Neurocomputing 132 (2014) 97–110

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Negotiation mechanism for self-organized scheduling system with collective intelligence

A. Madureira [a,*], I. Pereira [a], P. Pereira [a], A. Abraham [b]

[a] GECAD – Knowledge Engineering and Decision Support Research Center, School of Engineering, Polytechnic of Porto (ISEP/IPP), Portugal
[b] Machine Intelligence Research Labs (MIR Labs), USA

ARTICLE INFO

Article history:
Received 31 October 2012
Received in revised form
2 September 2013
Accepted 23 October 2013
Available online 27 November 2013

Keywords:
Negotiation in MAS
Self-organization
Swarm intelligence
Dynamic scheduling
Agile manufacturing

ABSTRACT

Current Manufacturing Systems challenges due to international economic crisis, market globalization and e-business trends, incites the development of intelligent systems to support decision making, which allows managers to concentrate on high-level tasks management while improving decision response and effectiveness towards manufacturing agility.

This paper presents a novel negotiation mechanism for dynamic scheduling based on social and collective intelligence. Under the proposed negotiation mechanism, agents must interact and collaborate in order to improve the global schedule. Swarm Intelligence (SI) is considered a general aggregation term for several computational techniques, which use ideas and inspiration from the social behaviors of insects and other biological systems. This work is primarily concerned with negotiation, where multiple self-interested agents can reach agreement over the exchange of operations on competitive resources. Experimental analysis was performed in order to validate the influence of negotiation mechanism in the system performance and the SI technique. Empirical results and statistical evidence illustrate that the negotiation mechanism influence significantly the overall system performance and the effectiveness of Artificial Bee Colony for *makespan* minimization and on the machine occupation maximization.

1. Introduction

For today's manufacturing environments, it is increasingly necessary that a close relationship between manufacturing decision making and corporate business strategy exists, so that manufacturing decisions complement and are fully aligned with the strategic objectives of organizations through agility concerns and requirements. Agility refers to the manufacturing systems ability to efficiently adapt to market and environmental changes in an cost-effective ways.

Real world scheduling requirements are related with complex systems operated in dynamic environments frequently subject to several kinds of imponderables and perturbations, such as:

- Scheduled orders could take more time than estimated;
- Machines could become unavailable or additional ones may be introduced;
- New orders arrive continuously to the system while scheduled orders could be cancelled;
- Unexpected events occur in the system (employees sickness, rush orders, lateness on raw-materials or components)

These scenarios make the current schedules easily outdated and unsuitable. Scheduling under this environment is known as dynamic, which could be defined as a continuous and ongoing reactive process where the real time information implies the revision and dynamic adaptation of current schedules to the perturbations [1,3].

A Job-Shop like manufacturing system has associated a dynamic nature observed through several kinds of perturbations on working conditions and requirements over time. For this kind of environment, it is important that the ability to efficiently and effectively adapt, on a continuous basis, existing schedules according to the referred disturbances, are mandatory for keeping business performance levels. The application of optimization techniques to the resolution of this class of real world scheduling problems seems really promising. Although, most of the known work on scheduling deals with optimization of classical Job Shop Scheduling Problems (JSSP) problems, on static and dynamic environments [1,2].

The problem of finding good solutions is very important to real manufacturing systems considering that production rate and production costs are very dependent on the schedules used for controlling the flow of work through the system. Production planning and distribution, transport planning, allocation of resources (raw materials, manpower or machines in time) and

* Correspondence to: GECAD, Dr. António Bernardino de Almeida, 431, 4200-072 Porto, Portugal. Tel.: +351 22 8340500; fax: +351 22 8321159.
E-mail address: amd@isep.ipp.pt (A. Madureira).
URL: http://www.gecad.isep.ipp.pt/ (A. Madureira).

0925-2312/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.neucom.2013.10.032

task scheduling are combinatorial optimization problems common in industrial reality. It is not possible to always adopt the optimal solution for two reasons: due to its complex nature, the resolution to optimality in an acceptable time for making decisions is normally intractable, and many problems in reality are so dynamic that when we process/execute the solution, the characteristics of the problem have already changed, and this is not the optimal solution for the new problem. Such dynamic scheduling has receiving increasing attention amongst researchers and practitioners [3–6]. However, scheduling is still having difficulties in real world environments and, hence, human intervention is required to maintain real-time adaptation and optimization.

The interest and research on Decision Support Systems (DSS) that exhibit self-organization properties is increasingly drawing to formalize some of the ideas from Autonomic Computing [7,8] for handling problems in complex manufacturing systems and to identify mechanisms that makes use of autonomous entities in solving hard computational problems and in modelling complex systems through Self-organized or Self-managed behaviours. Self-managed systems have the ability to manage themselves and to dynamically adapt to change in accordance with evolving or dynamic business policies and objectives, allowing the addition and removal of resources/tasks without service disruption [8]. This field of research has received much attention in Autonomic Computing (AC) paradigm [7]. As a result, managers and professionals can focus on tasks with higher value to the business process. Agent based Computing technology is well adapted to model and solve production planning problems in manufacturing systems and can easily integrate social issues and self-organized mechanisms into multi-agent architectures.

Nature provides several and diverse examples of social systems and collective intelligence, such as: insect colonies foraging behaviour for food; bacteria which appear able to act in a finalized way; the human brain considering that intelligence and mind arises from the interaction and coordination of neurons; the molecule and cell formation considering homeostasis and the capability of adapting and reproducing arise from protein interactions and antibody detection. Several efforts and contributions have been related on literature that take collective intelligence as an inspiration and basis for optimization algorithms developing based on analogy with social and self-organized behaviour [4,6,11,10,11]. These approaches have been generally referred as Swarm Intelligence (SI), and are based on assumption that an organized behaviour emerges from the interactions of many simple agents like observed in nature [9,10].

To address DSS for dynamic scheduling with self-organized capabilities, we intend to integrate and explore the following paradigms: Multi-Agent Systems (MAS) [12–14], Coordination and Competition [15,16], Autonomic Computing [7,8] and Swarm Intelligence [9,10].

In this research, we propose a novel negotiation mechanism, to the resolution of scheduling in real manufacturing systems, which is by nature intrinsically a Complex Adaptive System, through negotiation. *Complex* in the sense that manufacturing systems are composed of many components (jobs, operations, machines). *Adaptive* when referring to the fact that the system must dynamically adapt to external perturbations, like rush orders, or lateness on raw-materials, and *System* considering that all components are interconnected and interdependent. A negotiation mechanism is proposed considering the following assumptions: A set of autonomous resource agents, each implementing a SI method for Single Machine Scheduling Problems (SMSP) are engaged in finding the optimal or sub-optimal solution; A coordination mechanism combining the single solutions obtained by the resource agents into a global solution is performed; A negotiation mechanism to improve global solutions by machine idle times reducing could be established.

The remaining sections of this paper are organized as follows: in section 2 the scheduling problem definition is presented. Theoretical foundations, biological motivation and fundamental aspects of SI Paradigm namely with focalization on Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO) and Artificial Bees Colony (ABC) algorithms are summarized in Section 3. Section 4 presents some related work on negotiation for scheduling through MAS. In Section 5, the competitive architecture for the self-organized dynamic scheduling is presented and in Section 6 it is described the proposed negotiation mechanism, which integrates the ideas from collective intelligence and negotiation in a Multi-Agent System. The computational study and discussion of results is presented on Section 7. Finally, the paper presents some conclusions and provides some ideas for future works.

2. Problem definition

Real world scheduling problems have received a lot of attention in recent years. In this work, we consider the resolution of more realistic problems. Most real world multi-operation scheduling problems can be described as dynamic and extended versions of the classic Job-Shop scheduling combinatorial optimization problem. In practice, many scheduling problems include further restrictions and relaxation of others [1,2]. Thus, for example, precedence constraints among operations of the different jobs are common because, often, mainly in discrete manufacturing, products are made of several components that can be seen as different jobs whose manufacture must be coordinated. Additionally, since a job can be the result of manufacturing and assembly of parts at several stages, different parts of the same job may be processed simultaneously on different machines (concurrent or simultaneous processing). Moreover, in practice, scheduling environments tend to be dynamic, i.e. new jobs arrive at unpredictable intervals, machines breakdown, jobs can be cancelled and due dates and processing times can frequently change.

In this work, solutions are encoded by the direct representation, where the schedule is described as a sequence of operations, i.e. each position represents an operation index with initial and final processing times. Each operation is characterized by the index (i, j, k, l), where i defines the machine where the operation k is processed, j the job it belongs to, and l the graph precedence operation level (level 1 (one) corresponds to initial operations, without precedents [3].

The minimization of total completion time, also known as makespan [1,2] is given by

$$Min\ C_{\max} = max(F_j), \quad \forall j = 1, \ldots, n,$$

Subject to

$$ST_{ijkl} + p_{ijkl} \leq ST_{ij'k'l'} \quad \forall j = 1, \ldots, n, \quad \forall (O_{ijkl}, O_{ij'k'l'}) \tag{1}$$

The constraint from (1) represents the precedent relationship between two operations k and k'($k \neq k'$ and $k < k'$ and $l < l'$) of the same *job* j, that could be executed on different machines k and k', and at different levels l and l'.

$$ST_{ijkl} \geq t_{ijkl+1} \quad \forall O_{ijkl} \tag{2}$$

The constraint shown in (2) represents that the processing time to start operation O_{ijkl} must be greater or equal to the earliest start time for the same operation. The constraint, specified on (3), represents machine occupation, where only one operation could

be processed on each instant of time.

$$\sum X_{ijt} = 1 \quad \forall i, \quad j = 1, \ldots, n \quad \forall t \leq max(F_j) \tag{3}$$

The system machine occupation rate U, is given by

$$max\, U = max \frac{1}{M \times C_{\max}} \sum_{i=1}^{M} \sum_{k=1}^{n} p_{ijkl} \times 100 \quad \forall j,\ l \tag{4}$$

where p_{ijkl} is the processing time for each operation k on each machine i.

3. Swarm Intelligence

Recently, biological processes have been a source of inspiration for several fields in science and engineering [5,6,9,10]. Evolutionary computing is based on the Darwinian notions of survival of the fittest and on evolution, while particle swarm optimization is based on the theory of swarming insects or flocking birds. Several biologically inspired algorithms have been designed and applied, and many of them are effective for producing high quality solutions to a diversity of real world optimization problems, including scheduling, planning, logistics, space allocation, engineering design, bioinformatics and data mining, etc. The size and complexity of the optimization problems require the development of methods and solutions whose efficiency is measured by their ability to find acceptable solutions within a reasonable amount of time.

Swarm Intelligence (SI) refers to a research area that integrates efforts from computer science and artificial intelligence communities for the study, design and specification of efficient computational approaches for problem solving, inspired from the collective intelligence of biological populations that can be observed in nature such as ants, bees, fish, and birds.

3.1. Ant Colony Optimization

Ant Colony Optimization (ACO) algorithm takes inspiration from the foraging behavior of some ant species. These ants deposit pheromone on the ground in order to mark some favorable path that should be followed by other members of the colony. ACO exploits a similar mechanism for solving optimization problems and was initially proposed by Marco Dorigo [17]. The original idea has since then diversified to solve a wider class of numerical problems, and as a result, several problems have emerged, drawing on several aspects of the behavior of ants [18,19].

A general ACO algorithm is described in Table 1. After initialization, ACO iterates over three main steps: at each iteration, a number of solutions are constructed by the ants; these solutions could be then improved, optionally, through a local search, and finally the pheromone is updated through two possible events:

Table 1
Ant Colony Optimization Algorithm.

Algorithm 1: Ant Colony Optimization

```
Input: ACO Parameters and scheduling data problem
Output: Best solution
1:  Begin
2:      Set ACO parameters.
3:      Initialize pheromone trails
4:      While termination criteria not met do
5:        Construct AntSolutions
6:        Apply Localsearch (optional)
7:        Update Pheromones
8:        Memorize the best solution achieved so far
9:      EndWhile
10: End
```

evaporation and by increasing the pheromone levels associated with a chosen set of good solutions. A more detailed description of the three phases can be stated as follows [19]:

Construct Ant Solutions: A set of m artificial ants constructs solutions from elements of a finite set of available solution components $C=\{c_{ij}\}$, $i=1,\ldots, n$, $j=1,\ldots, |Di|$. A solution construction starts from an empty partial solution $s^p=\varnothing$. At each construction step, the partial solution s^p is extended by adding a feasible solution component from the set $N(s^p)\subseteq C$, which is defined as the set of components that can be added to the current partial solution sp without violating any of the constraints in Ω. The process of constructing solutions can be regarded as a walk on the construction graph $G_C=(V, E)$ as stated in [19]. The selection of a solution component from $N(s^p)$ is guided by a stochastic mechanism, which is biased by the pheromone associated with each of the elements of $N(s^p)$. The rule for the stochastic choice of solution components vary across the different proposed ACO algorithms but, in all of them, it is inspired by the Goss model (experimental setup for the double bridge experiment) of the behavior of real ants assuming that at a given moment in time m_1 ants have used the first bridge and m_2 the second one, the probability p_1 for an ant to choose the first bridge is given by [19]

$$p_1 = \frac{(m_1+k)^h}{(m_1+k)^h+(m_2+k)^h} \tag{5}$$

where parameters k and h are to be fitted to the experimental data. Monte Carlo simulations showed a very good fit for $k \approx 20$ and $h \approx 2$.

- **Apply Local Search**: Once solutions have been constructed, and before updating the pheromone, it is common to improve the solutions obtained by the ants through a local search. This phase, which is highly problem-specific, is optional although it is usually included in state-of-the-art ACO algorithms.
- **Update Pheromone**s: The aim of the global pheromone update is to increase the pheromone values associated with good or promising solutions, and to decrease those that are associated with bad ones. Usually, this is achieved by decreasing all the pheromone values through pheromone evaporation, and by increasing the pheromone levels associated with a chosen set of good solutions.

Several ACO algorithms have been proposed in the literature, which differ in some decisions characterizing the construction of solutions and update pheromone procedures [19]. Additional information about ACO based algorithms details of implementation could be found in [18,19].

In this work we consider an ACS for SMSP described in Madureira et al. [20], a particular ACO algorithm. The ACS differs from the previous Ant System due to three main aspects [18]: the state transition rule, the global updating rule, and the local updating rule. When applied to the SMSP, each ant constructs a feasible sequence by selecting an unscheduled job j to be on the ith position of the partial sequence constructed so far. This process is influenced by specific heuristic information η_{ij}, as well as the pheromone trails τ_{ij}.

The most interesting contribution of ACS [17,19] is the introduction of a local pheromone update and the pheromone update performed at the end of the construction process (named offline pheromone update). ACS algorithm can be stated as follows [17,19]: m ants are initially positioned on n cities chosen according to some initialization rule (randomly, for example). Each ant builds a tour (feasible solution) by repeatedly applying a stochastic greedy rule (the state transition rule). While constructing its tour/path, an ant also modifies the amount of pheromone on the

visited edges (cities) by applying the local updating rule. Once all ants have terminated their tour/path, the amount of pheromone on edges/cities is modified again, by global updating rule applying. Ants are guided, in building their solutions, by both heuristic information (they prefer to choose short edges), and by pheromone information (an edge with a high amount of pheromone is a very desirable choice). The pheromone updating rules are designed to give more pheromone to edges/cities which should be visited by ants.

The local pheromone update is performed by all ants after each construction step. Each ant applies it only to the last edge traversed:

$$\tau_{ij} = (1-\varphi)\tau_{ij} + \varphi\tau_0 \tag{6}$$

where $\varphi \in [0,1]$ is the pheromone decay coefficient, and τ_0 is the initial value of the pheromone.

The main goal of the local pheromone update is to introduce diversity in the search process performed by subsequent ants during an iteration by decreasing the pheromone concentration on the traversed edges, ants encourage subsequent ants to choose other edges and, hence, probably to produce different solutions. This mechanism makes it less likely that several ants produce identical solutions during one iteration.

The offline pheromone update, is applied at the end of each iteration by only one ant, which can be either the iteration-best (L_{ib}) or the best-so-far (L_{bs}). However, the update formula is slightly different:

$$\tau_{ij} = \begin{cases} (1-\rho)\times\tau_{ij} + \rho\times\Delta\tau_{ij} & \text{if } (i,j) \text{ belongs to the best tour,} \\ \tau_{ij} & \text{otherwise} \end{cases} \tag{7}$$

where $\tau_{ij} = 1/L_{\text{best}}$, where L_{best} can be either L_{ib} or L_{bs}.

Another important difference between ACS and AS is in the decision rule used by the ants during the construction process [18–20].

$$p_{ij}^{k} = \begin{cases} \dfrac{\tau_{ij}^{\alpha}\times\eta_{ij}^{\beta}}{\sum_{cij\in N(S^p)}\tau_{ij}^{\alpha}\times\eta_{ij}^{\beta}} & \text{if } cij \in N(S^p) \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

In ACS, the so-called pseudorandom proportional rule is used by the ants during the construction process: the probability for an ant to move from city i to city j depends on a random variable q uniformly distributed over [0,1], and a parameter q_0; if $q \le q_0$, $j = \text{argmax}_{c_{ij}\in N(S^p)}\{\tau_{il}\eta_{il}^{\beta}\}$ otherwise (8) is used.

Table 2
Particle Swarm Optimization Algorithm.

Algorithm 2: Particle Swarm Optimization

```
Input: PSO Parameters and scheduling data problem
Output: Best solution
1:  Begin
2:      Initialize particles population
3:      Evaluate fitness of individual particles and define
        pBest_i and Gbset
4:      While termination criteria not met do
5:        Modify velocities based on previous best and global best
6:            V_id = ω*V_i + r_1*C_1*(pBest_i - X_i) + r_2*C_2*(gBest - X_i)
7:        Move to the new position X_i = X_i + V_id
8:        Evaluate fitness of individual particles
9:        If f(X_i) < f(pBest_i) then pbest_i = X_i
10:       If f(X_i) < f(gBest) then gbest = X_i
11:     EndWhile
12: End
```

3.2. Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a population based stochastic optimization technique proposed by Kennedy and Eberhart [22], inspired by social behavior of bird flocking or fish schooling. The algorithm is initialized with a population of random solutions and searches for sub-optimal solution by updating generations.

There are N particles, each of these particles adjusts its direction based on their own experience and the experience of the rest of the population (group of particles). Each movement of each particle is based on three parameters: the sociability factor, the cognitive factor, and the maximum speed [10,22].

The algorithm starts by initializing particles population, defining initial current position and velocity of all the particles (Table 2). Then, at each iteration and for each particle, the velocities are updated, based on its previous best value $pBest_i$ and on best global value $gBest$, and a new position (for each particle) in the search space is defined according to (6) and (7), respectively. Each particle is treated as a point in a D-dimensional space. Considering all the current positions of each particle as $X_{id} = \{X_{i1}, X_{i2}, \ldots, X_{id}\}$, where X_i is the position of the particle i dimension d. The velocity of each particle is updated based on

$$V_{id} = w\times V_i + r_1\times C_1(pBest_i - X_i) + r_2\times C_2(gBest - X_i) \tag{9}$$

where C_1 (cognitive component reflecting personal experience) and C_2 (social component reflecting group experience) are positive constants, V_i is the velocity of particle i, and r_1 and r_2 are random numbers defined in the range [0,1]. ω constitutes the inertia weight provides a balance between global and local exploration and exploitation [22]. The velocity update consists (9) in three components: the *acceleration* that cannot be modified abruptly but adjusted considering current velocity and maximum speed; the *cognitive component* represents the learning from its personal flying experience; and the *social component* that represents group learning flying experience.

The current position of each particle is updated based on the

$$X_{id} = X_i + V_{id} \tag{10}$$

where X_{id} is the new position of particle i, X_i is the current position of particle i and V_{id} is the velocity of particle i.

PSO has become a popular optimization method and has been widely used in practical problem solving [9-11,22].

3.3. Artificial Bee Colony

Artificial Bee Colony (ABC) arise from the analogy with real bees as a social insects living in organized group called hive. In a beehive, the bees have some specific tasks performed by specialized bees. The main purpose of the colony is the maximization of the amount of nectar getting the utmost of the food sources. In 2005, Pham proposed a Bees Algorithm in a technical report [24] inspired in the foraging behavior of honey bees to find food sources, as an optimization algorithm to find an optimal solution. At the same time, Karaboga [23] proposes a similar algorithm named Artificial Bee Colony.

In a real bee colony, some tasks are performed by specialized individuals. These specialized bees try to maximize the nectar amount stored in the hive using collaboration and division of labor task trough self-organization. In this work we consider a modified Artificial Bee Colony described in [21] for single machine scheduling problems.

A modified ABC algorithm have three main phases, corresponding to three types of specialized bees, Employed, Onlooker and Scout, that represent a minimal model of the real swarm intelligent forage selection [25]. Employed bees are in the same number of food sources (solutions) and are responsible to explore one and

only one food source at the time and give information to other bees. When an employed bee left his food source becomes a scout bee. Onlooker bees turret in the hive for a information of a employed bees to establish a good food source. Scouts bees seek environment trying to find a new food source depending on an internal motivation or external clues or randomly. Half of the hive is composed by employed bees and the other half by onlooker bees. The food source position represents a solution that is measured by the nectar amount corresponds to the quality of the solution (Table 3).

In the initialization phase, the algorithm randomly generates $sn/2$ initial solutions, were sn is the size of the population, which will be the food field for the employed bees. Each x_i ($i=1, 2, sn/2$) is a dimensional vector D. Values between the limits of the parameterization are assigned to the solution and a $\texttt{failure}_i$ value is also added to analyze when this solution i must be abandoned. After validating the population, the algorithm repeats a specified number of cycles of employed, onlooker and scout bees phases.

3.3.1. Employed bees phase

An employed bee performs a change in their position of food source based on (11) and evaluates the nectar amount in the new position/solution [25]:

$$v_{ij} = \begin{cases} x_{ij} + \varnothing(x_{ij} - x_{kj}), & \text{if } R_j < MR \\ x_{ij} & \text{otherwise} \end{cases} \quad (11)$$

where $k \in \{1,2,\ldots,sn\}$ is a randomly chosen index that has to be different from i, and $\varnothing_{ij}$ is an uniformly distributed random real number in the range of $[-1,1]$. R_j is uniformly distributed random real number in the range of [0,1] and MR is a control parameter of ABC algorithm in the range of [0,1] which controls the number of parameters to be modified.

Then the algorithm selects the solution by the following rules:

- Two realizable solutions – selects the one with the best amount of nectar (fitness) value;
- One solution realizable and one unrealizable – select the realizable;
- Two unrealizable solution – select the one with the smaller degradation factor.

Finished the search, the employed bees share the information with the onlooker bees and the solutions are selected based on a probability by the value of fitness or violation of the solutions depending if they are realizable or not.

Table 3
Artificial Bee Colony Algorithm.

Algorithm 3: Modified ABC Algorithm

```
Input: ABC Parameters and scheduling data problem
Output: Best solution
1:  Begin
2:      Initialization of Bee Population and Food sources
3:      Cycle=1
4:      While cycle < > Maximum Cycle Number
5:          Solutions Evaluation
6:          Employed Bees Phase
7:          Calculate Probabilities for Onlookers
8:          Onlooker Bees Phase
9:          Scout Bees Phase
10:         Memorize the best solution achieved so far
11:         Increment Cycle
12:     EndWhile
13: End
```

3.3.2. Onlooker bees phase

Onlooker bees select their own food source based on a probabilistic rate according to the amount of nectar on the solution. The algorithm uses (8) to create a new food source, validating and adjusting the new solution according to the parameterization.

3.3.3. Scout bees phase

After the above steps, all food sources that will not be explored anymore are abandoned. The employed bees that left the food source get a new position from scouts search.

Several modified algorithms have been proposed since then in the literature and has been widely used in practical problem solving [10,25,21].

4. Negotiation in Multi-Agent System and Self-* systems

Competition has been studied in several fields, including psychology, sociology and anthropology. Social psychologists, for instance, study the nature of competition. They investigate the natural urge of competition and its circumstances. They also study dynamics group, to detect how competition emerges and what its effects are [15]. Several contributions have been stated for negotiation research area in several fields, including sociology, anthropology, philosophy, economics and political science.

Software systems developing involving autonomic interacting software agents present new challenges in Computer Science and Software Engineering. Agent based technologies provide a way to conceptualize complex and dynamic systems as comprising interacting social and autonomous entities, acting, learning and evolving separately in response to interactions and stimuli in their local environment [12–14]. Techniques to design and implement agent based systems could be categorized into three classes [12]: organization level (concerning organizational structure related to agent societies as a whole, trust, norms and obligations), interaction level (concerning agent communication, interaction and decision making) and Agent Level (concerning individual agents, like reasoning and learning).

A particularly challenging problem is the engineering of several forms of interaction among agents. Interaction may be aimed at enabling agents to coordinate their activities and behaviors, cooperate to reach common objectives, or compete to better achieve their individual objectives. Considering real manufacturing systems composed by multiple autonomous agents, negotiation is an important form of interaction that enables groups of agents to achieve at a mutual agreement regarding some objective or scheduling plan.

Multi-Agent Systems are composed of several agents, capable of mutual interaction. The interaction can be designed in the form of message passing, requesting, negotiating or producing changes in their common environment. MAS provide a way to conceptualize adaptive systems and self-organization as comprising interacting autonomous agents, each acting, learning or evolving individually in response to interactions on their own environments. MAS can manifest self-organization and complex behaviors even when the individual strategies of all their agents are simple [12].

Literature attempts to classify software agents according to different dimensions and criteria, which refer to the study of entities types and the investigation of agent's typology [27,28]. Based on the exhibition of ideal and primary attributes, Nwana [28] proposed a classification where agents may be classified considering several ideals and primary attributes which agents should exhibit such as: Autonomy, Cooperation and Learning.

Based on these three characteristics Nwana [28] proposed a classification with four types of agents: collaborative agents, collaborative learning agents, interface agents and truly smart agents (Fig. 1). Different MAS approaches are described on literature to

implement collaboration and negotiation between agents that could be categorized in two main classes [12–14,27,29]: first, each agent is able to communicate with the others requesting their needs to the group. It requires a higher degree of agents' intelligence, since they should be able to analyze the task and communicate with each other to obtain the solution. In the second class of approaches, a coordinator agent analyzes the problem and, based on their characteristics; send the tasks to each agent individually.

In MAS, agents should often work against each other due to the conflicts in their objectives, leading to competition. Competitive agents try to maximize their own benefits at the expense of others, and thus the success of one implies the failure of others [30]. A significant part of research in coordination of competitive agents is made on negotiation [31–34]. Negotiation can be defined as a form of interaction in which a group of agents with conflicting interests (and wish to collaborate) try to reach a mutual agreement for the allocation of scarce resources [33].

Negotiation can be defined as the process by which a joint decision is reached by two or more agents, each one trying to reach an individual objective. The agents first communicate their targets, which may be conflicted, and then try to reach an agreement by making concessions or searching for alternatives [30]. Mainly because competitive agents are autonomous and cannot be assumed to be benevolent, they must try to influence others in order to convince them to act in certain ways. Negotiation is thus decisive for managing such inter-agent dependencies [34]. Generally, literature defines the following negotiation methods: Contract Net Protocol [35]; Auctions [36]; Game Theory [37]; Argumentation [38].

One objective of negotiation is that the allocation of resources should be accepted by all participants. Since there are several different forms of agreements, negotiation can be seen as a distributed search through a space of possible agreements [38]. Negotiation mechanisms should consider some features such as [38]: simplicity, efficiency, distribution, symmetry, stability, and flexibility. Such mechanisms must lead to an agreement even if agents have not completed or corrected their private information related to their own decisions.

The protocol and strategy are the main components of a negotiation mechanism. The protocol defines the common rules among the participants in the act of negotiate. In general, it includes a set of norms that represents the constraints for the proposals that participants can do. The strategy defines the possible actions (or sequence of actions) that an agent plans to follow during the negotiation process [38].

Usually, the negotiation process consists in a group of rounds, with all agents making a proposal in each round. The agents' proposals are defined by its strategy and must be consistent with the defined protocol. The negotiation is concluded when an agreement is reached [36]. A complex aspect of the negotiation process is the number of agents involved and how these agents interact [36]. This interaction can be made as: One-to-one, where only one agent negotiates with another agent (e.g., a sale of a product to a costumer); One-to-many, in which an agent negotiates with a set of agents (e.g., auctions); Many-to-many, where multiple agents simultaneously negotiate with other agents (e.g., marketplace).

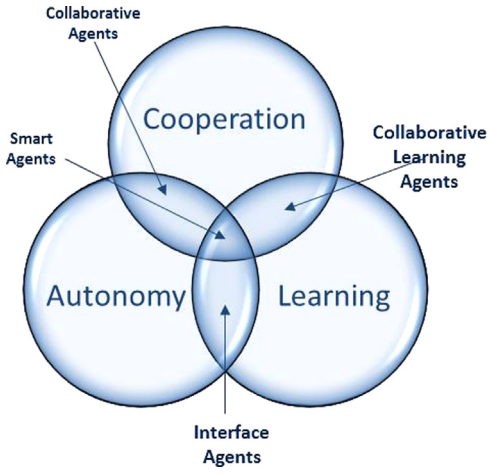


Fig. 1. Agent typology.
Adapted from Nwana's [28].

Several negotiation mechanisms have been proposed and referenced in the literature. In scheduling, negotiation is used generally to improve the quality of final solutions. For example, Zattar et al. [39] proposed the use of an operation-based time-extended negotiation protocol to allow decision-making for the real-time routing of job orders composed by operations in a job-shop environment. Singh et al. [40] proposed an improved Contract Net Protocol architecture named Contract Net Trust Establishment Protocol where two agents that are willing to cooperate in the achievement of the system's goal are ruled by process and resource managers. In Kim and Cho [41] negotiation agents have been used to allocate numerous orders to many participants for a supply chain formation. Adhau et al. [42] proposed a novel distributed multi-agent system using negotiation based on auctions for solving the resource conflicts and allocating multiple different types of shared resources amongst multiple competing projects.

In this paper it is used an adaptation of Contract Net Protocol in order to allow the negotiation between conflicting agents [46].

5. Collaborative Dynamic Scheduling architecture

The Collaborative Dynamic Scheduling architecture – Auto-DynAgents scheduling system – consists in a MAS in which a community of agents models a real manufacturing system subject to perturbations and imponderables (Fig. 2). Agents must be able to learn and manage their internal behavior and their relationships with other autonomic agents, by negotiation in accordance with business policies defined by managers and operational managers.

5.1. Collaborative architecture

Towards the distributed, autonomous and coordination features considered, the MAS technology is suitable to model real manufacturing systems, which can be mapped into the MAS where autonomous intelligent agents coordinate to solve scheduling problems.

Considering agents operational and subordination relations [26,27] defined in the proposed system, it is possible to consider the proposed architecture as a hybrid market based architecture with hybrid agents. AutoDynAgents is a Decision Support System for discrete manufacturing systems in dynamic environments and its application is flexible for any type of production system (Single Machine, FlowShop, JobShop) of products regarding as single items or multi-component assemblies, and different types of manufacturing environments, static or dynamic. In this scheduling approach, a divide-conquer methodology is used to decompose it into sub-problems, so that each SMSP can be solved separately [3,43]. The solutions could then be reassembled for an overall solution.

The scheduling problem under consideration is decomposed into a series of deterministic SMSP (considering that all release dates, processing times and due dates are known in advance), which are solved by a SI. The obtained solutions are then incorporated into the main problem and a repair mechanism is carried out, having into account job operation precedence and machine occupation times [43]. Its main objective is to guarantee the schedule feasibility. Then, the obtained solution is negotiated in order to refine the schedule based on three inter-related optimization objectives minimization of idle

times, minimization of *makespan* (completion time of all jobs) and maximization of machine utilization rate.

Whenever a new order arrives or some will be cancelled which disturbs the current schedule, in such a way that rescheduling must be done, the dynamic adaptation module integrates the event on a new deterministic problem [3]. Then, Resource Agents will be resubmitted to negotiation.

The AutoDynAgents system [3,4] and the embedded negotiation mechanism were developed in Java Language according to standardization of FIPA using the Jade platform for the development of agents [31] and the Eclipse as development environment.

5.2. System model

The system model envisages representing the main components of a dynamic scheduling process. The model is to be implemented in a multi-agent system designed to simulate resources and tasks in a scheduling decision making process involving coordination. The main objective is to support the operational manager in the decision making. In the proposed model there are agents representing tasks/jobs (Task Agents) and agents representing machines/resources (Resource Agents) in a manufacturing environment. The Resource Agents must be able to find an optimal or near optimal local solution through SI algorithms (ABC, ACS and PSO) for SMSP and to negotiate with other agents (Fig. 3). SMSP aims at sequencing a set of jobs on a single machine [1,2,43].

Additionally the proposed model considers a Coordinator agent (UI Coordinator agent) responsible to coordinate and integrate the single solutions obtained by each Resource Agent solution in order to obtain a global schedule for the original scheduling problem and self-* agents responsible for guarantee agility and adaptation.

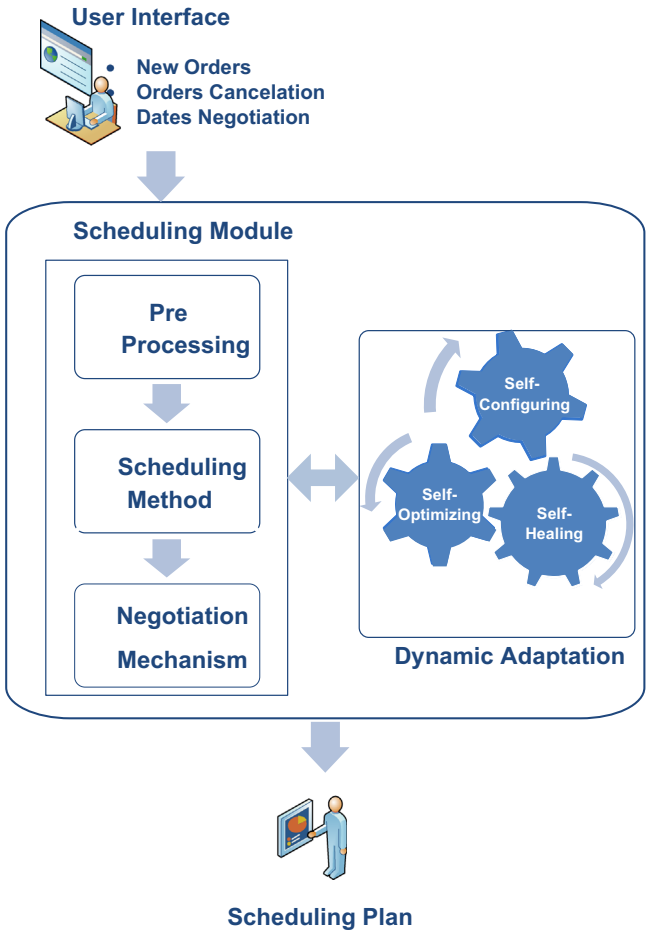


Fig. 2. System architecture.

The proposed self-managed model (Fig. 3) represents a more detailed view of the model described above. At this phase, we consider in the model three distinct agent types, here identified by self-* agents: Self-Configuration Agent, Self-Optimization Agent and Self-Healing agent. Considering classification schemes described in Section 4 proposed by Nwana [28], we propose a hybrid Multi-Agent architecture since it combine two or more approaches in a single agent that includes collaborative agents, collaborative learning agents, interface agents and smart agents.

The developed MAS for Scheduling problem resolution (AutoDynAgents) is a self-organized scheduling system and consists in a hybrid autonomous hierarchical architecture. There are agents representing jobs (or tasks) and agents representing resources (or machines). The system is able to [4]: find optimal or near optimal solutions through the use of MH; deal with dynamism (arriving of new jobs, cancelled jobs, changing jobs attributes, etc.); switch from one SI to another; and change/adapt the parameters of the algorithm according to the current situation.

The architecture model is based on four different types of agents: User Interface Coordinator Agent, Task Agents, Resource Agents and self-* Agents:

- **User Interface Coordinator agent** (Smart Agent), apart from being responsible for the user interface, dynamically generates the necessary Task Agents and Resource Agents, according to the number of jobs and machines that comprise the scheduling problem, and assign each job to the respective Task Agent. It is also responsible for the verification of feasible schedules and identification of constraint conflicts on each Task and the decision of which Resource Agent is responsible for solving a specific conflict.
- **Task Agents** (Collaborative Agents) process the necessary information regarding the task. Each one is responsible for the generation of the earliest and latest processing times and division of operations throughout the different Resource Agents.
- **Resource Agents** (Collaborative Agents) are responsible for scheduling the jobs' operations that require processing in the machine supervised by the Resource agent. These agents implement a SI algorithm (ABC, ACS or PSO) in order to find the best possible schedules, and communicate those solutions to the UI Coordinator Agent for later feasibility check. Resource agents are organized following the Market based architecture [25].

Additionally, to provide self-managing properties to the system, three agents representing three components of Autonomic Computing Self-CHOP (Configuring, Healing, Optimizing and Protecting) [7,8] were added to system in order to provide the system with autonomy, such as:

- **Self-Configuring Agent** (autonomic agent) is responsible for monitoring the system in order to detect changes occurred in the schedule, allowing a dynamic adaptation of the system. With this agent, the system is prepared to automatically handle with dynamism by adapting the solutions to external perturbations.
- **Self-Optimizing Agent** (interface agent) is responsible for automatically select a SI algorithm and tune the respective parameters, according to the problem. This parameters tuning is made through learning and experience, since it uses a CBR module [44].
- **Self-Healing Agent** (autonomic agent) monitors other agents in order to provide overall self-healing capabilities, providing

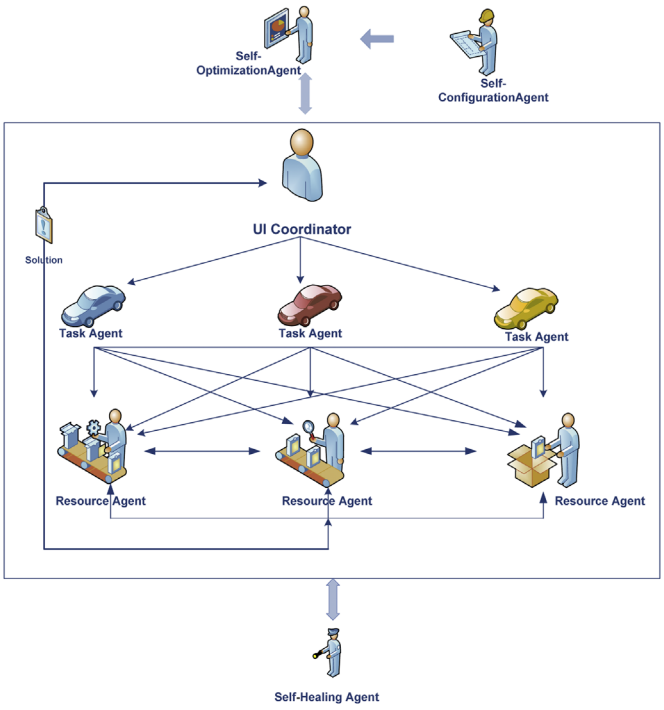


Fig. 3. System model.

the ability of recovering from failures. With this agent, the system becomes stable, even if some deadlocks or crashes can occur.

The used approach to deal with this problem is divided in two steps. In the first, the system waits for the solutions obtained by the Resource Agents and then applies a repair mechanism to adapt some operations in the generated schedules till a feasible solution is obtained. In the second step, a negotiation mechanism is established between related agents in the process, in order to interact with each other to pursuit their objectives through negotiation. This negotiation mechanism, described in the next sub-section, is prepared to accept agents subjected to dynamism (new jobs arriving, cancelled jobs, changing jobs attributes).

6. Negotiation mechanism

In this paper a negotiation mechanism is proposed for dynamic manufacturing scheduling systems. The approach seeks to provide the system with negotiation capability, so that the scheduling plan generated by the Resource Agents can be improved by reducing the idle times, and corresponding machine occupation rate improving.

Initially, the Resource Agents generate their own solution independently. The UI Agent analyzes this local solutions and applies a repair mechanism, according to precedence's constraints and occupation times on the different machines/resources [45]. At this stage, the system is entirely dependent of the initial solutions, becoming "blind" and incapable of improving the scheduling plans. As such, with this mechanism we intent to give the system, through a negotiation process, the capability to optimize the scheduling plans.

The implemented Negotiation Mechanism (NM) works on a continuous cycle, so that all idle times can be analyzed. As such, the mechanism is concluded when the process locks after trying to swap an operation and/or when a lack of credits exists.

The negotiation mechanism (Table 4) seeks to reduce/eliminate the idle times between operations, thus improving the utilization rate of each machine/resource, the overall delays and downtimes. Idle times in the Resource Agents are generated by operations precedence's constraints, i.e., an operation must wait to be processed until the end of its precedence operation. Therefore, the negotiator needs to anticipate the processing of the precedence operation or set another operation for an idle time. The negotiation mechanism is responsible for handling different scenarios and choosing the one that is more optimized.

In this context, we consider two types of agents, an initiator and a participant. At any time, an agent can be an initiator, a participant or both. In this sense, initiators are managers and participants are contractors. An Initiator could be an agent willing to buy something useful or wanting to sell the right to supply something useful. Participants would be agents wanting to sell something useful to buy the right to supply something useful. Each agent has a credit that is equal to the total amount of idle times found in the scheduling plan. This credit can be used as currency to buy something useful from other agents. As compensation for the changes to a scheduling plan the agent receives a value that is added to the credit. An agent that makes changes to its own scheduling plan pays himself.

Fig. 4 depicts the negotiation protocol established between two agents considering that negotiation is based on successive and multiple negotiation contacts between two agents involved in critical operations. During the negotiation process resource agents may exchange the following messages (request, refuse, accept):

- Request (A1, A2, Action Y) – agent A1 ask agent A2 to perform action (ex: exchange op1 with op2);
- Accept (A1, A2, Action Y) – agent A1 tell agent A2 that it accepts its request to perform the action Y;
- Reject/Refuse (A1, A2, Action Y) – agent A1 tell agent A2 that it cannot accept its request to perform the action Y.

In the negotiation algorithm, each agent aims to achieve a continuous scheduling plan with the biggest credit gain. The first Initiator agent is the one with the highest value of idle times between operations. After choosing the first Initiator agent the negotiation process begins with the algorithm described in Table 4. The Negotiation mechanism is established from the moment that a new deterministic scheduling plan has been defined by the system.

In order to better understand the negotiation mechanism a small practical example will be presented. The main goal is to produce a scheduling plan where the idle times are minimized. In this practical example each machine has an initial credit value that is equal to the sum of the idle times. The negotiation process will be started by the machine with the biggest credit value. Thus, the initial scheduling plan is represented in Fig. 5 and Table 5 summarizes the credits of each machine.

The negotiation process is started by agent M1, where the first candidate operation to be swapped is $T_{4,4}$ but a precedence constrains is in place by $T_{4,3}$ since this one only can start after $T_{4,2}$ has been concluded. The process is repeated but this time for $T_{3,3}$. The mechanism tests if it possible to swap with the operation on the right or with a precedence and concludes that the swap to the right is not possible due to the fact that $T_{4,4}$ only starts after $T_{4,3}$ ends. As such, it proceeds with a precedence swap between $T_{3,2}$ and $T_{1,2}$ with a credit cost of 4 units. A request is send to agent M2 to perform the swap between $T_{3,2}$ and $T_{1,2}$. Agent M2 makes the swap and receives 4 units. The process is repeated in a form of cycle until a feasible and possibly improved scheduling plan is achieved.

From Fig. 6 it is possible to analyze that the final scheduling plan managed to be improved in 13 time units and from Table 6

that agent M2 obtained the biggest credit gain. With the proposed negotiation mechanism we pretend to provide the system with the ability of optimize the global solution by negotiation, allowing it to evolve and produce better scheduling plans.

The Negotiation Mechanism (NM) is used when a global solution has been already attained by the scheduling module (based on integration of local solutions obtained by Resource Agents trough SI method), or when a disturbance (arrival of new jobs, canceled jobs, changes on due dates, etc.) occurs in the system and its adaptation has been incorporated in the current scheduling plan. Its main function is to improve the system performance (the current solution could not be degraded).

Table 4
Negotiation Mechanism Algorithm.

Algorithm 4: Negotiation Mechanism Optimization

```
Input: Scheduling plan obtained by scheduling module
Output: Scheduling plan optimized
1:  Begin
2:      Start Negotiation Mechanism
3:      If the mechanism is running for the first time then
4:        Update the data on each agent
5:        Start communication process between the agents
6:      EndIf
7:      While termination criteria not met do
8:        Get the solution from each agent
9:        Evaluate each operation relatively to their
          precedence's
10:       If the final plan is not feasible then
11:           Update agents with the previous valid solution
12:           End negotiation process
13:       EndIf
14:       If the solution is the best so far then
15:        Update the data on each agent
16:        Restart communication process between the agents
17:       Else
18:        Continues the negotiation process in the next
           operation
19:       EndIf
20:     EndWhile
21: End
```

7. Experimental analysis

A software tool was developed to support out the computational study aiming to analyze and evaluate the performance of the proposed negotiation mechanism, on minimizing the *makespan* (C_{max}). The computational tests were performed on an Intel® Core™ 2 Quad Q6600@ 2.40 GHz processor, 4 GB of RAM memory, a 250 GB 7200 rpm disc, and Windows 7 64-bit as operative system. The performance was tested on 20 benchmark instances of Job-Shop Scheduling Problem (JSSP) from different sizes, available at OR Library [47]. The instances were selected based on their dimension (number of jobs). Therefore, for this study we used different problem instances from Fisher and Thompson [48], Lawrence [49], Adams et al. [50], Storer et al. [51] and Yamada and Nakano [52].

The work reported in this paper is related to a Collaborative Dynamic Scheduling architecture – AutoDynAgents scheduling system – that consists in MAS in which a community of agents models a real manufacturing system subject to perturbations and imponderables. Agents must be able to learn and manage their internal behavior and their relationships with other autonomic agents, by negotiation in accordance with business policies defined by managers and operational managers. The Negotiation Mechanism (NM) is used when a global solution has been attained by the scheduling module, or when a disturbance (arrival of new jobs, canceled jobs, changes on due dates, etc.) occurs in the system and its adaptation has been implemented in the current scheduling plan.

7.1. Configuration and parameters tuning

According to system characteristics, considering decomposition of scheduling problem on single machine scheduling problems. SI parameters were defined for SMSP considering the minimization

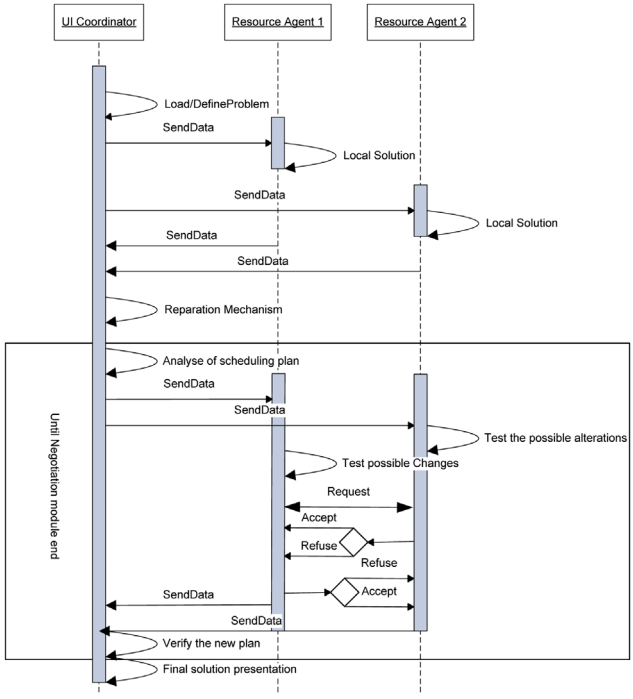


Fig. 4. Negotiation mechanism sequence diagram.

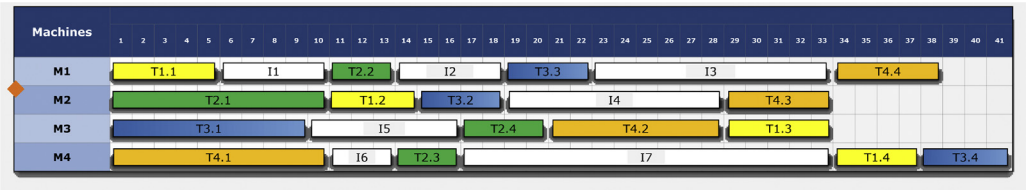


Fig. 5. Initial scheduling plan.

Table 5
Initial credits

	Stops	Credits gain	Credits lost	Credits
M1	21	0	0	21
M2	10	0	0	10
M3	7	0	0	7
M4	20	0	0	20

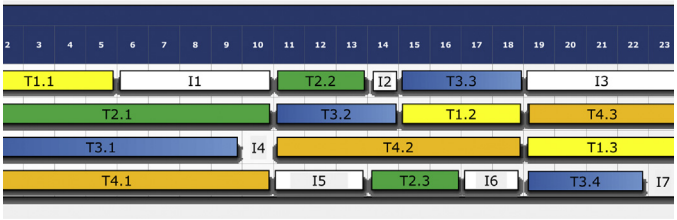


Fig. 6. Final scheduling plan.

Table 6
Final credits.

	Stops	Credits gain	Credits lost	Credits
M1	11	0	8	3
M2	0	8	0	8
M3	1	7	7	1
M4	6	5	5	6

of *makespan* (C_{max}). Additionally, the self-parameterization module was disabled in order to ensure selected SI technique maintenance, permitting inside analysis of system performance analysis trough NM influence and/or by SI technique advantage.

The tuning of parameters can allow greater flexibility and robustness but requires a careful initialization. The parameters can have a major influence on the efficiency and effectiveness of the search. Becomes not obvious, a priori, the setting of parameters to use. The values for the parameters depend on the problem, instances structure and the time available to solve the problem. There are no universal values for the parameters considered for SI based algorithms. Being widespread view, that its definition must result from a careful experimental effort, towards their tuning.

We consider the following common parameters to SI based techniques in analysis [43]:

- **Solution/individual representation** – the solutions/individuals are encoded by the natural representation (string), each position corresponds to a task and it is characterized by the quartet (i, j, k, l), where i indicates the machine where the operation k of the task j is processed, and l is the level of operation in the precedence graph. Additionally, the representation includes the time of start and processing conclusion of each operation. The position of the operation is the correspondent processing order in the corresponding machine. The number of positions on the string corresponds to the number of operations assigned to the single machine into consideration.
- **Initial solution generation mechanism** – the initial solution/individual (bee, ant or particle) is defined by the procedure rule SeqNivel [43] where the operations are sequenced in order of non-decreasing processing level (defined on precedence graph), giving priority to operations that are processed earlier. Avoiding the possibility of some operation that is processed at the end could be scheduled at the beginning of the plan what could have as consequence an infeasible solution. Thus, we expect to generate a good initial solution from which an initial colony will be obtained. Consider, for example an initial solution from machine 1,$(1,9,1,\mathbf{1}) \prec (1,10,2,\mathbf{2}) \prec (1,3,3,\mathbf{3}) \prec (1,7,4,\mathbf{5}) \prec (1,5,5,\mathbf{5}) \prec (1,2,6,\mathbf{7}) \prec (1,8,7,\mathbf{8}) \prec (1,4,8,\mathbf{8}) \prec (1,1,9,\mathbf{9}) \prec (1,6,10,\mathbf{10})$ The symbol "$\prec$" means in this context "immediately preceding in the sequence".
- **Initial population/colony generation mechanism** – the analogy with Genetic Algorithms for populations is followed. The initial bee/ant/particle colony generation process consists in applying PermNivelAdj mechanism [43] generator to the initial individual. So, new individuals, at first iteration, are generated consisting on exchanging operations belonging to the same processing level, based on initial individual. This procedure avoid that schedule mechanism try to schedule an operation of a job/task in a machine prior to its previous operation has been completed.

In order to evaluate the performance of the proposed negotiation mechanism three Swarm Intelligent techniques were used ACS, PSO and ABC. The SI based algorithm has a certain number of specific parameters that need to be set appropriately. An extensive computational effort has been made for parameter tuning of the SI techniques in order to unsure identical computational effort. In Table 7 the different set of parameterization values used for each SI techniques is presented.

7.2. Discussion of results

We considered several academic benchmark problems as an effective evaluation framework, since multiple authors and diverse application areas have used them over the years. Additionally, they allow us an insight on global behavior and performance for a significant class of scheduling problems, which are our main objective.

This computational study aims to evaluate how the Negotiation Mechanism influences system's performance, by measuring the *makespan* (C_{max}) values obtained by each SI technique, before and after negotiation, as well as the machine occupation rate (U). Additionally, we intend to analyze the performance of ABC, ACS and PSO with the negotiation mechanism and the overall system's performance. Each SI algorithm (ACS, PSO and ABC) was computed $n=5$ simulations for each instance under analysis, leading to 100 simulations in total, for each SI technique. For each instance

resolution through each SI algorithm were retrieved the C_{max} values of solutions before and after the negotiation mechanism (on each simulation) and machine occupation rate (U)

Initially it is our proposal to validate the contribution of Negotiation Mechanism on the system performance: obtained results will be analyzed and its significance verified trough nonparametric statistics techniques [53], considering that small samples ($n=20$ instances).

After the general exploratory results analysis about the behavior of the scheduling system through negotiation based on three different SI techniques a significance analysis of the results has been performed to identify possible dependencies mainly on the identification of SI performance on the minimization of makespan (C_{max}) and on the maximization of machine occupation rate (U). The Friedman test [53] for related samples was used, in both cases, to compare the performance of the three SI techniques considering that the results are mutually independent (results within one instance do not influence the results within other instance) and within each instance the observations (C_{max} objective and Machine occupation) can be ranked.

7.3. Negotiation mechanism influence in the overall system performance

The boxplot from Fig. 7 allows the analysis of location, dispersion and asymmetry of data, making its synthesis by ACS, PSO and ABC, before and after the negotiation mechanism processing. From its analysis it is possible to conclude that there are not outliers or extreme values and about the influence of mechanism in the system performance, in terms of minimization of *makespan* (C_{max}), in the resolution of the analyzed instances of JSSP, when compared mean values obtained before negotiation mechanism application. However, it is not clear, from the graph analysis, the negotiation mechanism influence in the overall system performance.

From the analysis of statistical sampling summary based on C_{max} minimization (Table 8), it is possible to conclude that exist some statistic evidence on the advantage of negotiation mechanism in the overall system performance. This conclusion can be supported either by central tendencies and dispersion measures. This evidence can be observed even on median and dispersion indicators. Regarding variability, through standard deviation and interquartile range analysis it possible to conclude that ABC presents the lowest variability, followed by PSO and ACS.

Additionally, it is possible to refer that, in general the difference in performance before and after negotiation mechanism is less significant on ABC than with PSO and ACS which can be mainly explained by the fact that ABC was able to get good solutions before negotiation mechanism applying, and therefore the mechanism offered few improvements to the solution (Fig. 7 and Table 8). This conclusion converge for the assumption that a global solution for a scheduling problem may emerge from a community of resource agents solving locally their schedules and negotiating with other machine agents that shares some relations between the operations/jobs (e.g. a precedence relation).

To evaluate the significance of Negotiation Mechanism influence on the performance of scheduling system on the resolution of scheduling problems, the Wilcoxon Signed Ranks Test [53] has been used. From inferential statistical analysis it is possible to conclude about statistical evidence that NM influence the performance of the system with $\alpha=5\%$ of significance level. For all SI techniques performed, ABC ($p=0.004<\alpha$), PSO ($p=0.0005<\alpha$) ACS ($p=0.009<\alpha$) the null hypothesis H_0, that consider NM does not influence significantly the performance of system, was rejected with 95% of confidence level.

7.4. Minimization of makespan (C_{max})

The bar graph from Fig. 8 allows the analysis of location, dispersion and asymmetry of data, making its synthesis by ACS, PSO and ABC for C_{max}. From its analysis it is possible to conclude that there are not outliers or extreme values and about the advantage of ABC in the resolution of the analyzed instances of JSSP when compared mean values with PSO and ACS methods. ABC is more effective in terms of minimization of *makespan* (C_{max}), followed by PSO and ACS techniques. This evidence can be observed even on median and dispersion indicators. Regarding variability, through standard deviation and interquartile range analysis it possible to conclude that ABC presents the lowest variability, followed by PSO and ACS. The lower median for C_{max} and the lowest variability show some statistical tendency on advantage ABC performance when compared with PSO and ACS.

After the general exploratory results analysis about the behavior of the scheduling system through negotiation based on three different SI techniques a significance analysis of the results has been performed to identify possible dependencies mainly on the identification of SI performance on the minimization of *makespan* (C_{max}). The Friedman test for related samples was used, to compare the difference of performance obtained by SI techniques. Considering a significance level $\alpha=5\%$, it is possible to conclude that exist at least one SI technique whose performance is different from at least one of the other SI technique ($\chi^2_{(2)}=21.7$; $p<0.001$). Having concluded that there exist some significant differences the post-hoc statistical procedure LSD has been used to characterize these differences and validate which algorithm is really more effective. Thus, is it possible to conclude based on the statistical evidence that allows us to say, with a confidence level of 95% that ABC was the most effective when the optimization objective is the minimization of *makespan* (C_{max}).

Table 7
ACS, PSO and ABC parameterization.

ACO		PSO		ABC	
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
Evaporation rate	80%	Minimum velocity	−4	Size of population	50/100
Number of colonies	1	Maximum velocity	4	Maximum failure	1000/2000
Alpha	1	Minimum inertia	40%	Number of cycles	3000/4500
Beta	1	Maximum inertia	95%		
Stopping criteria	95	C1	2.0		
Number of ants per colony	50 (150)	C2	2.0		
		Lower limit	0		
		Upper limit	4		
		Stopping criteria	1000/1500		
		Particles number	150/250		

7.5. Machine occupation rate

Manufacturing systems efficiency can often be improved by identifying the real reasons and true extent of manufacturing downtime, constraints or bottlenecks caused by machine downtime. An important aspect of manufacturing organizations is related with the improvement of resource utilization.

In Fig. 9 the obtained results are illustrated, for the machine occupation rate (%), where the main objective is to create a scheduling plan that reduce idle times and production delays while maximizing resource utilization/occupation (U).

When analyzing machine occupation percentage results (Fig. 9) it is possible to conclude about general system performance through SI based algorithms. In average the system performance was more effective on the machine idle times reduction with ABC when compared with PSO and ACS. Fig. 9 displays the boxplot of the resource occupation (%) for each technique in analysis. In terms of median rate for resource occupation, the ABC technique showed the best performance, followed by PSO technique and finally the ACS. Regarding the variability, the ABC and ACS techniques are similar, although the ACS technique presents lower rate values compared with ABC. The PSO technique has the highest variability. Summarizing the information provided by Fig. 9 and Table 9, it is possible to conclude that: the high median rate and the smaller variability support statistical evidence on the advantage regarding the performance of the ABC on the maximization of machine occupation rate (%).

The Friedman test was used, to compare the difference of performance obtained by SI techniques. Considering a significance level $\alpha=5\%$, it is possible to conclude that exist at least one SI technique whose performance is different from at least one of the other SI technique ($\chi^2_{(2)}=8.532$; $p=0.014<\alpha$). Having concluded that there exist some significant differences the post-hoc statistical procedure LSD has been used to characterize these differences and

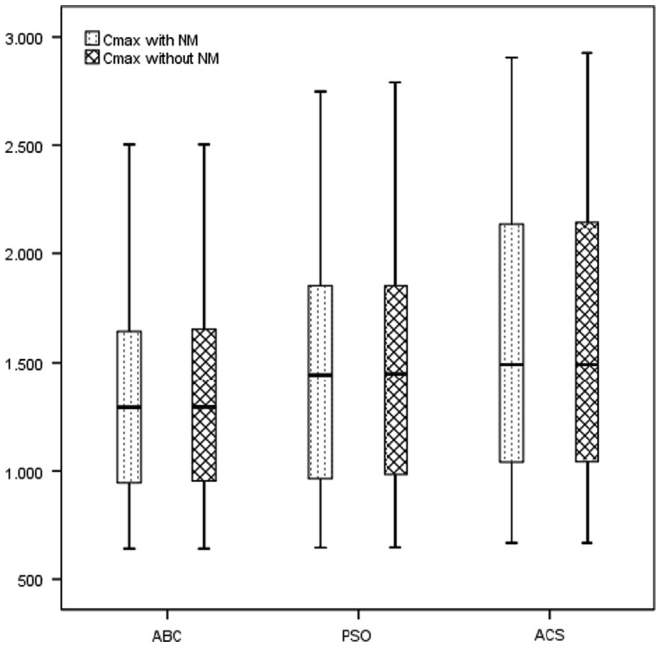


Fig. 7. Boxplot of the C_{max} values with and without Negotiation Mechanism using ABC, PSO and ACS.

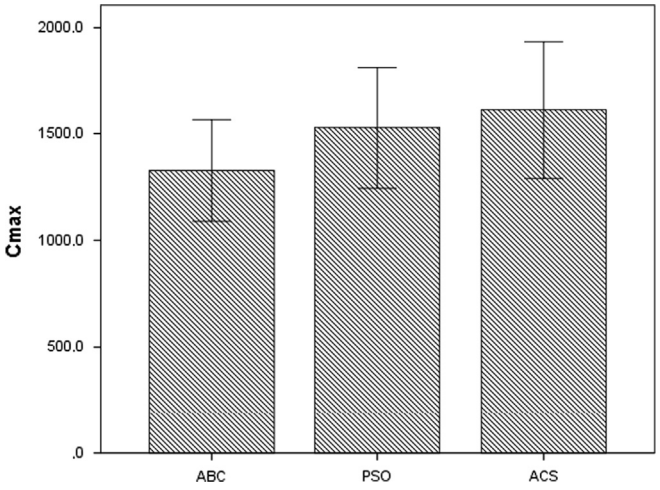


Fig. 8. Boxplot of the C_{max} minimization for ABC, PSO and ACS.

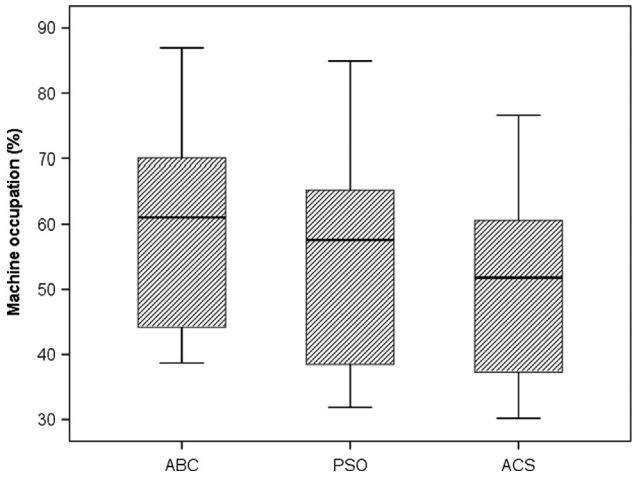


Fig. 9. Boxplot of the machine occupation % for ABC, PSO and ACS.

Table 8
Statistical sampling summary based on C_{max}.

	ABC		PSO		ACS	
	C_{max} with NM	C_{max} without NM	C_{max} with NM	C_{max} without NM	C_{max} with NM	C_{max} without NM
Mean	1327.040	1330.860	1528.460	1537.390	1611.700	1615.690
Median	1290.600	1292.900	1443.500	1448.800	1491.700	1491.700
Variance	258,753.796	260,105.323	371,849.285	371,113.800	473,759.682	477,087.076
Std. Deviation	508.678	510.0052	609.7945	609.1911	688.3020	690.7149
Interquartile range	727.850	728.4	924.3	897.9	1143.4	1153.3
Skewness	0.927	0.925	0.702	0.731	0.549	0.552
Kurtosis	0.412	0.407	−0.206	−0.117	−0.849	−0.834

Table 9
Statistical sampling summary based on machine occupation rate (U).

	ABC	PSO	ACS
Mean	59.3975	55.1285	51.2385
Median	61.0350	57.7650	51.7850
Variance	212.869	268.726	243.453
Std. Deviation	14.59003	16.39287	15.60299
Interquartile range	27.57	28.49	24.58
Skewness	0.313	0.514	0.358
Kurtosis	−0.690	−0.960	−1.072

validate which algorithm is really more effective. Thus, is it possible to conclude based on the statistical evidence that allows us to say, with a confidence level of 95% that ABC was the most effective when the optimization objective is the maximization of machine occupation.

Considering efficiency, most of the instances were solved in relatively short CPU time. For example, instance ABZ8 with 20 jobs and 15 machines took 2 s with ACS, 7 s with PSO and 19 s with ABC. In average the instances were solved in 5 s with ACS, 6 s with PSO, and 7 s with ABC.

8. Conclusions and further work

We proposed a novel framework that allows agents to coordinate their actions automatically, without human supervision, a requirement found in a wide variety of real world applications, such as the one proposed in this article.

The work reported in this paper is concerned with the resolution of real world scheduling problems by taking advantages from Swarm Intelligence paradigm, Negotiation in Multi-Agent Systems and Autonomic Computing. The main objective of this paper is the research of negotiation related issues for dynamic manufacturing systems in order to provide scheduling systems with collective intelligence and negotiation capabilities. A negotiation mechanism for dynamic scheduling based on Swarm Intelligence is proposed, where multiple self-interested agents can reach agreement over the operations exchange on competitive resources. Agents must collaborate to improve your local solution and global schedule. The proposed negotiation mechanism is able to analyze the scheduling plan generated by the Resource Agents and integrated by Coordinator Agent, and refine it by idle times reducing.

Experimental analysis was performed in order to validate the influence of the SI technique and negotiation mechanism in the system performance. From the obtained results it was possible to conclude about statistical evidence that negotiation mechanism influence significantly the overall system performance and about advantage of Artificial Bee Colony on effectiveness of *makespan* minimization and on the machine utilization maximization.

Future work includes the refinement of the Negotiation Mechanism, and the validation of the proposed system and negotiation mechanisms under dynamic environments subject to several random perturbations and imponderables.

Acknowledgements

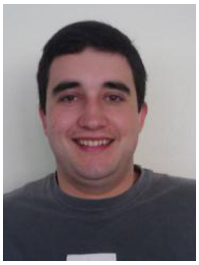
This work is supported by FEDER Funds through the “Programa Operacional Factores de Competitividade – COMPETE” program and by National Funds through FCT “Fundação para a Ciência e a Tecnologia” under the Project: FCOMP-01-0124-FEDER-PEst-OE/EEI/UI0760/2011 and PTDC/EME-GIN/109956/2009.

References

[1] M. Pinedo, Scheduling: Theory, Algorithms, and Systems, 4th edition, Springer, New York, NY, 2012.
[2] K. Baker, D. Trietsch, Principles of Sequencing and Scheduling, Wiley, Hoboken, NJ, 2007.
[3] A. Madureira, I. Pereira, Self-optimization for dynamic scheduling in manufacturing systems, in: Khaled Elleithy, et al., (Eds.), Technological Developments in Networking Education and Automation, Springer, Netherlands, 2010, pp. 421–426.
[4] A. Madureira, I. Pereira, Intelligent bio-inspired system for manufacturing scheduling under uncertainties, Int. J. Comput. Inf. Syst. Ind. Manage. Appl. 3 (2011) 72–79.
[5] F. Xhafa, A. Abraham, Metaheuristics for Scheduling in Industrial and Manufacturing Applications Series: Studies in Computational Intelligence, Springer, 2008.
[6] P. Siarry, Z. Michalewicz, Advances in Metaheuristics for Hard Optimization, Springer-Verlag, Berlin Heidelberg, 2008. (Natural Computing Series).
[7] EMA. Practical Autonomic Computing: Roadmap to Self-Managing Technology – A White Paper Prepared for IBM. Enterprise Management Associates, 2006.
[8] J. Kephart, D. Chess, The vision of autonomic computing, Computer 36 (2003) 41–50.
[9] M. Dorigo, Swarm Intelligence, Springer, New York, 2007.
[10] J. Kennedy, Swarm Intelligence, Handbook of Nature-Inspired and Innovative Computing, Springer-Verlag, Berlin Heidelberg, 2006.
[11] Y. Sun, L. Zhang, X. Gu, A hybrid co-evolutionary cultural algorithm based on particle swarm optimization for solving global optimization problems, NeuroComputing 98 (2012) 76–89.
[12] M. Luck, P. McBurney, O. Shehory, S. Willmoth, Agent Technology: Computing as Interaction. A Roadmap for Agent-Based ComputingAgentLink III, 2005.
[13] M. Wooldridge, N.R. Jennings, Intelligent agents: theory and practice, Knowl. Eng. Rev. 10 (2) (1995).
[14] N.R. Jennings, An agent-based approach for building complex software systems, Commun. ACM 44 (4) (2001) 35–41.
[15] L.G. Telser, A Theory of Effective Cooperation and Competition, Cambridge University Press, 1987.
[16] M. Allen-Williams, Coordination in Multi-Agent Systems (Ph.D. thesis), University of Southampton, 2005.
[17] M. Dorigo, Optimization, Learning and Natural Algorithms (Ph.D. thesis), Politecnico di Milano, Italy, 1992.
[18] M. Dorigo, L.M. Gambardella, Ant Colony System: a cooperative learning approach to the traveling salesman problem, IEEE Trans. Evol. Comput. 1 (1) (1997) 53–66.
[19] M. Dorigo, M. Birattari, T. Stützle, Ant colony optimization – artificial ants as a computational intelligence technique, IEEE Comput. Intell. Mag. 1 (2006) 28–39.
[20] A. Madureira, D. Falcão, I. Pereira, Ant colony system based approach to single machine scheduling problems – weighted tardiness scheduling problem, in: Proceedings of International Fourth World Congress on Nature and Biologically Inspired Computing (NaBIC'12), 2012, pp. 86–91, .
[21] A. Madureira, I. Pereira, A. Abraham, Towards scheduling optimization through artificial bee colony approach, in: Proceedings of International Fifth World Congress on Nature and Biologically Inspired Computing (NaBIC'13), 2013, pp. 252–257.
[22] J. Kennedy, R.C. Eberhart, Particle swarm optimization, in: Proceedings of the IEEE International Conference Neural Networks, 1995, pp. 1942–1948.
[23] D. Karaboga, An Idea Based on Honey Bee Swarm for Numerical Optimization, Technical Report TR06, Erciyes University, Engineering Faculty, Computer Engineering Department, 2005.
[24] D.T. Pham, A. Ghanbarzadeh, E. Koc, S Otri, S. Rahim, M. Zaidi, The Bees Algorithm, Manufacturing Engineering Centre, Cardiff University, United Kingdom, 2005.
[25] D. Karaboga, B. Akay, A modified Artificial Bee Colony (ABC) algorithm for constrained optimization problems, Appl. Soft Comput. 11 (2011) 3021–3031.
[26] N.R. Jennings, P. Faratin, A.R. Lomuscio, C. Sieera, M. Wooldridge, Automated negotiation: prospects, methods and challenges, Int. J. Group Decision Negotiation (GDN2000) 10 (2) (2000) 199–215.
[27] B. Horling, V. Lesser, A survey of multi-agent organizational paradigms, Knowl. Eng. Rev. 19 (4) (2005) 281–316.
[28] H.S. Nwana, Software agents: an overview, Knowl. Eng. Rev. 11 (3) (1996) 205–244.
[29] H.S. Kim, J.H. Cho, Supply Chain Formation Using Agent Negotiation, Decision Support Systems, 2010.
[30] G. Weiss, Multiagent Systems – A Modern Approach to Distributed Artificial Intelligence, The MIT Press, Cambridge, MA, 1999.
[31] F. Bellifemine, G. Caire, D. Greenwood, Developing Multi-Agent Systems with JADE, John Wiley & Sons, West Sussex, England, 2007. (Wiley Series in Agent Technology).
[32] H. Nwana, L. Lee, N. Jennings, Coordination in software agent systems, BT Technol. J. 14 (4) (1996) 79–88.
[33] D. Pruitt, Negotiation Behavior, Academic Press, New York, 1981.
[34] M. Beer, M. d'Inverno, N. Jennings, M. Luck, C. Preist, M. Schroeder, Negotiation in multi-agent systems, Knowl. Eng. Rev. 14 (3) (1999) 285–289.
[35] R. Smith, The Contract Net Protocol: high level communication and control in a distributed problemsolver, in: Proceedings of the First International

Conference on Distributed Computing Systems, IEEE, New York, 1979, pp. 185–192.
[36] M. Wooldridge, An Introduction to Multiagent Systems, John Wiley and Sons Ltd., West Sussex, England, 2002.
[37] T. Sandholm, eMediator: a next generation electronic commerce server, Comput. Intell. 18 (4) (2002) 656–676 (Special Issue on Agent Technology for Electronic Commerce).
[38] N. Jennings, An agent-based approach for building complex software systems, Commun. ACM 44 (4) (2001) 35–41.
[39] I. Zattar, J. Ferreira, J. Rodrigues, C. Sousa, A multi-agent system for the integration of process planning and scheduling using operation-based time-extended negotiation protocols, Int. J. Comput. Integrated Manuf. 23 (5) (2010) 441–452.
[40] A. Singh, D. Juneja, A.K. Sharma, Introducing Trust Establishment Protocol in Contract Net Protocol, in: Proceedings of the International Conference on Advances in Computer Engineering, 2010.
[41] H. Kim, J. Cho, Supply Chain Formation Using Agent Negotiation, Decision Support Systems, 2010.
[42] S. Adhau, M. Mittal, A. Mittal, A multi-agent system for distributed multi-project scheduling: an auction-based negotiation approach, Eng. Appl. Artif. Intell. (2012).
[43] A.M. Madureira, Meta-Heuristics Application to Scheduling in Dynamic Environments of Discrete Manufacturing (PhD Dissertation), University of Minho, Braga, Portugal, 2003 (in portuguese).
[44] I. Pereira, A. Madureira, Self-Optimization module for Scheduling using Case-based Reasoning, Applied Soft Computing, Elsevier, 2012. (in press).
[45] A. Madureira, C. RamosS.C. Silva,A Coordination Mechanism for Real World Scheduling Problems Using Genetic Algorithms, 2002 IEEE World Congress on Computational Intelligence, Hawai (EUA), 2002.
[46] C. Xueguang, S. Haigang, Further Extensions of FIPA Contract Net Protocol: Threshold plus DoA, ACM Symposium on Applied Computing, 2004.
[47] OR-Library – ⟨http://people.brunel.ac.uk/~mastjjb/jeb/info.html⟩.
[48] H. Fisher, G.L. Thompson, Probabilistic learning combinations of local job-shop scheduling rules, Industrial Scheduling, Prentice Hall, Englewood Cliffs, New Jersey (1963) 225–251.
[49] S. Lawrence, Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques, Graduate School of Industrial Administration, Carnegie-Mellon University, Pennsylvania, 1984.
[50] J. Adams, E. Balas, D. Zawack, The shifting bottleneck procedure for job shop scheduling, Manag. Sci. 34 (1988) 391–401.
[51] R.H. Storer, S.D. Wu, R. Vaccari, New search spaces for sequencing instances with application to job shop 38 (1992) 1495–1509Manage. Sci. 38 (1992) 1495–1509.
[52] T. Yamada, R. Nakano, A genetic algorithm applicable to large-scale job-shop instances, in: R. Manner, B. Manderick (Eds.), Parallel Instance Solving from Nature 2, North-Holland, Amsterdam, 1992, pp. 281–290.
[53] W.J. Conover, Practical Nonparametric Statistics, 3rd edition, Wiley Series in Probability and Statistics, 1999.

Ana Madureira was born in Moçambique, in 1969. She got her BSc degree in Computer Science Engineering in 1993 from ISEP, Master degree in Electrical and Computers Engineering–Industrial Informatics, in 1996, from FEUP, and the Ph.D. degree in Production and Systems, in 2003, from University of Minho, Portugal. She is Vice-Chair of IEEE Portugal Section and IEEE-CIS Portuguese chapter. She became IEEE Senior Member in 2010. Currently she is Coordinator Professor at the School of Engineering–Polytechnic of Porto (ISEP/IPP) and Ph.D. researcher of the GECAD Research Group. In the last few years, she was author of more than seventy scientific papers in scientific conference proceedings, journals and books



Ivo Pereira was born in 1984. His BSc degree in Computer Science Engineering was obtained in 2007 and his MSc degree was concluded in 2009, both in the Institute of Engineering–Polytechnic of Porto. Currently he is a Ph.D. student in University of Trás-os-Montes e Alto Douro. He is also a researcher of GECAD Research Group, where participated in three R&D projects. In the last few years, Ivo was author and co-author of more than twenty scientific papers in conference proceedings, journals and books. His main scientific areas of interest are Meta-Heuristics, Parameter Tuning, Machine Learning, Scheduling, and Intelligent Systems

Pedro Pereira was born in 1970. Completed his BSc degree in Food Engineering in 1994, at ESB-UCP, its master in Industrial Management at ISEP in 2010, and two post-graduate courses. Played for several year activities relates to quality, food, environmental, health and safety assurance and systems certification. Currently work as consultant in Food Safety and Health and Safety areas, as well as professional training. His main scientific areas of interest are Production Planning and Control, Quality Management, Sustainable manufacturing, Green economy, Green manufacturing.

Ajith Abraham received Ph.D. in Computer Science from Monash University, Melbourne, Australia. He is currently the Director of Machine Intelligence Research Labs (MIR Labs), Scientific Network for Innovation and Research Excellence, USA, which has members from more than 100 countries. He has a worldwide academic and industrial experience of over 23 years. He works in a multidisciplinary environment involving machine intelligence, network security, various aspects of networks, e-commerce, Web intelligence, computational grids, data mining, and their applications to various real-world problems. He has numerous publications/citations (h-index 54) and has also given more than 70 plenary lectures and conference tutorials in these areas. He is an Associate Editor of Neurocomputing, since 2003. Since 2008, he is the Chair of IEEE Systems Man and Cybernetics Society Technical Committee on Soft Computing and a Distinguished Lecturer of IEEE Computer Society representing Europe (since 2011). He is the founder of several IEEE technically sponsored conferences, which are now annual events for over a decade. More information at: http://www.softcomputing.net