations for each job. This dual coding vector is a representation of a feasible solution.

3.4. Population Initialization

To guarantee an initial population with quality, diversity and capability of avoiding falling in a local optimal, a hybrid way to generate the food sources was utilized. The machine assignment initialization uses three rules: random rule, local minimum processing time rule and global minimum processing time rule. The operation sequence initialization: random rule, most work remaining rule (MWR) and most number of operations remaining rule (MOR).

3.5. Crossover Operators

To evolve the machine assignment vector two crossover operators, the two-point crossover and the uniform crossover, are applied. Also, a method of a crossover called the Precedence Preserving Order-Based Crossover (POX) is implemented to evolve the operation sequence vector.

3.6. Mutation for Machine Assignment

To enhance the exploration capability in the employed bee search phase, a mutation operator for the machine assignment is embedded in the ABC algorithm.

3.7. Local Search Based on Critical Path

The local search strategy based on the critical path is proposed and embedded in the searching framework to enhance the local intensification capability for the onlooker bees.

3.7.1 Critical Path Theory

The earliest starting time of the operation O_{ji} is denoted as ST_{ji}^E and the latest starting time is ST_{ji}^L. If the operation O_{ji} is processed on the machine k, then the operation processed previously is PM_{ji}^k and operation processed next is NM_{ji}^k. $PJ_{ji} = O_{j-1i}$ is the operation of the job i that precedes O_{ji}. $NJ_{ji} = O_{j+1i}$ is the operation of the job j that is next to O_{ji}. The starting time of the dummy starting node $ST_E(0) = 0$. If the node has no job predecessor, the earliest completion time is $c^E(PJ_{ji}) = 0$. If it has no machine predecessor, the earliest completion time is $c^E(PM_{ji}^k) = 0$. The latest completion time of the ending node is equal to the makespan of the schedule, $c^L(N+1) = c_M(G)$.

The earliest completion time of the operation is described by equation 1 and the latest completion time by equation 2.

$$c_{ji}^E = ST_{ji}^E + PT_{jik} \tag{1}$$

$$c_{ji}^L = ST_{ji}^L + PT_{jik} \tag{2}$$

The earliest starting time is calculated by equation 3 and latest starting time of each node by equation 4.

$$ST_{ji}^E = max\{c^E(PJ_{ji}), c^E(PM_{ji}^k)\} \tag{3}$$

$$ST_{ji}^L = min\{S^L(NJ_{ji}), S^L(NM_{ji}^k)\}) \tag{4}$$

The total slack of operation is the amount of time that an activity can be delayed or anticipated without increasing makespan. The total slack of each node is calculated using equation 5.

$$TS_{ji} = ST_{ji}^L - ST_{ji}^E \tag{5}$$

The makespan of a schedule is defined by the length of its critical path, implying that any delay in the start of the critical operation will delay the schedule. The idea behind the local search of the critical path is to analyze all critical operations to verify the possibility of scheduling them earlier.

3.7.2 Moving Operations

In order to simplify the notation, the operation to be moved O_{ji} is called r and the candidate operation O_{lk} to have O_{ji} assigned before is called v.

$$TS_v \geq PT_r \tag{6}$$

$$ST_v^L \geq max\{c_{r-1}^E, c_{v-1}^E\} + PT_r \tag{7}$$

$$ST_{r+1}^L \geq max\{c_{r-1}^E, c_{v-1}^E\} + PT_r \tag{8}$$

The above moving operations process is repeated until all the critical operations of the present food source are moved or until the termination criteria for the local search is met. If the food source being searched has no more critical operations the search is terminated, otherwise the procedure is applied a certain number of times $move_{max}$ to have

better improvement of the final schedule. To accept the new solution generated one of the following conditions is satisfied: the new solution has a smaller makespan or the new solution has the same makespan but has fewer critical paths.

3.8. Termination Criteria

There are two termination conditions set to terminate the algorithm: a number of trials ter_{max} to improve the solution and the pre-defined number of iterations gen_{max}.

3.9. Verification and Validation

To verify and validate the implementation of the ABC algorithm for FJSSP in static environment, the algorithm was tested on the Brandimarte dataset, found in [3], and compared to known algorithms. The software used to implement the ABC algorithm was *Matlab 15b*.

3.9.1 Parameters

To perform the initialization of the population, the determination of several parameters is needed. The parameters were tuned by evaluating the combination in the same simulated conditions. The values for gen_{max} and ter_{max} were tuned using a multiple coefficient of 0.5 in the expressions defining the parameters in the mentioned paper. When those parameters were set, the three bees were also tuned using the same idea. The parameter $move_{max}$ was a new parameter added to the work and followed the same procedure. Hence, the parameters are: $gen_{max} = 2 \times n \times m$, the integer of $ter_{max} = 1,5 \times n \times m$, $move_{max} = n \times m$, $n_{eb} = 3 \times n$, $n_{ob} = 11 \times n$ and $n_{sb} = 0,2 \times n$.

3.9.2 Initial Population

Each of the rules mentioned for the population initialization are implemented with a certain probability of occurrence and the probabilities are determined by tuning the initial population creation. The values tuned for the initial population creation are 0.6 using random rule for the machine assignment and 0.8 using random rule for the operation sequence. This set of parameters has the highest fitness standard deviation and a relatively low mean fitness. Therefore, the probabilities of the MWR rule and MOR rule are 0.1 each, and the probabilities of the local and global minimum are 0.2 each.

3.9.3 Crossover and Mutation Operators

Following the idea described in subsection 3.9.2, the value for the population creation is 0.5 using uniform crossover and, therefore, 0.5 using two-point crossover. The value for the population creation is 0.9 using the mutation operator.

3.9.4 Comparation Algorithms

The results are compared in table 1 and all the results were obtained after twenty runs selecting the best individual. The proposed algorithm is within the best performing algorithms and it produces good results when compared to PVNS and ABC. PVNS achieves four optimal solutions, ABC achieves six and the proposed ABC also achieves six optimal solutions and one is a new reached lower bound. Comparing hGAJobs, it was better in six instances and equal in other three. Comparing to LEGA, it was equal in one instances and better in seven. From the comparation with KBACO, the proposed ABC is better in six instances and equal in three. When comparing TSPCB, it performed better in five datasest, and equally good in four of them. The good performance of the proposed implementation is guaranteed by the combination of different initial populations, including a moving operations technique and a local search method increasing the neighbourhood search that, as a result, improves the solution quality. It is very likely that with more time and maybe better tuned parameters, the proposed implementation would reach the optimum solutions in more instances.

Valuable to note, a new best known lower bound was reached for the mk05 benchmark of the Brandimarte dataset in static environment. The lower bound for mk05 is 168 and the previously last known lower bound found was 172 in [9] and [11]. The new reached lower bound is 169, three units of time smaller than the previous one.

4. Dynamic Scheduling

In the industrial world, scheduling systems often operate under dynamic and stochastic circumstances and it is inevitable to encounter some disruptions which are inherently stochastic and non-optimal. Therefore, algorithms which guarantee quick and good solutions for the scheduling are strongly needed.

In this work, heuristics were made in order to be possible to solve dynamic scheduling cases, through rescheduling. Not only the adaptations created in this dissertation are able to solve unpredictable scenarios, but also reoptimize the solution presenting a very good one.

5. Early Job Cancellation (ABC-R1)

The first scenario being studied is rescheduling when a job is cancelled. The ABC-R1 has several mechanisms included to improve the solution as much as possible.

Table 1: The results of the Brandimarte instances when solved using different algorithms.

Instances	**hGAJobs**[5]	**LEGA**[6]	**PVNS**[13]	**KBACO**[12]	**TSPCB**[9]	**ABC**[11]	**Proposed**
Mk01	40	40	40	39	40	40	39
Mk02	27	29	**26**	29	**26**	**26**	**26**
Mk03	**204**	N/	**204**	**204**	**204**	**204**	**204**
Mk04	**60**	67	**60**	65	62	**60**	**60**
Mk05	173	176	173	173	172	172	**169***
Mk06	64	67	67	67	65	60	**58**
Mk07	141	147	144	144	140	**139**	140
Mk08	**523**	**523**	**523**	**523**	**523**	**523**	**523**
Mk09	312	320	**307**	311	310	**307**	308
Mk10	211	229	208	229	214	208	

The bold numbers show the best result known for the instance
The * symbol means the result is the new best known
N/ means the result is not available

5.1. Early Job Cancellation Procedure

1. Load the static scheduling;
2. Initialize parameters: algorithm termination criteria (ter_{max} and gen_{max}), local search termination criteria ($move_{max}$), the deleted job (J^{delete}), the number of the employed bees (n_{eb}), the number of onlooker bees (n_{ob}) and the number of scout bees (n_{sb});
3. Initialize a population of food sources (FS_i) from the loaded static scheduling. In the machine assignment vector and in the operation sequence vector delete the operations belonging to the deleted job;
4. Calculate the objective value $f(FS_i)$ of the food source FS_i and then establish the best food source g_{best};
5. Onlooker bees phase
   (a) For every onlooker bee N_i generate new food sources F^2_{Si} from F^1_{Si} using local search according to its termination criteria;
   (b) Choose the best from all the N_i food sources generated and set it as the best;
   (c) If the stopping criteria are met go to the next point; otherwise, go back to point 5;
6. Initialize a population of food sources (FS^3_i) from the loaded static scheduling. In the machine assignment vector and in the operation sequence vector delete the operations belonging to the deleted job. The process will be done in a cycle, first deleting the operations one at each time, and then trying to anticipate the sequenced operations belonging to the same machine the deleted operation belongs to;
7. Onlooker bees phase
   (a) For every onlooker bee generate N_i new food sources FS^2_i from FS^1_i using local search according to its termination criteria;
   (b) Choose the best from all the N_i food sources generated and set it as FS^{best}_i;
   (c) If $f(FS^{best}_i) \leq f(FS^1_i)$, then $FS^1_i = FS^{best}_i$;
   (d) For all the employed bees, if $f(FS^{best}_i) \leq f(gbest)$, then $gbest = FS_i$;
   (e) If the stopping criteria is met go to the employed bee phase; otherwise, go back to point 7;
8. Employed bees phase
   (a) For every employed bee generate the new food source FS^1_i from FS_i;
      i. Applying crossover operators to the machine assignment vector;
      ii. Applying crossover operators to the operation sequence vector;
      iii. Applying mutation operator to the machine assignment vector;
      iv. Local search for the critical path according to the termination criteria;
      v. If $f(FS^{best}_i) \leq f(FS^1_i)$, then $FS^1_i = FS^{best}_i$;
      vi. For all the employed bees, if $f(FS^{best}_i) \leq f(gbest)$, then $gbest = FS_i$;
9. Scout bees phase
   (a) Initialize scout bees with random solutions and update g_{best}, if possible;
   (b) Determine the worst employed bees and replace them with the best scout bees if those are better;

10. If the stopping criteria are met the output is g_{best} and its objective value; otherwise, go back to point 4.

5.2. Parameters

The values for gen_{max} and ter_{max} were tuned using a multiple coefficient of 0.5 in the expressions mentioned in subsection 3.9.1. When those two were set, the three bees were also tuned. The parameter $move_{max}$ was a new parameter added to the work and followed the same procedure. Hence, the chosen parameters are: $gen_{max} = n \times m$, the integer of $ter_{max} = 0,4 \times n \times m$, $move_{max} = n$, $n_{eb} = 3 \times n$, $n_{ob} = 2 \times n$ and $n_{sb} = n$. The remaining parameters for the initialization of the population, for the crossover operators and for the mutation operator, won't be changed from the description presented in section 3.9.1.

6. Late Job Cancellation (ABC-R2)

The study has a late job cancellation order at a time defined as the time of job cancellation. At the order arrival, the previous conditions are wanted and it is important to generate a solution similar to the previous schedule.

6.0.1 Late Cancellation Procedure

1. Load the static scheduling;
2. Initialize parameters: the arrival time of the cancellation order (time of job cancellation) and the job which was cancelled (J^{delete});
3. Initialize the population of food source (FS_i) from the loaded static scheduling;
4. Using the machine assignment vector and the operation sequence vector, calculate the search space containing all the possible positions to anticipate the operations, according to the *time delete job*;
5. Each one of the operations which have the possibility to be anticipated will be introduced in the best position possible of the search space, respecting to the precedence constraints;
6. The output is the g_{best}.

7. Early Operation Cancellation (ABC-R3)

The ABC-R3 is similar to the procedure described for ABC-R1. The main difference is that ABC-R1 implies the cancellation of all the operations belonging to the job and ABC-R3 implies the cancellation of part of the job operations.

7.1. Early Operation Cancellation Procedure

The main differences are in the first, second and fourth steps. In the first step, a parameter describing which operation will be deleted is additionally initialized, namely ($delete_{operation}$). In the second step, the procedure to initialize the population is similar but operations processed before the operation cancelled are kept on the machine assignment and on the operation sequence vectors. The last difference in the fourth step: the operations processed before the cancelled operation are kept on the vectors. The parameters set in the algorithm ABC-R3 are described in 5.2.

8. Late Operation Cancellation (ABC-R4)

The ABC-R4 is a solution created when it is important to generate a solution similar to the previous one. This scenario has a late operation cancellation order at a time defined as the time of operation cancellation. The precedence constraints imply the cancellation of certain operations from the schedule, not only the one which was cancelled. The main differences to ABC-R2 are in the first and second steps. In the first step, the parameter setting which operation will be deleted is additionally initialized, called (O_{jj}^{delete}), and the variable of time initialized is the time of operation cancellation. In the second step, the procedure to initialize the population is similar but operations processed before the operation cancelled are kept.

8.0.1 Late Operation Cancellation Procedure

1. Load the static scheduling;
2. Initialize parameters: the arrival time of the cancellation order (time of operation cancellation) and the operation which was cancelled (O_{ji}^{delete});
3. Initialize the population of food source (FS_i) from the loaded static scheduling deleting the operations from the machine assignment vector and the operation sequence vector;
4. Using the new machine assignment vector and the operation sequence vector, the search space containing all the possible positions to anticipate the operations will be calculated, according to the *time of operation cancellation*;
5. Each one of the operations which have the possibility to be anticipated will be introduced in the best position possible of the search space, with respect to the precedence constraint;
6. The output is g_{best}.

9. Early New Job Arrival (ABC-R5)

The unexpected arrival of a new job is the disruption studied. The ABC-R5 was created as a more reactive model and has several mechanisms included to improve the solution as much as possible. The parameters set are described in 5.2.

9.1. Early New Job Arrival Procedure

1. Load the static scheduling;
2. Initialize parameters: algorithm termination criteria (ter_{max} and gen_{max}), local search termination criteria ($move_{max}$), the number of the employed bees (n_{eb}), the number of onlooker bees (n_{ob}), the number of scout bees (n_{sb}) and the new job (J^{new});
3. Initialize a population of food sources (FS_i);
   (a) The machine assignment vector is created using the explanation of 9.1.1;
4. Calculate the objective value $f(FS_i)$ of the food source FS_i and then establish the best food source g_{best};
5. Onlooker bees phase
   (a) For every onlooker bee generate N_i new food sources FS_i^2 from FS_i^1 using local search according to its termination criteria;
   (b) Choose the best from all the N_i food sources generated and set it as FS_i^{best};
   (c) If $f(FS_i^{best}) \leq f(FS_i^1)$, then $FS_i^1 = FS_i^{best}$;
   (d) For all the employed bees, if $f(FS_i^{best}) \leq f(gbest)$, then $gbest = FS_i$;
   (e) If the local search stopping criteria is met go to the employed bee phase; otherwise, go back to point 5;
6. Employed bees phase
   (a) For every employed bee generate the new food source FS_i^1 from FS_i;
      i. Applying crossover operators to the machine assignment vector;
      ii. Applying crossover operators to the operation sequence vector;
      iii. Applying mutation operator to the machine assignment vector;
      iv. Using the local search for the critical path according to the termination criteria;
      v. If $f(FS_i^{best}) \leq f(FS_i^1)$, then $FS_i^1 = FS_i^{best}$;
      vi. For all the employed bees, if $f(FS_i^{best}) \leq f(gbest)$, then $gbest = FS_i$;
7. Scout bees phase
   (a) Initialize scout bees with random solutions and update g_{best}, if possible;
   (b) Determine the worst employed bees and replace them with the best scout bees if those are better;
8. If the stopping criteria are met the output is g_{best} and its objective value; otherwise, go back to point 6;

9.1.1 New Machine Assignment Vector

The machine assignment vector from static scheduling is called u_{old}. The machine assignment for the new job is done independently of the remaining jobs, meaning a vector containing the machine assignment information of only the new job is initialized as explained in subsection 3.4. This vector is called u_{new}. In the end both vectors are joined in one vector u', first is the u_{old} and only after the u_{new}, creating the machine assignment vector for the new situation of $n+1$ jobs.

10. Late New Job Arrival (ABC-R6)

The ABC-R6 simulates a new order arrival and the need to introduce it on the system having the lowest disruption possible. It is considered that the static scheduling was already produced until the arrival time of the order, making it impossible to introduce the new operations before the time new job appears.

10.0.1 Late New Job Arrival Procedure

1. Load the static scheduling;
2. Initialize parameters: the arrival time of the new job (*time new job appears*) and the new number of jobs (n_{new});
3. Initialize the population of food source (FS_i) from the loaded static scheduling;
4. Using the machine assignment vector and the operation sequence vector, calculate the search space containing all the possible positions to introduce the operations of the new job, according to the *time new job appears*;
5. Each one of the new operations will be introduced in the best position possible of the search space, respecting the precedence constraints;
6. The output is g_{best}.

11. Results and Discussion

The software used to implement ABC-R1 until ABC-R6 was *Matlab 15b*.

11.1. Benchmarks

11.1.1 Benchmark 1

The problem is formulated in the paper [7], there are six machines, 13 different jobs and for each job the number of the operations is $3, 2, 3, 4, 3, 3, 2, 3, 2, 3, 3, 3, 3$, respectively. There is a total of 37 operations. The job 11 is cancelled as a dynamic occurrence.

11.1.2 Benchmark 2

In this paper [14], the benchmark treats the arrival of three new jobs ($J^{new} = \{14, 15, 16\}$) and each job has 3, 2 and 3 operations, respectively.

11.2. Early Job Cancellation (ABC-R1)

To analyze the performance of ABC-R1 the static scheduling of mk01 was used. All the jobs, from 1 to 10, one at each time, were cancelled. The makespan of the original static scheduling is 39 and the makespan of the results for each job cancelled are $36, 34, 38, 39, 38, 38, 37, 36, 36, 33$, respectively. The makespan of the dynamic cases is always smaller than the original. The run time was less than 1 minute and a half, the maximum run time was $78,6$ s and the minimum was $7,3$ s. A good solution was achieved in a short period of time to solve the problem.

11.2.1 Comparing Benchmark 1 and ABC-R1

In the case study using benchmark 1, the makespan of the initial scheduling is 66 and the makespan after the cancellation of the job 11 is 62. The results of the ABC-R1 algorithm are in table 2. The maximum makespan obtained is 52, which is considerably smaller than the makespan obtained in study 1, and an important fact is that the worst solution obtained using the implemented algorithm is 11,29% better. The maximum improvement of the solution obtained using ABC-R1 is 19,4% and the mean improvement is 17,2%. It is possible to conclude that ABC-R1 is highly competitive compared to the solution proposed in the case study using benchmark 1 and it always obtains a substantial lower makespan.

Table 2: Results of the ABC-R1 algorithm to solve the case study 1, after six runs

Min Makespan	50
Max Makespan	52
Mean Makespan	51,3
Standard Deviation Makespan	0,8
Min Run Time	62,1
Max Run Time	88,1
Mean Run Time	78,1
Standard Deviation Run Time	8,9

11.3. Late Job Cancellation (ABC-R2)

To test the performance of ABC-R2 mk04 was used to proceed to the tests. For job 6 and job 7, one at each time was set as a late cancellation of the job. All the results were obtained after three runs, selecting the best individual. The makespan of the original static scheduling is 60 and the results of the ABC-R2 are 55, respectively. The makespan of the dynamic cases is always smaller than the original makespan of the static scheduling and the run time was $1,5$ seconds for both cases. A good solution was achieved in a short period of time to solve the problem.

11.3.1 Comparing Benchmark 1 and ABC-R2

Only one result for ABC-R2 is presented because the algorithm has no stochastic behaviour and, as a consequence, the results obtained are the same for each run. The original makespan of the scheduling in the case study using benchmark 1, before the cancellation of the job 11 at 8 units of time, was 66 and after the cancellation is 62. Using the ABC-R2 a makespan of 55 is obtain in $1,7$ seconds. The makespan obtained with ABC-R2 is 7 units of time smaller and it has an improvement of 11,29%. In fact, the developed solution is substantially better than the solution presented in benchmark 1.

11.4. Early Operation Cancellation (ABC-R3)

To evaluate the performance of ABC-R3 mk04 was used. All the operations, from 2 to 5, were set one at each time as the cancellation of the operation. All the results were obtained after three runs, selecting the best individual. The makespan of the static scheduling of mk04 is 60. The makespan obtained using ABC-R3 is $38, 54, 42$ and 38, respectively, and is always smaller than the original of the static scheduling. Another important note is that the maximum run time was $99,2$s and the minimum was $0,3$s. A good solution was achieved in a short period of time to solve the problem.

11.5. Late Operation Cancellation (ABC-R4)

To test the performance of the ABC-R4 solving a late cancellation of the operation mk01 was used. For job 7 and job 10, one at each time were set as the dynamic event. All the results were obtained after three runs, selecting the best individual. The makespan of the static scheduling of mk04 is 60.

The makespan of these dynamic events is always smaller, being 37 for O_{72} and 33 for $O_{10,2}$ and the run time was $1,5$ seconds. A good solution was achieved to solve the problem of a late order to cancel one operation.

11.6. Early New Job Arrival (ABC-R5)

To verify the performance of the algorithm to dynamic events, the static scheduling of mk04 was used. Three dynamic events were studied: the arrival of a new job at the unit of time of 0, at the unit of time of 7 and at the unit of time of 45. All the results were obtained after three runs, selecting the best individual. The makespan of the static scheduling of mk04 is 60. All the solutions presented have a makespan of 60, meaning that there isn't an increase in the makespan. In a time between $1,2$ and $1,4$ seconds, the result is obtained. This has great applicability in the factory scheduling because several scenarios of introducing new orders in the scheduling can be done with the goal to improve the manufacturing production.

11.6.1 Comparing Benchmark 2 and ABC-R5

In the case using benchmark 2, the makespan of the initial scheduling is 66 and the one after the three orders arrival is 78. The results of the ABC-R5 for three new jobs arrival are in table 3. It is possible to conclude that ABC-R5 is highly competitive, when compared to the results obtained using benchmark 2, because even the maximum makespan obtained of 71 using the ABC-R5 is smaller than the makespan of 78 obtained using benchmark 2. Other reason to be considered a highly competitive solution is the significative improvement of the solution obtained using the ABC-R5 and it is important to notice that the worst improvement was 9%, which still a relevant improvement. The maximum improvement of the solution obtained using ABC-R5 algorithm comparatively to the solution obtained in the benchmark 2 is 21,8% and the mean improvement is 15,1%.

Table 3: Results of the ABC-R5 algorithm to solve the case study 2 with three new jobs inserted, after six runs.

Min Makespan	61
Max Makespan	71
Mean Makespan	66,2
Standard Deviation Makespan	3,8
Min Run Time	171,1
Max Run Time	232,0
Mean Run Time	201,2
Standard Deviation Run Time	20,5

11.7. Late New Job Arrival (ABC-R6)

11.7.1 Comparing Benchmark 2 and ABC-R6

Only one result of the ABC-R6 algorithm is presented because the algorithm has no stochastic behaviour. The original makespan of the scheduling in the case study using benchmark 2, before the new orders arrive, was 66. The time new job appears is 8 units of time and after the arrival of the three orders the makespan became 78. Using the ABC-R6, for three orders arrival at 8 units of time, a makespan of 66 is obtained in 1 second. Comparing this result with the result obtained in benchmark 2, there was an improvement of 15,38% in the solution. The cases of just one new order and two new orders arrival were also studied and, in both cases, a makespan of 66 was obtained. In fact, the developed solution is substantially better than the case study solution using benchmark 2.

12. Conclusions

The main objective of this dissertation was to develop tools capable of creating a schedule solution in a dynamic environment. To achieve this objective, a static scheduling algorithm was implemented using an Artificial Bee Colony algorithm and dynamic reactive heuristics were created for different dynamic environment states. For the static solution, the Artificial Bee Colony algorithm was successfully implemented. As a response to unpredicted or disruptive events, such as jobs cancellation, operations cancellation, and new job arrivals, six heuristics were created and implemented to solve the dynamic problem. Therefore, the Artifical Bee Colony algorithm was extended to innovative solutions for the dynamic environment.

To accomplish the first objective of this thesis, the implementation for the static environment was the Artificial Bee Colony algorithm in *Matlab 15b*. The algorithm uses a combination of strategies to generate the initial population and uses a coding scheme that prevents infeasible solutions. Furthermore, a local search based on the critical path was executed and also a move operations scheme with the capability of moving several operations was used. After testing the algorithm in benchmark problems and comparing it to other published algorithms, the implemented solution was verified to be a good solution and achieves the optimal solution in six of the ten instances. Valuable to note, a new optimal solution for one of the instances was found, it is three units of time smaller than the last one known and only one more than the lower bound.

Notwithstanding, this thesis primary goal was to create the six adaptations of the ABC algorithm and also successfully implement them in *Matlab 15b*.

The solutions solve the disturbance caused by job cancellations, operation cancellations and new job arrivals in two situations: the need of minimizing the disruption between the previous schedule and the new one and the need of a total reformulation. The scenarios were tested against the static scheduling obtained using the benchmark problems in the static environment, to fulfil the objective of evaluating the performance of the adapted algorithms and create instances for dynamic testing. All the implementations achieve good solutions in a very short time. Additionally, the solution obtained using ABC-R1, ABC-R2, ABC-R5 and ABC-R6 were compared to the solution obtained using benchmarks belonging to case studies. All the implementations are able to find a solution in a very short period of time. The makespan was always smaller, while compared to the benchmarks, and it was always several units of time smaller. It is important to refer, that the worst improvement of a solution obtained using one of the adaptations was 9,0%, which is still a relevant improvement comparatively to the benchmarks results and all the results have an improvement. All in all, very promising solutions are shown for dynamic scheduling.
To conclude, one static and six dynamic scheduling algorithms were successfully implemented. For a non-reactive plant, the Artificial Bee Colony algorithm is the right choice. On the other hand, if the plant is very dynamic with constant new order arrivals and cancellations, the implementation of the dynamic adaptations might be indicated.

References

[1] American Society for Quality. Manufacturing outlook survey. Accessed august 28,2018., 2014.

[2] Américo Azevedo and António Almeida. Factory templates for digital factories framework. *Robotics and Computer-Integrated Manufacturing*, 27(4):755–771, 2011.

[3] Paolo Brandimarte. Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations research*, 41(3):157–183, 1993.

[4] Company - PWC. Deutschland hinkt bei industrie 4.0 hinterher smart factory. Accessed august 28,2018., 2013.

[5] Mariana Cunha. Scheduling of flexible job shop problem in dynamic environments. Master's thesis, Instituto Superior Tcnico, 2017.

[6] Nhu Binh Ho, Joc Cing Tay, and Edmund M-K Lai. An effective architecture for learning and evolving flexible job-shop schedules. *European Journal of Operational Research*, 179(2):316–333, 2007.

[7] Yang Honghong and Wu Zhiming. The application of adaptive genetic algorithms in fms dynamic rescheduling. *International Journal of Computer Integrated Manufacturing*, 16(6):382–397, 2003.

[8] Dmitry Ivanov, Alexandre Dolgui, Boris Sokolov, Frank Werner, and Marina Ivanova. A dynamic model and an algorithm for short-term supply chain scheduling in the smart factory industry 4.0. *International Journal of Production Research*, 54(2):386–402, 2016.

[9] Jun-Qing Li, Quan-Ke Pan, PN Suganthan, and TJ Chua. A hybrid tabu search algorithm with an efficient neighborhood structure for the flexible job shop scheduling problem. *The international journal of advanced manufacturing technology*, 52(5-8):683–697, 2011.

[10] Fadi Shrouf, Joaquin Ordieres, and Giovanni Miragliotta. Smart factories in industry 4.0: A review of the concept and of energy management approached in production based on the internet of things paradigm. In *Industrial Engineering and Engineering Management (IEEM), 2014 IEEE International Conference on*, pages 697–701. IEEE, 2014.

[11] Ling Wang, Gang Zhou, Ye Xu, Shengyao Wang, and Min Liu. An effective artificial bee colony algorithm for the flexible job-shop scheduling problem. *The International Journal of Advanced Manufacturing Technology*, 60(1-4):303–315, 2012.

[12] Li-Ning Xing, Ying-Wu Chen, Peng Wang, Qing-Song Zhao, and Jian Xiong. A knowledge-based ant colony optimization for flexible job shop scheduling problems. *Applied Soft Computing*, 10(3):888–896, 2010.

[13] Mehdi Yazdani, Maghsoud Amiri, and Mostafa Zandieh. Flexible job-shop scheduling with parallel variable neighborhood search algorithm. *Expert Systems with Applications*, 37(1):678–687, 2010.

[14] Zalmiyah Zakaria and Sanja Petrovic. Genetic algorithms for match-up rescheduling of the flexible manufacturing systems. *Computers & Industrial Engineering*, 62(2):670–686, 2012.